OMNIA

OUR TEAM:

Ismaeel Rahaman
u20427906
u20427906@tuks.co.za
065 814 5362

Nawailah Tarmohamed
u21437972
u21437972@tuks.co.za
067 319 1637

Deshlan Pillay
u21481084
u21481084@tuks.co.za
067 127 6684

Sameer Ghela
u21445142
u21445142@tuks.co.za
066 239 5477

Sashin Gounden
u20487062
u20487062@tuks.co.za
072 520 1968

From left to right: Sameer Ghela, Ismaeel Rahaman (Group leader), Nawailah Tarmohamed, Sashin Gounden, and Deshlan Pillay.

Team 7: Omnia	Project: Water Resort administration system	System Name: Hydrotech

Iteration 7 – Technical Use Case Narratives

This document will define our Technical Narratives for all the use cases of the HydroTech System. Including the following Subsystems: User, Client, Accommodation, Ticketing, Events, Admin, Inventory, and Reporting.

Table of Contents

1. Document Introduction

This section contains all the technical use case diagrams which describe the standard procedures of actors alongside the system response, for each use case.

2. Technical Use Case Narratives

2.1. Subsystem 1 – User

USE CASE NAME:	Login	USE CASE TYPE
USE CASE ID:	1.1	Business Requirements:o
PRIORITY:	High	System Analysis: o
SOURCE:	Platinum Island resort	System Design: þ
PRIMARY BUSINESS ACTOR:	User	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes an event of the user logging into the system. The use case begins when the user selects the option for logging in. The system will display a login screen on which the user will have to enter their email address and their passwords in the respective fields. The system will validate their login credentials by comparing it to the ones stored in the database, and then the user will be able to login.	
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • User must be registered on the system.	
TRIGGER:	The user wishes to login to the system using their credentials	

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The user requests to login to the system using their credentials		**2.** The system responds by loading the '*Login*' screen which contains the following details: – An image of the resort covering half of the login screen – Label: "Welcome to platinum island resort" – Heading: "Log into your account" – Label: "Username"

			– Input field for the email address – Label: “Password” – Label: “Forgot password?” which is indented to the right of the password label – Input field for the user’s password – Submit button with the text “Log in”. – Label: “Do not have an account?” – Label: “Sign up” to the right of the previous label. [The “Sign Up” and “Forgot Password?” hyperlinks are enabled. The “Login” button is disabled and becomes enabled when all input fields have a value and have been validated using Angular Frontend validation]
	3. The user enters the email address and password in the appropriate fields.		**4.** The system will use the angular frontend to validate the entered information in the email and password fields by: – The input fields are not left empty. – The email address contains the “@” sign and the characters representing the domain name. [ALT]
	5. The user clicks on the “Log in” button. [ALT]		**6.** The system will send through the entered email address to the .Net controller which will call a SQL_Read query that will search for the entered email address according to the *UserName* and *Password* attributes in the **User** entity to ensure that the entered email and password address matches the information

			in the entity. Then using the *Role_ID* [FK] from the **User** entity the user's credentials are confirmed, and they are granted role specific access by referencing the permissions linked to the specific role on the system. The **Role** entity has the following attributes: o Role_ID [PK] o Name o Description The **Permission** entity has the following attributes: o Permission_ID [PK] o Name o Description A role and its permission are linked in the **RolePermission** entity with the following composite key: o Role_ID [PK,FK] o Permission_ID [PK,FK] The system will also send an SQL_Insert to the **Audit_Log** entity with the following attributes: o AuditLog_ID (int) [PK] o Employee_ID (int) [FK] o Date (datetime) o Time (time)
			7. The system successfully logs the user in and redirects them to the appropriate page.
ALTERNATE COURSES:	[ALT] **Step 4**: The username and password field is invalid. **Return to Step 3**.		

	[ALT] **Step 5a**: The user forgot the password. The user clicks on the "Forgot password?" hyperlink. The system invokes use case "1.3 Forgot password". **Terminate use case**. [ALT] **Step 5b**: The user does not have an account yet; the user will select the "Sign up" option. The system will load and display the "Register screen" which contains the following components: – Heading: "Register your account" – Input field with the Placeholder "First Name" – Input field with the Placeholder "Last Name" – Input field with the Placeholder "Cellphone number" – Input field with the text "Email address" – Label: "For a strong password use a combination of upper- and lower-case letters, numbers, and special characters. – Input field with the Placeholder "Password" – Submit button with the text "Sign up". – Label: "Already have an account" – Label: "Log in" to the right of the above label. [The sign-up button is disabled until all the information above is entered correctly, once that is complete the button will be enabled, and the user will be allowed to submit the information] The system will invoke use case "2.1 Register Client". [ALT] **Step 6a**: The username does not match an instance in the database. **Return to step 3**. [ALT] **Step 6b**: The password entered by the user does not match the *UserName* attribute found within the **User** entity record does not match an. **Return to step 3**. [ALT] **Step 7**: The system failed to grant the user access to the system. **Return to step 3**.
CONCLUSION:	The use case concludes when the user is logged into the system.
POST-CONDITION:	The user will be redirected to their appropriate page according to their role.
BUSINESS RULES:	Only registered users can login.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	Password in database must be hashed.
ASSUMPTIONS:	None

OPEN ISSUES:	None

USE CASE NAME:	Log out	USE CASE TYPE
USE CASE ID:	1.2	Business Requirements:o
PRIORITY:	High	System Analysis: o
SOURCE:	Platinum Island resort	System Design: þ
PRIMARY BUSINESS ACTOR:	User	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes the event where a user logs out of the system. The use case begins with the user requesting the log out of the system, the user clicks on the logout button on the screen. The system will respond by logging the user out. The use case concludes when the user is successfully logged out and redirected to the login page where they will be required to enter their details again.	
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • User must be logged into the system.	
TRIGGER:	The user wishes to log out of the system.	
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:
	1. The user wishes to logout of the system.	**2.** The HydroTech system will capture the user's logout request.
		3. The system will log the user out of the system. [ALT]
		4. The system will route the user to the login page with the following information: – An image of the resort covering half of the login screen – Label: "Welcome to platinum island resort" – Heading: "Log into your account"

		– Label: “Username” – Input field for the email address – Label: “Password” – Label: “Forgot password?” which is indented to the right of the password label – Input field for the user’s password – Submit button with the text “Log in”. – Label: “Do not have an account?” – Label: “Sign up” to the right of the previous label. [The “Sign Up” and “Forgot Password?” hyperlinks are enabled. The “Login” button is disabled and becomes enabled when all input fields have a value and have been validated using Angular Frontend validation]
ALTERNATE COURSES:	[ALT] **Step 3**: The system encountered an error while logging the user out and ask the user to attempt to log out again. Return to Step 1	
CONCLUSION:	The use case concludes when the user is successfully logged out of the system	
POST-CONDITION:	The user will be redirected to the log in page. The user’s accessibility has been removed from the system until they login again.	
BUSINESS RULES:	None	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None	
ASSUMPTIONS:	None	
OPEN ISSUES:	None	

USE CASE NAME:	Forgot password	USE CASE TYPE	
USE CASE ID:	1.3	Business Requirements:o	
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	User		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where a user forgets his password and they wish to change it. The use case starts when the user clicks on the forgot password hyperlink, they are then routed to the '*Forgot password*' screen where they will be prompted to enter their phone number, once the phone number is entered, the user will be sent an OTP which will have to be entered for the password fields to be enabled. Once the password is entered the user will be prompted to enter the password again in the confirm password text box. The use case concludes when the password is successfully reset and stored in the database.		
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • User must be registered		
TRIGGER:	The user wishes to set a new password as they have forgotten their current password.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:

	1. The user selects the forgot password option on the '*Login screen'.*		**2.** The Hydrotech system will respond by loading the '*Forgot password screen*' with the following information: – Label: "Please enter your phone number, you will receive an OTP in your inbox, once the OTP is submitted you will be allowed to change your password" – Heading "Forgot password" – Label: "phone number" – Input field with the placeholder: "Enter your phone number". – Button with the text "Enter". – Label: "OTP" – Input field with the placeholder: "Enter your OTP". – Button "Submit" – Label: "Password" – Input field with the placeholder: "Enter your new password". – Label: "Confirm Password" – Input field with the placeholder: "Confirm your new password". – Button with the text: "Update password" [The "Enter" button is enabled. The "Password" and "Confirm Password" fields are disabled and become enabled when the user enters the OTP that he received from the SMS. The "update password" button is disabled and becomes enabled once the fields of password and confirm password validated using

			Angular Frontend validation]
	3. The user enters their phone number in the input field.		**4.** The system validates the phone number to ensure that a phone number was entered and belongs to the user by searching for it in the User entity and validates that the phone number was entered in the correct format. [ALT]
			5. When the phone is located and verified the system will generate an OTP (One Time Password) and send it to the phone number that was entered.
			6. The system will prompt the user to enter the OTP sent to their phone number.
	7. The user enters the OTP they received.		**8.** The system validates the entered OTP by matching it with the OTP generated. The system successfully validated the entered OTP. [ALT]

			9. The system prompts the user to enter a new password and to confirm the password.
	10. The user enters a new password and confirms it		**11.** The system will validate the entered password to ensure that it is secure, and that the password matches the confirmed password entered. The system will also ensure that the new password does not match the user's old password by hashing the entered password and comparing it to the Password attribute in the User table. The system successfully validated the entered password. [ALT]
			12. The system updates the user's newly created password in the User table's Password attribute after hashing the new password.
			13. The system will route the user back to the login page
ALTERNATE COURSES:	[ALT] **Step 4**: The username or the phone number was invalid, or username was not found in database. Return to step 3. [ALT] **Step 8**: The OTP entered was invalid. Return to step 6. [ALT] **Step 11**: The entered password and confirmed password either do not match or do not comply with the security measures or the password is the same as the user's old password. Return to Step 10		
CONCLUSION:	The case ends when the user's password has been updated successfully and the system has returned to the login screen		
POST-CONDITION:	The user's password has been updated and they are able to log into the system with their newly created password.		
BUSINESS RULES:	• Password needs to be at least 8 characters long. • Password must have a combination of uppercase and lowercase characters.		

	• Password must include at least one number. • New password cannot be the same as the old password.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Update Password	USE CASE TYPE
USE CASE ID:	1.4	Abstract:¨
PRIORITY:	Medium	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	User (PBA)	
DESCRIPTION:	This use case describes the event where a user wants to change their saved password on the system. The use case begins when the user first confirms the email, and then proceeds to update their password on the system. The system will first request the user to enter their current password, which will be validated, and then the user will be able to enter a new password. The use case concludes when the system has successfully updated the user's password and displays a success notification to the user.	
PRE-CONDITION:	• The user must be logged into the system. • The system has loaded the *'View Profile'* screen. • The user scrolled to the "Account details" block.	

	Actor Actions	System Response
TYPICAL COURSE OF EVENTS:		**1.** The system will prompt the user to enter their current password.
	2. The user will enter their current password, new password, and confirmation of the new password and clicks the "Update Password" button. [ALT]	**3.** The system checks to see if the new password, old password, and password fields are input correctly and adhere to security specifications. The Password attribute in the User table is used by the system to compare the current password to the password that is already stored there. The system then checks to make sure the password entered by the user and the password currently stored for the user are different. [ALT]
		4. The system updates the user's password in the User table with the new password that they entered on the system after hashing the new password and displays a message indicating that the password was successfully updated.

ALTERNATE COURSES:	[ALT] **Step 3**: The user does not want to not change their password. **Terminate use case**. [ALT] **Step 4**: Password does not meet security requirements; password was not confirmed, current password does not match the current password stored on the system or new password matches old password. **Return to step 3**.
POST-CONDITION:	The password is updated and saved to the database.

USE CASE NAME:	Update Email Address	USE CASE TYPE
USE CASE ID:	1.5	Abstract:¨
PRIORITY:	Medium	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	User (PBA)	
DESCRIPTION:	This use case describes the event where a user wants to change their email address on the system. The use case begins when the user first confirms the email, and then proceeds to update their email address on the system. The use case concludes when the system has successfully updated the user's email address.	
PRE-CONDITION:	• The user must be logged into the system. • The system has loaded the *'View Profile'* screen for client or the "View Account" for the Admin. • The user scrolled to the "Account details" block.	

	Actor Actions	System Response
TYPICAL COURSE OF EVENTS:		**1.** The system will prompt the user to enter their current email address.
	2. The user will enter their current email address in the text box labelled "Current Email Address", and the new email address in the text box labelled "New Email Address" and clicks the "Update Email Address" button. [ALT]	**3.** The *Email* attribute in the **User** table is used by the system to compare the current email address to the email address that is already stored there. The system then checks to make sure the email address entered by the user contains @ symbol and that the email address currently stored for the user is different [ALT]
		4. The system updates the user's email in the **User** table with the new email that they entered on the system.
ALTERNATE COURSES:	[ALT] **Step 3**: The user does not want to not change their email address. **Terminate use case**. [ALT] **Step 4a**: New email matches' old email. **Return to step 3**.	

	[ALT] **Step 4b**: The email address does not contain “@” symbol. **Return to step 3**.
POST-CONDITION:	The email is updated and saved to the database.

2.2. Subsystem 2 – Client

USE CASE NAME:	Register Client	USE CASE TYPE	
USE CASE ID:	2.1	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when the client clicks on the “Sign Up” link on the “Login” screen which will result in the client being redirected to the “Register client” screen and the client will enter the relevant information on the screen. The use case ends once the client clicks the sign-up button, and their details are captured. A modal will then open up where the client will be required to enter the OTP that is sent to their number. The client is then redirected back to the login screen.		
PRE-CONDITION:	• The system displays the register client screen.		
TRIGGER:	The client requests to register an account on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The client will request to register an account on the system by clicking on the “Sign Up” link on the login screen.		**2.** The system will respond by displaying the “Register Client” screen. The following are visible on the screen: **Labels**: Heading Label: Register your Account. Body Label: For a strong password use a combination of upper- and lower-case

			letters, numbers, and special characters. **Textboxes**: Text Input Control: Placeholder: First name. Text Input Control: Placeholder: Last name. Text Input Control: Placeholder: Cell phone number. Text Input Control: Placeholder: Email address. Text Input Control: Placeholder: Password. **Buttons**: “Sign Up” (Register Button) “Cancel” (Return to login Button) The system prompts the client to add their details.
	3. The client will enter their details in the corresponding fields and click the” sign up” button. [ALT]		**4.** The system captures and validates the new client details entered by comparing the entered information with the information in the **Client** table to ensure that the client does not already exist and validates the fields using Angular to ensure that the information is correct and that all fields specified as required have been inputted. [ALT]
			5. The system will respond by loading the OTP modal screen with an input field for the OTP. The system prompts the client to add the OTP.
	6. The client will enter the OTP that is sent to their number in the required field in the pop-up modal. [ALT]		**7.** The system captures and validates the OTP details entered and validates the fields using Angular to ensure that the information is correct and that all fields specified as required have been inputted.

			8. The system will capture the information entered by the client and will populate the **Client** entity with the attributes: ○ First Name ○ Last Name ○ Email Address ○ Cell phone number The Client object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
			9. The system will make use of an SQL_Insert query in the controller to create the new record within the **Client Entity**: ○ Client_ID(PK) (value of the previous Client_ID, incremented by 1) ○ User_ID(FK) ○ ClientName(string) ○ ClientSurname(string) ○ ClientEmail(string) ○ ClientPhone(int) The system will make use of an SQL_Insert query in the controller to create the new record within the **User** Entity: ○ User_ID(PK) (value of the previous User_ID, incremented by 1) ○ Role_ID(FK) ○ UserName(varchar20) ○ Password (varchar20) [ALT] The client is redirected to the Login screen.
ALTERNATE COURSES:	[ALT] **Step 3**: The client clicks on the cancel button. **The use case is terminated**. [ALT] **Step 4**: The system detects that the information fields entered by the client was either left blank or was entered in incorrect format. The system will display error messages directing to which input field		

	has the error, prompting the client to re-enter their details. **Return to Step 3.** [ALT] **Step 6**: Invalid OTP. Request to Re-Send the OTP via the "Re-Send OTP" Button. [ALT] **Step 9**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).
CONCLUSION:	The client is now registered on the system and can log in.
POST-CONDITION:	The client is registered on the system and a new client record is added to the Client Table.
BUSINESS RULES:	None
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	View Profile		USE CASE TYPE
USE CASE ID:	2.2		Business Requirements: ☐
PRIORITY:	High		System Analysis: ☐
SOURCE:	Platinum Island Resort		System Design: ☑
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when the client navigates to the navigation bar on the top of the client home page and clicks on the user icon on the client home page screen and chooses the View Profile option on the dropdown. On this view the client will be able to view their details and update their details on this page. The use case ends when the system displays the view profile screen.		
PRE-CONDITION:	• The client is logged onto the system. • The system displays the client home page screen.		
TRIGGER:	The client requests to view the view profile screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Client** Entity which has the following attributes: ○ Client_ID (PK) ○ User_ID(FK) ○ ClientName (varchar 20)

			○ ClientSurname (varchar 20) ○ ClientEmail (varchar 50) ○ ClientPhone (varchar 10) ○ ClientIDNum (varchar 13)
			2. The system will load the 'View Profile' screen with the following elements: – Heading Label: "View Profile" – Heading Label: "Personal Details" – Label with text "First Name" – Textbox populated with Name attribute. – Label with text "Last Name" – Textbox populated with Last Name attribute. – Label with text "Cell Phone Number" – Textbox populated with cellphone number attribute. – Label with text "ID Number" – Textbox populated with ID Number attribute. – Button with text "Update Personal Details". – Heading label with text "Account Details" – Label with text "Current Email Address" – Textbox populated with *ClientName* attribute from **Client** Entity. – Label with text "New Email Address" – Empty text box – Button with text "Update Email Address"

			– Label with text "Current Password" – Textbox populated with *Password* attribute from the **Client** Entity. – Label with text "New Password" – Empty Textbox – Label with text "For a strong password use a combination of upper- and lower-case letters, numbers and special characters." – Label with text "Re-enter New Password" – Empty Textbox – Button with text "Update Password" – Button with text "Return to Home Page".
	3. The Client clicks the "Return to Home Page" Button. [ALT]		**4.** The system returns the client to the "Client Home Page".
ALTERNATE COURSES:	[ALT] **Step 3a**: The client requests to update client details. The system extends to Use Case 2.3 "Update Client Details". [ALT] **Step 3b**: The client requests to update their password. The system extends to Use Case 1.4 "Update Password". [ALT] **Step 3c**: The client requests to update their password. The system extends to Use Case 1.5 "Update email address".		
CONCLUSION:	The use case concludes once the client can view the View Profile screen.		
POST-CONDITION:	The client will be able to update their details on the "View Profile" screen.		
BUSINESS RULES:	None		

IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Update Client Details	USE CASE TYPE	
USE CASE ID:	2.3	Abstract:	☐
PRIORITY:	High	Extension:	x
SOURCE:	Platinum Island Resort		
PRIMARY BUSINESS ACTOR:	Client		
DESCRIPTION:	The use case begins when the client wants to update their details and selects the option to update their details by clicking “Manage Personal Details” button which confirms the changes made to any of the fields under the personal details section of the “View Profile” screen. In addition, the client can enter a new email address in the corresponding field and then confirm the change by clicking on the “Update Email Address” button. The client can also enter a new password in the required fields and confirm the updating of this information by clicking the “Update Password” button.		
PRE-CONDITION:	• The client is logged onto the system. • The system displays the view profile screen.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system will respond by loading the ‘*View Profile*’ screen with the following elements: – Heading Label: “View Profile” – Heading Label: “Personal Details” – Label with text “First Name” – Textbox populated with Name attribute. – Label with text “Last Name” – Textbox populated with Last Name attribute. – Label with text “Cell Phone Number” – Textbox populated with cellphone number attribute. – Label with text “ID Number” – Textbox populated with ID Number attribute.

			– Button with text “Update Personal Details” – Heading label with text “Account Details” – Label with text “Current Email Address” – Textbox populated with Email Address attribute. – Label with text “New Email Address” – Empty text box – Button with text “Update Email Address” – Label with text “Current Password” – Textbox populated with password attribute. – Label with text “New Password” – Empty Textbox – Label with text “For a strong password use a combination of upper- and lower-case letters, numbers and special characters.” – Label with text “Re-enter New Password” – Empty Textbox – Button with text “Update Password” – Button with text “Return to Home Page”. The system prompts the client to enter their updated details.
	2. The user updates the details they wish to update and submits the changes by clicking on the “Update Personal Details” button. [ALT]		**3.** The system captures and validates the updated client details entered by comparing the entered information with the information in the **Client** table to ensure that the client does not already exist and validates the fields using Angular to ensure that the information is correct and that all fields specified as required have been inputted. [ALT]

			4. The system will capture the information entered by the client and will populate the **Client** entity with the attributes: – First Name – Last Name – Email Address – Cell phone number The Client object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
			5. The system will make use of an SQL_update query in the controller to update the specific record within the **Client Entity**: ○ Client_ID(PK) (value of the previous Client_ID, incremented by 1) ○ User_ID(FK) ○ ClientName(string) ○ ClientSurname(string) ○ ClientEmail(string) ○ ClientPhone(int) [ALT]
ALTERNATE COURSES:	[ALT] **Step 2**: The client clicks the “Return to Home Page” button and is redirected to the Client Home Page. [ALT] **Step 3**: The system detects that the information fields entered by the client was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the client to re-enter their details. Return to Step 4. [ALT] **Step 5**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).		
POST-CONDITION:	A client’s record has been updated in the Client entity.		

USE CASE NAME:	View Resort	USE CASE TYPE	
USE CASE ID:	2.4	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when the client clicks on the “Make Booking” option on the client home page and is redirected to the “View Resort” screen. On this view the client will be presented with three different options of what type of booking they would like to make. In addition, the client will be able to view general information about the resort and its facilities and services.		
PRE-CONDITION:	• The client is logged onto the system. • The system displays the client home page screen. • The client clicks the “Make Booking” option on the client home page screen.		
TRIGGER:	The client requests to view the view resort screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system will respond by displaying the “View Resort” screen. The following are visible on the screen: ○ Card with text “Have a splash-tacular day. Get your water park ticket booked!”. ○ Button with text “Book Now” ○ Card with text “The ultimate water park escape awaits! Book a room for the perfect water park getaway:

			○ Button with text “Book Now”
			○ Card with text “Splash into the festivities! Reserve your spot to host an event!”
			○ Button with text “Book Now”
			○ Heading Label: “Platinum Island Resort Facilities & Services”
			○ Paragraph with description of the resort facilities and services with the following points:
			  ○ Comfortable accommodation options with scenic views
			  ○ Water park with thrilling water slides and attractions
			  ○ Relaxing pools and lazy river for a leisurely experience
			  ○ Exciting events and entertainment for all ages
			  ○ Delicious dining options with a variety of cuisines
			  ○ Spa and wellness center for ultimate relaxation
			  ○ Convenient amenities including parking and free Wi-Fi
			  ○ Professional event hosting services for special occasions

			○ Day visit passes for non-residents ○ Friendly and attentive staff to ensure a memorable experience.
	2. The client will hover on the card and click on the “Book Ticket” button. [ALT]		**3.** The system extends to “4.1 View Ticket Booking”.
ALTERNATE COURSES:	[ALT] **Step 3a**: The client clicks on the “Book Now” button on the room card. The system extends to Use Case 3.1 Filter Room Bookings”. [ALT] **Step 3b**: The client clicks on the “Book Now” button on the venue card. The system extends to Use Case 5.1 “View Event Booking”.		
CONCLUSION:	The use case concludes once the client can view the View Resort screen.		
POST-CONDITION:	The client can choose the type of booking they wish to make on the “View Resort” screen.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	The client has internet connection		
OPEN ISSUES:	None		

USE CASE NAME:	Add Review	USE CASE TYPE
USE CASE ID:	2.5	Abstract: ☐
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Client	
DESCRIPTION:	This use case describes the event where the client creates a new Review record on the system. The use case begins when the system prompts the client to enter the required information. The administrator enters the information required. The system verifies the information and then stores it within the **Review Entity**. The use case concludes when the Review record has successfully been created on the system.	
PRE-CONDITION:	• The client must be logged into the system. • The client is on the "View Client Homepage" screen. • The client clicked the "**+**" button (Add Review).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Review*" modal with the following elements: – Modal Heading Text: "Review and Feedback"; Close button. – Label: "Rating (out of 5)" – Rating numeric up down field with a default value set to "0". – Label: "Review Description" – A text input field with the placeholder text "Enter your description here…" – Submit button with the text "Submit". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]

		2. The system prompts the client to enter the new review details.
	3. The client will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Submit” button. [ALT]
	5. The client clicks the “Submit” button. [ALT]	**6.** The system will capture the information entered by the client and will populate a Review object with the attributes: – Rating – Description The Review object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **Review Entity**: ○ Review_ID (int) [PK] (value of the previous Item_ID, incremented by 1) ○ Client_ID (int) [FK] ○ Rating (int) ○ Description (string) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the client was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the client to re-enter the review details. **Return to Step 3**. [ALT] **Step 5**: The client selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).	
POST-CONDITION:	The new review record has been added to the system in the Review table.	

USE CASE NAME:	View Client Homepage	USE CASE TYPE	
USE CASE ID:	2.6	Abstract:	☐
PRIORITY:	High	Extension:	x
SOURCE:	Platinum Island Resort		
PRIMARY BUSINESS ACTOR:	Client		
DESCRIPTION:	The use case begins when the client clicks on the “Login” link on the “Login” screen which will result in the client being redirected to the “Client Homepage” screen. The use case ends once the client can view the client homepage.		
PRE-CONDITION:	• The client is successfully logged in. • The client clicks on the “Login” button on the Login screen.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Review** Entity which has the following attributes: ○ Review_ID (PK) ○ Client_ID.name (FK) ○ Rating (int) ○ Description(varchar30)
			2. The system will respond by displaying the “View client homepage” screen. The following are visible on the screen: – A user icon at the top left of the screen. ○ User icon has a dropdown with the following options:

			– "View Profile" with user icon (fa fa-user) – "Sign Out" (fa-sign-out) ○ Heading Label: "Platinum Island Resort" ○ Background image of the resort. ○ Paragraph with a description of a summary of the resort ○ Button with text "Book Now" The system will also respond by displaying the reviews section with the following: ○ Heading Label with text "Reviews & Feedback" ○ Button with "+" sign ○ Label with text "Add your feedback!" ○ Card that makes use of a Ngfor loop that goes though the review records from the **Review** entity and displays the following: – Client_ID.name – Rating (fa-star) – Description Link with text "View more" contains pagination functionality that displays 3 records at a time.
	3. The client clicks on the "Book Now" button. [ALT]		**4.** Extends to Use Case 2.4 View Resort.
ALTERNATE COURSES:	[ALT] **Step 3a**: The client clicks on the "View Profile" option. The system extends to Use Case 2.2 "View Profile".		

	[ALT] **Step 3b**: The client clicks on the “Sign Out” option. The system extends to Use Case 1.2 “Logout”.
POST-CONDITION:	The client can view the “View Client Homepage” screen.

2.3. Subsystem 3 - Accommodation

USE CASE NAME:	Filter Room Bookings	USE CASE TYPE	
USE CASE ID:	3.1	Abstract:	¨
PRIORITY:	High	Extension:	x
SOURCE:	Platinum Island Resort		
PARTICIPATING ACTORS:	Client (PBA) PayFast (ESA)		
DESCRIPTION:	The use case begins when the client requests to filter available accommodation booking options on the system. The system should allow the client to filter available room type bookings. The system will begin by prompting the client to select the check-in and check-out date where the system will filter the available rooms to the client. The use case concludes when room types are filtered according to the client's date selection following by the client will then select the room type of their choice.		
PRE-CONDITION:	• The client must be logged into the system. • The administrator clicked the 'Book Room' button.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system responds by loading the '*Filter room booking*' booking screen with the following elements: – Platinum Island logo – Label with the text: "check-in" – Label with the text: "check-out" – Angular date picker The system will prompt the client to select the check-in and check-out date of the client's duration at the resort.	
	2. The client will provide the check-in and check-out date within the necessary input fields.	**3.** The system will first validate that both input fields to ensure that: – Both the check-in and check-out input fields have been entered and are not blank.	

