Booking Management System

Lakeside Escapes

Contents

Background

The company I have been tasked with introducing an IT system into is Lakeside Lodge which is known to be a luxury hotel set in the vast lakeside estate. Lakeside lodge is privately owned by the Scott family and has been for many generations under their ownership the firm has won many prestigious awards and become renowned for its high-end facilities. The lakeside lodge offers many experiences for its guests such as a luxury spa an indoor pool and high-quality cuisine serving as a complete package for a great "get away". However, over recent years, the luxury hotel has had to cope with rising costs and the significant impacts of the recent pandemic. This has resulted in bookings falling dramatically which has led to a worrying decrease in the firm's annual profits. To counteract the recent circumstances the current business owners have made strategic business decisions to attract more customers in which they have developed "an entire holiday experience" held the Lakeside Lodge estate.

It has become evident that, managing the entire provision at lakeside escapes will be a challenging undertaking that will require a complete revision of the administrative systems and techniques. The system required must be fully integrated and be built into the separate developments of the Lodge this will allow the business to run at maximum efficiency and cut down on any discrepancies in the guest's timetables. The owner recognizes that any administrative deficiencies can be severely detrimental to the business and cause a loss in customer activity, considering this the system will have to be robust and comprehensive.

The area of the system I hope to design for the Lakeside Lodge is the Pod Booking microsystem. The key functions of this area of the business include tracking how long a customer chose to stay in a pod, ensuring their deposit is paid, applying any discounts that the customer is eligible for and recording the necessary data in the booking diary. My solution for the business will require the creation of a multi-faceted relational database that will be comprehensive enough to ensure adequate data is stored on each customer to maximize their experience while being user friendly enough for minimal staff training requirements to operate. This will be undertaken by through planning with some self-management techniques employed such as a developer diary paired with up-to-date research on the best practices for database creation. My solution will be based on the Microsoft Windows Forms App specifically the .NET framework, that will allow me to develop a fully functional relational database with a sleek user interface to be employed by the Lakeside Lodge.

Project Management Techniques

Project management techniques are useful addition to any project as they ensure that time and resources are not wasted during the development process. The project management techniques that I will research and consider for this application are Gantt and Pert charts. These techniques have been chosen as they are the most widely used among the IT industry and provide the best insight into efficient project development.

Gantt chart

A Gantt Chart visually displays the correlation between activities and their respective timelines. The management timelines and tasks are converted into horizontal bars (also called Gantt bars) to form a bar chart. Gantt charts aid the project management process as they can illustrate the development processes and timelines simplifying complex projects. The simplification of the project will allow for teams working on its development to better plan for deadlines and make sure that the proper amount of resources are allocated to specific project areas.

Advantages	Disadvantages
Visual Representation: Gantt charts provide a clear and simple visual representation of project tasks, timelines, and dependencies. **Task Order**: They allow project managers to order tasks effectively ensuring that tasks are completed in the correct order to prevent bottlenecks and delays. **Resource Allocation:** Gantt charts help in allocating resources efficiently by visualizing task durations and resource requirements. **Communication and Coordination:** Gantt charts facilitate communication and coordination among team members by providing a shared understanding of project timelines and deadlines.	**Complexity in Large Projects:** Gantt charts can become complex and difficult to manage in larger projects with many tasks, making them less effective. **Limited Flexibility:** Gantt charts may lack flexibility when accommodating changes or unexpected delays in project timelines (especially when agile development methodologies are employed). **Resource Management Challenges:** Managing Resources between tasks can be challenging, especially in projects with complex tasks, leading to potential inaccuracies in scheduling. **Time-Consuming to Maintain:** Maintaining and updating Gantt charts can be time-consuming, especially when changes occur frequently this man prevent accuracy and relevance.

PERT Charts

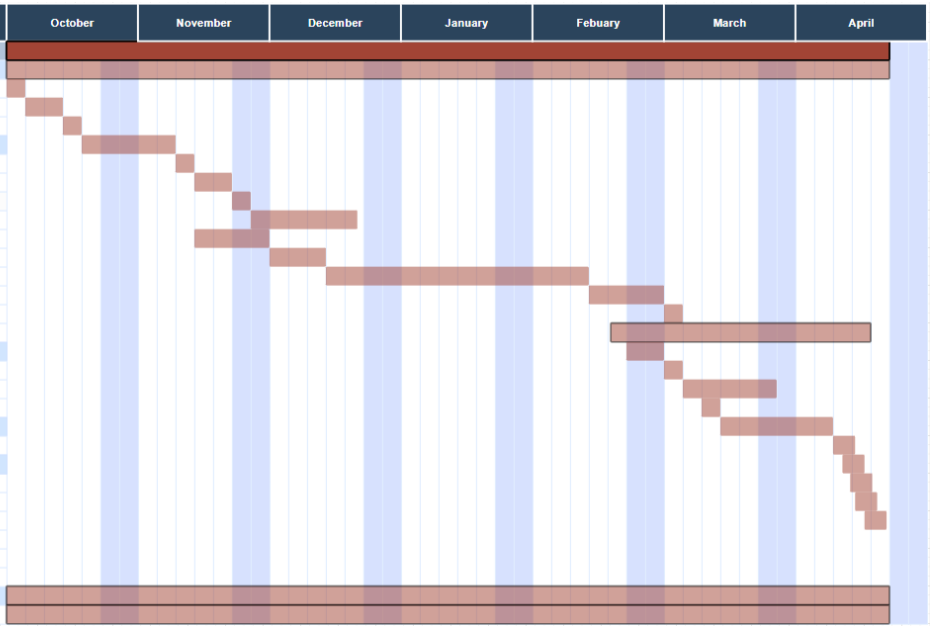
A PERT Chart is a project management tool used to schedule, organize, and map out tasks within a project. With PERT standing for Program Evaluation Review Technique it was originally used by the US Navy. A PERT Chart uses nodes (drawn as rectangles or circles) to represent events and milestones throughout the project. The nodes are connected by vectors (drawn as lines) which represent the various tasks that need to be completed. Each node will also contain the length of time that task will take to complete ensuring a clear visual representation of a project to its team and managers.

Advantages	Disadvantages
Visual Representation of a project: PERT charts provide a visual representation of the flow of a project, including tasks and milestones. **Identifies Critical Path:** PERT charts highlight the critical path, which is the sequence of tasks that determines the minimum duration required to complete the project. Identifying the critical path allows project managers to focus on the most time-sensitive tasks and allocate resources effectively. **Estimation of Project Duration:** PERT charts incorporate time estimates for each task, allowing for a more accurate understanding of the project's duration. This approach considers uncertainties and variations in task durations allowing for more accurate project timelines. **Facilitates Coordination and Communication:** PERT charts promote coordination and communication among project managers and team members by providing a clear view of project tasks and timelines.	**Difficult to setup:** Setting up a PERT chart can be complex and time-consuming, especially for large projects. It requires careful identification of tasks and accurate time estimates to be effective **Maintenance Challenges:** PERT charts require regular maintenance and updates to reflect changes in project schedules or timelines. Keeping the chart up-to-date can be challenging, particularly when there are frequent changes or unexpected delays, such as when using agile development methodologies. **Estimates have to be accurate:** PERT charts rely on time estimates for each task, which may not always be accurate. Inaccurate estimates can lead to set backs from the planned schedule and affect the success of the project. **Less focus on Resource Allocation**: PERT charts primarily focus on task order and project timelines instead of resource allocation and utilization. This may result in resource shortages during project development.