		– Will ensure that the date selected by the client does not occur before the current date which we can validate using the Date.Now() function with the typescript. [ALT] Once the information entered by the client has been successfully validated the system will then save the information provided by the client into the following variables: – checkInDate (which will be populated using a DateTime which was selected as the client's Check-In date.) – checkOutDate (which will be populated using a DateTime which was selected as the client's Check-Out date.) – dateDuration (which will be calculated by determining the difference in the number of days between the client's check-in and check-out dates.) The system will use the checkInDate and checkOutDate variables in the Angular Frontend to filter the available room options that the client will be able to choose from by reading the *RoomStatus_ID.Name [FK]* from the **Room** entity where the status is set to "Available" on the particular days chosen by the client to validate whether the room options are still available.

		4. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the following entities: **Room** Entity which has the following attributes: ○ Room_ID (int) [PK] ○ RoomType_ID (int) [FK] ○ RoomStatus_ID (int) [FK] ○ RoomNumber (int) ○ RoomFloor (int) The **RoomTypePrice** entity which uses the *RoomType_ID (int) [FK]* attribute to match the *Price* attribute to the *RoomType_ID (int) [PK]* from the **RoomType** entity with the following attributes: ○ RoomType_ID (int) [PK] ○ TypeName varchar(20) ○ Description varchar(30) ○ RoomCapacity (int) Which is displayed for each room accommodation option.
		5. The system will re- load the *"Filter Room Booking"* screen with the following elements: – Filter navigation bar with the following elements: ○ Platinum Island image logo ○ Font awesome icon, fa fa-calendar ○ Label with the text "Check-in"

		○ Label with the text "Check-out" ○ Angular material, datepicker Using an **NgFor loop*, the system will display each Room record within a card: – Picturebox containing the image of the room type. – Label populated with the {{*RoomFloor*}} and {{*RoomNumber*}} attributes. – Label populated with the {{*RoomType_ID.Name*}} attribute. – Card-content populated with the {{*Description*}} of the RoomType. – Card-content populated with the {{*RoomPrice_ID. Price*}} displaying the price of the room type. – Button with the text "Book Now".
	6. The client will click the "Book Now" button for the room option of their choice. [ALT]	**7.** The system will send the following variables from the component.html to the typescript: – checkInDate – checkOutDate – dateDuration – roomID (which will store the *Room_ID* attribute retrieved from the **Room** entity) – roomTypePrice (retrieved from the *Price* attribute from the **RoomTypePrice** entity) – roomTypeID (which will store the *RoomType_ID* attribute retrieved from the **RoomType** entity) – clientID (which will store the *Client_ID* attribute retrieved from the **Client** entity)

		Using these attributes, the system will instantiate an object within the *tempBooking* array within the typescript which will be pushed to a publicly declared array which can be accessed through the Data Service class.
		8. Extends to UC 3.2 Make Room Booking.
ALTERNATE COURSES:	[ALT] **Step 5**: The system detects that the client has not correctly provided the check-in and check-out dates. Return to Step 3. [ALT] **Step 8a**: The client decides to choose another check-in and/or check-out date. Return to Step 4. [ALT] **Step 8b**: The client decides to return to the client homepage. Terminate use case.	
POST-CONDITION:	The system will create and populate the following variable: – checkInDate – checkOutDate – dateDuration – roomID – roomTypePrice – roomTypeID – clientID with the information selected by the client when they click the “Book Now” button for the accommodation type of their choice.	

USE CASE NAME:	Make Room Booking	USE CASE TYPE
USE CASE ID:	3.2	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Client (PBA) PayFast (ESA)	
DESCRIPTION:	The use case begins when the client requests to complete an accommodation booking on the system. The system should allow the client to complete an accommodation booking. The system will prompt the client to provide the number of guests that will be occupying the room, the client will confirm their pre-populated information as well as indicate the number of guests that will be occupying the room. Once the client accepts the terms and conditions, they will proceed with the payment via the payment gateway, PayFast. Once the payment has been validated and is successful, the system will generate a unique reference number specific to the client's booking. The system will use the client email to generate and send an email detailing their booking summary information as well as will save the booking to the Booking entity. The use case concludes when the client receives an email summarising their booking.	
PRE-CONDITION:	• The client must be logged into the system. • The client clicked the 'Book Room' button on the "*Filter Room Bookings*". • The system has saved the appropriate variables, the client's check-in & check-out dates, the room type, the room type price as well the ID of the client, to the *tempBooking* array.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will send a request from the Angular frontend to the Data Service class where the service will return the object found within the *tempBooking* array which stores the following variables: ○ checkInDate (which will be populated using a DateTime which was selected as the client's Check-In date.) ○ checkOutDate (which will be populated using

		a DateTime which was selected as the client's Check-Out date.) ○ dateDuration (which will be calculated by determining the difference in the number of days between the client's check-in and check-out dates.) ○ roomID (which will store the *Room_ID* attribute retrieved from the **Room** entity of the room chosen by the client.) ○ roomTypePrice (retrieved the most recent Price from the *Price* attribute from the **RoomTypePrice** entity) ○ roomTypeID (which will store the *RoomType_ID* attribute retrieved from the **RoomType** entity of the selected type of room by the client.) ○ clientID (which will store the *Client_ID* attribute retrieved from the **Client** entity of the client that is currently

		logged in.) The system will then match the clientID variable to the *Client_ID* attribute within the **Client** entity to retrieve the matching record with the following attributes: ○ ClientName varchar(20) ○ ClientSurname varchar(20) ○ ClientEmail varchar(50) ○ ClientPhone varchar(10) The system will then match the roomTypeID variable to the *RoomType_ID* attribute within the **RoomType** entity to retrieve the matching record with the following attributes: ○ TypeName varchar(20) ○ Description varchar(30) ○ RoomCapacity (int) The system then fetches the most recent VAT record according to the *Date* attribute from the **VAT** entity with the following attributes: ○ VATAmount decimal(6,2) ○ Date (DateTime) The retrieved VAT amount will then be stored within a numeric variable, VATpercentage.

		The system performs a calculation by declaring a numeric variable, totalCost. Which is calculated by multiplying the dateDuration variable by the roomTypePrice variable *(dateDuration * roomTypePrice)* The system then performs a further calculation by declaring a numeric variable, VATCost. Which is calculated by multiplying the totalCost variable by the VATpercentage variable *(totalCost * VATamount)* [ALT]
		2. The system will load the "***Make Booking***" screen with the following elements: – Personal details container with the following elements: ○ heading label with the text "Personal Details" ○ label with the text "First Name" ○ text input field which is pre-populated with the client's first name – {{ClientName}} ○ label with the text "Last Name" ○ text input field which is pre-populated with the client's surname – {{ClientSurname}} ○ label with the text "Cell phone Number" ○ text input field which is pre-populated with the client's cell number – {{ClientPhone}} ○ label with the text "Email Address" ○ text input field which is pre-populated with the client's email

address – {{ClientEmail}}
    - label with the text "Number of Guests"
    - a numeric input field which is restricted with a minimum value = 1; and a maximum value = *RoomCapacity* attribute from the **RoomType** entity.
- Booking summary card to the right of the Personal Details Container with the following elements:
    - card header with the text "Booking Summary".
    - label populated with the checkInDate variable.
    - label populated with the checkOutDate variable.
    - label populated with the type of room – {{RoomType.TypeName}}.
    - paragraph tag populated with the description of the room type {{RoomType.Description}}
    - label with the text "TOTAL (VAT Inclusive)".
    - label populated with the text "R" + totalCost variable.
    - label with the text "(VAT 15%)".
    - label populated with the text "R" + VATCost variable.
    - label with the text "Amount Due".
    - label populated with the text "R" + totalCost variable.
- Terms and conditions check box container below the

		Personal Details container and Booking Summary card with the following elements: ○ terms and conditions label with the following hyperlinks which will redirect the client to a PDF: ○ "reservation" hyperlink detailing the terms and conditions of reserving the booking. ○ "cancellation" hyperlink detailing the cancellation and refund processes within the business. ○ "waiver forms" hyperlink detailing the health and safety terms and conditions. – Button to the left with the text "Back to Room Selection". – Button to the right with the text "Book Now Confirm with Payment".
		3. The system prompts the client to provide the number of guests that will be occupying the room.
	4. The client will input the number of guests that will be occupying the room within the numeric input field.	**5.** The system will prompt the client to accept the terms and conditions detailed.
	6. The client accepts the terms and conditions laid out by the business and proceeds with the payment of the booking by clicking on the "Proceed with Payment" button. [ALT]	**7.** The system will use the Angular frontend to validate that the input fields are not left blank and ensures the details entered by the client follows the following criteria: ○ The input fields are not left blank. ○ The number of guests fits within the limits of the minimum count = 1; and the maximum count = {{RoomType. Capacity}} ○ The terms and conditions checkbox are ticked and

		accepted by the client.
		8. Once the client's details are successfully validated, the system will prompt the client to choose their method of payment by loading the "*Choose Payment*" modal with the following elements: – Modal header with the text "Complete Payment". – Close modal button. – Modal content with the text "Please choose your preferred method of payment from the below options:". – Button with the text "PayPal". – Button with the text "Debit or Credit Card". – Label with the text "Powered by PayPal". [ALT]
	9. The client will provide their preferred method of payment.	**10.** The system will then redirect the client to the PayPal system depending on their payment selection where the system will pass through the t*otalCost* of the client's booking as well as the *ClientName* and *ClientSurname* to PayPal where they will be required to pay for their booking.
		11. Once the payment is successful, PayPal will send back a payment confirmation to the system. [ALT]
		12. The system will then create a unique reference number using a combination of randomly generated characters and numbers specific to the client's booking and make use of an SQL_Insert query to create a new record within the following entities: **RoomBooking** entity with the following attributes:

		○ RoomBooking_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomBookingStatus_ID (int) [FK] ○ Reminder_ID (int) [FK] ○ ReferenceNum varchar(10) ○ BookingDate DateTime ○ NumberOfGuests (int) ○ EntryDate (DateTime) ○ ExitDate (DateTime) **RoomPayment** entity with the following attributes: ○ RoomPayment_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomBooking_ID (int) [FK] ○ PaymentType_ID (int) [FK] ○ Amount decimal(6,2) The system will assign the status of “awaiting final payment” from the **RoomBookingStatus** entity to the client’s booking record with the following attributes: ○ RoomBookingStatus_ID (int) [PK] ○ Name varchar(20) ○ Description varchar(30) Using Angular frontend, the system will read the *ClientEmail* attribute from the **Client** entity to send an email using the MailKit plug-in, detailing the

		booking summary information of the client's booking as well as their unique booking reference number. The client's email contains the following elements: – Subject heading with the text "Platinum Island Resort Booking Confirmation" – Recipient is populated with the client's email address – {{ClientEmail}} – Salutation with the text "Dear {{ClientName}}" – Email body welcoming the client to the resort and details a summary of the booking made by the client. Additionally with the unique booking reference number. – Email closing with the text "We can't wait to see you there!" – Signature with the details of the resort. [ALT]
ALTERNATE COURSES:	[ALT] **Step 1**: The system will return a 404-code error, as the client's details are unable to be found within the **Client** entity. [ALT] **Step 6a**: The client refuses to accept the terms and conditions set out by the resort. **Terminate use case**. [ALT] **Step 6b**: The client chooses to cancel the booking by return to the booking homepage. **Terminate use case**.	

	[ALT] **Step 8**: The information entered by the client was invalid and/or input fields were left blank. The system will prompt the client to re-enter their details. **Return to Step 4**. [ALT] **Step 11**: PayPal will return a failed payment from the client. **Return to Step 9**. [ALT] **Step 12a**: The system failed to add the client's booking. The system alerts the client of the error and that the booking wasn't made successfully. **Terminate use case**. [ALT] **Step 12b**: The system failed to add the client's booking payment. The system alerts the client of the error, and that the payment record wasn't successfully added. **Terminate use case**.
POST-CONDITION:	• The booking has been stored in the **Room Booking** Entity and the room's booking status has been changed to "Awaiting Final Payment". • The client's payment has been stored in the RoomPayment entity.

USE CASE NAME:	Cancel Room Booking	USE CASE TYPE	
USE CASE ID:	3.3	Business Requirements:	o
PRIORITY:	Medium	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	The use case begins when the client requests to cancel their accommodation booking. The system should allow the client to cancel their accommodation reservation by calling into the business where the administrator will assist. The administrator will confirm the cancellation of the client's booking. Once the booking is cancelled, the client receives an email detailing the cancellation of their booking. The use case concludes when the booking is cancelled, and the client receives an email detailing the successful cancellation of their booking.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The client's accommodation booking status has to be "Booked" • The "View Accommodations" screen needs to be loaded. • The administrator clicked the "cancel" button.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by displaying a pop-up "*Cancel room booking*" modal with the following elements: – Modal Heading Text: "Confirm room booking cancellation" – Label: "Are you sure you want to cancel this booking"? – Submit button with the text "Confirm". – Cancel button with the text "Cancel". – Submit button with the text: "Submit".

		2. The administrator will confirm whether the client would like to cancel their accommodation booking.	
	3. The client will confirm the cancellation of their accommodation booking. [ALT]	**4.** The administrator clicks on the "Confirm" button.	**5.** Using the matching *ReferenceCode* attribute within the **Room booking** entity which corresponds to the appropriate matching record, the system will update the status of the client's booking by reading the *Name* attribute from the **RoomBookingStatus** entity depending on when the cancellation of the booking occurs: – If the clients' booking cancellation occurs within 48 hours before their check-in, which can be validated using the EntryDate attribute within the RoomBooking entity and the current date of cancellation using the Date.Now() function, the room booking will have a status of "Cancelled" – If the clients' booking cancellation occurs more than 48 hours before their check-in, which can be validated using the EntryDate attribute within the RoomBooking entity and the

			current date using the Date.Now() function of cancellation, the booking will have a status of “Eligible for Refund”. Where the client will be eligible for a 25% return on their booking payment. Using the *ClientEmail* attribute, which is retrieved from the **Client** entity, the system sends an Email to the client with the following information: – Subject: [Accommodation Cancellation - Important Update – Dear {{*ClientName*}}, – Email Body: “We regret to inform you that the booking scheduled for {{*Date*}} has been cancelled due to unforeseen circumstances. – Email Footer: “Best regards, The Platinum Island team”
ALTERNATE COURSES:	[ALT] **Step 3**: The client decides to decline their accommodation booking cancellation. Terminate use case.		
POST-CONDITION:	A reminder notification email will be sent to the client detailing their booking summary and check-in information.		

USE CASE NAME:	Send Accommodation Booking Reminder	USE CASE TYPE	
USE CASE ID:	3.4	Business Requirements:	o
PRIORITY:	Low	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Time		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	Client (ERA)		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when time prompts the system to send a reminder notification to the client. The system should send an email to the client before their booking check-in reminding them of their booking reservation. The use case concludes when the system sends an email to the client reminding them of the time of their check-in.		
PRE-CONDITION:	• The system detects that the client needs to be notified of when their accommodation reservation		
TRIGGER	Time prompts the system to send a reminder notification to the client.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Time prompts the system to send a reminder notification to the client		**2.** Using .NET Core backend, the system will periodically check for booking check-ins' which are due, by reading the *Reminder_Time* attribute from the **Reminder** entity to determine how many hours before the notification reminder is sent to the client, by reading the *BookingDate (DateTime)* attribute from each record in the **RoomBooking** entity. Which then prompts the use

			of a LINQ Query which utilizes an SQL_Read query to retrieve the Client_ID.*ClientEmail* (varchar(50)) [FK] attribute from the **RoomBooking** entity by matching the *Client_ID (int) [FK]* attribute to the *Client_ID (int) [PK]* attribute within the **Client.** [ALT]
			3. The system will then use the MailKit library in the backend to construct an email reminder containing the booking summary details particular to the client's booking by reading the following attributes from the **RoomBooking** entity: ○ RoomBooking_ID (int) [PK] ○ Client_ID.ClientEmail (varchar(50)) [FK] ○ Client_ID. Name (varchar(20)) [FK] ○ ReferenceNum varchar(10) ○ BookingDate (DateTime) The system will then use the *ClientEmail* attribute retrieved to generate an email which is sent to the client's email to notify them of their accommodation reservation in 24 hours' time which contains the following elements: – Subject heading with the text "Platinum Island Resort Booking Reminder" – Recipient is populated with the client's email

			address – {{ClientEmail}} – Salutation with the text “Dear {{ClientName}}” – Email body reminding the client of their booking and provides a summary of the booking made by the client. Additionally with the unique booking reference number. – Email closing with the text “We can’t wait to see you there!” – Signature with the details of the resort.
ALTERNATE COURSES:	None.		
CONCLUSION:	The use case concludes when the system sends an email to the client detailing the time of their check-in, in 24 hours’ time.		
POST-CONDITION:	A reminder notification email will be sent to the client detailing their booking summary and check-in information.		
BUSINESS RULES:	• The accommodation reservation notification is sent to the client 24 hours before their check-in.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	View room type	USE CASE TYPE
USE CASE ID:	3.5	Business Requirements: ☐
PRIORITY:	High	System Analysis: ☐
SOURCE:	Platinum Island Resort	System Design: ☑
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes the event where the administrator would like to view the room type record available of the system. The use case begins with the administrator requests to access the 'Room type' option by navigating to the side navigation bar. On this view, the administrator will be able to create, update and delete room types on the system. The use case concludes when the system displays all the room type records.	
PRE-CONDITION:	• The administrator needs to be logged onto the system.	
TRIGGER:	The administrator requests to view Room type screen on the system.	

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Administrator will request to view the *View Supplier Screen* by hovering over the 'Supplier' tab option where the side navigation bar will display three routing options: ○ 'Room'		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to

	○ 'Room types' The administrator will then click on "Room type" side-navbar option.		retrieve the records from the **RoomType Entity** which has the following attributes: ○ RoomType_ID (int) [PK] ○ TypeName (varchar (20)) ○ Description (varchar (30)) ○ RoomCapacity (int) The system retrieves the Price attribute from the **RoomTypePrice** entity by using the RoomType_ID [FK] in the **RoomTypePrice** entity that corresponds to the RoomType_ID [PK] in the **RoomType** entity
			3. The system will load the '*Supplier*' screen with the following elements: – Label with the text "Room type" – "**+**" button – Label with the text "Add type" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter Room type". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right. Supplier Table with the following headers: ○ "Name" ○ "Description" ○ "Capacity" ○ "Price" ○ Empty Header

			The system will populate each row within the Supplier Table with the records read from the **Supplier Entity** as follows: – "Name" with the *Name* – *"Description"* with the *Description* – *"Capacity"* with the room capacity – *"Price"* with the price of the room *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*
	4. The administrator requests to add a room type record. [ALT]		**5.** . The system extends to "3.6 Add room type".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search a room type. The system will prompt the administrator to enter the room type details within the search bar. The administrator will enter the 'Name', 'Description, 'Capacity 'or 'price of the room. The system retrieves and displays a list of all the room type records that match the search criteria entered by the administrator using the following attributes from the **RoomType Entity**: ○ "Name" ○ "Description" ○ "Capacity" ○ "Price" [ALT] **Step 4b**: The administrator requests to update a Room type. The system extends to Use Case 3.7 "Update Room type". . [ALT] **Step 4c**: The administrator requests to delete a Supplier. The system extends to Use Case 3.8 "Delete Room type".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the RoomType **entity**.		

POST-CONDITION:	The administrator will be able to add a Room Type on the *'Room Type* screen.
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Rom type records on the system.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Add Room Type	USE CASE TYPE
USE CASE ID:	3.6	Abstract: ☐
PRIORITY:	High	**Extension:** x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new RoomType record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the RoomType record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'RoomType'* screen. • The administrator clicked the "**+**" button (Add RoomType).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add RoomType*" modal with the following elements: – Modal Heading Text: "Add RoomType"; Close button. – Label: "Type Name" – Type name text input field with the placeholder text "Enter Room Type". – Label: "Type Description" – Type Description text input field with the placeholder text "Enter description". – Label: "Room Capacity" – Capacity numeric up down with the placeholder text "Enter Capacity". – Label: "Room Price" – Price numeric up down with the placeholder "Enter room price"

		– Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new RoomType details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Submit” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a RoomType object with the attributes: – Name – Description – Room Capacity – Room price The RoomType object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **RoomType Entity**: ○ RoomType_ID (int) [PK] (value of the previous ItemCategory_ID, incremented by 1) ○ TypeName (varchar (20)) ○ Description (varchar (30)) ○ RoomCapacity (int) ○ RoomType_ID.Price (int)

		[ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the RoomType details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).	
POST-CONDITION:	The new RoomType record has been added to the system in the RoomType table.	

USE CASE NAME:	Update Room Type	USE CASE TYPE
USE CASE ID:	3.7	Abstract: ☐
PRIORITY:	High	**Extension:** x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates a RoomType record on the system. The use case begins when the administrator chooses to update a RoomType record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the RoomType record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'RoomType'* screen. • The administrator clicked the update button.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit RoomType*" modal. Using the RoomType_ID sent from the Angular front-end to the .NET Core backend, the system will match the ItemCategory_ID selected to a specific record in the **RoomType Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the RoomType. The "*Edit RoomType*" modal has the following elements: – Modal Heading Text: "Add RoomType"; Close button. – Label: "Name" – RoomType name text input field with the pre-populated name text – {{Name}}

		– Label: “Type Description” – RoomType Description text input field with the prepopulated text {{Description}} – Label: “Room Capacity” – RoomType Capacity numeric updown input field with the prepopulated Capacity text {{Capacity}} – Label: “Room Price” – RoomType price numeric updown input field with the prepopulated Capacity text {{RoomType_ID.Price}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated RoomType details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a RoomType object with the attributes: – Name – Description – Room Capacity

		– Room price The RoomType object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the RoomType record within the **RoomType Entity**: ○ RoomType_ID (int) [PK] (value of the previous ItemCategory_ID, incremented by 1) ○ TypeName (varchar (20)) ○ Description (varchar (30)) ○ RoomCapacity (int) ○ RoomType_ID.Price (int)
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the RoomType details. Return to **Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. The use case terminates.	
POST-CONDITION:	The relevant RoomType information has been successfully updated in the **RoomType Entity**.	

USE CASE NAME:	Delete RoomType	USE CASE TYPE
USE CASE ID:	3.8	Abstract: ☐
PRIORITY:	High	**Extension:** x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific RoomType record. The use case begins with the system verifying there are no sale items associated with the RoomType. The system will confirm the RoomType deletion before the administrator confirms the deletion of the record. The use case concludes when the RoomType record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'RoomType'* screen. • The administrator clicked the delete button. • The RoomType must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system matches the RoomType_ID [PK] with the RoomType selected by the administrator to complete the. The system will do this by performing an SQL_Read query in the .NET Core controller.
		2. The system responds by displaying a pop-up "*Delete RoomType*" modal with the following elements: – Modal Heading Text: "Confirm RoomType deletion"; Close button. – Label: "Are you sure you want to delete this RoomType?" – Submit button with the text "Confirm". – Cancel button with the text "Cancel". –

		3. The system prompts the administrator if they would like to delete the selected record.
	4. The administrator clicks the "Confirm" button. [ALT]	**5.** The system deletes the RoomType record from the **RoomType Entity** with the following attributes: ○ RoomType_ID (int) [PK] (value of the previous ItemCategory_ID, incremented by 1) ○ TypeName (varchar (20)) ○ Description (varchar (30)) ○ RoomCapacity (int) ○ RoomType_ID.Price (int) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 4**: The administrator selects the Cancel button. The use case terminates.	
POST-CONDITION:	The relevant RoomType information has been successfully deleted from the **RoomType Entity**.	

2.4. Subsystem 4 - Ticketing

USE CASE NAME:	View ticket booking	USE CASE TYPE	
USE CASE ID:	4.1	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes an event where the client would like to view the Event Booking screen. The use case begins when the client requests to access the 'event booking' view. On this view, the client will be able to view details regarding the events of the platinum island. The client can also click on the book now button which will redirect the client to the event booking form.		
PRE-CONDITION:	• The client needs to be logged into the system. • The client chooses to select the day visit and clicks the *'Book Now'* on the view resort screen		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
			1. The system responds by loading the '*View ticket booking*' screen with the following information: – Heading: "Plan your adventurous day." – Second heading: "PLATINUM ISLAND" – Label: "Ready, set, splash!"

			– Submit button: “Reserve your spot”. On the footer of the page there Is a label with the following text: – “Adults: R100 | Pensioners: R80 | Kids: R70.”
	2. The client chooses to select the “Reserve your spot” button.		**3.** The system extends to Use case “4.2 Make ticket booking”
ALTERNATE COURSES:	None		
CONCLUSION:	The view ticket booking screen is loaded.		
POST-CONDITION:	The screen is extended to the view ticket booking screen.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Make ticket booking	USE CASE TYPE	
USE CASE ID:	4.2	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
OTHER PARTICIPATING ACTORS:	• PayPal (ESA) • Client		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when the client wishes to make a day visit booking on the system. The system should allow the client to complete a day visit booking. The system will prompt the client to provide the number of adults, children and pensioners that will be arriving on the day, the client will confirm their pre-populated information, Once the client accepts the terms and conditions, they will proceed with the payment via the payment gateway, PayPal. Once the payment has been validated and is successful, the system will generate a QR Code specific to the client's booking. The system will use the client email to generate and send an email detailing their booking summary information as well as will save the booking to the Booking entity. The use case concludes when the client receives an email summarising their booking.		
PRE-CONDITION:	• The client must be logged into the system. • The client must have internet access.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by loading the *'Make ticket booking'* screen with the following information: – Heading: "Personal details" – Label: "First name" – Textbox containing the clients pre-populated name {{Client.ClientName}} – Label: Last Name

			– Textbox containing the clients pre-populated last name {{Client.ClientSurname}} – Label: “Email address” – Textbox containing the clients pre-populated email address {{Client.Emailaddress}} – Heading: “Ticket details” – Label: “Children: R80 per person | Adults: R100 per person | Pensioners: R70 per person. – Label: “Date of visit” – DatePicker: Days before the current date are disabled. – Label: Number of children (6 years to 17 years) – Numeric up down: initial value set to 0. Client cannot select a number below 0. – Label: Number of adults (18 years to 64 years) – Numeric up down: initial value set to 0. Client cannot select a number below 0. – Label: Number of pensioners (65 years and older) – Numeric up down: initial value set to 0. Client cannot select a number below 0. – Button: “View summary booking” Outside of the make ticket booking form is the following details: – Checkbox: “I understand that by clicking “Book Now” I have read and agreed to the terms and

			conditions of the: "waiver document" where the underlined text is a hyperlink that displays the rules and regulations of the day visits. – Button with an arrow facing the left with the text: "Bookings page". – Buttons with an arrow facing the right with the text: "Book Now. Confirm with Payment" The system prompts the client to complete the required information.
	2. The client completes the required information by selecting a date and the number of children, adults and pensioners accompanying him on the day visit.		**3.** The system validates that the maximum capacity has not been reached by referring to the "Capacity" attribute in the **DayVisitDate** entity and enables the "View Day visit summary" button. [ALT] The system then prompts the client to view the day visit summary.
	4. The client confirms that all the information is correct and clicks on the terms and conditions checkbox. [ALT]		**5.** Using Angular the system enables the "Day Visit Summary" button and prompts the client to select the button.
	6. The client clicks "View summary booking" button.		**7.** The system responds by loading the "Day Visit Summary" modal containing the following information: – Heading "Day Visit Summary" – The current date.

			– The number of tickets, followed by the type of tickets they are using the price attribute in the **DayVisitTicketPrice** entity and the TypeName attribute in the **DayVisitType** entity. – Label: “Subtotal (VAT exclusive)” with the total price due indented to the right of the label. – Label “(VAT 15%)” with the total VAT due indented to the right of label. – Label: “Amount due” and the total amount due indented to the right of the label – Button with the text “Paypal” – Button with the text “Debit/Credit”
	8. The client will provide their preferred method of payment.		**9.** The system will redirect the client to PayPal where the system will pass through the total Cost of the client’s booking to PayPal where they will be required to pay for their booking. The system prompts the client to make a booking.
	10. The client makes the payment on PayPal		**11.** Once the payment is successful, the system will generate a QR Code specific to the client’s booking and will create a new record within the following entity: **DayVisitTicket** entity with the following attributes: ○ Ticket_ID [PK] ○ DayVisit_ID [FK] ○ GuestName ○ QRCode

			The system will send an email to the client using the client informing of the following information: – Heading: “Platinum Island resort day visit booking confirmation” – Email Body: “Thank you for booking a day visit with Platinum Island resort. You have made a booking for *xx adults, xx children and xx pensioners.*” – Label: the total due for your payment is: *TotalCost,* where TotalCost is the amount that was paid by the client. – Here is a QR code that you will have to scan in at the entrance. – Footer: ○ “We hope you have spash-tecular day”. ○ The platinum island team. [ALT]
ALTERNATE COURSES:	[ALT] **Step 3**: The maximum capacity has been reached. The “View Day visit summary” button will remain disabled. [ALT] **Step 4**: The client does not select the checkbox. The “View summary” button Will remain disabled. [ALT] **Step 11**: The payment is not successful. **Go to step 9**.		
CONCLUSION:	The use case concludes when the booking is made and sent to client via email.		
POST-CONDITION:	The client receives a QR Code in which he will scan at the entrance of the resort.		
BUSINESS RULES:	None.		
IMPLEMENTATION CONSTRAINTS	None.		

AND SPECIFICATIONS:	
ASSUMPTIONS:	The client has internet access.
OPEN ISSUES:	None

USE CASE NAME:	View day visit prices	USE CASE TYPE	
USE CASE ID:	4.3	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where an administrator wishes to view the prices of the day visits on the system. The use case begins when the admin clicks on the view day visit prices option in the side navbar. The system will then load the day visit prices screen. On this view, the administrator will be able to update the ticket prices from the list. The use case concludes when the system displays all the ticket price records.		
PRE-CONDITION:	• The administrator must have access to internet. • The administrator must be logged onto the system.		
TRIGGER:	The administrator wishes to view the day visit prices and clicks the on the day visit side navbar option		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The administrator will request to view the *'day visit prices'* screen by clicking on the day visit prices option on right panel side navigation bar.		**2.** The system will send a request from the angular frontend to the data service class where the service will make a HttpGet request to the .Net core backend which makes use of a linq and lambda query which utilizes the SQL_READ query to retrieve records from the **DayVisitTicketPrice** entity and displays the following attributes:

			○ Price_ID [PK] (int) ○ DayVisitType_ID [FK] (int) attribute of the DayVisitType entity where the DayVisitType _ID in the DayVisitType entity corresponds to the DayVisitType _ID in the DayVisitTicketPrice entity. ○ Price (int) ○ Date (DateTime) **DayVisitType** ○ DayVisitType_ID (PK) ○ TypeName (varchar 20) ○ LowerAge (int) ○ UpperAge (int)
			3. The system responds by loading the '*View Day visit prices'* screen details with the following information: – Heading: "Day visit prices" – Table with the headings: ○ Types of tickets ○ Price ○ Empty header – The system will prepopulate the first columns rows within the table with the attribute from the **dayvisitType** entity as follows: ○ "Type of ticket" with the TypeName – The system will prepopulate the second columns rows within the table with the attribute from the **dayvisitTicketPrice** entity as follows:

			○ "Price" with the Price of the ticket. – Empty header with the edit button with a pencil icon.
	4. The employee wishes to update the day visit ticket prices. [ALT]		**5.** The system Invokes Use case "4.4 Update Day visit prices".
ALTERNATE COURSES:	[ALT] **Step 4**: The administrator wishes to close the *'View Day visit Prices'* screen. Terminate use case.		
CONCLUSION:	The case ends when the administrator can view the price details are on the system.		
POST-CONDITION:	• The view day visit screen is displayed to the administrator.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Update Ticket Price	USE CASE TYPE	
USE CASE ID:	4.4	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PARTICIPATING ACTORS:	Administrator (PBA)		
DESCRIPTION:	This use case describes the event where an administrator wants to update the ticket prices of a specific category. The use case begins when the *'View ticket price'* screen is loaded, and the administrator selects the edit button to edit a price. The use case ends once the updated price is saved in the database.		
PRE-CONDITION:	• The user must have access to internet. • The user must be logged onto the system. • The view ticket price screen is loaded.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system loads the update ticket price edit modal popup. Using the DayVistType_ID sent from the Angular front-end to the .NET Core backend, the system will match the DayVistType_ID selected to a specific record in the DayVisitTicketPrice Entity using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the Ticket Type. The *'edit ticket prices'* modal has the following elements: – Heading: Edit ticket prices – Label: Ticket type – Disabled textbox with the prepopulated data of the selected category from the attribute {{TypeName}} in the **DayVisitType** entity.

			– Label: Ticket price – Numeric up down with the prepopulated price of the different prices based on the selected category from the attribute {{Price}} in the **DayVisitTicketPrice** entity. – Submit Button with the text: “Save”. – Cancel Button with the text: “Cancel”. [The submit button is disabled until all the fields are validated. The cancel button is enabled on default] The system would prompt the administrator to update the prepopulated price details.
	2. The administrator will enter the new price of the selected category in the numeric updown		**3.** The system will use the angular frontend to ensure that none of the input fields are left blank and will enable the save button.
	4. The administrator clicks on the save button. [ALT]		**5.** The system will capture the information entered by the administrator and will populate the dayvisitticketprices object with the attributes: – Type Name – Prices The DayVisitTicketPrice object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.

			6. The system will make use of an SQL_Update query in the controller to update the DayVisitTicketPrices record within the DayVisitTicketPrices Entity: ○ Price_ID (int) [PK] ○ DayVisitType_ID (int) [FK] ○ Prices (int) ○ Date (date)
ALTERNATE COURSES:	[ALT] **Step 4**: The administrator chooses to not update the selected field and selects the cancel button. Terminate use case.		
POST-CONDITION:	• The new price details are displayed on the '*View dayvisit price'* screen.		

2.5. Subsystem 5 – Events

USE CASE NAME:	View event booking	USE CASE TYPE	
USE CASE ID:	5.1	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes an event where the client would like to view the Event Booking screen. The use case begins when the client requests to access the 'event booking' view. On this view, the client will be able to view details regarding the events of the platinum island. The client can also click on the book now button which will redirect the client to the event booking form.		
PRE-CONDITION:	• The client needs to be logged into the system. • The client chooses to select the events and clicks the *'Book Now'* on the view resort screen		
TRIGGER:	The client requests to view the event booking screen.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	AUTOMATED ACTION:
	1. The client requests to view the event booking screen.		**2.** The system responds by loading the event booking screen with the following elements. – Heading: "Create Moments That Last: Book Your Unforgettable Event" – Subheading: "Weddings and Conferences

			at Platinum Island Resort” – Label: “Celebrate your special day or host your business meeting at our scenic resort in Brits. We offer elegant gardens and halls occupying up to 100 guests.” – Label: “Our team will offer the best customer service to ensure the best experience.” – Submit button with the text: “Book Now” On the footer of the page, the system will display the following information: Label: “Only one hour travel from Pretoria you will find Platinum Island Resort. Whether big or small, we caters for your every wedding need. Enjoy a wedding in the lush green gardens of the Resort, beautifully decorated hall, the possibilities are endless. Tailor made menus, organized banqueting team and superb cuisine is sure to provide you with peace of mind and an unforgettable day. Platinum Island Resort offers a wide range of

			Conference Facilities catering for delegates from 10 to 100. All conference rooms are equipped with modern conference equipment. An array of facilities and activities are offered on the Resort. Conferences are tailor made according to your needs ensuring the best service and standards at all times.”
	3. The client clicks the “Make event booking button”		**4.** The system extends to Use case “5.2 Make event booking”
ALTERNATE COURSES:	None		
CONCLUSION:	The view event booking screen is loaded.		
POST-CONDITION:	The even booking screen is loaded and the client can select the book now button.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Make Event Booking	USE CASE TYPE	
USE CASE ID:	5.2	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
OTHER PARTICIPATING ACTORS:	• PayPal (ESA) • Client (PBA)		
OTHER INTERESTED STAKEHOLDERS:	Administrator		
DESCRIPTION:	The use case begins when the client wishes to make an event booking on the system. The system should allow the client to complete an event booking. The system will prompt the client to fill in the required details and the client will confirm their pre-populated information, Once the client accepts the terms and conditions, they will proceed with the payment via the payment gateway, PayPal. Once the payment has been validated and is successful, the system will generate a booking reference number specific to the client's booking. The system will use the client email to generate and send an email detailing their booking summary information as well as will save the booking to the Booking entity. The use case concludes when the client receives an email summarising their booking and will change the status of the event booking "Booked".		
PRE-CONDITION:	• The client must be logged into the system. • The client clicked the 'Book Event Now' button on the "*View Event Booking*" screen.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system uses an HttpGet which is sent from the Angular Frontend to the backend where the system will retrieve the *Client_ID* attribute within the **Client** entity to retrieve the record elements specific to the client which is logged in: ○ ClientName ○ ClientSurname

			○ ClientEmail ○ ClientPhone The system will then use the information retrieved from the client's record to pre-populate the input fields The system also retrieves the most recent VAT record according to the *Date* attribute from the **VAT** entity with the following attributes: ○ VATAmount ○ Date The retrieved VAT amount will then be stored within a numeric variable, *VATamount*. The system then fetches the most recent price from the **VenuePrice** entity with the following attributes: ○ VenuePrice_ID [PK] ○ Venue_ID [FK] where Venue_ID [PK] in the **Venue** entity corresponds to the Venue_ID [FK]. ○ VenuePriceAmount ○ Date The system then responds by loading the *'Make Event Booking'* screen with the following information: – Form Heading: "Personal details" – Label: "First name" – Textbox containing the clients pre-populated name {{ClientName}} – Label: Last Name – Textbox containing the clients pre-populated last name {{ ClientSurname}}

			– Label: “Select Event Type”
			– Event type dropdown with the following categories:
			  ○ Birthday
			  ○ Wedding
			  ○ Conference
			– Label: “Email address”
			– Textbox containing the clients pre-populated email address {{ClientEmail}}
			– Label: “Phone number”
			– Textbox containing the clients pre-populated email address {{ClientPhone}}
			
			– Heading: “Event Details”
			– Label: “Total flat rate of R2 500 is charged for usage of the venue | Venue capacity is limited to 100 persons”
			– Label: “Date of Event”
			– Angular DatePicker (which is validated by disabling days before the current date by using the Date.Now() function).
			– Label: “Upload Guest List”
			– A “Choose file” button which will be used to upload files.
			– Label with the text “*Only pdf upload file types are accepted”
			– Label: “No file chosen” to the right of the “Choose file” button
			
			Below the event booking form is the following details:

			– Terms and conditions container below the Personal Details container and Booking Summary card with the following elements:
			  ○ Acceptance checkbox
			  ○ Terms and conditions label with the following hyperlinks which will redirect the client to a PDF:
			  ▪ “reservation” hyperlink detailing the terms and conditions of reserving the booking.
			  ▪ “cancellation” hyperlink detailing the cancellation and refund processes within the business.
			  ▪ “waiver forms” hyperlink detailing the health and safety terms and conditions.
			– Button to the left with the text “View Event Page”.
			– Button to the right with the text “View Booking Summary”.
			
			[ALT]

			2. The system will prompt the client to provide the date and type of the event as well as the prompt them to upload their guest list.

	3. The client will select an event booking date of their choice within the DatePicker, the event type as well as uploads their guestlist as a PDF document.		**4.** The system will use Angular Frontend to validate that the input fields are not left blank and ensures the details entered by the client follows the following criteria: ○ The input fields are not left blank. ○ The system would use the Date.Now() to restrict the user from selecting a date before the current time. ○ The document uploaded by the client is restricted to only PDF uploads. ○ The event type description field is not left blank. Once validated, the system will prompt the client to accept the terms and conditions checkbox stipulated by the business regarding events. [ALT]
	5. The client accepts the terms and conditions laid out by the business and proceeds with the payment of the booking by clicking on the "View Booking Summary" button. [ALT]		**6.** The system will use Angular Frontend to validate that the terms and conditions checkbox are ticked and accepted by the client and will also validate whether the date of the Event booking is available to the client by reading the *Date* attribute within the **Event** entity and will read the associated event status of the date chosen by the client by using the *EventStatus_ID [FK]* attribute to read if the Name attribute within the **EventStatus** entity is available. Once the information entered by the client has been successfully validated the system will then save the information provided by the client into the following variables:

			– eventDate (which will be populated using a DateTime which was selected as the client's event date.) – eventPrice (which is populated by retrieving the most recent *VenuePriceAmount* attribute from the **VenuePrice** entity) – totalCost (which is populated by multiplying the eventPrice by the date duration (eventPrice * 1)) – VATamount (which is populated by multiplying the *totalCost* by the VAT amount) – VATexc (which is calculated by declaring a numeric variable, *VATexc*. Which is calculated by subtracting the *VATamount* from the *totalCost* (*totalCost - VATAmount*)) [ALT]
			7. The system responds by loading the "Event Summary" modal containing the following information: – Heading "Event Summary" – Label: "Your event is currently reserved for the:"

			with the date of booking chosen by the client. – Label with the text "Reserved for:" and populated with the text of the name of the client booking the event – {{ClientName}} – Label: "Subtotal:" with the total before vat. – {{VATexc}} – Label: "(VAT 15%)" with the VAT amount due. – {{VATamount}} – Amount due: with the total amount due – {{totalCost}} – Button with the text "Paypal" – Button with the text "Debit/Credit" The system will then prompt the client to choose their method of payment.
	8. The client will choose their preferred method of payment.		**9.** The system will then redirect the client to the PayPal system depending on their payment selection where the system will pass through the t*otalCost* of the client's booking as well as the *ClientName* and *ClientSurname* to PayPal where they will be required to pay for their booking.
			10. Once the payment is successful, the system will then generate a reference number specific to the client's booking and will create a new record within the following entity: **Event** entity with the following attributes: o Event_ID [PK] (int)

			○ EventStatus_ID [FK] (int) ○ Client_ID [FK] (int) ○ Venue_ID [FK] (int) ○ Reminder [FK] (int) ○ Date (Date) ○ ReferenceCode (varchar (6)) ○ Guest list (varbinary) ○ EventTotal (decimal 6,2) ○ Description (varchar (30)) **EventPayment** entity with the following attributes: ○ EventPayment_ID [PK] (int) ○ Event_ID [FK] (int) ○ Client_ID [FK] (int) ○ PaymentType_ID (int) ○ Amount (decimal 6,2) The name attribute in the **EventStatus** entity will be changed to “Booked”. Using Angular frontend, the system will read the *ClientEmail* attribute from the **Client** entity to send an email using the MailKit plug-in, detailing the booking summary information of the client’s booking as well as their unique booking code. The client’s email contains the following elements: – Subject heading with the text “Platinum Island Resort Booking Confirmation”. – Recipient is populated with the client’s email address – {{ClientEmail}}

			– Salutation with the text “Dear {{ClientName}}” – Email body welcoming the client to the resort and details a summary of the booking made by the client. Additionally with the unique booking reference number. – Email closing with the text “We can’t wait to see you there!” – Signature with the details of the resort. [ALT]
ALTERNATE COURSES:	[ALT] **Step 1**: The system will return a 404-code error, as the client’s details are unable to be found within the **Client** entity. [ALT] **Step 4**: The system will return a validation error after reading the details entered by the client. **Return to Step 3**. [ALT] **Step 5a**: The client refuses to accept the terms and conditions set out by the resort. **Terminate use case**. [ALT] **Step 5b**: The client chooses to cancel the booking by return to the event booking homepage. **Terminate use case**. [ALT] **Step 6a**: The system notifies the client that terms and conditions was not accepted and will not allow the client to proceed with the booking unless terms and conditions are accepted. **Return to Step 4**. [ALT] **Step 6b**: The event booking date chosen by the client is already taken. The system will prompt the client to choose a new event date. **Return to Step 3**. [ALT] **Step 10a**: PayPal will return a failed payment from the client. **Return to Step 8**. [ALT] **Step 10b**: The system failed to add the client’s booking. The system alerts the client of the error and that the booking wasn’t made successfully. **Terminate use case**. [ALT] **Step 10c**: The system failed to add the client’s booking payment. The system alerts the client of the error, and that the payment record wasn’t successfully added. **Terminate use case**.		

POST-CONDITION:	The client receives a reference number in which he will give upon the entrance of the resort and will change the status of the event booking “Booked”.

USE CASE NAME:	Cancel event booking	USE CASE TYPE	
USE CASE ID:	5.3	Business Requirements:	o
PRIORITY:	Medium	System Analysis:	þ
SOURCE:	Platinum Island Resort	System Design:	o
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	This use case describes an occurrence in which the client wishes to cancel their booking details. The use case begins when the client calls the administrator of platinum island, and provides the administrator with the reference number, the administrator locates the event and cancels it as indicated, and it concludes when the booking is cancelled, and the client is notified via email.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The client had to have booked an event booking at Platinum Island. • The system has loaded the *'Events'* screen. • The administrator clicked the "cancel" button.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	MANUAL ACTION	SYSTEM RESPONSE:
			1. The system responds by displaying a pop-up "*Cancel Event*" modal with the following elements: – Modal Heading Text: "Confirm event cancellation" – Label: "Are you sure you want to cancel this booking"? – Submit button with

			the text "Confirm". – Cancel button with the text "Cancel". – Submit button with the text: "Submit".
		2. The administrator confirms that the client would like to cancel their booking.	
	3. The client chooses to cancel their booking. [ALT]	**4.** The administrator selects the Confirm button.	**5.** Using the matching *ReferenceCode* attribute within the **Event** entity which corresponds to the appropriate matching record, the system will update the status of the client's booking by reading the *Name* attribute from the **EventStatus** entity depending on when the cancellation of the booking occurs: – If the clients' event cancellation occurs within 48 hours before their event check-in, which can be validated using the Date attribute within

			the Event entity and the current date of cancellation using the Date.Now() function, the event booking will have a status of “Cancelled” – If the clients’ event cancellation occurs more than 48 hours before their event check-in, which can be validated using the Date attribute within the Event entity and the current date using the Date.Now() function of cancellation, the event booking will have a status of “Refund Pending”. Where the client will be eligible for a 25% return on their event payment. Using the *ClientEmail* attribute, which is retrieved from the **Client** entity, the system sends an Email to the client with

			the following information: – Subject: [Event Cancellation - Important Update – Dear {{*ClientName*}}, – Email Body: “We regret to inform you that the event scheduled for {{*Date*}} has been cancelled due to unforeseen circumstances. – Email Footer: “Best regards, The Platinum Island team”
ALTERNATE: COURSES:	[ALT] **Step 3**: the client does not wish to cancel the booking. The administrator selects the cancel button. Terminate use case.		
POST-CONDITION:	• The venue booking status has been updated in the **Event** entity to “cancelled”.		

USE CASE NAME:	Send Event Booking Reminder		USE CASE TYPE
USE CASE ID:	5.4		Business Requirements: o
PRIORITY:	High		System Analysis: o
SOURCE:	Platinum Island Resort		System Design: þ
PRIMARY BUSINESS ACTOR:	Time		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	Client (ERA)		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case begins when the system detects that the current time is 24 hours before the client's check-in time. The system should send an email to the client 24 hours before their booking check-in reminding them of their booking reservation. The use case concludes when the system sends an email to the client detailing the time of their check-in, in 24 hours' time.		
PRE-CONDITION:	• The system detects that the client's event reservation is due in 24 hours.		
TRIGGER	Time detects it is 24 hours before the client's check-in time.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Time detects it is 24 hours before the client's check-in time.		**2.** Using .NET Core backend, the system will periodically check for booking check-ins' which are due, by reading the *Reminder_Time* attribute from the **Reminder** entity to determine how many

			hours before the notification reminder is sent to the client, by reading the *Date (DateTime)* attribute from each record in the E**vent** entity. Which then prompts the use of a LINQ Query which utilizes an SQL_Read query to retrieve the Client_ID.*ClientEmail* (varchar(50)) [FK] attribute from the E**vent** entity by matching the *Client_ID (int) [FK]* attribute to the *Client_ID (int) [PK]* attribute within the **Client.** [ALT]
			3. The system will then use the MailKit library in the backend to construct an email reminder containing the booking summary details particular to the client's booking by reading the following attributes from the E**vent** entity: ○ Event_ID [PK] (int) ○ EventStatus_ID [FK] (int) ○ Client_ID [FK] (int) ○ Venue_ID [FK] (int)

			○ Gueslist (varbinary) ○ ReferenceCode (varchar (6)) ○ Date (Date) ○ EventTotal (decimal 6,2) The system will then use the *ClientEmail* attribute retrieved to generate an email which is sent to the client's email to notify them of their accommodation reservation in 24 hours' time which contains the following elements: – Subject heading with the text "Platinum Island Resort Booking Reminder" – Recipient is populated with the client's email address – {{ClientEmail}} – Salutation with the text "Dear {{ClientName}}" – Email body reminding the client of their booking and provides a summary of the

			booking made by the client. Additionally with the unique booking reference number. – Email closing with the text "We can't wait to see you there!" – Signature with the details of the resort.
ALTERNATE COURSES:	[ALT] **Step 2**: The system confirms that there are no check-in's due to the resort in 24 hours by reading the *Date* attribute from the **Event** entity. Terminate use case.		
CONCLUSION:	The use case concludes when the system sends an email to the client detailing the time of their check-in, in 24 hours' time.		
POST-CONDITION:	A reminder notification email will be sent to the client detailing their booking summary and check-in information.		
BUSINESS RULES:	• The accommodation reservation notification is sent to the client 24 hours before their check-in.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	View Event Prices	USE CASE TYPE	
USE CASE ID:	5.5	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where an administrator wishes to view the prices of the events on the system. The use case begins when the admin clicks on the view event prices option in the side navbar. The system will then load the event prices screen. On this view, the administrator will be able to update the event prices from the list. The use case concludes when the system displays all the event price records.		
PRE-CONDITION:	• The user must have access to internet. • The administrator must be logged onto the system.		
TRIGGER:	The administrator wishes to view the event prices and clicks the event prices side navbar options.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The administrator will request to view the *'event prices'* screen by clicking on the event prices option on right panel side navigation bar.		**2.** The system will send a request from the angular frontend to the data service class where the service will make a HttpGet request to the .Net core backend which makes use of a linq and lambda query which utilizes the SQL_READ query to retrieve records from the **VenuePrice** entity and displays the following attributes:

			○ VenuePrice_ID [PK] (int) ○ Venue_ID [FK] (int) ○ VenuePriceAmount (decimal 6,2) ○ Date (date)
			3. The system responds by loading the '*View Event Price*' screen details with the following information: – Heading: "Events Prices" – Table with the headings: ○ Event ○ Price ○ Empty header – The system will prepopulate the second column's rows within the table with the *VenuePriceAmount* attribute from the **VenuePrice** entity as follows: ○ "Price" with the Price of the venue. – The system will prepopulate the first columns rows within the *VenueName* attribute from the **Venue** entity as follows: ○ {{VenueName}} with the Name of the venue. – Empty header with the edit button with a pencil icon.
	4. The employee wishes to update a venue event price.		**5.** The system Invokes Use case "5.6 Update Event Price".
ALTERNATE COURSES:	None		

CONCLUSION:	The case ends when the user can view the event price details that are displayed to the administrator.
POST-CONDITION:	• The "*View Event Price*" screen is displayed to the administrator.
BUSINESS RULES:	None
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Update Event Price	USE CASE TYPE	
USE CASE ID:	5.6	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PARTICIPATING ACTORS:	Administrator (PBA)		
DESCRIPTION:	This use case describes the event where an administrator wants to update the ticket prices of a specific category. The use case begins when the *'event price'* screen is loaded, and the administrator selects the edit button to edit a price. The use case ends once the updated price is saved in the database.		
PRE-CONDITION:	• The user must have access to internet. • The user must be logged onto the system. • The view event price screen is loaded.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system loads the "*Edit Event Price*" modal popup. Using the *Venue_ID* sent from the Angular front-end to the .NET Core backend, the system will match the *Venue_ID* selected to a specific record in the **VenuePrice** Entity using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the event price record. The "*Edit Event Price*" modal has the following elements: – Heading: Edit Event Price – Label: Venue Name – Disabled textbox with the prepopulated data of the selected venue with the attribute

			{{VenueName}} in the **Venue** entity. – Label: Venue Price – Numeric up down with the prepopulated price of the selected venue price with the attribute ▪ Which is restricted from selecting a numeric value below the amount of 0. – {{VenuePrice}} in the **VenuePrice** entity. – Submit Button with the text: “Save”. – Cancel Button with the text: “Cancel”. [The submit button is disabled until all the fields are validated. The cancel button is enabled on default] The system would prompt the administrator to update the prepopulated venue price details.
	2. The administrator will enter the new price of the event in the numeric updown.		**3.** The system will use the angular frontend to ensure that none of the input fields are left blank and will enable the save button. [ALT]
	4. The administrator clicks on the “Save” button. [ALT]		**5.** The system will capture the information entered by the administrator and will populate the *eventPrice* object with the attributes: – VenuePriceAmount – Date (which will be populated using the Date.Now() function)

			The *eventPrice* object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
			6. The system will make use of an SQL_Update query in the controller to update the VenuePrice record within the **VenuePrice** Entity: ○ VenuePrice_ID (int) [PK] ○ Venue_ID (int) [FK] ○ VenuePriceAmount (int) ○ Date (date)
ALTERNATE COURSES:	[ALT] **Step 3**: The information entered by the administrator returns a validation error which is displayed to the administrator. Return to Step 2. [ALT] **Step 4**: The administrator chooses to not update the selected field and selects the cancel button. Terminate use case.		
POST-CONDITION:	• The new price details are displayed on the '*View event price'* screen.		

2.6. Subsystem 6 – Administration

USE CASE NAME:	View Employee	USE CASE TYPE	
USE CASE ID:	6.1	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view employee records available of the system. The use case begins with the administrator requests to access the 'Employee' View by navigating from the navigation bar. On this view, the administrator will be able to create, update and delete employees on the system. The use case concludes when the system displays all the employee records.		
PRE-CONDITION:	• The administrator needs to be logged onto the system.		
TRIGGER:	The administrator requests to view Employee screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
	1. Administrator will request to view the *Employee Screen* by clicking the "View Employee" tab option on the navigation bar.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Employee**

			Entity which has the following attributes: ○ Employee_ID (int) [PK] ○ EmployeeType_ID (int) [FK] attribute of the EmployeeType entity where the EmployeeType_ID in the EmployeeType entity corresponds to the EmployeeType_ID in the Employee entity. ○ User_ID (int) [FK] attribute of the User entity where the User_ID in the User entity corresponds to the User_ID in the Employee entity. ○ ID_Number (varchar 20) ○ Cell_Num (varchar 20) ○ EmployeeName (varchar 20) ○ HireDate (date)
			3. The system will load the ***'Employee'*** screen with the following elements: – Label with the text "Employees" – "**+**" button – Label with the text "Add Employee" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter employee name". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right.

			Employee Table with the following headers: ○ "Full Name" ○ "ID Number" ○ "Employee Type" ○ "Cell Number" ○ Empty Header The system will populate each row within the Employee Table with the records read from the **Employee Entity** as follows: – "Full Name" with the *EmployeeName* – "ID Number" with the *ID_Number* – "Cell Number" with the *Cell_Num* The system will populate each row within the Employee Table with the records read from the **Employee Type Entity** as follows: – "Employee Type" with the *TypeName.* *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*
	4. The administrator requests to add an employee record. [ALT]		5. . The system extends to "6.2 Add Employee".

ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an Employee. The system will prompt the administrator to enter the Employee details within the search bar. The administrator will enter either the 'Name' OR 'ID Number' OR 'Cell Number' of the employee. The system retrieves and displays a list of all the employee records that match the search criteria entered by the administrator using the following attributes from the **Employee Entity**: ○ EmployeeName ○ ID_Number ○ Cell_Num [ALT] **Step 4b**: The administrator requests to update an employee. The system extends to Use Case 6.3 "Update Employee". [ALT] **Step 4c**: The administrator requests to delete an Employee. The system extends to Use Case 6.4 "Delete Employee".
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Employee Table.
POST-CONDITION:	The administrator will be able to add an employee on the *'Employee'* screen.
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Employee records on the system.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Add Employee	USE CASE TYPE
USE CASE ID:	6.2.	Abstract: ¨
PRIORITY:	High	**Extension:** x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new Employee record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it within the **Employee Entity**. The use case concludes when the Employee record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'View Employee'* screen. • The administrator clicked the "**+**" button (Add Employee).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Employee*" modal with the following elements: – Modal Heading Text: "Add Employee"; Close button. – Label: "Full Name" – Employee full name text input field with the placeholder text "Enter employee's full name". – Label: "ID Number" – Employee ID number with the placeholder text "Enter valid ID". – Label: "Cell Number" – Employee cell number text input field with the placeholder text "Enter employee's cell number". – Label: "Employee Type" – A combo box dropdown which is populated using the 'TypeName' attribute from the **Employee Type Entity**. With the placeholder text "Select an option…" – Submit button with the text "Save".

		– Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		3. The system prompts the administrator to enter the new Employee details.
	4. The administrator will enter the required information within the text input fields.	**5.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Submit” button. [ALT]
	6. The administrator clicks the “Submit” button. [ALT]	**8.** The system will capture the information entered by the administrator and will populate an employee object with the attributes: – Full Name – ID Number – Employee Type – Cell Number The Employee object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		9. The system will make use of an SQL_Insert query in the controller to create the new record within the **Employee Entity**: ○ Employee_ID (int) [PK] (value of the previous Employee_ID, incremented by 1) ○ EmployeeType_ID (int) [FK] ○ User_ID (int) [FK] ○ ID_Number (varchar 20) ○ Cell_Num (varchar 20) ○ EmployeeName (varchar 20) ○ HireDate (Date) (when the employee is added on the add event handler, it will create a new date object, when the button is clicked, and will save it to HireDate

		attribute) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employee details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).	
POST-CONDITION:	The new Employee record has been added to the system in the Employee table.	

USE CASE NAME:	Update Employee	USE CASE TYPE	
USE CASE ID:	6.3	Abstract:	¨
PRIORITY:	High	**Extension:**	x
SOURCE:	Platinum Island Resort		
PARTICIPATING ACTORS:	Administrator		
DESCRIPTION:	This use case describes the event where the administrator updates an employee record on the system. The use case begins when the administrator chooses to update an employee record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Employee record has been successfully updated on the system.		
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Employee'* screen. • The administrator clicked the update button.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system responds by displaying a pop-up "*Edit Employee*" modal. Using the Employee_ID sent from the Angular front-end to the .NET Core backend, the system will match the Employee_ID selected to a specific record in the **Employee Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the Employee. The "*Edit Employee*" modal has the following elements: – Modal Heading Text: "Edit Employee"; Close button. – Label: "Full Name" – Employee name text input field with the pre-populated Full name text - {{EmployeeName}} – Label: "ID Number" – Employee ID Number text input field with the pre-populated ID	

		Number text – {{ID_Number}} – Label: “Cell Number” – Employee Cell Number text input field with the pre-populated cell number text – {{Cell_Num}} – Label: “Employee Type” – A combo box dropdown which is pre-populated using the ‘TypeName’ attribute from the **Employee Type Entity** {{EmployeeType.TypeName}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated Employee details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an employee object with the attributes: – Full Name – ID Number – Employee Type – Cell Number The Employee object will be sent to the Data Service where

		it will send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the Employee record within the **Employee Entity**: ○ Employee_ID (int) [PK] ○ EmployeeType_ID (int) [FK] ○ User_ID (int) [FK] ○ ID_Number (varchar 20) ○ Cell_Num (varchar 20) ○ EmployeeName (varchar 20)
ALTERNATE COURSES:	[ALT] **Step 4a**: The system detects that the information fields entered by the administrator was left blank. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employee details. **Return to Step 3**. [ALT] **Step 4b**: The system detects that the information fields entered by the administrator was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employee details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant Employee category information has been successfully updated in the **Employee Entity**.	

USE CASE NAME:	Delete Employee	USE CASE TYPE	
USE CASE ID:	6.4	Abstract:	¨
PRIORITY:	High	**Extension:**	x
SOURCE:	Platinum Island Resort		
PARTICIPATING ACTORS:	Administrator		
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific employee record. The system displays a modal to confirm the deletion, the admin confirms the employee deletion. The use case concludes when the employee record has been deleted from the system.		
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Employee'* screen. • The administrator clicked the delete button. • The employee must exist on the system before it can be deleted.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system responds by displaying a pop-up "*Delete Employee*" modal with the following elements: – Modal Heading Text: "Confirm Employee Deletion"; Close button. – Label: "Are you sure you want to delete this employee record?" – Submit button with the text "Confirm". – Cancel button with the text "Cancel".	
		4. The system prompts the administrator if they would like to delete the selected record.	
	5. The administrator clicks the "Confirm" button. [ALT]	**6.** The system deletes the employee record using an HttpDelete request from the **Employee Entity** with the following attributes: ○ Employee_ID (int) [PK] ○ EmployeeType_ID (int) [FK] ○ User_ID (int) [FK] ○ ID_Number (varchar 20) ○ Cell_Num (varchar 20) ○ EmployeeName (varchar 20) ○ HireDate (date)	

		The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 4**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant employee information has been successfully deleted from the **Employee Entity**.	

USE CASE NAME:	View Employee Type	USE CASE TYPE	
USE CASE ID:	6.5.	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view Employee Type records available of the system. The use case begins with the administrator requests to access the 'Employee Type' by navigating through the side navigation bar. On this view, the administrator will be able to create, update and delete employees' type on the system. The use case concludes when the system displays all the Employee Type records.		
PRE-CONDITION:	• The administrator needs to be logged onto the system.		
TRIGGER:	The administrator requests to view Employee Type screen on the system.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Administrator will request to view the *Employee Type Screen* by hovering over the 'Employees' tab option where the side navigation bar will display two routing options: ○ 'View Employees' ○ 'Employee Type'		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records

	The administrator will then click on "Employee Type" side-navbar option.		from the **Employee Type Entity** which has the following attributes: ○ Employee Type_ID (int) [PK] ○ TypeName (varchar 20) ○ TypeDescription (varchar 30)
			3. The system will load the ***'Employee Type'*** screen with the following elements: – Label with the text "Employee Type" – "**+**" button – Label with the text "Add Employee Type" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter Employee Type". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right. Employee Type Table with the following headers: ○ "Employee Type" ○ "Employee Description" ○ Empty Header The system will populate each row within the Employee Type Table with the records read from the **Employee Type Entity** as follows: – "Employee Type" with the *TypeName.*

			– "Employee Description" with the *TypeDescription.* *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*
	4. The administrator requests to add an Employee Type record. [ALT]		**5.** The system extends to "6.6. Add Employee Type".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an Employee Type. The system will prompt the administrator to enter the Employee Type details within the search bar. The administrator will enter either the 'Employee type OR 'Employee Description' of the Employee Type. The system retrieves and displays a list of all the Employee Type records that match the search criteria entered by the administrator using the following attributes from the **Employee Type Entity**: – TypeName – TypeDescription [ALT] **Step 4b**: The administrator requests to update an Employee Type. The system extends to Use Case 6.7. "Update Employee Type". [ALT] **Step 4c**: The administrator requests to delete an Employee Type. The system extends to Use Case 6.8. "Delete Employee Type".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Employee Type Table.		
POST-CONDITION:	The administrator will be able to add an Employee Type on the *'Employee Type'* screen.		
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Employee Type records on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		

ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Add Employee type	USE CASE TYPE
USE CASE ID:	6.6	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new EmployeeType record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it within the **EmployeeType Entity**. The use case concludes when the Employee record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'EmployeeType'* screen. • The administrator clicked the "**+**" button (Add EmployeeType).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Employee Type*" modal with the following elements: – Modal Heading Text: "Add Employee Type"; Close button. – Label: "Employee Type" – Employee type text input field with the placeholder text "Enter Employee Type". – Label: "Description" – Employee Description with the placeholder text "Provide enter a description, max of 30 characters". – Submit button with the text "Submit". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all

		text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new Employee type details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the "Submit" button. [ALT]
	5. The administrator clicks the "Submit" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an EmployeeType object with the attributes: – TypeName (varchar 20) – TypeDescription (varchar 30) The EmployeeType object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **EmployeeType Entity**: ○ EmployeeType_ID (int) [PK] (value of the previous Employee_ID, incremented by 1) ○ TypeName (string) ○ TypeDescription (string) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employeetype details. **Return to Step 3**.	

	[ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request. (400 bad request).
POST-CONDITION:	The new Employee type record has been added to the system in the EmployeeType table.

USE CASE NAME:	Update Employee Type	USE CASE TYPE
USE CASE ID:	6.7	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates an Employee Type record on the system. The use case begins when the administrator chooses to update an Employee Type record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Employee Type record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Employee Type'* screen. • The administrator clicked the update button.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit Employee Type*" modal. Using the EmployeeType_ID sent from the Angular front-end to the .NET Core backend, the system will match the EmployeeType_ID selected to a specific record in the **Employee Type Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the Employee type. The "*Edit Employee Type*" modal has the following elements: – Modal Heading Text: "Edit Employee Type"; Close button. – Label: "Employee Type"

		– Employee Type name text input field with the pre-populated name text {{TypeName}} – Label: “Description” – Employee Type description text input field with the pre-populated description text {{Description}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated Employee type details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Employee Type object with the attributes: – TypeName – TypeDescription The Employee Type object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the Employee Type record within the **Employee Type Entity**: ○ TypeName (string) ○ TypeDescription (string)

ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employee type details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.
POST-CONDITION:	The relevant Employee type information has been successfully updated in the **Employee Type Entity**.

USE CASE NAME:	Delete Employee Type	USE CASE TYPE
USE CASE ID:	6.8	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific Employee type record. The use case begins with the system verifying there are no Employees associated with the Employee type. The system will confirm the Employee type deletion before the administrator confirms the deletion of the record. The use case concludes when the Employee type record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Employee Type* screen. • The administrator clicked the delete button. • The Employee type must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system matches the EmployeeType_ID [PK] for the employee type selected by the administrator from the **EmployeeType Entity**, with the EmployeeType_ID [FK] retrieved from the **Employee Entity**, to ensure no employees are associated with the selected employee type. *The system will do this by performing an SQL_Read query in the .NET Core controller.* [ALT]
		2. The system responds by displaying a pop-up "*Delete Employee Type*" modal with the following elements: – Modal Heading Text: "Confirm Employee Type Deletion"; Close button.

		– Label: “Are you sure you want to delete this Employee type?” – Submit button with the text “Confirm”. – Cancel button with the text “Cancel”.
		3. The system prompts the administrator if they would like to delete the selected record.
	4. The administrator clicks the “Confirm” button. [ALT]	**5.** The system deletes the Employee type record from the **EmployeeType Entity** with the following attributes: ○ EmployeeType_ID (int) [PK] ○ TypeName (string) ○ TypeDescription (string) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 1**: The system determines that there are Employees that are associated with the selected Employee Type. The system will throw an error. **Terminate use case**. [ALT] **Step 4**: The administrator selects the Cancel button. The use case terminates.	
POST-CONDITION:	The relevant Employee type information has been successfully deleted from the **EmployeeType Entity**.	

USE CASE NAME:	View Accommodations	USE CASE TYPE	
USE CASE ID:	6.9	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view the accommodation bookings that were made in the resort. The use case begins with the administrator requests to access the 'Rooms' View by through the dashboard. On this view, the administrator will be able to check-in and check-out client accommodation bookings as well as update and cancel client accommodation bookings at their request. The use case concludes when the system displays all of the room booking records that need to be checked-in and checked-out.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system displays the dashboard screen.		
TRIGGER:	The administrator requests to view Rooms screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
	3. Administrator will request to view the "*View Accommodations*" screen by hovering over the 'Accommodation' tab option where the side navigation bar will display two routing options:		**4.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilises an SQL_Read query to retrieve the records from the **RoomBooking** entity which has the following attributes: ○ RoomBooking_ID (int) [PK] ○ Client_ID (int) [FK]

	○ ‘Rooms’ ○ ‘Room Types’ The administrator will then click on “Rooms” side-navbar option.		○ RoomBookingStatus_ID (int) [FK] ○ Reminder_ID (int) [FK] ○ ReferenceNum varchar(10) ○ BookingDate (DateTime) ○ NumberOfGuests (int) ○ EntryDate (DateTime) ○ ExitDate (DateTime) Using the LINQ Query, the system will save which records are due to be checked-in by matching the *EntryDate* attribute to the current date variable as well which records are due to be checked-out by matching the *ExitDate* attribute from the **RoomBooking** entity to the current date variable. The system will save the retrieved records from the check-in and check-out to their appropriate Lists which will be retrieved by the frontend, Angular. [ALT]
			5. The system will load the ‘*View Accommodations*’ screen with the following elements: – Label with the text “Accommodation” – Search bar with the placeholder text of “Enter reference number”. – Font Awesome Icon ‘*fa fa-search*’ within the search bar, on the right. – Label with the text “Check-In” – Check-In Table with the following headers: ○ “Reference Number” ○ “Full Name” ○ “Email Address” ○ “Contact No.” ○ “Check-In Date” ○ Empty Header #1 ○ Empty Header #2 The system will populate each row within the Check-In Table with the records read from the **RoomBooking** entity that have

			been filtered and retrieved from the ASP.NET backend as follows:
			– "Reference Number" with the {{*ReferenceNum}}* attribute
			– "Full Name" with the concatenation of the client's first and last name {{*Client_ID.ClientName* + " " + *Client_ID.ClientSurname}}*
			– "Email Address" with the {{Client_ID.ClientEmail}}
			– "Contact No." with the {{Client_ID.ClientPhone}}
			– "Check-In Date" with the {{EntryDate }}
			
			– *Empty header #1 with:*
			  ○ *{{RoomBookingStatus_ID.Name}} which is text that placed within a button.*
			– *Empty header #2 with:*
			  ○ Delete button with a trash-can icon (*fa fa-trash-o).*
			
			– Label with the text "Check-Out"
			– Check-Out Table with the following headers:
			  ○ "Reference Number"
			  ○ "Full Name"
			  ○ "Email Address"
			  ○ "Contact No."
			  ○ "Check-Out Date"
			  ○ Empty Header
			
			The system will populate each row within the Check-Out Table with the records read from the **RoomBooking** entity that have been filtered and retrieved from the ASP.NET backend as follows:
			– "Reference Number" with the {{*ReferenceNum}}* attribute
			– "Full Name" with the concatenation of the client's first and last name {{*Client_ID.ClientName* + " " + *Client_ID.ClientSurname}}*
			– "Email Address" with the {{Client_ID.ClientEmail}}
			– "Contact No." with the {{Client_ID.ClientPhone}}
			– "Check-In Date" with the {{EntryDate }}

			– *Empty header with:* ○ *{{RoomBookingStatus_ID.Name}} which is text that placed within a button.*
	6. The administrator clicks the "Check-In" button. [ALT]		**7.** The system extends to "6.10 Room Check-In".
ALTERNATE COURSES:	[ALT] **Step 2**: The system returns null records that are due to be checked-in or checked-out. [ALT] **Step 4a:** The administrator requests to search a client's accommodation booking. The system will prompt the administrator to enter the reference number of the client's accommodation booking within the search bar. The system retrieves and displays the selected room booking record that matches the search criteria entered by the administrator using the following attributes from the **RoomBooking** entity: – *ReferenceNum* [ALT] **Step 4b**: The administrator requests to check-out a client from the resort. The system extends to Use Case 6.11 "Room Check-Out". [ALT] **Step 4c**: The administrator requests to cancel a client's accommodation booking. The system extends to Use Case 3.3 "Cancel Room Booking".		
CONCLUSION:	The use case concludes when the system displays all of the room booking records that need to be checked-in and checked-out.		
POST-CONDITION:	The administrator will be able to check-in, check-out room bookings as well as cancel them on the *'View Accommodations'* screen.		
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Room Booking records on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Room Check-In	USE CASE TYPE	
USE CASE ID:	6.10	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	The use case begins when the client initiates the check-in process into their room booking. The system should allow the administrator to check in a client's room booking. Whereby the client will be asked to confirm their booking before their room booking record is retrieved from the RoomBooking entity. Once the administrator has found and confirmed the client's room booking, the status of their room booking will be changed to "Checked-in" and the admin will hand over the room keys to the client. The use case concludes when the administrator hands over the room keys to the client and their room booking status is changed to "Checked-In"		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The client's accommodation must have been made prior to check-in. • The screen displayed the "View Accommodations" screen..		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
		1. The administrator requests the client to provide their room booking details, such as their name, surname, or reference number.	
	2. The client will provide their room	**3.** The employee navigates to the search bar where the	**4.** The system sends the search criteria from the component.html to the component.ts, where the search criteria will be stored within a string variable which can be

	booking details.	employee enters the provided info and clicks on the search icon button.	used to filter the Room Booking displayed on the system using a LINQ Query on the room booking records retrieved from the **RoomBooking** entity. The system populates the Check-In table with the result of the search criteria. [ALT]
		5. The administrator clicks the "Check-In" button specific to the client's room booking found within the Check-In table.	**6.** The system responds by retrieving the client's accommodation booking by using the *RoomBooking_ID* attribute send from the Angular frontend as parameters to the .NET Core backend. Using an HttpGet request, the system will retrieve the client's record from the **RoomBooking** entity with the following attributes: ○ RoomBooking_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomBookingStatus_ID (int) [FK] ○ Reminder_ID (int) [FK] ○ ReferenceNum varchar(10) ○ BookingDate (DateTime) ○ NumberOfGuests (int) ○ EntryDate (DateTime) ○ ExitDate (DateTime) The system will then load the "Check-In Room" Modal with the following elements: – Modal header with the text "Check-In Room Booking" – Label with the text "Room Booking Reference" – Text input field which is pre-populated with the client's room booking reference number – {{*ReferenceNum*}} – Button with the text "Confirm Booking Check-In"

			[ALT]
		7. The administrator requests the client to confirm their room booking reference number to the booking on the system.	
	8. The client provides their room booking reference number.	**9.** The administrator will match the reference number given by the client to the reference number pulled up on the screen and will click the "Confirm Booking Check-In" button. [ALT]	**10.** The system will respond by removing the client's room booking record from the Check -in table and storing the Room Booking record within an array which will be retrieved when the client's check-out date is due. The system updates the *{{RoomBookingStatus_ID.Name}}* in the **RoomBookingStatus** entity to "Checked-In" as an updated record.
ALTERNATE COURSES:	[ALT] **Step 4**: No record could be found that match the search criteria given by the client. **Return to step 2**. [ALT] **Step 6**: The system could not retrieve the Room Booking and Room Booking status information from the **RoomBooking** entity. **Terminate use case**. [ALT] **Step 9a**: The reference number given by the client does not match the reference number displayed on the system. **Return to step 8**. [ALT] **Step 9b**: The room is not yet ready to be checked-in. Inform Client to come back later. **Return to step 1**.		
POST-CONDITION:	The room booking status of the client's accommodation is updated to "Checked-In"		

USE CASE NAME:	Room check-out	USE CASE TYPE	
USE CASE ID:	6.11	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	This use case describes the event where the Client request to check-out after using the venue. The administrator responds by facilitating the check-out process. The administrator will search for the client specific event booking on the system. The system will retrieve the details of the event entity. Thereafter the administrator will confirm details from the client. The use case concludes when the administrator checks the client out of the system. The status in the **RoomBookingStatus** entity will update to "Checked Out"		
PRE-CONDITION:	• Administrator needs to be logged in. • The *'View accommodations'* screen is already loaded. • The client has already checked-in the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
		1. The administrator asks the client for the booking reference number.	
	2. The client provides their booking reference number.	**3.** The employee navigates to the search bar where the administrator enters the provided information.	**4.** The search criteria of the Booking submitted by the administrator will be used to retrieve all the event bookings that match the search criterion with the Database Model Array by using the Search Booking function which retrieves all the Bookings from the Database Model which includes the search phrase by comparing it with the booking reference attributes in the **RoomBooking** Entity. The system populates the Booking table with the result of the search criteria. The **RoomBooking** entity has the following attributes: ○ Roombooking_ID [PK]

			○ RoomBookingStatus_ID [FK] ○ Client_ID [FK] ○ Reminder_ID [FK] ○ ReferenceNum ○ BookingDate ○ NoOfGuests [ALT]
		5. The employee navigates to the “Check-Out” button and clicks the button	**6.** The object will be sent to the booking service where it will send an http put request to the .Net Core controller which will use an Entity Framework Update method to update the **RoomBooking** details. The system updates the Name in the **RoomBookingStatus** Entity to “Checked Out” as an updated record. A Success message will be displayed to notify the Employee that the client was successfully checked out.
		7. The administrator informs the client that the check-out was successful.	
ALTERNATE COURSES:	[ALT] **Step 5**: No matches found. The system could not retrieve any bookings.		
POST-CONDITION:	• The Name attribute in the **RoomBookingStatus** entity has been changed to “Checked Out”.		

USE CASE NAME:	View VAT	USE CASE TYPE
USE CASE ID:	6.12	Business Requirements:o
PRIORITY:	Medium	System Analysis:o
SOURCE:	Platinum Island Resort	System Design:þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes the event where the administrator would like to view the VAT information on the system. The use case begins with the administrator requesting to view the VAT information on the system. The system will load and display it. The use case concludes when the administrator views the VAT screen.	
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system displays the dashboard screen.	
TRIGGER:	The administrator requests to view the VAT screen on the system.	

	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
TYPICAL COURSE OF EVENTS:	**1.** The administrator requests to view the VAT by navigating to the side nav bar and will hover on the settings option where a dropdown will be shown having the following options: o VAT o Audit Log o Reminder Timer The administrator will click on the VAT option.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **VAT Entity** which has the following attributes: o VAT_ID (int) [PK] o VatAmount (numeric)

			o Date (datetime)
			3. The system will load the '*Settings*' screen with the following elements: - Heading Label with the text "VAT" - "**+**" button - Label with the text "Add VAT" to the right of the "**+**" button. - VAT Table with the following headers: o "VAT Percentage" o "Date Set" o Empty Header o Edit button with a pencil icon (*fa fa-pencil*). The system will populate each row within the VAT Table with the records read from the **VAT Entity** as follows: - "VAT Percentage" with the *VatAmount* attribute - "Date Set" with the *Date* attribute.
	4. The administrator requests to add a VAT record. [ALT]		5. . The system extends to "7.13 Create VAT".

ALTERNATE COURSES:	[ALT] **Step 4**: The administrator requests to update a VAT record. The system extends to Use Case 7.14 "Update VAT".
CONCLUSION:	The use case concludes when the system displays the VAT screen.
POST-CONDITION:	The system retrieves the VAT information, and the system displays the "View VAT" screen.

BUSINESS RULES:	The administrator only has access.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	The administrator has internet connection.
OPEN ISSUES:	None

USE CASE NAME:	Create VAT	USE CASE TYPE
USE CASE ID:	6.13	Abstract:¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new VAT record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the VAT record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'VAT'* screen. • The administrator clicked the "**+**" button (Add VAT).	

	Actor Actions	System Response
TYPICAL COURSE OF EVENTS:		**1.** The system responds by displaying a pop-up "*Create VAT*" modal with the following elements: - Modal Heading Text: "Create VAT"; Close button. - Label: "VAT Percentage" - VAT percentage input field - Submit button with the text "Submit". - Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the submit button is disabled until the text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new VAT percentage details.
	3. The administrator will enter the required information within the VAT percentage input field.	**4.** The system will use the Angular frontend to validate that the input field is not left blank and ensures the input entered by the administrator follows the following criteria:

		o The VAT percentage entered is a valid numeric input. o The VAT amount fits within the limits of the minimum percentage = 0; and the maximum percentage = 100. The system will enable the "Submit" button once the input field has been validated. [ALT]
	5. The administrator clicks the "Submit" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a VAT object with the attributes: o VATAmount o Date (which will create a new date object when the submit button is clicked in the format YYYY-MM-DD) The VAT object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **VAT Entity**: o VAT_ID (int) [PK] (value of the previous VAT_ID, incremented by 1) o VatAmount (decimal) o Date (datetime)
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the VAT percentage details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The new VAT record has been added to the system in the **VAT** table.	

USE CASE NAME:	Update VAT	USE CASE TYPE
USE CASE ID:	6.14	Abstract:¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates a specific VAT record on the system. The use case begins when the system prompts the administrator to enter the required updated information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the VAT record has successfully been updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'VAT'* screen. • The administrator clicked the edit button.	

	Actor Actions	System Response
TYPICAL COURSE OF EVENTS:		**1.** The system responds by displaying a pop-up "*Edit VAT*" modal. Using the VAT_ID sent from the Angular front-end to the .NET Core backend, the system will match the VAT_ID selected to a specific record in the **VAT Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the VAT record. The "*Edit VAT*" modal has the following elements: – Modal Heading Text: "Edit VAT"; Close button. – Label: "VAT Percentage" – VAT percentage input field with the pre-populated VatAmount – {{VatAmount}} – Submit button with the text "Save".

	– Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the submit button is disabled until the text input fields have been entered and validated.]
	2. The system prompts the administrator to enter the updated VAT percentage details.
3. The administrator will enter the required information within the VAT percentage input field.	**4.** The system will use the Angular frontend to validate that the input field is not left blank and ensures the input entered by the administrator follows the following criteria: ○ The VAT percentage entered is a valid numeric input. ○ The VAT amount fits within the limits of the minimum percentage = 0; and the maximum percentage = 100. The system will enable the "Save" button once the input field has been validated. [ALT]
5. The administrator clicks the "Save" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a VAT object with the attributes: - VatAmount The VAT object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
	7. The system will make use of an SQL_Update query in the controller to update the specific record within the **VAT Entity** with the following attributes: ○ VAT_ID (int) [PK] ○ VatAmount (decimal) ○ Date (datetime)

ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the VAT percentage details. Return to Step 3. [ALT] **Step 5**: The administrator selects the Cancel button. The use case terminates.	
POST-CONDITION:	The specific VAT record has been successfully updated on the system in the **VAT** table.	

USE CASE NAME:	View Audit Log	USE CASE TYPE	
USE CASE ID:	6.15	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view the audit log on the system. The use case begins with the administrator requesting to view the audit log on the system. The system will load and display it. The use case concludes when the administrator views the audit log.		
PRE-CONDITION:	• The administrator must be logged in to the system.		
TRIGGER:	The administrator requests to view the view audit log screen.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The administrator requests to view the audit log by navigating to the side nav bar and will click on the Audit Log option.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **AuditLog** entity which has the following attributes: ○ AuditLog_ID [PK] ○ UserID [FK] ○ Date (DateTime) ○ Time (DateTime)

			3. The system will respond by displaying the “View Audit Log” screen. The following are visible on the screen: ○ Heading Label: “Audit Log”. The system will populate each row within the Audit Log Table with the records read from the **AuditLog Entity** as follows: ○ Label text “User Name” ○ Table cell populated with User Name attribute – {User_ID.UserName}}. ○ Label text “Date” ○ Table cell populated with Date attribute – {{Date}}. ○ Label text “Time” ○ Table cell populated with Time attribute – {{Time}}.
ALTERNATE COURSES:	None		
CONCLUSION:	The system displays the “*View Audit Log*” screen.		
POST-CONDITION:	The system retrieves the audit log information, and the system displays the “*View Audit Log*” screen.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	The administrator has internet connection		

OPEN ISSUES:	None

USE CASE NAME:	View Reminder Timer	USE CASE TYPE	
USE CASE ID:	6.16	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view the reminder timer screen on the system. The use case begins with the administrator requesting to view the reminder timer on the system. The system will load and display it. The use case concludes when the administrator views the reminder log.		
PRE-CONDITION:	• The administrator must be logged in to the system.		
TRIGGER:	The administrator requests to view the reminder timer screen.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The administrator requests to view the reminder timer by navigating to the side nav bar and will hover on the settings option where a dropdown will be shown having the following options: – General Settings – Audit Log – Reminder Timer		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Reminder Entity** which has the following attributes:

	The administrator will click on the Reminder Timer option.		○ Reminder_ID (PK) ○ Reminder_Type (varchar 20) ○ Reminder_Time (Time)
			3. The system will respond by displaying the "View Reminder Timer" screen. The following are visible on the screen: ○ Heading Label: "Reminder Timer". ○ Table with heading labels: ○ Label text "ID" ○ Label text "Reminder Type" ○ Label text "Reminder Time" ○ Edit button with a pencil icon (*fa fa-pencil*)
	4. The administrator requests to update the reminder timer.		**5.** The system extends to Use Case 6.17 Update Reminder Timer.
ALTERNATE COURSES:	None		
CONCLUSION:	The use case concludes when the system displays the "View Reminder Timer" screen.		
POST-CONDITION:	The administrator will be able to edit a reminder timer on the "View Reminder Timer" screen.		
BUSINESS RULES:	Only the administrator can view the reminder timer on the syste,		
IMPLEMENTATION CONSTRAINTS	None		

AND SPECIFICATIONS:	
ASSUMPTIONS:	The administrator has internet connection
OPEN ISSUES:	None

USE CASE NAME:	Update Reminder Timer	USE CASE TYPE	
USE CASE ID:	6.17	Abstract:	☐
PRIORITY:	High	Extension:	x
SOURCE:	Platinum Island Resort		
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator updates an audit log record on the system. The system will retrieve the information of the selected record, where the system will prompt the administrator to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Audit Log record has been successfully updated on the system.		
PRE-CONDITION:	• The administrator must be logged onto the system. • The system has loaded the “Audit Log” screen. • The administrator clicked the update button.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by displaying a pop-up “*Edit Reminder Timer*” modal. Using the Reminder_ID sent from the Angular front-end to the .NET Core backend, the system will match the Reminder_ID selected to a specific record in the **Reminder Entity**

			using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the Reminder. The "*Edit Reminder Timer*" modal has the following elements: – Modal Heading Text: "Edit Reminder Timer"; Close button. – Label: "Reminder Type" – Reminder Type text input field with the pre-populated reminder type text – {{Reminder_Type}} – Label: "Reminder Timer" – Reminder Timer time input field – Submit button with the text "Save". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
			2. The system prompts the administrator to enter the updated reminder details.
	3. The administrator will enter the required updated information within the text input fields.		**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the "Save" button. [ALT]

	5. The administrator clicks the "Save" button. [ALT]		**6.** The system will capture the information entered by the administrator and will populate a reminder object with the attributes: ○ Reminder Type ○ Reminder Timer The Reminder object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
			7. The system will make use of an SQL_Update query in the controller to update the Reminder record within the **Reminder Entity**: ○ Reminder_ID (int)[PK] (value of the previous Reminder_ID, incremented by 1) ○ Reminder_Type(varchar 20) ○ Reminder_Time (Time) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Employee details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).		
POST-CONDITION:	The relevant Reminder information has been successfully updated in the **Reminder Entity**.		