Selected Chart

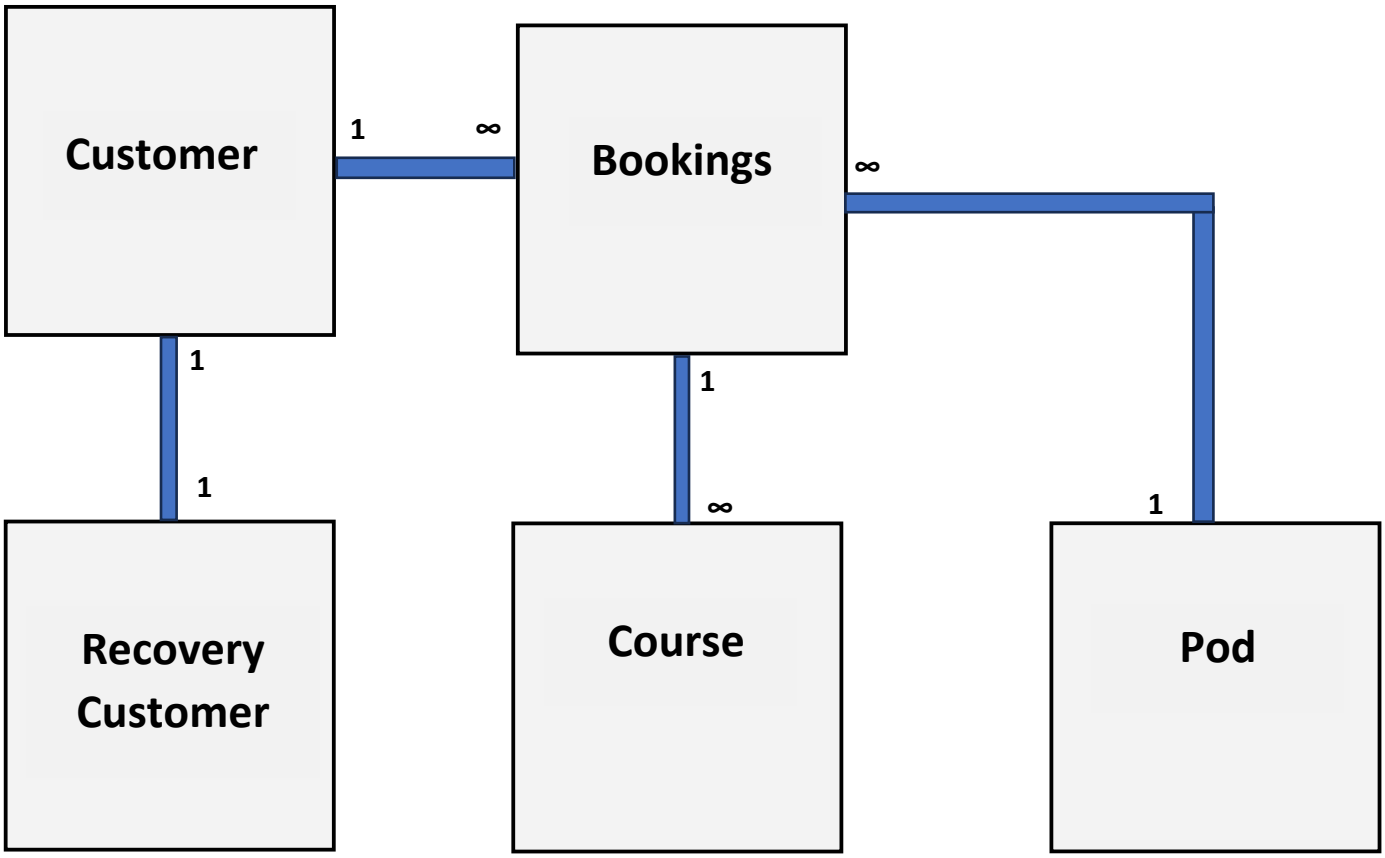
The Gantt chart has been selected for the documentation process of this project. After my research it became evident that a Gantt chart would be a useful tool to keep the development of the LakeSide Escapes Pod Booking System on track and on time. This is because of its simple and intuitive nature and that it is a less time consuming chart to develop. It is easier to make amendments to this chart when compared to the PERT chart, this becomes especially useful as this system will be created using an agile development methodology.

	Task Name	Duration	Start	ETA
1	Complete project execution	276 days	16.09.23	1.03.24
2	Planning and research	35 days	16.04.23	1.06.23
3	Project Management Techniques	1 day	16.04.23	16.04.23
4	Case Study Overview	2 days	17.04.23	18.04.23
5	User Requirements	1 day	19.04.23	19.04.23
6	Design	3 days	20.04.23	24.04.23
7	Range of Possible Solutions	27 days	25.04.23	31.05.23
8	Entity Relationship Diagram	1 day	1.06.23	1.06.23
9	Normalisation	20 days	24.04.23	21.05.23
10	Data Dictionary	2 days	24.04.23	25.04.23
11	Use Case Diagram	2 days	26.04.23	27.04.23
12	Class Diagram	3 days	30.04.23	2.05.23
13	Storyboards	10 days	3.05.23	16.05.23
14	Walkthrough/Pseudocode	2 days	17.05.23	18.05.23
15	Test Plan	1 day	21.05.23	21.05.23
16	Pseudocode	31 days	19.04.23	31.05.23
17	Testing	28 days	19.04.23	27.04.23
18	Unit Testing	11 days	30.04.23	3.05.23
19	System Testing	9 days	4.05.23	22.05.23
20	User Acceptance Testing	8 day	23.05.23	23.05.23
21	User guide	7 days	24.05.23	29.05.23
22	Creating User Guide	7 days	30.05.23	31.05.23
23	Evaluation	18 Days	24.05.23	29.05.23
24	Evaluation of user requiremnts	3 Days	24.04.23	25.04.23
25	Evaluation of Testing	3 Days	26.04.23	27.04.23
26	Evaluation of Chosen Methodology	3 Days	30.04.23	2.05.23
27	Evaluation of Project Management Techniques	3 Days	3.05.23	16.05.23
28	Evaluation of Solution	3 Days	25.04.23	31.05.23
29	Evaluation of Performance	3 Days	1.06.23	1.06.23
30	Programming Solution	150 Days	1.10.23	28.02.24
31	Programing	150 Days	1.10.23	28.02.24



Entity Relationship Diagram

An ERD (Entity Relationship Diagram) is a structural outline that displays how different entities within a system relate to one another. ERD's are vital to visualize and understand how data will be connected and are useful when modeling a relational database. They are useful for both the designer and customer as they show the defined structure and in turn functionality of the database in an easy to understand way. Entity relationship diagram's utilize images to show relationships between entities (tables) of a relational database such as numbers, symbols and connecting lines.



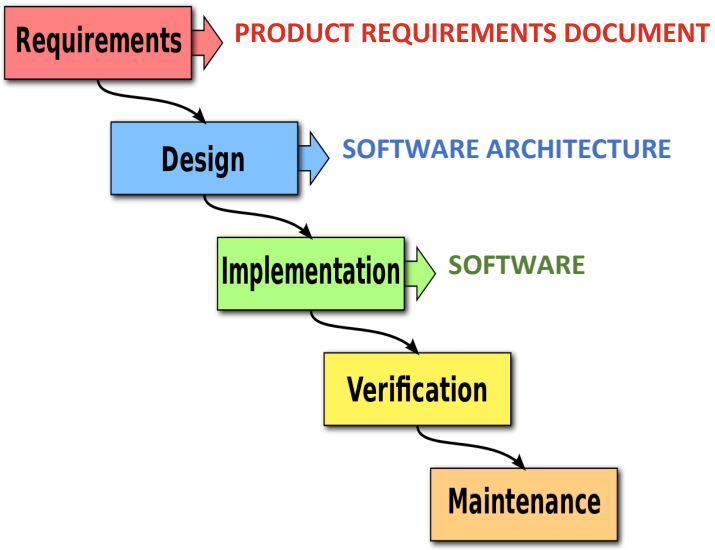
The Entity Relationship Diagram states that :

- Customers can be recovered if deleted.
- One customer can make many bookings.
- One booking will hold booking details.
- One available booking will have many booking details

Design methodologies

Design methodology refers to the development of a system or method for a unique situation. The term is most often applied to technological fields in reference to web design, software or information systems design. In relation to the development of Lakeside Escapes relational database a design methodology will involve a strategy to ensure that the project is completed within the allocated timeframe. There are two primary methods of design with some processes being agile and some being more structured.

Waterfall



The waterfall model, initially introduced by Winston W. Royce in 1970, wasn't originally labeled as such. It represented the pioneering approach publicly documented for managing software development processes. Despite early problems regarding the delayed inclusion of testing, companies readily embraced this structured methodology. The term "waterfall" was coined in a paper penned by Bell and Thayer. This method adheres to a systematic progression: analysis, design, development, and testing, with each stage concluding before the next one begins. Its efficacy hinges significantly on thorough upfront efforts, ensuring documentation for precise time estimations and the establishment of foreseeable release timelines.

However, the waterfall approach faces challenges when confronted with evolving project requirements, as its rigid structure makes adaptation less fluid compared to more agile methodologies.