USE CASE NAME:	View day visit booking	USE CASE TYPE
USE CASE ID:	6.18	Abstract: ☐
PRIORITY:	High	Extension: ☐
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Guard (PBA)	
DESCRIPTION:	This use case describes an event where the system displays the total amount of tickets bought per category. The use case starts when the client already scanned the QR Code on the and the system then displays the total amount of people who can enter the resort. The use case ends once the scan was successful, and the guard allows the client access to the resort.	
PRE-CONDITION:	• The guard has internet access. • The scan was successful	

	Actor Actions	System Response
TYPICAL COURSE OF EVENTS:		**1.** The ionic frontend will send through the RefCode that was scanned to the .Net core backend to retrieve the specific records in the **DayVisit** entity to retrieve the following attributes: ○ DayVisit_ID [PK] (int) ○ DayVisitStatus [FK] (int) ○ Client_ID [FK] (int) ○ DayVisitDate_ID (int) ○ RefCode (int) ○ Total (decimal (6,2)) The day visit type records will be retrieved via DayVisitType entity using the DayVisitType_ID[PK]which is connected to the DayVisitTicket entity via the DayVisitType_ID [FK] in the DayVisitTicket entity. The system then determines the number of DayVisitTickets that belong to each to DayVisitTicketType and populates the following variables as follows: – The number of kids selected as *totalKids*

		– The number of pensioners selected as *totalPensioners* – The number of total adults selected as *totalAdults* – The total amount of tickets booked as *TotalBooked*
		2. The system the displays the *View Day visit* screen with the following elements: – Heading: QR Code Valid: Admit {{*TotalBooked*}} – In an ionic item card, there are the following details: ○ Subheading: “Age Groups” ○ Label: “Number of kids” and the total number of kids as *totalKids* ○ Label: “Number of pensioners” and the total number of pensioners as *totalPensioners* ○ Label: “Number of adults” and the total number of adults as *totalAdults*
	3. The guard allows the total number of guests to enter the resort.	
ALTERNATE COURSES:	None	
POST-CONDITION:	The system displays the total amount of people that can enter.	

USE CASE NAME:	Check in ticket	USE CASE TYPE	
USE CASE ID:	6.19	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Client		
PRIMARY THE SYSTEM ACTOR:	Guard		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes an event where the client provides the QR code for the guard to scan for him to enter the resort, the system responds by loading the '*Check in' animation* screen which loads upon switching on the ionic application. The use case ends once the QR code is scanned and the client enters the resort.		
PRE-CONDITION:	• The client has made a ticket booking previously.		
TRIGGER:	The client wishes for their QR Code to be scanned by the guard and enter the resort.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The client wishes for their QR Code to be scanned by the guard and enter the resort.	**2.** The guard opens QR code scanner ionic application	**3.** The system responds by loading an animation screen with the following elements: – water droplet animation – Ripple effects once the raindrop stops. – A circle with the text: "HydroTech"

			4. The system then loads the screen with the following information: – Heading: Platinum Island QR scanner” – Button with the text: “Start Scan”. – Button with the text: “Stop scan”. – Button with the text: “Reset”. The system prompts the guard to scan a QR code.
		5. The guard requests the QR code details.	
	6. The client provides his QR Code.	**7.** The guard then scans the clients QR code.	**8.** The system verifies the QR code, to check that the embedded reference code matches the code in the database using an httpget method [ALT]
			9. The object will be sent to the DayVisit service where it will send an http put request to the .Net Core controller which will use an Entity Framework Update method to update the **DayVisitStatus** details. The system updates

			the Name in the **DayVisitStatus** Entity to "Checked In" as an updated record. The system then displays a link in a pop up as well as an ion-item with the following elements: – Heading: QR Code – The link that was displayed in the pop up. The system finally prompts the client to open the link.
		10. The guard then clicks the link popup that displays on top of the '*Check in*' ticket screen.	**11.** The system extends to use case "6.18 View Day visit"
ALTERNATE COURSES:	[ALT] **Step 8**: The scan was unsuccessful because it has expired. **Terminate use case**.		
CONCLUSION:	The system completes the scan and displays the link pop up.		
POST-CONDITION:	The popup link is displayed and opened by the guard		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	View events	USE CASE TYPE
USE CASE ID:	6.20	Business Requirements:o
PRIORITY:	High	System Analysis: o
SOURCE:	Platinum Island resort	System Design: þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes an event where the administrator would like to view the event records that are available on the system. The use case begins when the administrator requests to access the 'events view'. On this view, the administrator will be able to check clients in, check clients out, update bookings and cancel bookings. The use case concludes when the system displays all the event records.	
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system.	
TRIGGER:	The administrator wishes to view the event records on the system.	

	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
TYPICAL COURSE OF EVENTS:	**1.** The administrator wishes to view the event screen by hovering over the Events tab option where the side navigation bar will display two routing options. o 'Events' o 'Events Prices' The administrator will then click on the 'Events' option.		**2.** The system will send a request from the angular frontend to the data service class where the service will make a HttpGet request to the .Net core backend which makes use of a linq and lambda query which utilizes the SQL_READ query to retrieve records from the **Event** entity which has the following attributes: o Event_ID [PK] (int) o EventStatus_ID [FK] (int)

			o Reminder_ID [FK] (int) o Venue_ID [FK] (int) o Client_ID [FK] (int) o Date (date) o GuestList (varbinary) o ReferenceCode (varchar 6) o EventTotal (decimal 6,2)
			3. The system will load the *'View Events'* screen with the following details: – Heading: Events – Search bar with the placeholder: "Enter booking details". – Heading: "Events schedule" – Table with the headings: o "Reference code" o "Full name" o "Email address" o "Contact No." o "Events date" o "Guest list" o Empty header o Empty header Using the Client_ID attribute in the **Client** entity that corresponds to the Client_ID in the **Event** entity, the system populates each row within as follows: – "Full name" as the client's full name – "Email address" as the client's email address – "Contact No." as the client's

			cell phone number. Using the information retrieved from the **Event** entity, the system populates each row in the table as follows: – "Reference code" as the reference number – "Events date" as the date booked of the event. – "Guest list" as the guest list that was uploaded. – Empty header with the pencil icon and 'x' icon which will be used to update bookings and cancel bookings respectively. Using the EvenStatus_ID attribute in the **EventStatus** entity that corresponds to the EventStatus_ID in the **Event** entity, the system populates each row within as follows: – Empty header for the status of the booking

	4. The administrator wants to cancel a booking. [ALT]		**5.** The system extends to "5.3 Cancel Event booking".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an event booking. The system will prompt the administrator to enter the booking details within the search bar. The administrator will enter the 'Full name', 'Reference number', 'Contact Number' or 'Booking date'. The system retrieves a list of all the event records that match the criteria entered by the administrator using the following attributes from the **Event** and **Client** entity. [ALT] **Step 4b**: The client arrives at the resort and wishes to be checked in. The system extends to use case "6.21 Event check-in".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Event Table.		
POST-CONDITION:	The administrator will be able to update and cancel an event on the '*View Events*' screen.		
BUSINESS RULES:	• Only authorised users of the system will be permitted to view the events on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Event check-in	USE CASE TYPE	
USE CASE ID:	6.21	Business Requirements:o	
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PARTICIPATING ACTOR:	Client (PBS) Administrator (PSA)		
DESCRIPTION:	This use case shows the scenario in which a client arrives at the event and starts the check-in procedure. In response, the administrator helps with check-in. The administrator will look up the system's booking of a certain client event. The system will retrieve the **Event** entity's information. The administrator will then confirm the information with the client. The use case ends when the client is checked-In and the status in the **EventStatus** entity is changed to “Checked In”.		
PRE-CONDITION:	• Administrator needs to be logged in. • The *‘View event’* screen is already loaded. • The client must have a valid booking. • The client has arrived on the correct check-in date.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The Client arrives at the event and requests to check in.	**2.** The administrator responds by requesting the client to provide the booking reference number from the person who made the booking.	
	3. The client provides their booking reference number.	**4.** The administrator navigates to the search bar where they enter the provided information.	**5.** The search criteria of the Booking submitted by the administrator will be used to retrieve all the event bookings that match the search criterion with the Database Model Array by using the Search Booking function which retrieves all the Bookings from the Database Model which includes the search phrase by comparing it with the booking reference attributes in the **Event** Entity. The system

			populates the Event booking table with the result of the search criteria. The **Event** entity has the following attributes: o Event_ID [PK] (int) o EventStatus_ID [FK] (int) o Client_ID [FK] (int) o Venue_ID [FK] (int) o Reminder_ID [FK] (int) o GuestList (varbinary) o ReferenceCode (varchar (6)) o Date (Date) o EventTotal (decimal (6,2)) [ALT]
		6. The employee navigates to the "Check-In" button and clicks the button	**7.** The object will be sent to the event service where it will send an http put request to the .Net Core controller which will use an Entity Framework Update method to update the **EventStatus** details. The system updates the Name in the **EventStatus** Entity to "Checked In" as an updated record. A Success message will be displayed to notify the Employee that the Guest was successfully checked in.
		8. The administrator informs the client that the check in was successful and allows the client to use the venue.	
ALTERNATE COURSES:	[ALT] **Step 5**: No matches found. The system could not retrieve any bookings.		

POST-CONDITION:	• The Name attribute in the **EventStatus** entity has been changed to “Checked-In”.

USE CASE NAME:	Event check-out	USE CASE TYPE	
USE CASE ID:	6.22	Business Requirements:o	
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island resort	System Design:	þ
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	This use case describes the event where the Client request to check-out after using the venue. The administrator responds by facilitating the check-out process. The administrator will search for the client specific event booking on the system. The system will retrieve the details of the event entity. Thereafter the administrator will confirm details from the client. The use case concludes when the administrator checks the client out of the system. The status in the **EventStatus** entity will update to "Checked Out"		
PRE-CONDITION:	• Administrator needs to be logged in. • The *'View event'* screen is already loaded. • The client has already checked-in the system.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The client arrives at the resort and wishes to get checked-out by the administrator.	**2.** The administrator asks the client for the booking reference number.	
	3. The client provides their booking reference number.	**4.** The employee navigates to the search bar where the administrator enters the provided information.	**5.** The search criteria of the Booking submitted by the administrator will be used to retrieve all the event bookings that match the search criterion with the Database Model Array by using the Search Booking function which retrieves all the Bookings from the Database Model which includes the search phrase by comparing it with the booking reference attributes in the **Event** Entity. The system populates the Booking table with the result of the search criteria.

			The **Event** entity has the following attributes: o Event_ID [PK] (int) o EventStatus_ID [FK] (int) o Client_ID [FK] (int) o Venue_ID [FK] (int) o Reminder_ID [FK] (int) o GuestList (varbinary) o ReferenceCode (varchar (6)) o Date (Date) o EventTotal (decimal (6,2)) [ALT]
		6. The employee navigates to the "Check-Out" button and clicks the button	**7.** The object will be sent to the eventbooking service where it will send an http put request to the .Net Core controller which will use an Entity Framework Update method to update the **EventStatus** details. The system updates the Name in the **EventStatus** Entity to "Checked Out" as an updated record. A Success message will be displayed to notify the Employee that the Guest was successfully checked out.
		8. The administrator informs the client that the check-out was successful.	
ALTERNATE COURSES:	[ALT] **Step 5**: No matches found. The system could not retrieve any bookings.		
POST-CONDITION:	The Name attribute in the **EventStatus** entity has been changed to "Checked Out".		

USE CASE NAME:	View Dashboard	USE CASE TYPE	
USE CASE ID:	6.23	Business Requirements:	o
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view the administrator-side dashboard. The use case begins with the administrator requests to view the administrator-side dashboard. On this view, the administrator will be able to update their account details, view event and accommodation reservations due on the current day as well as view room ratings through a visual graph. The use case concludes when the system displays the administrator-side dashboard to the administrator.		
PRE-CONDITION:	• The administrator needs to be logged onto the system.		
TRIGGER:	The administrator requests to view the system's dashboard.		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. The administrator requests to view the system's dashboard.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilises an SQL_Read query to retrieve the records from the following entities:

			RoomBooking entity which has the following attributes: ○ RoomBooking_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomBookingStatus_ID (int) [FK] ○ ReferenceNum varchar(10) ○ BookingDate (DateTime) ○ NumberOfGuests (int) ○ EntryDate (DateTime) ○ ExitDate (DateTime) The system will retrieve the current date by using the Date.Now() function which will be used to compare against each **RoomBooking** entity record where the current date matches the *EntryDate* attribute and will add the record to a binding list. **Event** entity which has the following attributes: ○ Event_ID (int) [PK] ○ Client_ID (int) [FK] ○ EventStatus_ID (int) [FK] ○ Venue_ID (int) [FK] ○ ReferenceCode varchar(6) ○ Date (DateTime) ○ GuestList (varbinary) ○ EventTotal (decimal(6,2)) ○ Description (varchar(30)) The system will retrieve the current date by using the Date.Now() function which will be used to compare against each **Event** entity record where the current date matches the *Date* attribute and

			will add the record to a binding list. **Review** entity which has the following attributes: ○ Review_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomType_ID (int) [FK] ○ Rating (int) ○ Description (varchar(30)) The system will use the *RoomType_ID [FK]* within the **Review** enity and will match the record to the *RoomType_ID [PK]* in the **RoomType** entity where we retrieve the records *TypeName* attribute. The system will then group each rating according to their associated *TypeName.* [ALT]
			3. The system will load the '*View Dashboard*' screen with the following elements: – Label with the text "Dashboard" – Label with the text "Accommodation" – Accommodation Table with the following table headers: ○ "Name" ○ "Surname" ○ "Contact Number" ○ "Reference Code" ○ "Check-out Date" The system will populate each row within the Accommodation Table with the records read from the **RoomBooking** entity that have been filtered and retrieved from the ASP.NET backend as follows: – "Name" with the client's first name {{*Client_ID.ClientName*}}

			– "Surname" with the client's first name {{*Client_ID.ClientSurname}}*
			– "Contact Number" with the {{Client_ID.ClientPhone}}
			– "Reference Number" with the {{*ReferenceNum}}* attribute
			– "Check-out Date" with the {{ExitDate }}
			
			– Label with the text "Events"
			– Accommodation Table with the following table headers:
			  ○ "Name"
			  ○ "Surname"
			  ○ "Contact Number"
			  ○ "Reference Code"
			  ○ "Check-out Date"
			
			The system will populate each row within the Event Table with the records read from the **Event** entity that have been filtered and retrieved from the ASP.NET backend as follows:
			– "Name" with the client's first name {{*Client_ID.ClientName}}*
			– "Surname" with the client's first name {{*Client_ID.ClientSurname}}*
			– "Contact Number" with the {{Client_ID.ClientPhone}}
			– "Reference Number" with the {{*ReferenceCode}}* attribute
			– "Check-out Date" with the {{Date }}
			
			– To the right of the Accommodation table, will be a clickable content card with the text "Welcome back!", "Click here to update your account details"
			– To the right of the Event table, will be a content card populated with the current DateTime which provides a live Date and Time update.

			– Below the booking tables will be a pie chart, which displays the average ratings for each room type, which consists of a single, duo and family room types. Thus, there is 3 different pieces which are coloured differently per each room type. – Key detailing which room type is associated to which colour
	4. The administrator clicks the clickable content card to view their account details.		**5.** . The system extends to "6.24 View Account".
ALTERNATE COURSES:	[ALT] **Step 2**: The system returns null records where no *EntryDate* and *Date* attributes match the current date retrieved.		
CONCLUSION:	The use case concludes when the system displays the dashboard screen, and the administrator can update their account details.		
POST-CONDITION:	The administrator will be able to update their account details as well as view current check-ins for event and room bookings on the *'View Dashboard'* screen.		
BUSINESS RULES:	None		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	View account	USE CASE TYPE
USE CASE ID:	6.24	Abstract: o
PRIORITY:	High	x Extension:
SOURCE:	Platinum Island resort	
PARTICIPATING ACTOR:	Administrator	
DESCRIPTION:	This use case describes the event where a user wants to view their account details on the system. The use case begins when the user clicks on the view account section on the dashboard, the system will then open the modal with the account details, which the user will then be able to update or close.	
PRE-CONDITION:	• The user must have access to internet. • The user must be logged onto the system. • The view account screen is loaded.	

	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
TYPICAL COURSE OF EVENTS:			**1.** The system responds by loading the '*View account*' modal details with the following information: – Heading Label: "Welcome back" – Label : "Click here to update details" – Label: "Email address" – Input field prepopulated with the user's email address. – Label: "Password" – Input field with the prepopulated password – Submit button with the text: "Save".

			2. The system prompts the user to enter the updated account details
	3. The user enters their updated information and clicks the “Save” button. [ALT]		**4.** The system captures and validates the updated user details entered by comparing the entered information with the information in the User table to ensure that the user does not already exist and validates the fields using Angular to ensure that the information is correct and that all fields specified as required have been inputted. [ALT]
			5. The system will capture the information entered by the user and will populate the **User** entity with the attributes: - UserName - Password The User object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
			6. The system will make use of an SQL_update query in the controller to update the specific record within the **User Entity**: ○ User_ID) (value of the previous User_ID, incremented by 1) ○ Role_ID(FK) ○ UserName (varchar) ○ Password (varchar) [ALT]

ALTERNATE COURSES:	[ALT] **Step 3**: The user wishes to close the *'View account'* screen. **Terminate use case**. [ALT] **Step 4**: The system detects that the information fields entered by the user was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the user to re-enter their details. **Return to Step 4**. [ALT] **Step 6:** The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).
POST-CONDITION:	• The view account screen is displayed to the administrator.

USE CASE NAME:	View refunds	USE CASE TYPE	
USE CASE ID:	6.25	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where an administrator wishes to view the refunds that are due to the clients. The use case begins when the admin clicks on the refunds option in the side navbar. The system will then load the view refunds screen. On this view, the administrator will be able to see the list of clients that are due for a refund and select the 'refund' button indicating the client has been refunded. The use case concludes when the system displays all the refunds that are due.		
PRE-CONDITION:	• The client must have access to internet. • The client must be logged onto the system. • The administrator clicks the 'cancel' button on the view accommodations or view events screen.		
TRIGGER:	The administrator wishes to view the refunds and clicks the on the refunds option on the side navbar.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
	4. The administrator will request to view the *'refunds'* screen by clicking on the refunds option on right panel side		**5.** The system will send a request from the angular frontend to the data service class where the service will make a HttpGet request to the .Net core backend which makes use of a linq and lambda query which utilizes the SQL_READ query to retrieve records from the **Client, RoomBooking** and **Event** entities. The system reads the following from the **client** entity: ○ ClientName (varchar (20)) ○ ClientSurname (varchar (20))

	navigation bar.		○ ClientEmail (varchar (30)) ○ ClientPhone (varchar (10)) The system would read the following attributes from the **RoomBooking** entity: ○ ReferenceNum (varchar (6)) ○ Using the RoomBooking_ID [FK] in the **RoomBookingRefund** entity the system retrieves the {{RoomBooking.Amount}} attribute. The system would read the following attributes from the **Event** entity: ○ ReferenceCode (varchar (6)) ○ Using the Event_ID [FK] in the **EventRefund** entity the system retrieves the {{Event.Amount}} attribute.
			6. Using the current date, the system checks that the client cancelled their booking after 48 hours before the booking and is due a 25% return of their total amount. The system does the calculation by using the Amount attribute in the **RoomBookingRefund** multiplies it by 0.25 and stores it in an array called *RoomRefund.* Using the current date, the system checks that the client cancelled their booking after 48 hours before the booking and is due a 25% return of their total amount. The system does the calculation by using the Amount attribute in the **EventRefund** multiplies it by 0.25 and stores it in an array called *EventRefund.*

			7. The system responds by loading the '*View Refunds'* screen details with the following information: – Main Heading: "Refunds" – Subheading: "Accommodations" – Table with the headings: o Name o Surname o Email address o Contact number. o Reference Code o Amount due. o Refund status – The system will prepopulate the first table with the following details from the **client** and **RoomBookingRefund** entities: – Name as the name of the ClientName – Surname as the ClientSurname – Email address as the ClientEmail – Contact Number as the ClientPhone – Reference Code as the ReferenceNum – The *RoomRefund* array that was used to calculate the amount due to the client. – A button with the text: "Refund" The system then loads the second table that displays the events to be refunded for with the following attributes: – Subheading: "Events" – Table with the headings: o Name o Surname o Email address o Contact number. o Reference Code o Amount due. o Refund status – The system will prepopulate the first table with the

			following details from the **client** and **Event** entities: – Name as the name of the ClientName – Surname as the ClientSurname – Email address as the ClientEmail – Contact Number as the ClientPhone – Reference Code as the bookings ReferenceCode – The *Eventrefund array* that was used to calculate the amount due to the client. – A button with the text: “Refund”
	8. The employee wishes to refund a room booking. [ALT]		**9.** The system Invokes Use case “6.26 Refund room booking”.
ALTERNATE COURSES:	[ALT] **Step 5**: The employee wishes to refund an event booking. Invoke “use case 6.27 Refund event booking”		
CONCLUSION:	The case ends when the administrator can view the refunds due on the system.		
POST-CONDITION:	• The view refunds screen is displayed to the administrator.		
BUSINESS RULES:	• Cancellation after 48 hours will result in the client being returned 25% of the fee. • Cancellation within 48 hours before the booking will result in no fee being returned.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Refund Room Booking	USE CASE TYPE	
USE CASE ID:	6.26	Business Requirements:	o
PRIORITY:	High	System Analysis:	þ
SOURCE:	Platinum Island Resort	System Design:	o
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	The use case begins when the client initiates the refund process of their room booking reservation. The system should allow the administrator to refund a client's room booking. Whereby the client will be asked to confirm their booking before their room booking record is retrieved from the **RoomBooking** entity. Once the administrator has found the record, the administrator will request confirmation on the client's room booking refund and will then complete the client's refund payment off of the system. The system will then change the status of their room booking to "Refunded". The use case concludes when the administrator when the administrator completes the refund booking EFT, the transaction is logged on the system and the system updates room booking status of the client's booking.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system has loaded the *'View Refunds'* screen. • The client's accommodation booking must have been cancelled first. • The client's room booking status must have a status of "Refund Pending"		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	MANUAL ACTION	SYSTEM RESPONSE:
		1. The administrator requests the client to provide their room booking reference number.	
	2. The client provides their room booking referenc	**3.** The administrator searches and selects the relevant room booking reservation.	

	e number.	The administrator requests confirmation on the refund process by communicating the amount due to the client. [ALT]	
	4. The client will confirm the refund process as well as the amount due. [ALT]	**5.** The administrator requests the client to provide their banking account details.	
	6. The client will provide their bank account details.	**7.** The administrator completes an online EFT payment by making use of the client's bank account details provided and the *Amount* attribute value from the **RoomPayment** entity	
		8. Once the transaction has been approved and completed. The administrator will click the "Refund" button.	**9.** Using the *RoomBooking_ID* sent from the Angular front-end to the .NET Core backend, the system will match the *RoomBooking_ID* selected to a specific room booking record in the **RoomBooking** entity using an SQL_Read query. The system then retrieves the client's accommodation from the

			RoomBooking entity with the following attributes: ○ RoomBooking_ID (int) [PK] ○ Client_ID (int) [FK] ○ RoomBookingStatus _ID (int) [FK] ○ ReferenceNum (varchar(6)) ○ BookingDate (DateTime) ○ NumberOfGuests (int) ○ EntryDate (DateTime) ○ ExitDate (DateTime) The system will also retrieve the relevant RoomPayment record by using the *RoomBooking_ID (int) [FK]* within the **RoomPayment** entity to retrieve the *Amount (decimal(6,2))* attribute. The system retrieves the "Refunded" status by reading the *Name (varchar(20))* attribute within the **RoomBookingStatus** entity, using the *RoomBookingStatus_ID (int) [FK]* within the **RoomBooking** entity, and updates the status of the client's room bookings. The system will make use of an SQL_Insert query in the controller to create the new record within the **RoomBookingRefund** entity with the following attributes: ○ Refund_ID (int) [PK] (value of the previous Refund_ID, incremented by 1)

			○ RoomBooking_ID (int) [FK] ○ Amount (decimal(6,2)) [ALT]
ALTERNATE: COURSES:	[ALT] **Step 3a**: The reference number given by the client does not match any of the reference numbers displayed on the system **Return to Step 2**. [ALT] **Step 3b**: The client's room booking was not valid for a room booking refund as it was cancelled less than 48 hours before the *EntryDate* attribute within the **RoomBooking** entity. **Terminate Use Case**. [ALT] **Step 4**: The client refuses to continue with their refund room booking process. **Terminate Use Case**. [ALT] **Step 9a**: The transaction was not approved and was unsuccessful. **Return to Step 8**. [ALT] **Step 9b**: The system was unable to create a new record within the **RefundRoomBooking** entity and will return a Bad Request 404 error to the system. **Terminate Use Case**.		
POST-CONDITION:	• The room booking status of the of the client's room booking is changed to "Refunded" and a new record is added to the **RoomBookingRefund** entity.		

USE CASE NAME:	Refund Event Booking	USE CASE TYPE	
USE CASE ID:	6.27	Business Requirements:	o
PRIORITY:	High	System Analysis:	þ
SOURCE:	Platinum Island Resort	System Design:	o
PARTICIPATING ACTORS:	Client (PBA) Administrator (PSA)		
DESCRIPTION:	The use case begins when the client initiates the refund process of their event booking reservation. The system should allow the administrator to refund a client's event booking. Whereby the client will be asked to confirm their booking before their event booking record is retrieved from the **Event** entity. Once the administrator has found the record, the administrator will request confirmation on the client's event booking refund and will then complete the client's refund payment off of the system. The system will then change the status of their event booking to "Refunded". The use case concludes when the administrator when the administrator completes the refund booking EFT, the transaction is logged on the system and the system updates event booking status of the client's booking.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system has loaded the *'View Refunds'* screen. • The client's event booking must have been cancelled first. • The client's event booking status must have a status of "Refund Pending"		

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	MANUAL ACTION	SYSTEM RESPONSE:
		1. The administrator requests the client to provide their event booking reference code.	
	2. The client provides their event booking reference code.	**3.** The administrator searches and selects the relevant event booking reservation.	

		The administrator requests confirmation on the refund process by communicating the amount due to the client. [ALT]	
	4. The client will confirm the refund process as well as the amount due. [ALT]	**5.** The administrator requests the client to provide their banking account details.	
	6. The client will provide their bank account details.	**7.** The administrator completes an online EFT payment by making use of the client's bank account details provided and the *Amount* attribute value from the **EventPayment** entity	
		8. Once the transaction has been approved and completed. The administrator will click the "Refund" button.	**9.** Using the *Event_ID* sent from the Angular front-end to the .NET Core backend, the system will match the *Event_ID* selected to a specific event booking record in the **Event** entity using an SQL_Read query. The system then retrieves the client's event record from the **Event** entity with the following attributes: ○ Event_ID (int) [PK]

			○ Client_ID (int) [FK] ○ Venue_ID (int) [FK] ○ EventStatus_ID (int) [FK] ○ ReferenceCode (varchar(6)) ○ Date (DateTime) ○ GuestList (varbinary) ○ EventTotal (decimal(6,2)) ○ Description (varchar(30)) The system will also retrieve the relevant EventPayment record by using the *Event_ID (int) [FK]* within the **EventPayment** entity to retrieve the *Amount (decimal(6,2))* attribute. The system retrieves the “Refunded” status by reading the *Name (varchar(20))* attribute within the **EventStatus** entity, using the *EventStatus_ID (int) [FK]* within the **Event** entity, and updates the status of the client’s event bookings. The system will make use of an SQL_Insert query in the controller to create the new record within the **EventRefund** entity with the following attributes:

			○ Refund_ID (int) [PK] (value of the previous Refund_ID, incremented by 1) ○ Event_ID (int) [FK] ○ Amount (decimal(6,2)) [ALT]
ALTERNATE: COURSES:	[ALT] **Step 3a**: The reference code given by the client does not match any of the reference codes displayed on the system. **Return to Step 2**. [ALT] **Step 3b**: The client's event booking was not valid for an event booking refund as it was cancelled less than 48 hours before the *Date* attribute within the **Event** entity. **Terminate Use Case**. [ALT] **Step 4**: The client refuses to continue with their refund event booking process. **Terminate Use Case**. [ALT] **Step 9a**: The transaction was not approved and was unsuccessful. **Return to Step** 8. [ALT] **Step 9b**: The system was unable to create a new record within the **EventRefund** entity and will return a Bad Request 404 error to the system. **Terminate Use Case**.		
POST-CONDITION:	The event booking status of the of the client's event booking is changed to "Refunded" and a new record is added to the **EventRefund** entity.		

2.7. Subsystem 7 – Inventory

USE CASE NAME:	View Item	USE CASE TYPE
USE CASE ID:	7.1	Business Requirements: o
PRIORITY:	Low	System Analysis: o
SOURCE:	Platinum Island Resort	System Design: þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes the event where the administrator would like to view item records available of the system. The use case begins with the administrator requests to access the 'Items' View by navigating through the side navigation bar. On this view, the administrator will be able to create, update and delete items on the system. The use case concludes when the system displays all of the item records.	
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system displays the dashboard screen.	
TRIGGER:	The administrator requests to view the Item's screen on the system.	

	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
TYPICAL COURSE OF EVENTS:	**1.** Administrator will request to view the *Items Screen* by hovering over the 'Inventory Management' tab option where the side navigation bar will display three routing options: ○ 'Items' ○ 'Item Type' ○ 'Item Category' The administrator will then click on "Items" side-navbar option.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Item** Entity which has the following attributes: ○ Item_ID (int) [PK] ○ ItemType_ID (int) [FK] ○ Name (string) ○ Description (string) ○ QtyOnHand (int)
			3. The system will load the '*Item*' screen with the following elements: – Label with the text "Items" – **"+"** button – Label with the text "Add Items" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter item name".

			– Font Awesome Icon '*fa fa-search*' within the search bar, on the right. – Item Table with the following headers: ○ "Name" ○ "Description" ○ "Item Type" ○ "Quantity on Hand" ○ Empty Header The system will populate each row within the Item Table with the records read from the **Item Entity** as follows: – "Name" with the *{{Name}}* – "Description" with the *{{Description}}* – "Item Type" with the *{{ItemType_ID.Name}}* – *"*Quantity on Hand" with the *{{QtyOnHand}}* *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*

	4. The administrator requests to add an item record. [ALT]		**5.** The system extends to "7.2 Add Item".

ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an item. The system will prompt the administrator to enter the item details within the search bar. The administrator will enter either the 'Name', 'Description' or the 'Item Type' of the item. The system retrieves and displays a list of all the item records that match the search criteria entered by the administrator using the following attributes from the **Item Entity**: – Name – Description – Item Type [ALT] **Step 4b**: The administrator requests to update an item. The system extends to Use Case 7.4 "Update Item". [ALT] **Step 4c**: The administrator requests to delete an item. The system extends to Use Case 7.5 "Delete Item".
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Item Table.
POST-CONDITION:	The administrator will be able to add an item on the *'Item'* screen.
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Item Category records on the system.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Add Item	USE CASE TYPE
USE CASE ID:	7.2	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new Item record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it within the **Item Entity**. The use case concludes when the Item record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Items'* screen. • The administrator clicked the "**+**" button (Add Item).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Item*" modal with the following elements: – Modal Heading Text: "Add Item"; Close button. – Label: "Name" – Item name text input field with the placeholder text "Enter an item name". – Label: "Item Type"

		– A combo box dropdown which is populated using the 'Name' attribute from the **Item Type Entity**. With the placeholder text "Select an option…" – Label: "Quantity" – Numeric UpDown with a default value set to '0'. The system will restrict the numeric updown from counting below the value of 0. – Label: "Description" – Item Description with the placeholder text "Provide a description describing the item, maximum of 30 characters". – Submit button with the text "Submit". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new item details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the "Submit" button.

		[ALT]
	5. The administrator clicks the "Submit" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item object with the attributes: – Name – Item Type – Quantity – Description The Item object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **Item Entity**: ○ Item_ID (int) [PK] (value of the previous Item_ID, incremented by 1) ○ ItemType_ID (int) [FK] ○ Name (string) ○ QtyOnHand (int) ○ Description (string) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. The use case terminates.	

	[ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).
POST-CONDITION:	The new item record has been added to the system in the Item table.

USE CASE NAME:	Update Item	USE CASE TYPE
USE CASE ID:	7.3	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates an Item record on the system. The use case begins when the administrator chooses to update an Item record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Item record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item'* screen. • The administrator clicked the update button.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit Item*" modal. Using the *Item_ID* sent from the Angular front-end to the .NET Core backend, the system will match the *Item_ID* selected to a specific record in the **Item** entity using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the item. The "*Edit Item*" modal has the following elements: – Modal Heading Text: "Edit Item"; Close button. – Label: "Name" – Item name text input field with the pre-populated name text – {{Name}} – Label: "Item Type"

		– A combo box dropdown which is pre-populated using the 'Name' attribute from the **Item Type Entity** – {{ItemType.Name}} – Label: "Quantity" – Numeric UpDown with a pre-populated QtyOnHand – {{QtyOnHand}}. – Label: "Description" – Item description text input field with the pre-populated description text – {{Description}} – Submit button with the text "Save". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated item details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the "Save" button. [ALT]
	5. The administrator clicks the "Save" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item object with the attributes: – Name – Item Type – Quantity – Description The Item object will be sent to the Data Service where it will

		send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the Item record within the **Item Entity**: ○ Item_ID (int) [PK] (value of the previous Item_ID, incremented by 1) ○ ItemType_ID (int) [FK] ○ Name (string) ○ QtyOnHand (int) ○ Description (string)
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant item category information has been successfully updated in the **Item Entity**.	

USE CASE NAME:	Delete Item	USE CASE TYPE
USE CASE ID:	7.4	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific item record. The use case begins when the system will request confirmation on the item deletion before the administrator confirms the deletion of the record. The use case concludes when the item record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item'* screen. • The administrator clicked the delete button. • The item must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will use the *Item_ID* sent from the Angular front-end to the .NET Core backend, the system will match the *Item_ID* selected to a specific record in the **Item** entity using an SQL_Read query. The system will then respond by displaying a pop-up "*Delete Item*" modal with the following elements: – Modal Heading Text: "Item Deletion Confirmation"; Close button. – Label: "Are you sure you want to delete this item?" – Submit button with the text "Confirm". – Cancel button with the text "Cancel".
		2. The system prompts the administrator if they would like to delete the selected record.

	3. The administrator clicks the "Confirm" button. [ALT]	**4.** The system deletes the item record using an HttpDelete request from the **Item** entity with the following attributes: ○ Item_ID (int) [PK] ○ ItemType_ID (int) [FK] ○ Name (string) ○ QtyOnHand (int) ○ Description (string) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 3**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant item information has been successfully deleted from the **Item** entity.	

USE CASE NAME:	View Item Type	USE CASE TYPE	
USE CASE ID:	7.5	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island Resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view item type records available of the system. The use case begins with the administrator requests to access the 'Item Type' View by navigating through the side navigation bar. On this view, the administrator will be able to create, update and delete item types on the system. The use case concludes when the system displays all of the item type records.		
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system displays the *'View item type'* screen.		
TRIGGER:	The administrator requests to view the Item Type screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
	1. Administrator will request to view the '*Item Type' Screen* by hovering over the 'Inventory Management' tab option where the side navigation bar will display three routing options: ○ 'Items' ○ 'Item Type' ○ 'Item Category'		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Item Type Entity** which has the following attributes: ○ ItemType_ID (int) [PK]

	The administrator will then click on “Item Type” side-navbar option.		○ ItemCategory_ID (int) [FK] ○ Name (string) ○ Description (string)
			3. The system will load the ‘*Item Type*’ screen with the following elements: – Label with the text “Item Types” – “**+**” button – Label with the text “Add Item Type” to the right of the “**+**” button. – Search bar with the placeholder text of “Enter type name”. – Font Awesome Icon ‘*fa fa-search*’ within the search bar, on the right. – Item Table with the following headers: ○ “Name” ○ “Description” ○ “Item Category” ○ Empty Header The system will populate each row within the Item Table with the records read from the **Item Type Entity** as follows: – “Name” with the *Name* – “Description” with the *Description* – “Item Category” with the *ItemCategory_ID.Name* – Empty header with*:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*

	4. The administrator requests to add an item type record. [ALT]		5. The system extends to "7.6 Add Item Type".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an item type. The system will prompt the administrator to enter the item type details within the search bar. The administrator will enter either the 'Name', 'Description' or the 'Item type of the item type. The system retrieves and displays a list of all the item type records that match the search criteria entered by the administrator using the following attributes from the **Item Type Entity**: – Name – Description – Item Category [ALT] **Step 4b**: The administrator requests to update an item type. The system extends to Use Case 7.7 "Update Item Type". [ALT] **Step 4c**: The administrator requests to delete an item type. The system extends to Use Case 7.8 "Delete Item Type".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Item Type Table.		
POST-CONDITION:	The administrator will be able to add an item type on the *'Item Type'* screen.		
BUSINESS RULES:	• Only administrators will be permitted to view Item Type records on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Add Item Type	USE CASE TYPE
USE CASE ID:	7.6	Abstract: ☐
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new Item Type record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it within the **Item Type Entity**. The use case concludes when the Item record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Type'* screen. • The administrator clicked the "**+**" button (Add Item Type).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Item Type*" modal with the following elements: – Modal Heading Text: "Add Item Type"; Close button. – Label: "Name" – Item name text input field with the placeholder text "Enter item Type". – Label: "Item type" – A combo box dropdown which is populated using the 'Name' attribute from the **Item Category Entity**. With the placeholder text "Select an option…" – Label: "Description" – Item Description with the placeholder text "Provide enter a description". – Submit button with the text "Submit".

		– Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new item type details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Submit” button. [ALT]
	5. The administrator clicks the “Submit” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item Type object with the attributes: – Name – Description – Item Category The Item Type object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **Item Type Entity**: ○ ItemType_ID (int) [PK] (value of the previous Item_ID, incremented by 1) ○ ItemCategory_ID (int) [FK] ○ Name (string) ○ Description (string) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item type details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	

	[ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).
POST-CONDITION:	The new item record has been added to the system in the Item Type table.

USE CASE NAME:	Update Item Type	USE CASE TYPE
USE CASE ID:	7.7	Abstract: ☐
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates an Item Type record on the system. The use case begins when the administrator chooses to update an Item Type record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Item Type record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Type'* screen. • The administrator clicked the update button.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit Item Type*" modal. Using the ItemType_ID sent from the Angular front-end to the .NET Core backend, the system will match the ItemType_ID selected to a specific record in the **Item Type Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the item type. The "*Edit Item Type*" modal has the following elements: – Modal Heading Text: "Edit Item Type"; Close button. – Label: "Name" – Item Type name text input field with the pre-populated name text – {{Name}}

		– Label: “Item Category” – A combo box dropdown which is pre-populated using the ‘Name’ attribute from the Item Category Entity. With the placeholder text – {{ItemCategory_ID. Name}} – Label: “Description” – Item Type description text input field with the pre-populated description text – {{Description}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated item type details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item Type object with the attributes: – Name – Item Category – Description The Item Type object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the Item

		Type record within the **Item Type Entity**: ○ ItemType_ID (int) [PK] ○ ItemCategory_ID (int) [FK] ○ Name (string) ○ Description (string)
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item type details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant item type information has been successfully updated in the **Item Type Entity**.	

USE CASE NAME:	Delete Item Type	USE CASE TYPE
USE CASE ID:	7.8	Abstract: ☐
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific item type record. The use case begins with the system verifying there are no items associated with the item type. The system will confirm the item type deletion before the administrator confirms the deletion of the record. The use case concludes when the item type record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Type* screen. • The administrator clicked the delete button. • The item type must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system matches the ItemType_ID [PK] for the item type selected by the administrator to the item type from the ItemType Table, with the ItemType_ID [FK] retrieved from the Items Table, to ensure no inventory item are associated with the selected item type. The system will do this by performing an SQL_Read query in the .NET Core controller. [ALT]
		2. The system responds by displaying a pop-up "*Delete Item Type*" modal with the following elements: – Modal Heading Text: "Item Type Deletion Confirmation"; Close button. – Label: "Are you sure you want to delete this item type?" – Submit button with the text "Confirm".

		– Cancel button with the text “Cancel”.
		3. The system prompts the administrator if they would like to delete the selected record.
	4. The administrator clicks the “Confirm” button. [ALT]	**5.** The system deletes the item type record from the **ItemType Entity** with the following attributes: ○ ItemType_ID (int) [PK] ○ ItemCategory_ID (int) [FK] ○ Name (string) ○ Description (string) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 1**: The system determines that there are Items that are associated with the selected Item Type. The system will throw an error with the following information “Cannot delete this record as there are associated records” **Terminate use case**. [ALT] **Step 4**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant item type information has been successfully deleted from the **ItemType Entity**.	

USE CASE NAME:	View Item Category	USE CASE TYPE
USE CASE ID:	7.9	Business Requirements:o
PRIORITY:	High	System Analysis:o
SOURCE:	Platinum Island Resort	System Design:þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes the event where the administrator would like to view item category record available of the system. The use case begins with the administrator requests to access the 'Item Category' View by navigating from the 'Item' View. On this view, the administrator will be able to create, update and delete item categories on the system. The use case concludes when the system displays all of the item category records.	
PRE-CONDITION:	• The administrator needs to be logged onto the system. • The system displays the dashboard screen.	
TRIGGER:	The administrator requests to view Item Category screen on the system.	

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Administrator will request to view the *Item Category Screen* by hovering over the 'Inventory Management' tab option where the side navigation bar will display three routing options: ○ 'Items'		2. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to

	○ 'Item Type' ○ 'Item Category' The administrator will then click on "Item Category" side-navbar option.		retrieve the records from the **Item Category Entity** which has the following attributes: ○ ItemCategory_ID (int) [PK] ○ Name (string) ○ Description (string)
			3. The system will load the '*Item Category*' screen with the following elements: – Label with the text "Item Category" – "**+**" button – Label with the text "Add Item Category" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter category name". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right. Item Category Table with the following headers: ○ "Name" ○ "Description" ○ Empty Header The system will populate each row within the Item Category Table with the records read from the **Item Category Entity** as follows: – "Name" with the *Name* – "Description" with the *Description* *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*).

			○ Delete button with a trash-can icon (*fa fa-trash-o).*
	4. The administrator requests to add an item category record. [ALT]		**5.** . The system extends to "7.10 Add Item Category".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search an item category. The system will prompt the administrator to enter the item category details within the search bar. The administrator will enter either the 'Name' or 'Description' of the item category. The system retrieves and displays a list of all the item category records that match the search criteria entered by the administrator using the following attributes from the **Item Category Entity**: - Name - Description - [ALT] **Step 4b**: The administrator requests to update an item category. The system extends to Use Case 7.11 "Update Item Category". [ALT] **Step 4c**: The administrator requests to delete an item category. The system extends to Use Case 7.12 "Delete Item Category".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Item Category Table.		
POST-CONDITION:	The administrator will be able to add an item category on the *'Item Category'* screen.		
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Item Category records on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Add Item Category	USE CASE TYPE
USE CASE ID:	7.10	Abstract:¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new Item Category record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Item Category record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Category'* screen. • The administrator clicked the "**+**" button (Add Item Category).	

TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Item Category*" modal with the following elements: – Modal Heading Text: "Add Item Category"; Close button. – Label: "Name" – Item Category name text input field with the placeholder text "Please enter Category Name". – Label: "Description" – Item Category Description with the placeholder text "Provide a description for the item category". – Submit button with the text "Submit". – Cancel button with the text "Cancel". [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input

		fields have been entered and validated.]
		2. The system prompts the administrator to enter the new item category details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the "Submit" button. [ALT]
	5. The administrator clicks the "Submit" button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item Category object with the attributes: ○ Name ○ Description The Item Category object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **Item Category Entity**: ○ ItemCategory_ID (int) [PK] (value of the previous ItemCategory_ID, incremented by 1) ○ Name (string) ○ Description (string) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item category details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	

	[ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).
POST-CONDITION:	The new item category record has been added to the system in the ItemCategory table.

USE CASE NAME:	Update Item Category	USE CASE TYPE
USE CASE ID:	7.11	Abstract: ¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates an Item Category record on the system. The use case begins when the administrator chooses to update an Item Category record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Item Category record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Category'* screen. • The administrator clicked the update button.	

TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit Item Category*" modal. Using the ItemCategory_ID sent from the Angular front-end to the .NET Core backend, the system will match the ItemCategory_ID selected to a specific record in the **Item Category Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the item category. The "*Edit Item Category*" modal has the following elements: – Modal Heading Text: "Add Item Category"; Close button. – Label: "Name" – Item Category name text input field with the pre-

	populated name text – {{Name}} – Label: “Description” – Item Category description text input field with the pre-populated description text – {{Description}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
	2. The system prompts the administrator to enter the updated item category details.
3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate an Item Category object with the attributes: ○ Name ○ Description The Item Category object will be sent to the Data Service where it will send an HttpPut request to the backend .NET Core.
	7. The system will make use of an SQL_Update query in the controller to update the Item Category record within the **Item Category Entity**: ○ ItemCategory_ID (int) [PK] ○ Name (string) ○ Description (string)

ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the item category details. Return to Step 3. [ALT] **Step 5**: The administrator selects the Cancel button. The use case terminates.
POST-CONDITION:	The relevant item category information has been successfully updated in the **Item Category Entity**.

USE CASE NAME:	Delete Item Category	USE CASE TYPE
USE CASE ID:	7.12	Abstract:¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific item category record. The use case begins with the system verifying there are no sale items associated with the item category. The system will confirm the item category deletion before the administrator confirms the deletion of the record. The use case concludes when the item category record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Item Category'* screen. • The administrator clicked the delete button. • The item category must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system matches the ItemCategory_ID [PK] for the item category selected by the administrator to the item category from the ItemCategory Table, with the ItemCategory_ID [FK] retrieved from the ItemType Table, to ensure no inventory item types are associated with the selected item category. The system will do this by performing an SQL_Read query in the .NET Core controller. [ALT]
		2. The system responds by displaying a pop-up "*Delete Item Category*" modal with the following elements: – Modal Heading Text: "Item Category Deletion

		Confirmation"; Close button. – Label: "Are you sure you want to delete this item category?" – Submit button with the text "Confirm". – Cancel button with the text "Cancel".
		3. The system prompts the administrator if they would like to delete the selected record.
	4. The administrator clicks the "Confirm" button. [ALT]	**5.** The system deletes the item category record from the **ItemCategory Entity** with the following attributes: ○ ItemCategory_ID (int) [PK] ○ Name (string) ○ Description (string) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 1**: The system determines that there are Item Types that are associated with the selected Item Category. The system will throw an error. **Terminate use case**. [ALT] **Step 4**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant item category information has been successfully deleted from the **Item Category Entity**.	

USE CASE NAME:	View Supplier	USE CASE TYPE	
USE CASE ID:	7.13	Business Requirements:	o
PRIORITY:	Low	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes the event where the administrator would like to view supplier records available of the system. The use case begins with the administrator requests to access the 'Supplier' View by navigating from the navigation bar. On this view, the administrator will be able to create, update and delete suppliers on the system. The use case concludes when the system displays all the supplier records.		
PRE-CONDITION:	• The administrator needs to be logged onto the system.		
TRIGGER:	The administrator requests to view Supplier screen on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
	1. Administrator will request to view the *Supplier Screen* by clicking the "Supplier" tab option on the navigation bar.		**2.** The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve the records from the **Supplier**

			Entity which has the following attributes: – Supplier_ID (int) [PK] – FullName (varchar 20) – Email (varchar 50) – ContactNum (varchar 10) – Description (varchar 30)
			3. The system will load the ***'Supplier'*** screen with the following elements: – Label with the text "Suppliers" – "**+**" button – Label with the text "Add Supplier" to the right of the "**+**" button. – Search bar with the placeholder text of "Enter Supplier name". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right. Supplier Table with the following headers: ○ "Full Name" ○ "Email Address" ○ "Contact No." ○ "Description" ○ Empty Header The system will populate each row within the Supplier Table with the records read from the **Supplier Entity** as follows: – "Full Name" with the *Name.*

			– "Email Address" with the *SupplierEmail.* – "Contact No."" with the *SupplierPhone* – "Description" with the Description. *Empty header with:* ○ Edit button with a pencil icon (*fa fa-pencil*). ○ Delete button with a trash-can icon (*fa fa-trash-o).*
	4. The administrator requests to add a Supplier record. [ALT]		**5.** . The system extends to "7.14 Add Supplier".
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search a Supplier. The system will prompt the administrator to enter the Supplier details within the search bar. The administrator will enter either the 'Name' OR 'Email Address' OR 'Contact No.' of the Supplier. The system retrieves and displays a list of all the Supplier records that match the search criteria entered by the administrator using the following attributes from the **Supplier Entity**: – Name – Supplier Email – Supplier Phone – Description [ALT] **Step 4b**: The administrator requests to update a Supplier. The system extends to "Use Case 7.15 Update Supplier". [ALT] **Step 4c**: The administrator requests to delete a Supplier. The system extends to "Use Case 7.16 Delete Supplier".		
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the Supplier Table.		
POST-CONDITION:	The administrator will be able to add a Supplier on the *'Supplier'* screen.		
BUSINESS RULES:	• Only authorised users of the system will be permitted to view Supplier records on the system.		

IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Add supplier	USE CASE TYPE
USE CASE ID:	7.14.	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator creates a new Supplier record on the system. The use case begins when the system prompts the administrator to enter the required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Supplier record has successfully been created on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Supplier'* screen. • The administrator clicked the "**+**" button (Add Supplier).	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Add Supplier*" modal with the following elements: – Modal Heading Text: "Add Supplier"; Close button. – Label: "Full Name" – Supplier name text input field with the placeholder text "Enter suppliers full name". – Label: "Email address" – Supplier email text input with the placeholder text "Enter suppliers email address". – Label: "Cell number" – Supplier cell number text input with the placeholder text "Enter suppliers cell number". – Label: "Description" – Supplier description text input with the placeholder text "Enter a description describing

		the supplier, maximum of 30 characters”. – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the new Supplier details.
	3. The administrator will enter the required information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Submit” button. [ALT]
	5. The administrator clicks the “Submit” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a Supplier object with the attributes: – FullName – Email – ContactNum – Description The Supplier object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
		7. The system will make use of an SQL_Insert query in the controller to create the new record within the **Supplier Entity**: ○ Supplier_ID (int) [PK] (value of the previous ItemCategory_ID, incremented by 1) ○ Name (Varchar 20) ○ Description (Varchar 30) ○ SupplierEmail (Varchar 50)

		○ SupplierPhone (Varchar 10) [ALT]
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Supplier details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**. [ALT] **Step 7**: The system will return a bad request when the system fails to retrieve the records from the backend to the angular frontend (400 bad request).	
POST-CONDITION:	The new Supplier record has been added to the system in the Supplier table.	

USE CASE NAME:	Update Supplier	USE CASE TYPE
USE CASE ID:	7.15	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator updates a Supplier record on the system. The use case begins when the administrator chooses to update a Supplier record on the system. The system will retrieve the information of the selected record, where the system will prompt the user to enter the updated required information. The administrator enters the information required. The system verifies the information and then stores it. The use case concludes when the Supplier record has been successfully updated on the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Supplier'* screen. • The administrator clicked the update button.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system responds by displaying a pop-up "*Edit Supplier*" modal. Using the Supplier_ID sent from the Angular front-end to the .NET Core backend, the system will match the ItemCategory_ID selected to a specific record in the **Supplier Entity** using an SQL_Read query. All input fields will be pre-populated with the current, saved attributes of the Supplier. The "*Edit Supplier*" modal has the following elements: – Modal Heading Text: "Add Supplier"; Close button. – Label: "Name" – Supplier name text input field with the pre-populated name text – {{Name}}

		– Label: “Email address” – Supplier email text input field with the prepopulated email address text {{Email}} – Label: “Cell number” – Supplier email text input field with the prepopulated cell number text {{cell number}} – Label: “Description” – Supplier description text input field with the pre-populated description text {Description}} – Submit button with the text “Save”. – Cancel button with the text “Cancel”. [The cancel and close buttons are enabled on default; however, the save button is disabled until all text input fields have been entered and validated.]
		2. The system prompts the administrator to enter the updated Supplier details.
	3. The administrator will enter the required updated information within the text input fields.	**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Save” button. [ALT]
	5. The administrator clicks the “Save” button. [ALT]	**6.** The system will capture the information entered by the administrator and will populate a Supplier object with the attributes: – Name – SupplierEmail – SupplierPhone – Description The Supplier object will be sent to the Data Service where it will

		send an HttpPut request to the backend .NET Core.
		7. The system will make use of an SQL_Update query in the controller to update the Supplier record within the **Supplier Entity**: ○ Supplier_ID (int) [PK] ○ Name (varchar 20) ○ SupplierEmail (varchar 50) ○ SupplierPhone (varchar 10) ○ Description (varchar 30)
ALTERNATE COURSES:	[ALT] **Step 4**: The system detects that the information fields entered by the administrator was either left blank or was entered in incorrect format. The system will display error messages directing to which input field has the error, prompting the administrator to re-enter the Supplier details. **Return to Step 3**. [ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant Supplier information has been successfully updated in the **Supplier Entity**.	

USE CASE NAME:	Delete Supplier	USE CASE TYPE
USE CASE ID:	7.16	Abstract: ¨
PRIORITY:	Low	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to delete a specific Supplier record. The use case begins with the system verifying there are no supplier orders associated with the Supplier. The system will confirm the Supplier deletion before the administrator confirms the deletion of the record. The use case concludes when the Supplier record has been deleted from the system.	
PRE-CONDITION:	• The administrator must be logged into the system. • The system has loaded the *'Supplier'* screen. • The administrator clicked the delete button. • The Supplier must exist on the system before it can be deleted.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system matches the **Supplier_ID** [PK] for the Supplier selected by the administrator to the Supplier from the Supplier Table, with the Supplier_ID [FK] retrieved from the Supplier_Order Table, to ensure no orders that are associated with the selected Supplier. The system will do this by performing an SQL_Read query in the .NET Core controller. [ALT]
		2. The system responds by displaying a pop-up "*Delete Supplier*" modal with the following elements: – Modal Heading Text: "Confirm supplier deletion"; Close button. – Label: "Are you sure you want to delete this Supplier?"