Advantages:

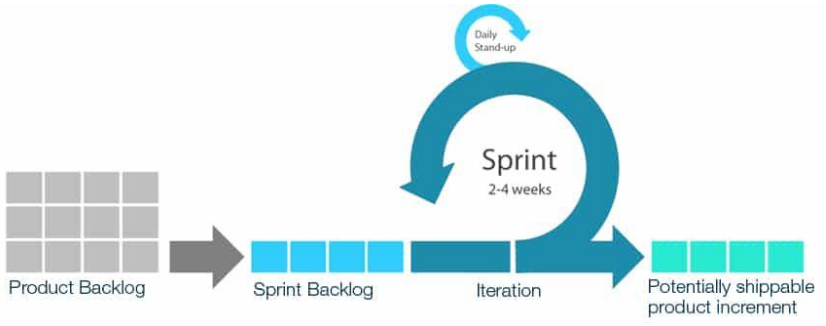
- From the start of development, the requirements are very well documented, clear and fixed.

- Before the next phase of development, the previous phase must be completed.

Disadvantages:

- It isn’t flexible enough to accommodate for a change in requirements and adjusting the scope during a life cycle can end a project.

Scrum



Jeff Sutherland played a pivotal role in the inception of the Scrum methodology in 1993. He coined the term "scrum" after seeing a paper in the Harvard Business Review from, where the author showed similarites between high-performing development teams and rugby teams utilizing the scrum formation for restarts. Scrum emerged as a response to the shortcomings of the traditional linear sequential approach known as waterfall, which was considered overly rigid and inflexible, especially when facing changing requirements or significant flaws uncovered during testing.

Scrum represents a methodology for software development rooted in iterative and incremental processes. It serves as an adaptable, swift, and flexible agile framework aimed at continuously delivering value to the customer throughout the software development lifecycle. The primary goal of Scrum is to address customer needs through the active involvement of the client alongside the development team. The development process begins with a basic concept, followed by the elaboration of a prioritized list of features and requirements, known as the product backlog.

Advantages:

- It is adaptable as it allows for a change in requirements throughout the process.

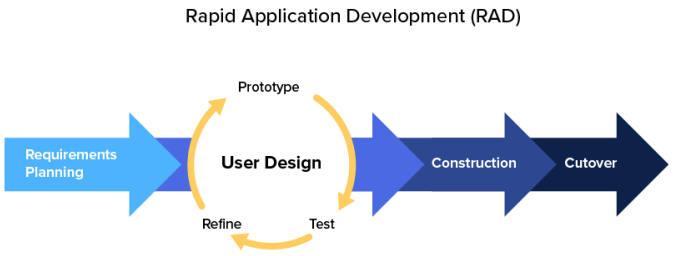
- Scrum encourages active stakeholder and Product Owner involvement throughout the development process. This transparency helps to ensure that expectations and requirements are effectively managed.

Disadvantages:

- Scrum can sometimes cause scope creep due to the lack of a definite end date. If individuals within the scrum team aren’t committed or cooperative, the chances of project failure is quite high.

- When using an Agile Methodology such as Scrum, there is very little documentation from the offset meaning that new team members will find it difficult to integrate into the project.

RAD



James Martin pioneered the Rapid Application Development (RAD) methodology during the 1980s while collaborating with IBM. His book titled "Rapid Application Development" serves as a testament to his commitment to fostering a more flexible approach to software development. Martin's motivation stemmed from the common pitfalls of linear project models, often resulting in delays or going over budget.

RAD, as articulated by James Martin, is a development lifecycle engineered to yield faster outcomes and superior quality compared to traditional methods. Its design capitalizes on cutting-edge development tools to maximize efficiency. Martin saw RAD as an agile methodology tailored to address the inherent rigidity of conventional development approaches, which struggle to accommodate changes once development initiates. Instead, RAD embraces flexibility, welcoming new inputs and features at every stage of the development journey.

Advantages:

- By using RAD, there is a quicker delivery of the product to the client or end user. It promises faster end delivery of the product due to it being highly iterative and can work towards the end goal quicker.

- RAD can accommodate changes throughout the development process. It gives a framework for midstream adjustments to be made quickly.

Disadvantages:

- It requires modularity. It can only be applied to projects that can be broken down into small components. Normally this means RAD will only work with small to medium-sized projects.

- RAD requires a team of highly skilled developers and also strong team collaboration throughout the development process.

User Requirements

Functional	
UR1	Effective programming to minimise load times and system requirements
UR2	The application must have a main menu screen
UR3	The application must use an SQL based database to store data
UR4	The application must allow users to add, edit and delete bookings
UR5	The application must allow users to add, edit and delete pods
UR6	The application must allow users to add, edit and delete courses
UR7	The application must allow users to add, edit and delete guests
UR8	The database should store guest information, including contact details and create a unique guest ID
UR9	The database should store Pod information such as price and maximum occupancy
UR10	All bookings must be visible to staff so that they understand the true occupancy of the hotel
UR11	The application must allow customers to reserve pods for specific dates, indicating the start and end of their stay
UR12	The application must perform validation checks to ensure accurate data entry, preventing errors in customer information and reservations
UR13	The application must not be vulnerable to cyber-attacks (SQL injection)
UR14	The system should automatically apply eligible discounts to the total cost of the booking reservation
UR15	The system should update pod availability preventing double booking in

	the same pod
UR16	The user must be able to close the application from anywhere in the application (And not have to use the IDES Stop function)
UR17	The system must have a reporting system in order to create structured pod, courses and monthly revenue reports, that are based on certain criteria or parameters, that must update automatically
UR18	Normalised database to prevent data redundancy.
UR19	When a guest is deleted, all associated bookings must be deleted in order to maintain data integrity
UR20	When a Pod is deleted, all associated bookings must be deleted in order to maintain data integrity
UR21	When a Course is deleted, all associated bookings must be deleted in order to maintain data integrity
Non-Functional	
UR22	Aesthetic and intuitive user interface
UR23	The application should be stable and be able to handle numerous bookings per day without crashing
UR24	The application should be reliable and if a failure would occur no data loss should result
UR25	The system should be modular and well-documented to facilitate future enhancements and maintenance

Story Boards

This is the first form that is displayed to the user when the application is run. It acts a gateway to all other aspects of the application; this is handled through the use of a panel system that populates the primary viewing panel with whatever form the user has selected to travel to. The panel system allows for the user to travel directly form one part of the application to another reducing the time spent in menus. The exit button exists as a way to stop the application from running without having to use the IDE's built in stop program function.

Booking viewer form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: "Booking Control"
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641 Colour: Gray (CustomImage.jpeg Is Loaded when program compiles)

		Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
GuestBTN	Button	Size: 136, 33 Colour: Dim Gray (Fill White) Text: Guest Outline colour: Gray
PodBTN	Button	Size: 136, 33 Colour: Dim Gray (Fill White) Text: Pod Outline colour: Gray
BookingsBTN	Button	Size: 136, 33 Colour: Dim Gray (Fill White) Text: Bookings Outline colour: Gray
CoursesBTN	Button	Size: 136, 33 Colour: Dim Gray (Fill White) Text: Courses Outline colour: Gray
HubBTN	Button	Size: 136, 33 Colour: Dim Gray (Fill White) Text: Hub Outline colour: Gray

The Booking form will handle all functions tied to the booking table within the database. This includes Adding, Editing, Deleting, Paying Deposits and Viewing bookings. The booking builder provides a visual representation of the booking to the staff and will prevent the wrong information being put into the database such as the wrong guest on a booking. The Data Grid View allows for the viewing of bookings and is more user friendly than a traditional booking diary.

LINK TO CASE STUDY OF A 'MESSY BOOKING DIARY'

Booking form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: "Booking Control"
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641

		Colour: Gray (CustomImage.jpeg Is Loaded when program compiles) Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
bookingManagerLBL	Label	Size: 402, 55 Colour: Black Text: “Booking Manager” Font Size: 36
dateOfStayLBL	Label	Size: 120, 25 Colour: Black Text: “Date Of Stay”
guestStayingLBL	Label	Size: 120, 25 Colour: Black Text: “Guest Staying”
podIDLBL	Label	Size: 120, 25 Colour: Black Text: “Pod ID”
courseIDLBL	Label	Size: 120, 25 Colour: Black Text: “Course ID”
numberOfOccupantsLBL	Label	Size: 120, 25 Colour: Black Text: “Number Of Occupants”
bookingBuilderLBL	Label	Size: 120, 25 Colour: Black Text: “Booking Builder”
booingPreviewLBL	Label	Size: 120, 25 Colour: Black Text: “”
bookingPreview2LBL	Label	Size: 120, 25 Colour: Black Text: “”
bookingPreview3LBL	Label	Size: 120, 25 Colour: Black Text: “”
selectForDepositLBL	Label	Size: 120, 25 Colour: Black Text: “Select a Booking to Pay its Deposit”

selectForDeleteLBL	Label	Size: 120, 25 Colour: Black Text: “Select a Booking to Delete”
dateOfStayDateTimePicker	DateTimePicker	Size: 120, 25 Colour: Control Text: DateTime.Now
LengthOfStayComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
guestIDComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
podIDComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
courseIDComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
noOccupantsTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
payDepositComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
deleteComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
addBTN	Button	Size: 80, 30 Colour: Gray Text: “ADD”
updateBTN	Button	Size: 80, 30 Colour: Gray Text: “UPDATE”
payDepositBTN	Button	Size: 80, 30 Colour: Gray Text: “PAY DEPOSIT”
deleteBTN	Button	Size: 80, 30 Colour: Gray Text: “DELETE”
bookingDGV	Data Grid View	Size: 960, 300 Colour: AppWorkSpace Data: BookingTable

Form1(BookingForm)	Form	Size: 960, 640 Colour: Control FormBoderStyle: None

The Guest form will handle all functions tied to the guest table within the database. This includes Adding, Editing, Deleting and Viewing Guests. The Data Grid View allows for the viewing of guests and is more user friendly than a traditional booking diary. The database will also automatically record data on if a guest has booked before, this functionality is vital as it will ensure that adequate discounts are applied which may increase sales of pods in the hotel.