		– Submit button with the text “Confirm”. – Cancel button with the text “Cancel”.
		3. The system prompts the administrator if they would like to delete the selected record.
	4. The administrator clicks the “Confirm” button. [ALT]	**5.** The system deletes the Supplier record from the **Supplier Entity** with the following attributes: ○ Supplier_ID (int) [PK] ○ Name (string) ○ SupplierEmail (string) ○ SupplierPhone (string) ○ Description (string) The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework.
ALTERNATE COURSES:	[ALT] **Step 1**: The system determines that there are Supplier Orders that are associated with the selected Supplier. The system will throw an error. **Terminate use case**. [ALT] **Step 4**: The administrator selects the Cancel button. **The use case terminates**.	
POST-CONDITION:	The relevant Supplier information has been successfully deleted from the **Supplier Entity**.	

USE CASE NAME:	View Supplier order	USE CASE TYPE	
USE CASE ID:	7.17	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes an event where the administrator would like to view the Supplier order records that are available on the system. The use case begins when the administrator requests to access the 'Supplier orders' view. On this view, the administrator will be able to place an order for new items, mark the items as received in the checkbox and cancel the order by clicking on the cancel button. The use case concludes when the system displays all the supplier order records.		
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system.		
TRIGGER:	The administrator wishes to view the supplier order records on the system.		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE: MANUAL ACTION:	SYSTEM RESPONSE: AUTOMATED ACTION:
	1. The administrator wishes to view the supplier order screen by hovering over the supplier tab option where the side navigation bar will display two routing options. ○ 'View suppliers' ○ 'View supplier order' The administrator will then click on		**2.** The system will send a request from the angular frontend to the data service class where the service will make a HttpGet request to the .Net core backend which makes use of a linq and lambda query which utilizes the SQL_READ query to retrieve records from the **Supplier_Order** entity which has the following attributes:

	the 'View Supplier Order' option.		○ Order_ID [PK] (int) ○ Supplier_ID [FK] (int) ○ User_ID [FK] (int) ○ OrderStatus_ID [FK] (int) ○ Item_ID [FK] (int) ○ Date (date) ○ Quantity (int)
			3. The system will load the '*Supplier Order'* screen with the following elements: – Heading with the text: "Supplier Order" – A "+" button. – Label with the text "Place Supplier Order" to the right of the "+" button – A search bar with the search bar text "Search". – Font Awesome Icon '*fa fa-search*' within the search bar, on the right. – Supplier Order table with the following headers: ○ "Supplier Name" ○ "Item Name" ○ "Quantity" ○ "Received". ○ "Cancel". The system will populate each row within the Supplier table with the records read from the **Supplier_Order** entity as follows: – "Quantity" with the Quantity needed.

			Using the Order_ID attribute in the **Order_Line** table that corresponds to the Order_ID attribute in the **Supplier_Order** table, the system reads the {{Order_ID.Name}} records as follows: – "Supplier Name" as the name of the supplier. Using the Item_ID attribute in the **Order_Line** table that corresponds to the Item_ID attribute in the **Item** table, the system reads the {{Item_ID.Name}} records as follows: – "Item Name" with ItemName – A button with the text: "Received" in the column with the Received heading. – A trash icon in the column with the Cancel heading.
	4. The administrator wants to place an order. [ALT]		**5.** The system extends to "7.18 Place Supplier Order"
ALTERNATE COURSES:	[ALT] **Step 4a**: The administrator requests to search a supplier order. The system will prompt the administrator to enter the Supplier order details within the search bar. The administrator will enter the 'Supplier Name', 'Item Name' or 'Quantity' of the order. The system retrieves a list of all the supplier order records that match the criteria entered by the administrator using the following attributes from the **Item** and **Supplier_Order** entity: – Quantity – Supplier Name – Item name		

	[ALT] **Step 4b**: The administrator wants to cancel the order. The system extends to use case “7.19 Cancel Supplier Order”. [ALT] **Step 4c**: The administrator wants to check the checkbox indicating that the order is received. The system extends to use case “7.20 Receive Supplier Order”.
CONCLUSION:	The use case concludes when the system retrieves and displays the appropriate records within the **Supplier_Order** and **Item** entities.
POST-CONDITION:	The administrator will be able to place an order on the ‘*Supplier Order’* screen.
BUSINESS RULES:	• Only administrators can view supplier orders on the system.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	None
OPEN ISSUES:	None

USE CASE NAME:	Place Supplier order	USE CASE TYPE	
USE CASE ID:	7.18	Business Requirements:	☐
PRIORITY:	High	System Analysis:	☐
SOURCE:	Platinum Island resort	System Design:	☑
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	This use case describes an event where the administrator would like to place an order with a certain supplier. The use case begins when the administrator clicks on the "+" icon next to the "place supplier order" text. The system then prompts the administrator to fill in the required details of the order, so that the order can be sent to the supplier. The use case ends when the order is sent to supplier via email informing him of what the platinum island resort would need.		
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • The administrator clicks on the "+" button (place supplier order)		
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by loading the place supplier order pop-up modal with the following information: – Heading: "Place supplier order" – Label: "Supplier name" – Dropdown with the supplier names with the placeholder "Select a supplier" – Label: "Item name" – Dropdown with the list of items and placeholder "Select an item". – Label: "Quantity"

			– Numeric up down with the start number set as 1. – Submit Button with the text “Place order”. – Cancel Button with the text “Cancel”.
			2. The system prompts the administrator to enter the details for the order.
	3. The administrator will enter the required information within the text input fields.		**4.** The system will use the Angular frontend to validate that none of the input fields are left blank and enables the “Place order” button.
	5. The administrator clicks the “Place order” button. [ALT]		**6.** The system will make use of an SQL_Insert to capture the information entered by the administrator using a supplier order object with the following information: – Supplier Name – Item Name – Quantity The Supplier object will be sent to the Data Service where it will send an HttpPost request to the backend .NET Core.
			7. The system will make use of an SQL_Insert query in the controller to create a new record in the **Supplier_Order** entity: ○ Order_ID [PK] (int) ○ Supplier_ID [FK] (int) ○ User_ID [FK] (int) ○ Item_ID (int) ○ Date (date) ○ Quantity (int) The OrderStatus in the **Supplier_Order** entity is set to “Placed”. The Supplier object will be sent to the Data Service where it will send an HttpPost request to the .NET Core backend.

			8. Finally, the system sends an email using mailkit in the .Net Core backend, to the selected supplier informing the business of the order in the following format: – Email Heading: “Place Order” – Email body: Platinum Island would like to place an order for {{SupplierOrder.Quantity}} {{Item.Name}}
ALTERNATE COURSES:	[ALT] **Step 5**: The administrator selects the Cancel button. **The use case terminates**.		
CONCLUSION:	The use case concludes when the email is sent to the supplier.		
POST-CONDITION:	An email is sent to the supplier informing them about the order that was placed to the company.		
BUSINESS RULES:	• Only administrators can place supplier orders on the system.		
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None		
ASSUMPTIONS:	None		
OPEN ISSUES:	None		

USE CASE NAME:	Cancel Supplier order	USE CASE TYPE
USE CASE ID:	7.19	Business Requirements:o
PRIORITY:	High	System Analysis: o
SOURCE:	Platinum Island resort	System Design: þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes an event where the administrator would like to cancel an order that was placed with a certain supplier. The use case begins when the administrator clicks on the button with the trash icon. After the cancel button is clicked, the system will display a pop-up modal confirming to the administrator that he would like to want to cancel this order. Once the administrator confirms that he would like to cancel the order, an email is sent to the supplier informing him of the cancellation.	
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • The administrator clicks on the trash (*fa fa trash)* button (cancel order).	

TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by loading the cancel supplier order pop-up modal with the following information: – Heading: “Cancel supplier order” – Label: “Are you sure you want to cancel the order that was placed” – Submit Button with text “Confirm”. – Cancel button with the text “Cancel”.

			2. The system prompts the administrator to confirm the cancellation of the order.
	3. The administrator clicks on the confirm button. [ALT]		**4.** The system deletes the Supplier order record from the **Order_Line** Entity with the following attributes: ○ Item_ID [PK, FK] (int) ○ Order_ID (int) ○ Quantity ○ Order_total The system does this by performing an SQL_DELETE query in the .Net Core controller using LINQ Lambda and Entity Framework. The record is also removed from the table. The system changes the status of the supplier order from the **SupplierOrderStatus** Entity with the following attributes: ○ SupplierOrderStatus_ID [PK, FK] (int) ○ Name ○ Description The system does this by performing an SQL_UPDATE query in the .Net Core controller using LINQ Lambda and Entity Framework. The name attribute is changed to “cancelled”.
			5. The system sends an automated email to the supplier using mailkit in the .Net Core backend to send the email.
ALTERNATE COURSES:	[ALT] **Step 3**: The administrator chooses to cancel the email the email instead of sending it. **Terminate use case**.		

CONCLUSION:	The use case concludes when the email is sent to the supplier informing him about the cancellation.
POST-CONDITION:	An email is sent to the supplier informing them about the order that was cancelled. The record is deleted from the database and removed from the table.
BUSINESS RULES:	• Only authorised users of the system will be permitted to place an order to the supplier records on the system.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None.
ASSUMPTIONS:	None.
OPEN ISSUES:	None

USE CASE NAME:	Receive Supplier order	USE CASE TYPE
USE CASE ID:	7.20	Business Requirements:o
PRIORITY:	High	System Analysis: o
SOURCE:	Platinum Island resort	System Design: þ
PRIMARY BUSINESS ACTOR:	Administrator	
PRIMARY THE SYSTEM ACTOR:	None	
OTHER PARTICIPATING ACTORS:	None	
OTHER INTERESTED STAKEHOLDERS:	None	
DESCRIPTION:	This use case describes an event where the administrator would like to tick the check box and mark the order as received. The use case begins once the administrator receives the ordered stock and marks it as checked in the check box. After the checkbox is marked as checked, a pop-up modal is displayed prompting the administrator to confirm that the order was received. When he clicks on the confirm button an email is sent to supplier informing the supplier that the order was received.	
PRE-CONDITION:	• The admin must have access to internet. • The admin must be logged onto the system. • The administrator clicks on the checkbox (cancel order).	

		SYSTEM RESPONSE:	
TYPICAL COURSE OF EVENTS:	ACTOR ACTION:	MANUAL ACTION:	AUTOMATED ACTION:
			1. The system responds by loading the cancel supplier order pop-up modal with the following information: – Heading: “Receive the order?” – Label: “Are you sure that the order has arrived?” – Submit Button with text “Confirm”. – Cancel button with the text “Cancel”.

			2. The system prompts the administrator to confirm the arrival of the order.
			3. The system changes the status of the supplier order from the **SupplierOrderStatus** Entity with the following attributes: ○ SupplierOrderStatus_ID [PK, FK] (int) ○ Name ○ Description The system does this by performing an SQL_UPDATE query in the .Net Core controller using LINQ Lambda and Entity Framework. The name attribute is changed to "received".
	4. The administrator clicks on the button indicating that the order is correct and has been received. [ALT]		**5.** The system responds by sending an automated email to the supplier informing him that the order has been received by the administrator.

ALTERNATE COURSES:	[ALT] **Step 3**: The administrator chooses to click the cancel button. **Terminate use case**.
CONCLUSION:	The use case concludes when the email is sent to the supplier informing him about the arrival of the stock.
POST-CONDITION:	An email is sent to the supplier informing them that the order has been received by the administrator.
BUSINESS RULES:	• Only authorised users of the system will be permitted to receive an order from the supplier.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None.
ASSUMPTIONS:	None.

OPEN ISSUES:	None

2.8. Subsystem 8 – Reports

USE CASE NAME:	View Reports	USE CASE TYPE	
USE CASE ID:	8.1	Business Requirements:o	
PRIORITY:	High	System Analysis:	o
SOURCE:	Platinum Island Resort	System Design:	þ
PRIMARY BUSINESS ACTOR:	Administrator		
PRIMARY THE SYSTEM ACTOR:	None		
OTHER PARTICIPATING ACTORS:	None		
OTHER INTERESTED STAKEHOLDERS:	None		
DESCRIPTION:	The use case describes the event when the administrator requests to view the list of available reports on the “View Reports” screen. The use case begins with the administrator requesting to view the list of available reports on the system. The system will load and display it. The use case concludes when the administrator views the list of available reports on the system.		
PRE-CONDITION:	• The administrator is logged into the system.		
TRIGGER:	The client requests to view the view reports screen.		

	ACTOR ACTION:	SYSTEM RESPONSE:	
		MANUAL ACTION:	AUTOMATED ACTION:
TYPICAL COURSE OF EVENTS:	**1.** The administrator requests to view the audit log by navigating to the side nav bar and selecting the “Reports” option. The administrator will click the “Reports” option.		**2.** The system responds by displaying the View Reports screen with the following elements: – Heading label with text “Reports” – Heading label with text “Available Reports” – Card Option with text “peak season day visit summary report”

			– Card Option with text “accommodation summary report” – Card Option with text “room rating report” – Card Option with text “customer demographic report” – Card Option with text “event type report” – Card Option with text “booking type remuneration report”. – Card Option with text “supplier order list report” – Card Option with text “view employee list report” – Card Option with text “available rooms list report” – Card Option with text “view supplier list report”
	3. The administrator selects the “peak season day visit summary report” Card Option. [ALT]		**4.** The system extends to Use Case 8.2 Generate peak season day visit summary report.
ALTERNATE COURSES:	[ALT] **Step 3a**: The administrator selects the “accommodation report” card option. The system extends to Use Case 8.3 Generate Accommodation summary report. [ALT] **Step 3b**: The administrator selects the “room rating report” card option. The system extends to Use Case 8.4 Generate room rating report.		

	[ALT] **Step 3c**: The administrator selects the “customer demographic report” card option. The system extends to Use Case 8.5 Customer demographic report. [ALT] **Step 3d**: The administrator selects the “event type report” card option. The system extends to Use Case 8.6 Generate event type report. [ALT] **Step 3e**: The administrator selects the “booking type remuneration report” card option. The system extends to Use Case 8.7 Booking type remuneration report. [ALT] **Step 3f**: The administrator selects the “supplier order list report” card option. The system extends to Use Case 8.8 Supplier order report list. [ALT] **Step 3g**: The administrator selects the “view employee list report” card option. The system extends to Use Case 8.9 Generate Employee list. [ALT] **Step 3h**: The administrator selects the “available rooms list report” card option. The system extends to Use Case 8.10 Generate Available rooms list. [ALT] **Step 3i**: The administrator selects the “view supplier list report” card option. The system extends to Use Case 8.11 Generate Supplier list.
CONCLUSION:	The use case concludes when the system displays the “View Reports” screen.
POST-CONDITION:	The administrator will be able to select to view any of the available reports.
BUSINESS RULES:	The administrator only has access.
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS:	None
ASSUMPTIONS:	The administrator has internet connection
OPEN ISSUES:	None

USE CASE NAME:	Generate peak season day visit report	USE CASE TYPE	
USE CASE ID:	8.2	Abstract:	☐
PRIORITY:	High	Extension:	☐
SOURCE:	Platinum Island resort		
PARTICIPATING ACTORS:	Administrator (PBA)		
DESCRIPTION:	In this case, the administrator asks for the creation of a detailed report that shows day visits at the busiest times of the year. The system effectively collects information on reservations for day trips and groups them according to each month. The algorithm then dynamically creates a line graph that visually illustrates the variances in ticket sales throughout various months. The use case concludes when the administrator downloads the report.		
PRE-CONDITION:	• The administrator is logged onto the system. • The '*peak season day visit*' option was selected on the '*view reports*' screen.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system loads the '<u>Peak season day visit Report</u>' screen with the following elements: – Heading with the text: "Peak season day visit summary" – Font awesome icon, fa fa-calendar – Label with the text "Start". – Label with the text "End". – Angular material, datepicker – Button with the text "Generate Report ". – Button with the text "Print" The system prompts the administrator to enter the date period for the accommodation summary report within the datepicker input fields.	
	2. The administrator provides their date period of choice within the datepicker input fields and clicks the "Generate report" button	**3.** The system will use the Angular frontend to validate that the datepicker input fields are not left blank and ensures the details entered by the administrator follows the following criteria: – The input fields are not left blank. – The duration between the start date and end date chosen does not exceed a 12-month period.	

		[ALT]
		4. Once the date elements have been validated, the system will send through start-date and end-date as parameters via an Angular frontend request to the Report service where the service will make an HttpGet request to the .NET Core backend, along with the parameters selected by the administrator, which makes use of a Lambda LINQ Query which creates a SQL Read query to retrieve all the items from the **DayVisit** entity. The following attributes will be retrieved from the **DayVisit** entity: ○ DayVisit_ID [PK] (int) ○ DayVisitStatus [FK] (int) ○ Client_ID [FK] (int) ○ DayVisitDate_ID [FK] (int) ○ RefCode (varchar (6)) ○ Total (decimal (6,2)) In the .Net core backend, the system will then use a LINQ Query to filter the records according to the duration selected by the administrator using the *Date* attribute in the **DayVisitDate** entity. The Angular frontend will retrieve save the data sent from the .NET Core backend to the Report Service within an empty declared array which is called within the component.ts. [ALT]
		5. The system then reloads the <u>'Peak season day visit report'</u> screen with the updated following: – Heading with the text: "Peak season day visit summary" – Font awesome icon, fa fa-calendar – Label with the text "Start". – Label with the text "End".

		– Angular material, datepicker – Button with the text "Generate Report ". – A line chart which displays the total count of the monthly ticket sales filtered per month. – Button with the text "Print"
	6. The administrator clicks the "Print" button. [ALT]	**7.** The system generates PDF for the download of the report with the currently selected date aggregation which contains the following elements: – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text "Platinum Island Resort" within the ribbon – Heading with the text: "Peak season day visit summary" – A line chart which displays the total count of the monthly ticket sales filtered per month. – Text detailing who generated the report populated by the employee's name {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text "END"
ALTERNATE COURSES:	[ALT] **Step 3**: The system does not verify or validate the information provided by the administrator, there is an error. **Return to step 2**. [ALT] **Step 4**: The system could not retrieve any data regarding the number of ticket sales for the year and displays the following error message: – "Error! Could not retrieve any data". [ALT] **Step 6a**: The administrator does not select to print report. Use Case Terminates. [ALT] **Step 6b**: The administrator selects a new start and end date to generate the report. **Return to step 2**	
POST-CONDITION:	The administrator can now view the data on which month of the year had the most sales.	
CONCLUSION:	The use case concludes when the system generates the peak season day visit report and displays it in PDF format for download.	

USE CASE NAME:	Generate Accommodation Summary	USE CASE TYPE
USE CASE ID:	8.3	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator (PBA)	
DESCRIPTION:	The use case begins when the administrator requests to generate an accommodation summary report on the system. The system should allow the administrator to generate an accommodation summary report. The system will request the time period they would like to generate the report for. The system will then retrieve all the room bookings with their associated room booking types for the booked room bookings and then display the total count that the room type was booked per month in a triple bar graph. The use case concludes when the admin downloads the report.	
PRE-CONDITION:	• The administrator must be logged into the system. • The administrator clicks add on “Accommodation Summary Report” on the “View report” screen.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system loads the ‘*Accommodation Summary Report*’ screen with the following elements: ○ Heading with the text “Accommodation Summary Report” ○ Font awesome icon, fa fa-calendar ○ Label with the text “Start” ○ Label with the text “End” ○ Angular material, datepicker ○ Button with the text “Generate Report” The system then prompts the administrator to enter the date period for the accommodation summary report within the datepicker input fields.
	2. The administrator provides their date period of choice within the datepicker input fields and clicks the “Generate Report” button.	**3.** The system will use the Angular frontend to validate that the datepicker input fields are not left blank and ensures the details entered by the

		administrator follows the following criteria: ○ The input fields are not left blank ○ The duration between the start date and end date chosen does not exceed a 12-month period. [ALT]
		4. Once the date elements have been validated, the system will send through start-date and end-date as parameters via an Angular frontend request to the Report service where the service will make an HttpGet request to the .NET Core backend, along with the parameters selected by the administrator, which makes use of a Lambda LINQ Query which creates a SQL Read query to retrieve all the items from the **RoomBooking** entity. The following attributes will be retrieved from the **RoomBooking** entity: ○ RoomBooking_ID (int) [PK] ○ Room_ID (int) [FK] which will be used to access the *TypeName (*varchar(20)*)* attribute within the **RoomType** entity where *RoomType*_ID (int) is foreign key within the **Room** entity {{Room_ID.RoomType_ID.TypeName}}. ○ EntryDate (DateTime) ○ ExitDate (DateTime) In the .NET Core backend, the system will use a LINQ query to filter the room bookings retrieved according to

		the date duration selected by the administrator by checking the *EntryDate* and *ExitDate* attributes from the **RoomBooking** entity fall within the date period. The system then calculates the number of instances related to a specific Room Type as well as the total number of instances per Room Type according to the selected date period. The system will make use of a further LINQ Lambda query to first group each room type then count each instance within the **RoomBooking** entity linked to a specific **RoomType** entity instance and returns the total count per room type. The Angular frontend will retrieve save the data sent from the .NET Core backend to the Report Service within an empty declared array which is called within the component.ts. [ALT]
		5. The system then reloads the '*Accommodation Summary Report*' screen with the updated following: ○ Heading with the text "Accommodation Summary Report" ○ Font awesome icon, fa fa-calendar ○ Label with the text "Start" ○ Label with the text "End"

		○ Angular material, datepicker ○ Button with the text “Generate Report” ○ Chart header with the text “Accommodation Summary Report” ○ A bar chart which displays the total count per room type filtered. Each room type represents a different bar within the bar chart. ○ Above the bar chart is the labels for each room type with its associated colour. ○ Button with the text “Print”
	6. The administrator clicks the “Print” button. [ALT]	**7.** The system generates PDF for the download of the report with the currently selected date aggregation which contains the following elements: ○ Platinum Island Resort Logo to the left of a colour ribbon. ○ Heading with the text “Platinum Island Resort” within the ribbon. ○ Heading with the text “Accommodation Summary Report” ○ Description text describing the report ○ Text detailing the date criteria selected by the administrator ○ A bar chart which displays the total count per room type filtered per month. Each room type represents a different bar within the bar chart. ○ Above the bar chart is the labels for each room type with its associated colour. ○ Text detailing who generated the report populated by the

		employee's name – {{EmployeeName}} ○ Date at which the report was generated which uses the Date.Now() method. ○ Label with the text "END".
ALTERNATE COURSES:	[ALT] **Step 3**: The system does not verify or validate the information provided by the administrator, there is an error. **Return to step 2**. [ALT] **Step 4**: The system is unable to retrieve the necessary data. Display appropriate error message. **Use Case Terminates**. [ALT] **Step 6a**: The administrator does not select to print report. **Use Case Terminates**. [ALT] **Step 6b**: The administrator selects a new start and end date to generate the report. **Return to step 2**.	
POST-CONDITION:	• The system prints the accommodation summary report.	

USE CASE NAME:	Generate Room Rating Report	USE CASE TYPE
USE CASE ID:	8.4	Abstract:¨
PRIORITY:	High	Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the admin would like to generate a room rating report. The use case begins when the administrator requests to generate a Room Rating Report. The system will then retrieve all the Room ratings for each room type and display the different proportions of ratings each type of room receives. The use case concludes when the administrator downloads the report.	
PRE-CONDITION:	• The admin must be logged into the system. • The admin selects the Room Rating Report option.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve all the records from the **Review** Entity and the **RoomType** Entity. The Reviews will be retrieved using the Review_ID as the primary key inside the Review Entity where all Reviews will be retrieved. The Room Types will be retrieved using the RoomType_ID as the primary key inside the RoomType Entity where all Room Types will be retrieved. The RoomType entity is connected to the Review entity via the RoomType_ID (FK). Once the system retrieves the number of intances belonging to each Review_ID. The

		following attributes from the Review Entity are retrieved: ○ Rating [ALT]
		2. The system performs a calculation using the above-mentioned information to calculate the average of ratings given to each room type for each room booking generated. Using Angular the system displays the bar chart with the average rating per room type for all room bookings up to date. The average is calculated using the Average function for every rating from all room bookings recorded.
		3. The system loads the "Generate Room Rating Report" with the following elements: – Heading Label with text "Room Rating Report" – A bar chart is displayed under the heading, which displays the average ratings for each room type, which consists of a 2-sleeper room, a 3-sleeper room, and a 4-sleeper room. Thus, there is 3 different lines which are coloured differently per each room type. To the left of the bar chart there is labels for each room type and their associated colours. – Button with the text "Print"
	4. The administrator clicks the download button. [ALT]	**5.** The system then downloads the report in the following format: – Platinum Island Resort Logo to the left of a colour ribbon.

		– Heading with the text “Platinum Island Resort” within the ribbon. – Heading Label with text “Room Rating Report” – Description text describing the report. – A bar chart is displayed under the heading, which displays the average ratings for each room type, which consists of a 2-sleeper room, a 3-sleeper room, and a 4-sleeper room. Thus, there is 3 different lines which are coloured differently per each room type. Above the bar chart there is labels for each room type and their associated colours. – Text detailing who generated the report populated by the employee’s name – {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text “END”.
ALTERNATE COURSES:	[ALT] **Step 1**: The system is unable to retrieve the necessary data. Display appropriate error message. **Use Case Terminates**. [ALT] **Step 4**: The administrator does not choose to download the report. **Use Case Terminates**.	
POST-CONDITION:	The system successfully retrieved the information, and the administrator can download the report.	

USE CASE NAME:	Customer demographic report	USE CASE TYPE
USE CASE ID:	8.5.	Abstract: ☐
PRIORITY:	High	Extension: ☐
SOURCE:	Platinum Island resort	
PARTICIPATING ACTORS:	• Administrator (PBA)	
DESCRIPTION:	The use case describes an event in which the administrator wishes to generate the customer demographic report. The use case begins when the administrator selects the 'customer demographic report' option on the 'view reports' screen. The system displays a doughnut graph using the retrieved data on the different types of tickets bought over the period of the year. The use case ends when the administrator can view the customer demographic report and download it.	
PRE-CONDITION:	• The administrator is logged onto the system. • The "*Customer demographic report* " option was selected on the "*View reports*" screen.	

	Actor Actions	System Response
sTYPICAL COURSE OF EVENTS:		**1.** The system will send a request form the Angular frontend to the Report service where the service will make a http get request to the .NET Core backend, along with the currents years beginning date and end date, which makes use of a Lambda Linq Query which creates a SQL Read query to retrieve all the items from the **DayVisit, DayVisitTicket** and **DayVisitType** entities. The **DayVisit** entity has the following attributes: ○ DayVisit_ID [PK] (int) ○ DayVisitStatus [FK] (int) ○ Client_ID [FK] (int) ○ DayVisitDate_ID [FK] (int) ○ NumberOfGuests (int) ○ Total (decimal (6,2)) The **DayVisitTicket** entity has the following attributes: ○ Ticket_ID [PK] (int) ○ DayVisit_ID[FK] (int) Where the DayVisit_ID in the **DayVisit** entity, corresponds with the DayVisit_ID in **DayVisitTicket** entity. ○ Client_ID [FK] (int) ○ DayVisitType_ID [FK] (int) Where the DayVisitType_ID

		in the **DayVisitType** entity, corresponds with the DayVisitType_ID in **DayVisitTicket** entity. ○ NumberOfGuests (int) The **DayVisitType** entity has the following attributes: ○ DayVisit_ID [PK] (int) ○ Category (varchar (20)) ○ Price (decimal (6,2)) ○ Date (date) [ALT]
		2. The system will then do a calculation on the total amount of ticket sales done in the past year and using the GROUP function in SQL, the system will then group them by each category using the category attribute In the **DayVisitType** entity.
		3. The system then reloads the '<u>Customer demographic</u>' screen with the updated following: – Heading with the text: "Customer demographic report" – A pie chart with the percentages of ticket sales in the last year per category. – Button with the text "Print" The system prompts the administrator to print the report.
	4. The administrator clicks the "Print" button. [ALT]	**5.** The system generates PDF for the download of the report which contains the following elements: – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text "Platinum Island Resort" within the ribbon – Heading with the text: "Customer demographic report." – A pie chart with the percentages of ticket sales

		in the last year per category. – Text detailing who generated the report populated by the employee's name {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text "END"
ALTERNATE COURSES:	[ALT] **Step 1**: The system could not retrieve any data regarding the number of ticket sales for the year and displays the following error message: – "Error! Could not retrieve any data". [ALT] **Step 4**: The administrator does not select to print report. **Use Case Terminates**.	
POST-CONDITION:	The administrator can now view they report that was generated by the system.	
CONCLUSION:	The use case concludes when the system generates the customer demographic report and displays it in PDF format for download.	

USE CASE NAME:	Generate event type report	USE CASE TYPE
USE CASE ID:	8.6	Abstract: ☐
PRIORITY:	High	**Extension:** ☐
SOURCE:	Platinum Island resort	
PARTICIPATING ACTORS:	• Administrator (PBA)	
DESCRIPTION:	The use case describes an event in which the administrator wishes to generate the event type report. The use case begins when the administrator selects the '*Generate event type report*' option on the *'view reports'* screen. The system displays a bar graph using the retrieved data on the different types of events that the venue was used for. The use case ends when the administrator can view the event type report and download it.	
PRE-CONDITION:	• The administrator is logged onto the system. • The '*Generate event type report*' option was selected on the *'view reports'* screen.	

TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will then create a service will make an HttpGet request to the .NET Core backend, along with the parameters selected by the administrator, which makes use of a Lambda LINQ Query which creates a SQL Read query to retrieve all the items from the **Event** entity. The system then retrieves the following attributes that are from the **Event** entity: ○ Description (varchar (10)) In the .Net core backend, the system will then use a LINQ Query to filter the records according to the duration selected by the administrator using the *Date* attribute in the **Event** entity. The .Net core backend then makes use of a LINQ lambda expression in order to count every instance of the different options from the

		description attributes in the **Event** entity. The Angular frontend will retrieve save the data sent from the .NET Core backend to the Report Service within an empty declared array which is called within the component.ts. [ALT]
		2. The system then reloads the '<u>Event type report</u>' with the following updated elements: – Platinum island resort logo – Label with the text: "Platinum Island resort" – Heading with the text: "Event type report" – A list depicting the number of number of selected event types per category. – Table with the following headers: o Birthdays o Conferences o Weddings The table is then populated with the counts from the array for each different event description. – Button with the text "Print" – Text detailing who generated the report populated by the employee's name {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text "END"

	2. The administrator clicks the "Print" button. [ALT]	**3.** The system generates PDF for the download of the report with the currently selected date aggregation which contains the following elements: – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text "Platinum Island Resort" within the ribbon – Heading with the text: "Event type report" – A list depicting the number of number of selected event types per category. – Text detailing who generated the report populated by the employee's name {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text "END"
ALTERNATE COURSES:	[ALT] **Step 2**: The system could not retrieve any data regarding the different event types booked. [ALT] **Step 3**: The administrator does not select to print report. **Use Case Terminates**.	
POST-CONDITION:	The administrator can now view the data on which event type has had the most frequent selection.	
CONCLUSION:	The use case concludes when the system generates the event type report and displays it in PDF format for download.	

USE CASE NAME:	Generate Booking Type Remuneration Report	USE CASE TYPE
USE CASE ID:	8.7	Abstract: ¨
PRIORITY:	High	Extension: x
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator (PBA)	
DESCRIPTION:	This use case describes the event where the administrator would like to generate a booking type renumeration report. The system should allow the administrator to generate this financial report. The system retrieves all the payments related to each type of booking made on the system. The system will then populate a table that will display the overall amount paid per each subcategory of a booking type, the overall total for each booking type and the overall total of all the booking payments made on the system. The use case concludes when the administrator chooses to print the report	
PRE-CONDITION:	• The administrator must be logged into the system. • The administrator clicks add on “Booking Type Remuneration Report” on the “View report” screen.	

TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will send a request form the Angular frontend to the Report service where the service will make a http get request to the .NET Core backend, which makes use of a Lambda LINQ Query which creates multiple SQL_Read queries to retrieve all the items from the **DayVisitPayment**, **DayVisit**, **DayVisitTicket**, **DayVisitType**, **RoomPayment**, **RoomBooking**, **Room**, **RoomType**, **RoomBookingRefund**,**EventPayment**, **EventRefund**,and **Event** entities. All the above are linked in the database as follows: – The day visit payment records will be retrieved via **DayVisitPayment** entity using the *DayVisitPayment_ID [PK]* which is connected to the **DayVisit** entity via the *DayVisit_ID [FK]* in the **DayVisitPayment** entity. – The day visit ticket records will be retrieved via **DayVisitTicket** entity using the *DayVisitTicket_ID [PK]* which is

		connected to the **DayVisit** entity via the *DayVisit_ID [FK]* in the **DayVisitTicket** entity. – The day visit type records will be retrieved via **DayVisitType** entity using the *DayVisitType_ID [PK]* which is connected to the **DayVisitTicket** entity via the *DayVisitType_ID [FK]* in the **DayVisitTicket** entity. – The room booking payment records will be retrieved via **RoomPayment** entity using the *RoomPayment_ID [PK]* which is connected to the **RoomBooking** entity via the *RoomBooking_ID [FK]* in the **RoomPayment** entity. – The room booking refund records will be retrieved via **RoomBookingRefund** entity using the *Refund_ID [PK]* which is connected to the **RoomBooking** entity via the *RoomBooking_ID [FK]* in the **RoomBookingRefund** entity. – The **RoomBooking** entity is connected to the **RoomType** entity via the **Room** entity which is joined to it via the *Room_ID [FK].* The **Room** entity is then connected to the **RoomType** entity via the *RoomType_ID [FK]*. – The event booking payment records will be retrieved via **EventPayment** entity using the Event*Payment_ID [PK]* which is connected to the **Event** entity via the *Event_ID [FK]* in the **EventPayment** entity. – The event booking refunds records will be retrieved via **EventRefund** entity using the Refund*_ID [PK]* which is connected to the **Event** entity via the *Event_ID [FK]* in the **EventRefund** entity. [ALT]

		2. In the .NET Core backend, the system will respond to the retrieval of entities by performing the following processing:
		– Each record within the **RoomPayment** entity will be linked a room booking record through the use of the *RoomBooking_ID [FK]*. Within the each of those records, the system will find the corresponding room which was booked within the room booking by linking the *Room_ID [FK]* within the **RoomBooking** entity to the *Room_ID [PK]* in the **Room** entity. Within the **Room** entity, the system will read the *TypeName* attribute of the room types using the *RoomType_ID [FK].* The system will then read the *Amount* attribute from each record found within the **RoomPayment** entity and group each record found into its associated room type category. The system will then total each *Amount* record found within the **RoomPayment** entity according to which room type the payment is associated to. The system will then declare three list objects which will be used to store each category of room type amounts retrieved:
		  – RoomType1Arr
		  – RoomType2Arr
		  – RoomType3Arr
		  The system will group and store the amounts per room type according to the following criteria:
		  – Where the payment record has an associated *RoomType_ID == 1*, the system will store the values retrieved within a List Object labelled “roomType1Arr”

		– Where the payment record has an associated *RoomType_ID == 2*, the system will store the values retrieved within a List Object labelled "roomType2Arr" – Where the payment record has an associated *RoomType_ID == 3*, the system will store the values retrieved within a List Object labelled "roomType3Arr" – Each record within the **RoomBookingRefund** entity will be linked a room booking record through the use of the *RoomBooking_ID [FK]* which is linked to the *RoomBooking _ID [PK]* found within the **RoomBooking** entity. The system will then total each *Amount* record found within the **RoomBookingRefund** entity. The system will then declare a list object which will be used to store the room refund amounts retrieved – roomRefundArr – Each record within the **EventPayment** entity will be linked an event booking record through the use of the *Event_ID [FK]* which is linked to the *Event_ID [PK]* found within the **Event** entity. Within the **Event** entity, the system will read the *Description* attribute which details the type of event chosen by the client. The system will then read the *Amount* attribute from each record found within the **EventPayment** entity and group each record found into its associated event type category. The system will then total each *Amount* record found within the

		EventPayment entity according to which event type the payment is associated to. The system will then declare three list objects which will be used to store each category of event type amounts retrieved: – EventType1Arr – EventType2Arr – EventType3Arr The system will group and store the amounts per event type according to the *description* attribute found within the **Event** entity with the following criteria: – Where the payment record has an associated *description == "conference"*, the system will store the values retrieved within a List Object labelled "eventType1Arr" – Where the payment record has an associated *description == "wedding"*, the system will store the values retrieved within a List Object labelled "eventType2Arr" – Where the payment record has an associated *description == "birthday"*, the system will store the values retrieved within a List Object labelled "eventType3Arr" – Each record within the **EventRefund** entity will be linked an event booking record through the use of the *Event_ID [FK]* which is linked to the *Event_ID [PK]* found within the **Event** entity. The system will then total each *Amount* record found within the **EventRefund** entity. The system will then

		declare a list object which will be used to store the event refund amounts retrieved – eventRefundArr – Each record within the **DayVisitPayment** entity will be linked to a day-visit purchase booking record through the use of the *DayVisit_ID [FK]* which is linked to the *DayVisit_ID [PK]* found within the **DayVisit** entity. Each day visit booking is comprised of multiple tickets which is identified by the *Ticket_ID [PK]* within the **DayVisitTicket** entity. Each ticket is categorised into a type, the system will use the *DayVisitType_ID [FK]* attribute to determine what each type of ticket per day visit purchase is categorised into by reading the *Category* attribute within the **DayVisitType** entity. The system will then read the *Amount* attribute from each record found within the **DayVisitPayment** entity and group each record found into its associated day visit ticket type category. The system will then total each *Amount* record found within the **DayVisitPayment** entity according to which day visit ticket types the payment is associated to. The system will then declare three list objects which will be used to store each category of day visit types amounts retrieved: – DayVisitAdultArr – DayVisitChildArr – DayVisitPensionerArr The system will group and store the amounts per event type according to the day visit types according to the following criteria: – Where the payment record has an

		associated *category* == *"Adult"*, the system will store the values retrieved within a List Object labelled "DayVisitAdultArr"
		– Where the payment record has an associated *category* == *"Children"*, the system will store the values retrieved within a List Object labelled "DayVisitChildArr"
		– Where the payment record has an associated *category* == *"Pensioner"*, the system will store the values retrieved within a List Object labelled "DayVisitPensionerArr"
		
		In the backend Report Controller, the system will declare and populate the following variables by performing the appropriate calculations:
		– roomType1 (which will be populated by summing each index value retrieved within the roomType1Arr array)
		– roomType2 - (which will be populated by summing each index value retrieved within the roomType2Arr array)
		– roomType3 (which will be populated by summing each index value retrieved within the roomType3Arr array)
		  – roomTotal (which is calculated by adding each room type variable – {*roomType1 + roomType2 + roomType3*})
		  – roomRefund (which will be populated by summing each index value retrieved within the roomRefundArr array)

		– conferenceTotal (which will be populated by summing each index value retrieved within the eventType1Arr array) – weddingTotal (which will be populated by summing each index value retrieved within the eventType2Arr array) – birthdayTotal (which will be populated by summing each index value retrieved within the eventType3Arr array) – eventTotal (which is calculated by adding each event type variable – {*conferenceTotal* + *weddingTotal* + *birthdayTotal*}) – EventRefund (which will be populated by summing each index value retrieved within the eventRefundArr array) – adultTotal (which will be populated by summing each index value retrieved within the DayVisitAdultArr array) – childrenTotal (which will be populated by summing each index value retrieved within the DayVisitChildArr array) – pensionerTotal (which will be populated by summing each index value retrieved within the DayVisitPensionerArr array) – ticketTotal (which is calculated by adding each day visit type variable – {*adultTotal* + *childrenTotal* + *pensionerTotal*}) – overallTotal (which is the overall total calculated for the booking types –

		{*roomTotal + eventTotal + ticketTotal*}) ▪ The Angular frontend will retrieve the populated variables sent from the .NET Core backend to the Report Service which will be called within the "Remuneration component.ts". – netTotal (which is the net total calculated for all the booking types minus refunds – {*overallTotal – eventRefund - roomRefund* })
		3. The system then loads the '*<u>Booking Type Remuneration Report</u>*' screen with the updated following: – Heading with the text "Booking Type Remuneration Report" – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text "Platinum Island Resort" within the ribbon. – Description text describing the report – A table with the following elements: ○ Table header with the text "Booking Type" ○ Table header with the text "Total Sales"

		○ Table row populated with the following elements: • Row label with the text “Conference” • Row data with the total amount for conference event payments – {{conferenceTotal}} ○ Table row populated with the following elements: • Row label with the text “Wedding” • Row data with the total amount for wedding event payments – {{weddingTotal}} ○ Table row populated with the following elements: • Row label with the text “Birthday” • Row data with the total amount for birthday event payments – {{birthdayTotal}} ○ Table row populated with the following elements: • Row label with the text “Total Sales Across All Event Types” • Row data with the total accumulated amount for all events payments – {{eventTotal}} ○ Table row populated with the following elements: • Row label with the text “Duo” • Row data with the total amount for Duo room booking payments – {{roomType1}} ○ Table row populated with the following elements: • Row label with the text “Family”

		• Row data with the total amount for Family room booking payments – {{roomType2}} ○ Table row populated with the following elements: • Row label with the text “Single” • Row data with the total amount for Single room booking payments – {{roomType3}} ○ Table row populated with the following elements: • Row label with the text “Total Sales Across All Room Types” • Row data with the total accumulated amount for all room booking payments – {{roomTotal}} ○ Table row populated with the following elements: • Row label with the text “Adults” • Row data with the total amount for Adult ticket purchase payments – {{adultTotal}} ○ Table row populated with the following elements: • Row label with the text “Kids” • Row data with the total amount for Children ticket purchase payments – {{childrenTotal}} ○ Table row populated with the following elements: • Row label with the text “Seniors” • Row data with the total amount for Pensioner ticket purchase payments – {{pensionerTotal}}

		○ Table row populated with the following elements: • Row label with the text “Total Sales Across All Categories” • Row data with the total accumulated amount for all ticket purchase payments – {{ticketTotal}} ○ Table row populated with the following elements: • Row label with the text “Total:” • Row data with the total accumulated amount for all booking type payments – {{overallTotal}} ○ Table row populated with the following elements: • Row label with the text “Total Refunds Across All Room Types:” • Row data with the total accumulated amount for all room refunds – {{roomRefund}} ○ Table row populated with the following elements: • Row label with the text “Total Refunds Across All Event Types:” • Row data with the total accumulated amount for all event refunds – {{eventRefund}} ○ Table row populated with the following elements: • Row label with the text “NET INCOME:” • Row data with the total accumulated amount sales minus refund amounts – {{netTotal}}

		– Text detailing who the report was generated by populated by the employee's name – {{EmployeeName}} – The date at which the report was generated using the Date.Now() method. – Label with the text "END" – Button with the text "Print"
	3. The administrator clicks the "Print" button. [ALT]	**4.** The system generates PDF for the download of the report which contains the following elements: – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text "Platinum Island Resort" within the ribbon. – Heading with the text "Booking Type Remuneration Report" – Description text describing the report – A table which details the finances behind each subcategory of each booking type, the total for each booking type as well as the overall total of all booking types. – Text detailing who generated the report populated by the employee's name – {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text "END".
ALTERNATE COURSES:	[ALT] **Step 1**: The system is unable to retrieve the necessary data. Display appropriate error message. Use Case Terminates. [ALT] **Step 3**: The administrator does not select to print report. **Use Case Terminates**.	
POST-CONDITION:	• The system prints the booking type remuneration report.	

USE CASE NAME:	Generate Supplier Order List	USE CASE TYPE
USE CASE ID:	8.8	Abstract:¨
PRIORITY:	High	x Extension:
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	Administrator	
DESCRIPTION:	This use case describes the event where the administrator would like to generate a supplier order list report. The use case begins when the administrator requests to generate a Supplier Order List Report. The system will then retrieve all the supplier order records and display them in a list report. The use case concludes when the administrator downloads the report.	
PRE-CONDITION:	• The admin must be logged into the system. • The admin selects the supplier order list report option.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will send a request from the Angular frontend to the Data Service class where the service will make a HttpGet request to the .NET Core backend which makes use of a LINQ Query which utilizes an SQL_Read query to retrieve all the records from the **Supplier** Entity, the **Order_Line** Entity and the **SupplierOrder** Entity. The Supplier Orders will be retrieved using the Order_ID as the primary key in the **Supplier_Order** entity where all the Supplier Orders will be retrieved. The **Supplier_Order** entity is connected to the Supplier entity via the Supplier_ID(FK). The **Supplier_Order** entity is also connected to the **Supplier_Order_Status** entity via the OrderStatus_ID (PK, FK). The Supplier_Order entity is also connected to the **Item** entity via the Item_ID (PK, FK). Once the system retrieves the number of instances belonging to each Order_ID. The following attributes from the

		Supplier_Order entity are retrieved: ○ Date ○ Quantity The system will also retrieve attributes belonging to each instance of the Supplier_ID from the **Supplier** entity: ○ FullName The system will also retrieve attributes belonging to each instance of the Item_ID (PK, FK) from the **Item** entity: ○ Name [ALT]
		2. Using Angular the system displays the list with all the past Supplier Orders.
		3. The system loads the "Supplier Order List Report" with the following elements: - Heading label with text "Supplier Order List Report" - A list is displayed under the heading which displays all past supplier orders. The list consists of columns made up of the attributes mentioned above for each supplier order. - Button with the text "Print"
	4. The administrator clicks the download button. [ALT]	**5.** The system then downloads the report in the following format: – Platinum Island Resort Logo to the left of a colour ribbon. - Heading with the text "Platinum Island Resort" within the ribbon. - Heading label with text "Supplier Order List Report" - Description text describing the report.

		- A list is displayed under the heading which displays all past supplier orders. The list consists of columns made up of the attributes mentioned above for each supplier order. - Text detailing who generated the report populated by the employee's name – {{EmployeeName}} - Date at which the report was generated which uses the Date.Now() method. - Label with the text "END".
ALTERNATE COURSES:	[ALT] **Step 1**: The system is unable to retrieve the necessary data. Display appropriate error message. **Use Case Terminates**. [ALT] **Step 4**: The administrator does not choose to download the report. **Use Case Terminates**.	
POST-CONDITION:	The system successfully retrieved the information, and the administrator can download the report.	

USE CASE NAME:	View Employee List	USE CASE TYPE
USE CASE ID:	8.9.	Abstract: ☐
PRIORITY:	Moderate	Extension: ☐
SOURCE:	Platinum Island Resort	
PARTICIPATING ACTORS:	• Admin (PBA)	
DESCRIPTION:	This use case describes the event where the admin would like to generate a list of all employees. The system should allow the admin to generate a list of employees by retrieving all the employee's details and populates the list by displaying the employees full name, ID number, Employee type, contact number and HireDate. The use case concludes when the admin downloads the report	
PRE-CONDITION:	• The "*View employee list*" option was selected on the "*View reports*" screen.	
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response
		1. The system will send a request form the Angular frontend to the Report service where the service will make a http get request to the .NET Core backend, making use of a Lambda Linq Query which creates a SQL Read query to retrieve all the items from the **Employee Entity, and Employee type entity.** The **Employee** entity contains the following attributes: ○ ID_Num (Varchar (20)) ○ Cell_Num (Varchar(20)) ○ EmployeeName (varchar(20)) ○ HireDate (Datetime) The *Type_name* attribute will be retrieved from the *Employee_Type* table by referencing the EmployeeType_ID attribute in the Employee table where the following attributes will be retrieved: **Employee_Type:** ○ TypeName (Varchar (20)) [ALT]
		2. The system then loads a '<u>View employee list</u>' report with the following elements: – Platinum Island resort logo

		– Header: “Employee list” – Label: “This report displays the employees that are stored on the system” **Table headings:** – “Full Name” – “ID Number” – “Employee type” – “Contact No.” – “Hire Date” – Button with the text “Print” The system prompts the administrator to print the report.
	3. The administrator clicks the “Print” button. [ALT]	**4.** The system generates PDF for the download of the report with the currently selected date aggregation which contains the following elements: – Platinum Island Resort Logo to the left of a colour ribbon. – Heading with the text “Platinum Island Resort” within the ribbon – Heading with the text: “Peak season day visit summary” **Table headings:** – “Full Name” – “ID Number” – “Employee type” – “Contact No.” – “Hire Date” – Text detailing who generated the report populated by the employee’s name {{EmployeeName}} – Date at which the report was generated which uses the Date.Now() method. – Label with the text “END”
ALTERNATE COURSES:	[ALT] **Step 1**: An error occurred on the system and the employee list data could not be retrieved. The system displays the following error message: Error! Could not retrieve any data.	

	[ALT] **Step 3a**: The administrator does not select to print report. **Use Case Terminates**. [ALT] **Step 3b**: The administrator selects a new start and end date to generate the report. **Return to step 2**
POST-CONDITION:	The admin can now view a list of all the employee's stored on the system with their details.

USE CASE NAME:	Available rooms list	USE CASE TYPE	
USE CASE ID:	8.10	Abstract:	☐
PRIORITY:	High	Extension:	☐
SOURCE:	Platinum Island resort		
PARTICIPATING ACTORS:	• Administrator (PBA)		
DESCRIPTION:	The use case describes an event in which the administrator wishes to generate a list displaying a list of available rooms in a given period of time. The use case begins when the administrator selects the dates in which he wants to generate the list. The system will then retrieve data using the given dates and create a list of available rooms. The use case ends when the administrator can download the available rooms list from the system.		
PRE-CONDITION:	• The administrator is logged onto the system. • The '*Generate available rooms list'* option was selected on the *'view reports'* screen.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system loads the '<u>Available rooms list report</u>' screen with the following elements: – Heading with the text: "Available rooms list" – Button with the text "Generate Report ". – Button with the text "Download Report."	
		2. The system will send through the current date as parameters via an Angular frontend request to the Report service where the service will make an HttpGet request to the .NET Core backend, along with the parameters selected by the administrator, which makes use of a Lambda LINQ Query which creates a SQL Read query to retrieve all the items from the **Room** and **RoomType** entity where the status in the **RoomBookingStatus** entity is available The system retrieves the relevant data from the **Room** entity with the following attributes: ○ Room_ID [PK] (int) ○ RoomType_ID [FK] (int) where the RoomType_ID in	

		the Room entity corresponds with the RoomType_ID in the **RoomType** entity. ○ RoomNumber_ID (int) ○ Reminder_ID [FK] (int) ○ RoomFloor Using the RoomType_ID in the **Room** entity, the system will retrieve the following attributes in the **RoomType** entity: ○ TypeName (Varchar (20)) ○ RoomCapacity (int) In the .Net core backend the system will use LINQ query to filter out all the records according to Name attribute in the **RoomStatus_ID** where the details of the Name attribute are “Available”. [ALT]
		3. Using the dates selected, the system then loads the ‘Available rooms list’ with the following details: – Heading with the text: “Available rooms list” – Button with the text “Generate Report “. – Label: “This report displays the list of rooms currently available at the resort” – Button with the text: “Download Report" **Table headings:** – “Room Number” – “Room Floor” – “Type Of Room” – “Room Capacity.” The system prompts the administrator to download the report.
	4. The administrator wishes to download the available rooms list and selects the “Download report” button. [ALT]	**5.** The system generates PDF for the download of the report with the current date which contains the following elements:

		– Platinum Island Resort Logo to the left of a colour ribbon.
		– Heading with the text “Platinum Island Resort” within the ribbon
		– Heading with the text: “Peak season day visit summary”
		– A Table with the following headings:
		  ○ “Room Number”
		  ○ “Room Floor”
		  ○ “Type Of Room”
		  ○ “Room Capacity.”
		  ○ “Description”
		
		– Text detailing who generated the report populated by the employee’s name {{EmployeeName}}
		– Date at which the report was generated which uses the Date.Now() method.
		– Label with the text “END”

ALTERNATE COURSES:	[ALT] **Step 2**: The system could not retrieve any data regarding the available rooms for the selected time period displays the following error message: – “Error! Could not retrieve any data”. [ALT] **Step 4a**: The administrator does not select to print report. **Use Case Terminates**. [ALT] **Step 4b**: The administrator selects a new start and end date to generate the report. **Return to step 2**	
POST-CONDITION:	The administrator can now view the data on the different available rooms in a given time period.	
CONCLUSION:	The use case concludes when the system generates the available rooms list and displays it in PDF format for download.	

USE CASE NAME:	View Supplier List	USE CASE TYPE	
USE CASE ID:	8.11.	Abstract:	☐
PRIORITY:	Moderate	Extension:	☐
SOURCE:	Platinum Island Resort		
PARTICIPATING ACTORS:	• Admin (PBA)		
DESCRIPTION:	This use case describes the event where the admin would like to generate a list of all suppliers. The system should allow the admin to generate a list of suppliers by retrieving all the supplier's details and populates the list by displaying the suppliers full name, Email, contact number and the description. The use case concludes when the admin downloads the report		
PRE-CONDITION:	• The "*View supplier list*" option was selected on the "*View reports*" screen.		
TYPICAL COURSE OF EVENTS:	Actor Actions	System Response	
		1. The system will send a request form the Angular frontend to the Report service where the service will make a http get request to the .NET Core backend, making use of a Lambda Linq Query which creates a SQL Read query to retrieve all the items from the **Supplier Entity.** The above entity is linked in the database as follows: **Supplier:** ○ FullName (Varchar(20)) ○ Email (Varchar(20)) ○ ContactNum (varchar(20)) ○ Description (varchar(20)) [ALT]	
		2. The system then loads a View supplier list report with the following elements: – Platinum Island resort logo – Header: "View supplier list" – Label: "This report displays the suppliers that are stored on the system" – Button with the text: "Download Report" **Table headings:** – "Full Name" – "Email Address" – "Contact No."	

		– "Description" The system prompts the administrator to download the report.

	3. The admin wishes to download the View supplier list report and selects the "Download report" button.
	[ALT]
		4. The system generates PDF for the download of the report with the currently selected date aggregation which contains the following elements:
		
		– Platinum Island Resort Logo to the left of a colour ribbon.
		– Heading with the text "Platinum Island Resort" within the ribbon
		– Heading with the text: "Peak season day visit summary"
		– A Table with the following headings:
		  ○ "Full Name"
		  ○ "Email Address"
		  ○ "Contact No."
		  ○ "Description"
		
		– Text detailing who generated the report populated by the employee's name {{EmployeeName}}
		– Date at which the report was generated which uses the Date.Now() method.
		– Label with the text "END"

ALTERNATE COURSES:	[ALT] **Step 1**: The system could not retrieve any data regarding the Suppliers and displays the following error message: – "Error! Could not retrieve any data". [ALT] **Step 3**: The administrator does not select to print report. **Use Case Terminates**.	
POST-CONDITION:	The admin can now view a list of all the supplier's stored on the system with their details.	

3. Document Conclusion

The above document contained the technical narratives for the User, Client, Accommodation, Ticketing, Events, Admin, Inventory, and Reporting Subsystem. This concludes the detailed descriptions given for each technical use case.

4. Team sign-off

I, Ismaeel Rahaman, confirm that I have read the requirement changes of the Platinum Island Resort and agree to all of the information contained in this document. I confirm that the information stated above is factual and that I am aware of any and all changes made to this document between drafts. Furthermore, I confirm that I have contributed to this document equally as much as my teammates.

……..

Ismaeel Rahaman

I, Nawailah Tarmohamed, confirm that I have read the requirement changes of the Platinum Island Resort and agree to all of the information contained in this document. I confirm that the information stated above is factual and that I am aware of any and all changes made to this document between drafts. Furthermore, I confirm that I have contributed to this document equally as much as my teammates.

……..

Nawailah Tarmohamed

I, Deshlan Pillay, confirm that I have read the requirement changes of the Platinum Island Resort and agree to all of the information contained in this document. I confirm that the information stated above is factual and that I am aware of any and all changes made to this document between drafts. Furthermore, I confirm that I have contributed to this document equally as much as my teammates.

……..

Deshlan Pillay

I, Sameer Ghela, confirm that I have read the requirement changes of the Platinum Island Resort and agree to all of the information contained in this document. I confirm that the information stated above is factual and that I am aware of any and all changes made to this document between drafts. Furthermore, I confirm that I have contributed to this document equally as much as my teammates.

……..

Sameer Ghela

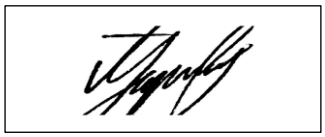
I, Sashin Gounden, confirm that I have read the requirement changes of the Platinum Island Resort and agree to all of the information contained in this document. I confirm that the information stated above is factual and that I am aware of any and all changes made to this document between drafts. Furthermore, I confirm that I have contributed to this document equally as much as my teammates.

……..

Sashin Gounden

5. Client sign-off

I Ya'qoob Tayob, on behalf of the Platinum Island resort, acknowledge that I have received and reviewed the work and confirm that it is up to quality standards.



Signature