LINK TO CASE STUDY HARD TO KEEP TRACK OF GUEST INFORMATION SUCH AS IF THEY HAVE STAYED WITH THE HOTEL BEFORE.

Guest form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: “Booking Control”
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641

		Colour: Gray (CustomImage.jpeg Is Loaded when program compiles) Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
firstNameLBL	Label	Size: 120, 25 Colour: Black Text: “First Name”
lastNameLBL	Label	Size: 120, 25 Colour: Black Text: “Last Name”
emailLBL	Label	Size: 120, 25 Colour: Black Text: “Email”
phonenNumberLBL	Label	Size: 120, 25 Colour: Black Text: “Phone Number”
titleLBL	Label	Size: 120, 25 Colour: Black Text: “Title”
addressLine1LBL	Label	Size: 120, 25 Colour: Black Text: “Address Line 1”
addressLine2LBL	Label	Size: 120, 25 Colour: Black Text: “Address Line 2”
selectaGuestForDeleteLBL	Label	Size: 120, 25 Colour: Black Text: “Select a Guest to Delete”
guestManagerLBL	Label	Size: 120, 25 Colour: Black Text: “Guest Manager”
addBTN	Button	Size: 80, 30 Colour: Gray Text: “ADD”
updateBTN	Button	Size: 80, 30 Colour: Gray Text: “UPDATE”
deleteBTN	Button	Size: 80, 30

		Colour: Gray Text: “DELETE”
guestDGV	Data Grid View	Size: 960, 300 Colour: AppWorkSpace Data: GuestTable
deleteComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
titleComboBox	Combo Box	Size: 200, 25 Colour: Gray Text: N/A
firstNameTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
lastNameTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
emailTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
phoneNumberTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
addresssLine1TextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
addressLine2TextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
GuestForm	Form	Size: 960, 640 Colour: Control FormBoderStyle: None

The Pod form will handle all functions tied to the Pod table within the database. This includes Adding, Editing, Deleting and Viewing Pods. The Data Grid View allows for the viewing of Pods and is more user friendly than a traditional booking diary. The program will automatically calculate the deposits and total costs of a guests booking using the information entered here such as the pods base cost.

LINK TO CASE STUDY OF POD PRICES NOT BEING READILY AVAILABLE AND ERRORS WHEN CALCULATING THE BOOKINGS OVERALL COST

Pod form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: “Booking Control”
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641 Colour: Gray (CustomImage.jpeg Is Loaded when program compiles)

		Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
addBTN	Button	Size: 80, 30 Colour: Gray Text: “ADD”
updateBTN	Button	Size: 80, 30 Colour: Gray Text: “UPDATE”
deleteBTN	Button	Size: 80, 30 Colour: Gray Text: “DELETE”
podDGV	Data Grid View	Size: 960, 300 Colour: AppWorkSpace Data: PodTable
podTypeComboBox	ComboBox	Size: 200, 25 Colour: Gray Text: N/A
podDeleteComboBox	ComboBox	Size: 200, 25 Colour: Gray Text: N/A
podCapacityTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
costPerNightTextBox	TextBox	Size: 180, 25 Colour: Control Text: N/A
PodForm	Form	Size: 960, 640 Colour: Control FormBoderStyle: None
podTypeLBL	Label	Size: 120, 25 Colour: Black Text: “Pod Type”
podCapacityLBL	Label	Size: 120, 25 Colour: Black Text: “Pod Capacity”
costPerNightLBL	Label	Size: 120, 25 Colour: Black Text: “Cost Per Night”
selectAPodTODeleteLBL	Label	Size: 120, 25 Colour: Black

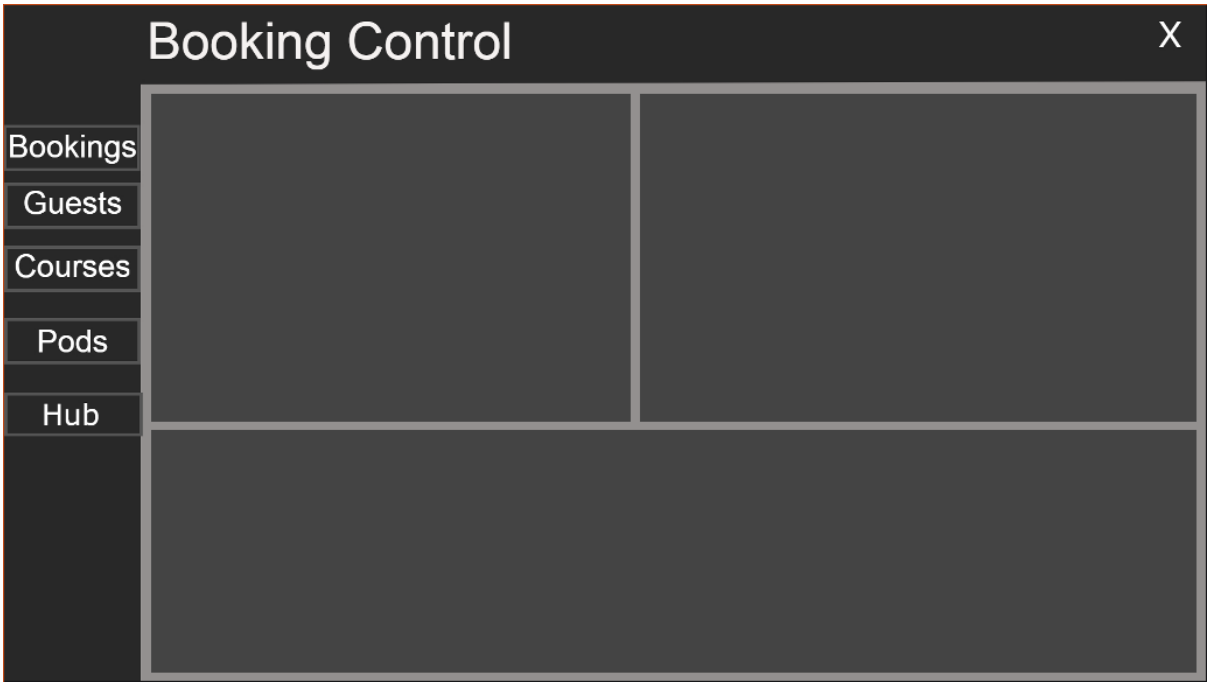
		Text: “Select a Pod to Delete”

The Course form will handle all functions tied to the Course table within the database. This includes Adding, Editing, Deleting and Viewing Courses. The Data Grid View allows for the viewing of Courses and is more user friendly than a traditional booking diary. The program will automatically calculate the deposits and total costs of a guests booking using the information entered here such as the Courses base cost.

LINK TO THE CASE STUDY OF COURSE OPTIONS BEING AVAILABLE WHEN A GUEST MAKES A BOOKING SUCH AS YOGA, POTTERY AND MEDITATION

Course form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: “Booking Control”
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641

		Colour: Gray (CustomImage.jpeg Is Loaded when program compiles) Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
addBTN	Button	Size: 80, 30 Colour: Gray Text: “ADD”
updateBTN	Button	Size: 80, 30 Colour: Gray Text: “UPDATE”
deleteBTN	Button	Size: 80, 30 Colour: Gray Text: “DELETE”
courseDGV	Data Grid View	Size: 960, 300 Colour: AppWorkSpace Data: CourseTable
CourseForm	Form	Size: 960, 640 Colour: Control FormBoderStyle: None
courseNameLBL	Label	Size: 120, 25 Colour: Black Text: “Course Name”
courseCostLBL	Label	Size: 120, 25 Colour: Black Text: “Course Cost”
selectACourseToDeleteLBL	Label	Size: 120, 25 Colour: Black Text: “Select a Course to Delete”
deleteCourseComboBox	ComboBox	Size: 200, 25 Colour: Gray Text: N/A



The hub form contains the systems reports. These reports will provide an overview to the manager Ian on how his business is preforming. The information displayed will be a breakdown of pod popularity, course popularity and revenue per month. This will allow for the proper adjustments to be made enabling the business to grow and evolve to sell more bookings by displaying what is working and what needs improvement.

LINK TO CASE STUDY OF NEEDING AN OVERVIEW OF HOW THE HOTEL IS OPERATING

Hub form		
Components	**Type**	**Properties**
BookingViewer	Form	Size: 1100, 700 Colour: White FormBorderStyle: None
PanelHeader	Panel	Size: 1100, 59 Colour: Dim Gray Text: “Booking Control”
PanelSide	Panel	Size: 136, 641 Colour: Dim Gray Text: N/A
PanelMain	Panel	Size: 964, 641

		Colour: Gray (CustomImage.jpeg Is Loaded when program compiles) Text: N/A
closeProgramLBL	Label	Size: 50, 50 Colour: White Text: “X” Font Size: 28 (Sans Serif)
HubForm	Form	Size: 960, 640 Colour: Control FormBoderStyle: None
reportViewer1	Report Viewer	Size: 1100, 59 Colour: Control Text: N/A Data: Report1
reportViewer2	Report Viewer	Size: 136, 641 Colour: Control Data: Report2 Text: N/A
reportViewer3	Report Viewer	Size: 964, 641 Colour: Control Data: Report3 Text: N/A

Reports

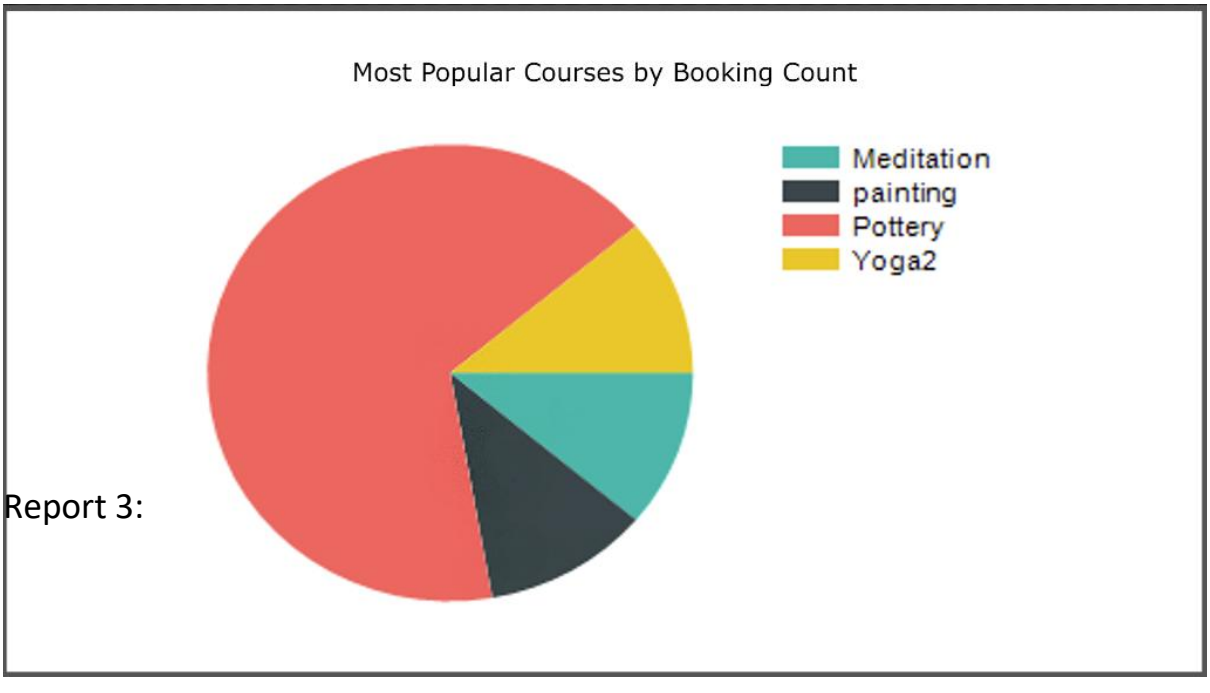
Reports display an overview of how effectively the hotel is operating. The reports will be tied to report viewers on the hub form. The information displayed will include Top 5 Most Popular Pods, Most Popular Courses and a Breakdown of Revenue per Month in 2024 (The first year this system will be integrated into the LakeSide Escapes Business). Reports off a clear and simple real-time display of how effective marketing and or research and development has been in acquiring more bookings for the hotel. The design for these reports will be shown below:

Report 1:

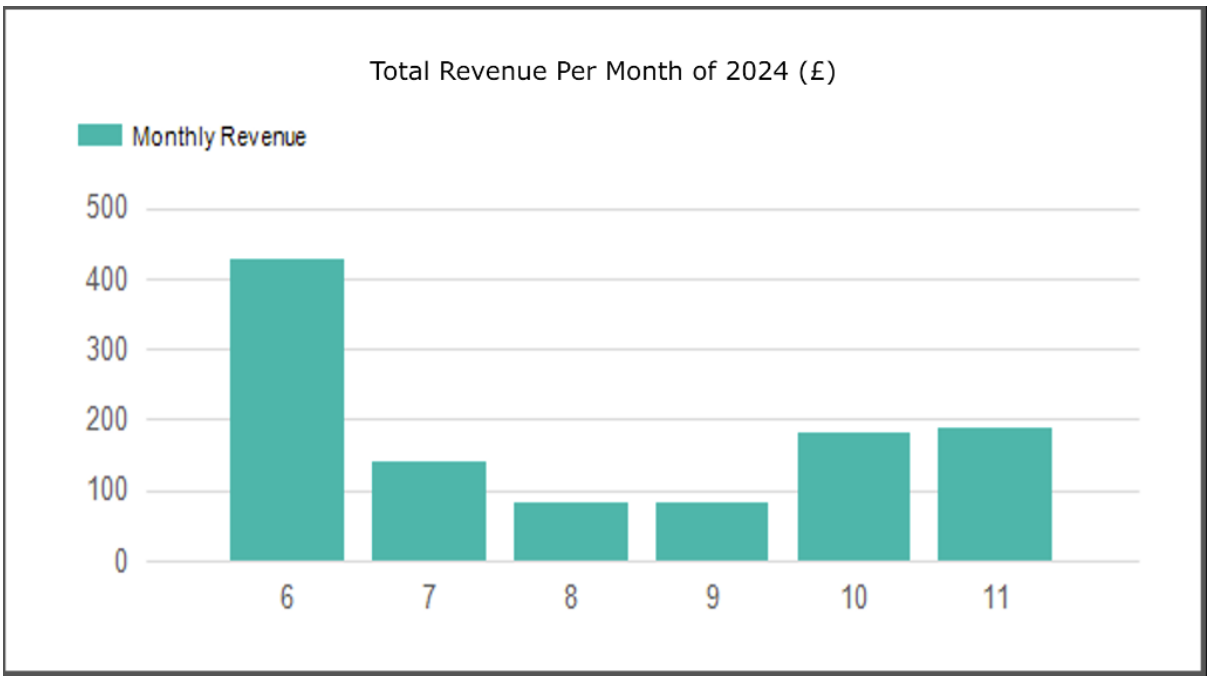
Top 5 Pods in Descending Order by Booking Count

Top 5	Pod ID	Pod Type	Capacity	Booking Count
1	3	Luxury	4	23
2	5	Luxury	4	22
3	12	Luxury	4	19
4	7	Standard	2	12
5	4	Standard	2	7

Report 2:



Report 3:


Total Revenue Per Month of 2024 (£)
Monthly Revenue
500
400
300
200
100
0
6
7
8
9
10
11

Normalisation

Database normalisation is the process of organising data within a database. Normalisation involves creating tables and curating relationships between tables to protect the data and to make the database more flexible by eliminating redundancy and improving data integrity. There are 3 defined stages of normalisation:

Unnormalized Form – Involving the atomisation of all fields, such as name to first name and last name.

1st Normal Form - Remove all repeating attributes

2nd Normal Form - Remove partial key dependencies

3rd Normal Form - Remove transitive dependencies

UNF**:** GuestID,Title, Address Line 1, Adress Line 2, PreviouslyBooked, DateBooked, FirstName, LastName, Email, PhoneNumber, PodID, PodType, Capacity, BaseCostPerNight, CourseID, CourseName, CourseCostPerPerson, BookingID, GuestID, PodID, CourseID, NumberOfOccupants, CheckInDate, CheckOutDate, BookingStatus

1NF**:**

Guest Table – GuestID, FirstName, LastName, Email, Phone, Title, Address Line 1, Address Line 2, PodID, PodType, Capacity, BaseCostPerNight,CourseID, CourseName, CourseCostPerPerson, NumberOfOccupants, CheckInDate, CheckOutDate, BookingStatus, Previously Booked, Date Booked

Booking Table - BookingID, DepositAmount, DiscountPercentage, TotalCost

2NF**:**

Guest Table - GuestID, FirstName, LastName, Email, Phone, Title, Address Line 1, Address Line 2

Pod Table - PodID, PodType (Standard, Luxury), Capacity, BaseCostPerNight

Course Table - CourseID (Primary Key), CourseName, CourseCostPerPerson

Booking Table - BookingID, #GuestID, #PodID, #CourseID, NumberOfOccupants, CheckInDate, CheckOutDate, BookingStatus (Provisional, Permanent), DepositAmount, DiscountPercentage, TotalCost, Date Booked

3NF:

Guest Table - GuestID, FirstName, LastName, Email, PhoneNumber, Title, Address Line 1, Address Line 2

Pod Table - PodID, PodType (Standard, Luxury), Capacity, BaseCostPerNight

Course Table - CourseID (Primary Key), CourseName, CourseCostPerPerson

Booking Table - BookingID, #GuestID, #PodID, #CourseID, NumberOfOccupants, CheckInDate, CheckOutDate, BookingStatus (Provisional, Permanent), DepositAmount, DiscountPercentage, TotalCost, Date Booked

Data Dictionaries

Database Name	BookingDatabase.mdf	**Table Name**	Booking	**Primary Key Field**	BookingID

Table description: Table holds all data on bookings (provisional and permanent)						
Booking						
Field Name	PK/FK	Data Type	Length	Default Value	Description	Example Data
BookingID	PK	INT Identity (1,1)		0	A number to uniquely identify a booking	1
GuestID	FK	INT		0	A number to uniquely identify a guest	1
PodID	FK	INT		0	A number to uniquely identify a pod	1
CourseID	FK	INT		0	A number to uniquely identify a course	1
NumberOfOccupants		NVAR CHAR		Null	Number of people on the booking	4
CheckInDate		DateTime		Null	The starting date of the booking	25/02/2024
CheckOutDate		DateTime		Null	The ending date of the booking	28/02/2024
BookingStatus		NVAR CHAR		Null	Identifier if the booking is provisional or permanent	Provisional
DepositAmount		NVAR CHAR		Null	Cost of the deposit for the booking	100
DiscountPercentage		NVAR CHAR		Null	Discount, if any, applied to the booking	3
TotalCost		NVAR CHAR		Null	Total cost of the booking	250
DateBooked		DateTime		Null	Date of when the booking was made	25/02/2024

Table description: Table holds all data on guests

Guest						
Field Name	PK/FK	Data Type	Length	Default Value	Description	Example Data
GuestID	PK	INT Identity (1,1)		0	A number to uniquely identify a guest	1
FirstName		NVAR CHAR		Null	A guests first name	James
LastName		NVAR CHAR		Null	A guests last name	Bond
Email		NVAR CHAR		Null	A guests email address	bond@gmail.com
Phonenumber		NVAR CHAR		Null	A guests phone number	12345678911
Title		NVAR CHAR		Null	A guests title	Mr
AddressLine1		NVAR CHAR		Null	A guests first address line	4
AddressLine2		NVAR CHAR		Null	A guests second address line	Hill Drive

Table description: Table holds all data on pods						
Pod						
Field Name	PK/FK	Data Type	Length	Default Value	Description	Example Data
PodID	PK	INT Identity (1,1)		0	A number to uniquely identify a pod	1
PodType		NVAR CHAR		Null	Type of pod (standard or luxury)	Luxury
Capacity		NVAR CHAR		Null	Capacity of the pod	6
BaseCostPerNight		NVAR CHAR		Null	Cost of the pod per night	10

Table description: Table holds all data on courses						
Course						
Field Name	PK/FK	Data Type	Length	Default Value	Description	Example Data
CourseID	PK	INT Identity (1,1)		0	A number to uniquely identify a course	1
CourseName		NVAR CHAR		Null	The name of the course	Meditation
CourseCostPerPErson		NVAR CHAR		Null	The cost of the course per person	10

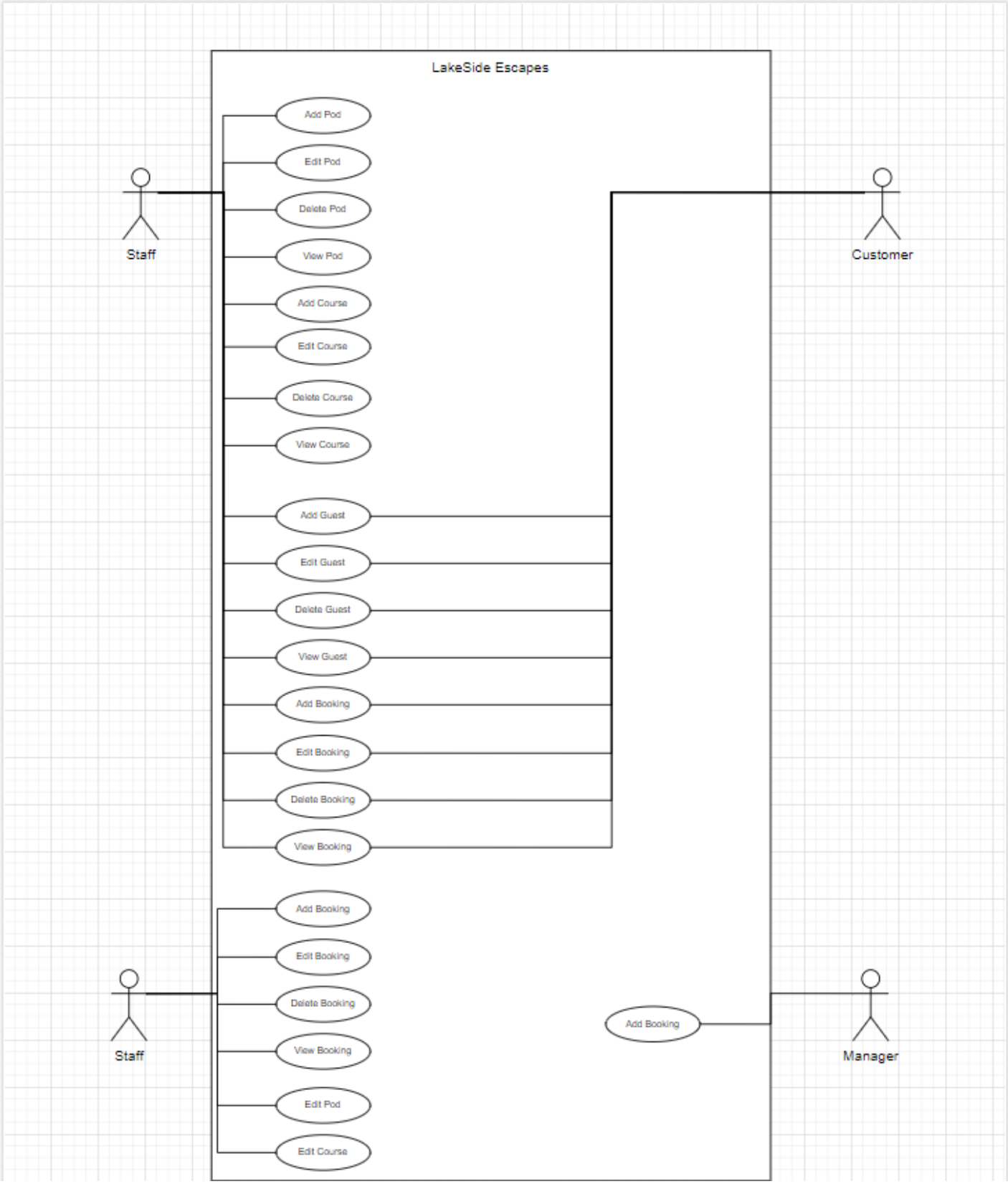
UML

UML (Unified Modelling Language) is a generalised modelling language used in software engineering. It allows for the functionalities, design and structure of a system to be visualised, allowing for easier understanding of complex programs. The two examples of UML I am using to showcase my software's functionality are Use Case Diagrams and Class Diagrams.

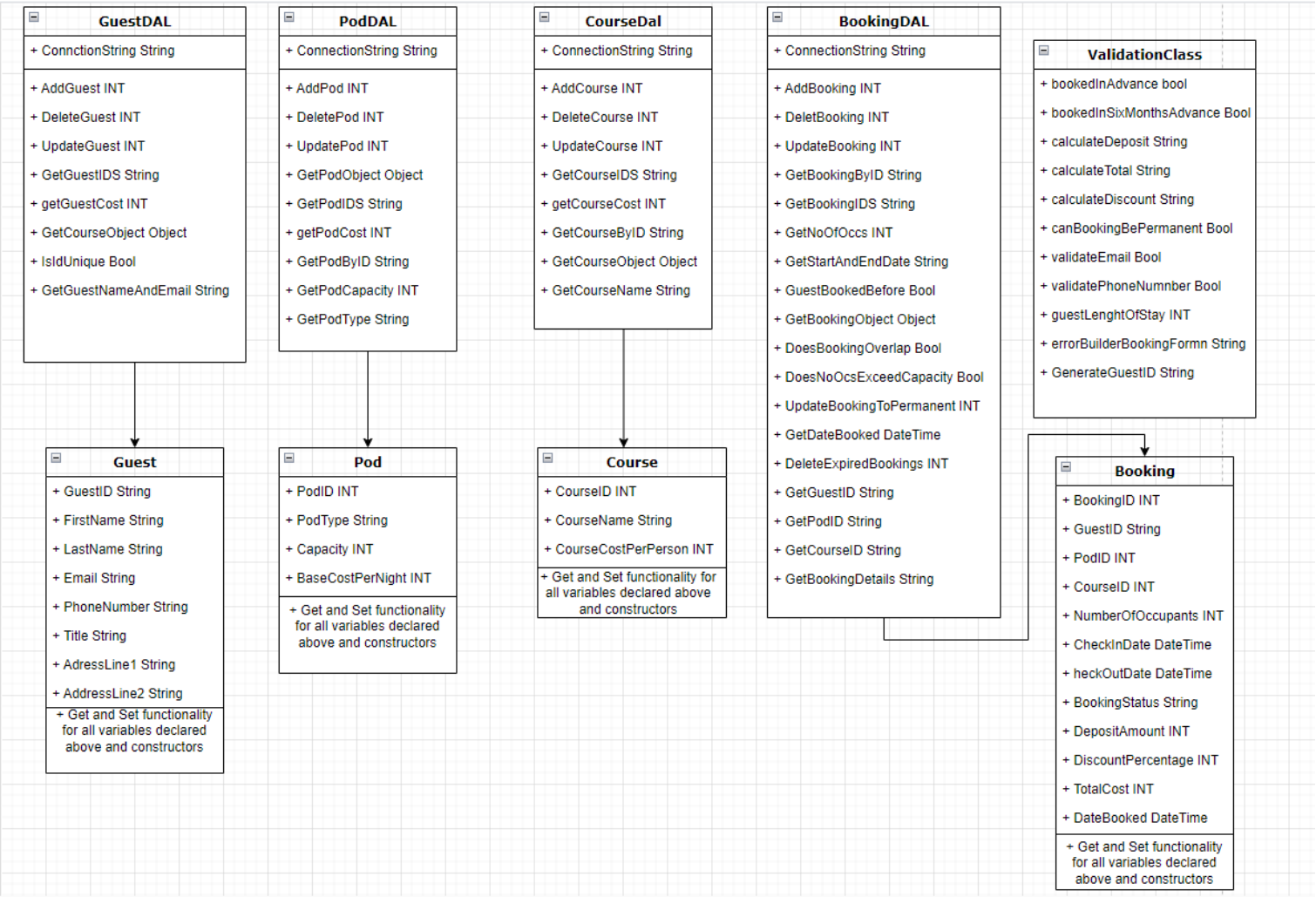
Use Case Diagrams – Through the use of this diagram, the primary functions of the system are outlined at a higher level ensuring that the uses of the program can be communicated effectively, while excluding possibly overwhelming technical jargon.

Class Diagrams – These diagrams are impactful tools that can show the models and characteristics of the program . A Class Diagram can serve as a blueprint for the structure of a system this will ensure that an technical oversights are caught before programming begins.

Use Case Diagram



Class Diagram



Range of Possible Solutions

After analysis of the Lakeside Escapes business a range of possible solutions will be examined for the development of their Booking System. Each possible solution outlines multiple advantages and disadvantages. At the conclusion of the discussion one possible solution framework will be chosen with it being most applicable to the Lakeside Escapes business.

Solution 1

C# Windows Forms application with a locally hosted SQL database

A Windows Forms based system would allow for the creation of a desktop application that will be user friendly and demand low amounts of processing power two factors that are essential for the Lakeside Escapes business as this will reduce staff training costs and business infostructure costs. A locally hosted database will also allow for the solution to be used without internet access which is necessary for a hotel located in the countryside.

The solution would be compiled on Microsoft's Visual Studio IDE as it allows for the compatibility of windows forms and an SQL database.

Advantages	Disadvantages
A locally hosted SQL Database gives the client full control over the data allowing them to change how they utilize the system. A Locally hosted database also allows for better data security. Therefore, there will be less issues raised surrounding GDPR due to the data being more protected. The intended solutions language C# is a well-documented programming language. The testing process can be simplified as the User Acceptance Testing is only for the Lakeside Escapes business allowing for both the developer and the client to save time and money. The application can be readily downloaded onto the store computer.	Updates for this type of system may have to be done locally as it can be used offline making it vulnerable to being outdated. With the database being stored locally large amounts of data may be lost if the operating machine is damaged. Costs for the business may increase as this form of solution has to be accessed on a windows machine. C# is not the fastest programming language available for this type of system. The database can also not be accessed concurrently reducing how user friendly this form of solution is.

Solution 2

Mobile application (IOS) with a locally hosted SQLite database

An IOS based mobile application with a locally hosted database would be a very user friendly and cost effective way of implementing a Booking System into the Lakeside Escapes company. An IOS based machine such as an iPad has enough processing capacity and battery life to run a booking management system while allowing for employees to show guests the hotels facilities while making a booking. The programming language used would have to be Apples proprietary language Swift.

Advantages	Disadvantages
A locally hosted SQL Database gives the client full control over the data allowing them to change how they utilize the system. A Locally hosted database also allows for better data security. Therefore, there will be less issues raised surrounding GDPR due to the data being more protected. iPads can be an inexpensive way to set up a Booking System compared to windows machines. Swift is a faster and more modern programming language than C#. iPad's are very user friendly due to touchscreen capabilities and allow for employees to show guests the hotels available facilities while making a booking.	Updates for this type of system may have to be done locally as it can be used offline making it vulnerable to being outdated. With the database being stored locally large amounts of data may be lost if the operating machine is damaged. As Swift is a very recent language it is not as well documented as C# This form of solution has to be accessed on apple devices which may be inconvenient for the business. SQLite databases can be difficult to scale which could be an issue if the company decide to expand in the future.

Solution 3

Web- based application with a hosted oracle database

A web-based application would allow for multiple employees to interact with the system concurrently which may increase sales for the business. This solution would involve Amazon Web Services (AWS) as it would host all data in an Oracle database. Controls can be tailored to fit the business's needs. Paired with, a securely hosted PHP powered web application by a third party provider that is hosted on the web. This would be accessible through any device with an internet connection.

Advantages	Disadvantages
The application is accessible by any location on any device that has an internet connection. As the application is hosted from one server updates are easier to implement allowing for this term be a longer term solution. The cloud hosted database has a great variety implementable security features and also has data recovery services. Any internet connected device can access this database making it less costly for the business as expensive machines don't have to be purchased.	A website may be excessive for the Lakeside Escapes business as the company will only have to access the database from the primary computer or till. The development process for PHP applications can be lengthy and costly due to the various testing needed for the many devices that the application would run on. Data security could be an issue if passwords are compromised due to the nature of the database being online. This solution requires constant and reliable internet access that may not always be possible for the Lakeside Escapes business as they are located in a rural area.

Chosen Solution

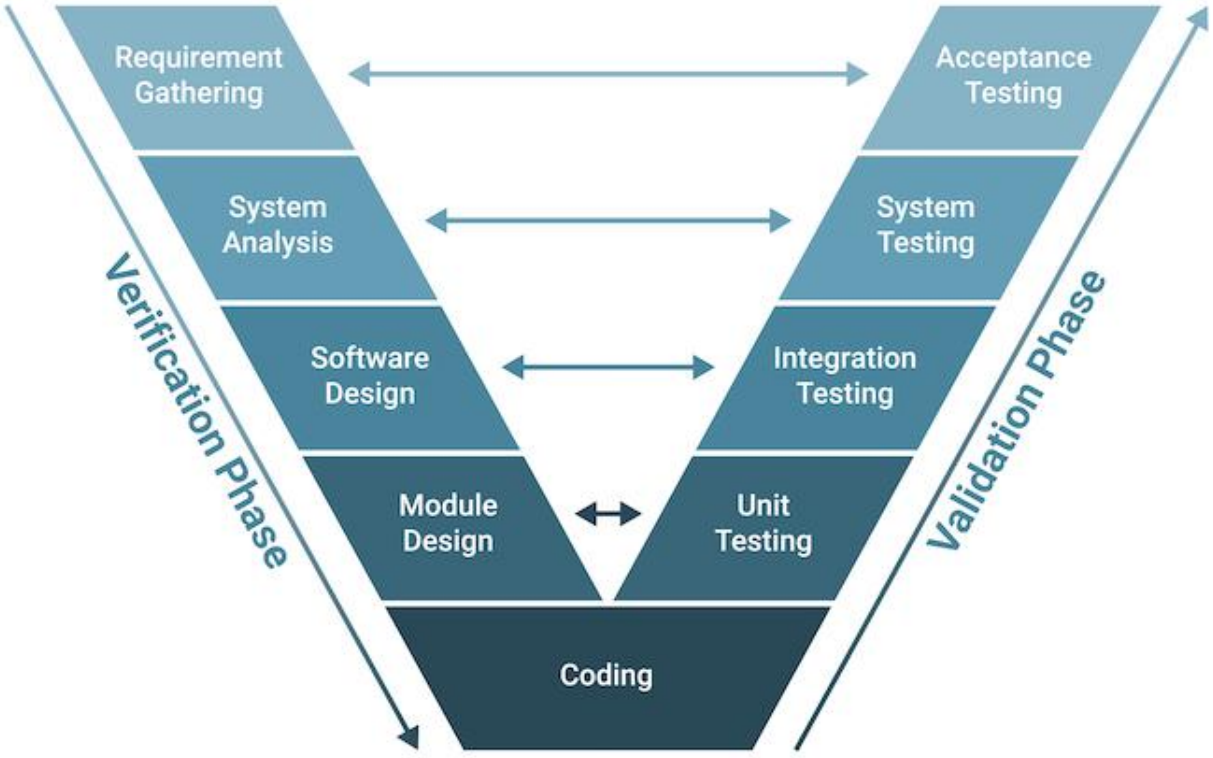
After extensive analysis of the range of possible solutions, I have decided that a C# Windows Forms application with a locally hosted SQL database will be the most suitable option. This solution will allow for the development of a robust, user friendly and secure Booking System that will enable the Lakeside Escapes company to better manage their bookings. The primary factors that where considered when choosing this solution were: the higher security nature of a locally hosted database, the ability to operate the system without an internet connection and the well documented and powerful nature of C# driven applications.

Test Plan

Thorough testing is essential to developing a software solution. Testing allows for the programmer to find and fix errors, rectify oversights in design and to ensure their solution meets the set user requirements. The importance of testing cannot be overlooked in the solution to Lakeside Escapes Booking System because guests debts would be at risk of miscalculation this would reflect negatively not only on the business but also on myself as the engineer to the solution.

The chosen test plan will be based on the Validation-model. It is the industry standard testing format for applications such as this Booking System and will provide a systematic and visual representation of the software development process. With its ability to locate errors in software at the early stages of development, I believe that it is the right choice for the development of this application.

Diagram of the Validation model:



Outline of testing

Data Entry Testing:

Data Entry Testing ensures that all data entry points within the solution only accept valid data. This is tested by entering not only valid data but also invalid data to all points within the application where the end user can enter information. With area's being tested in isolation the probability of errors occurring later in the development process is reduced. Testing like this will ensure that a Lakeside Escapes employee may not inadvertently reduce the data integrity of the database such as by using SQL characters in entered data.

Link Testing:

Link testing ensures that all modules of the application operate correctly in combination. In the case of the solution for Lakeside Escapes Booking System this form of testing will involve all the controls and user interactable functions within the program. This is to ensure that users can reach all parts of the program that they can interact with to make full use of the application. It may also involve preventing users from utilizing certain controls if they try to use them incorrectly such as adding no information into a created booking and trying to add it to the database.

User Acceptance Testing:

User Acceptance Testing ensures that all of the predefined user requirements have been met by the application. This form of testing will involve a overview of the entire system generally tested in the end users place of work on their machines. User Acceptance Testing is broken down into two parts Alpha and Beta. Alpha testing is performed by test engineers in a controlled environment where the data entered and functions used can be predetermined . Whereas Beta testing is performed by the end user in an uncontrolled environment where data entered or functions used could be unexpected this increases the likelihood of errors occurring and thus, is a vital form of testing before the final product is released.

Test Outline	**Data Entry Testing -** The following table denotes the data entry tests carried out on the 'Lakeside Escapes Booking System' application.				
Booking Form					
Test NO	**Test**	**Valid/Invalid Data**	**Test Data**	**Expected outcome**	**Actual Outcome**
1	Date of stay	Invalid	10 March 2024	Message box should appear with 'Bookings must be made 2 months in advance of the date of stay'	Function worked as intended
		Valid	26 January 2025	Message box should appear with 'Success' Booking added to database	Function worked as intended
2	Length of stay dropdown displays correct values	N/A	N/A	Four length of stay options should appear '3, 5, 7 and 14'	The combo box is not displaying '14'
3	Guest staying dropdown displays correct values	N/A	N/A	All guests in the database should be displayed	Function worked as intended
4	Pod ID dropdown displays correct values	N/A	N/A	All pods in the database should be displayed	Function worked as intended
5	Course ID dropdown displays correct values	N/A	N/A	All courses in the database should be displayed	Function worked as intended
7	Booking builder Guest label update when Guest staying dropdown value is changed	N/A	N/A	Guest information should be displayed in the booking builder	Function worked as intended
8	Booking builder Pod label update when Pod ID dropdown value is changed	N/A	N/A	Pod information should be displayed in the booking builder	Function worked as intended

9	Booking builder Course label update when Course ID dropdown value is changed	N/A	N/A	Course information should be displayed in the booking builder	Function worked as intended
10	Number of occupants has a correct value that does not exceed pod capacity or is 0	Invalid	0	Message box should appear with 'Invalid number of occupants'	Function worked as intended
		Invalid	100	Message box should appear with 'Invalid number of occupants'	Function worked as intended
		Valid	4	Message box should appear with 'Success' Booking added to database	Function worked as intended
11	ADD button click with no entered data	Invalid	N/A	Message box should appear with 'Incomplete booking information'	Function worked as intended
	Add button click with valid booking data	Valid	Date of stay: 19th June Length of stay: 3 Guest staying: BS297 Pod ID: 12 Course ID: 12 Number of Occupants: 2	Message box should appear with 'Success' Booking added to database	Program did not allow booking to be added to the database despite correct values
12	Delete booking dropdown displays correct values	N/A	N/A	All bookings in the database should be displayed in this dropdown	No bookings were displayed in this dropdown
13	Delete booking button click	N/A	Booking ID - 1	Message box should appear with 'Are you sure?' 'Yes ' ' No'	Function worked as intended
			Yes	Message box should appear with 'Successful delete' and booking	Function worked as intended

				should be deleted form database	
			No	Message box should appear with 'Delete cancelled' and booking should remain in database	Function worked as intended
14	Update button in DataGrid populates data	N/A	N/A	All booking information should be populated into corresponding textboxes and dropdowns from the selected record	Function worked as intended
15	Update button	N/A	N/A	Selected record should be updated with all the new information put in to the database	Function worked as intended
16	Pay Deposit of booking dropdown displays correct values	N/A	N/A	All bookings in the database should be displayed in this dropdown	Function worked as intended
17	Pay bookings deposit button click	N/A	Booking ID - 12	Message box should appear with "Booking is now permanent"	Function worked as intended
17	Delete booking button click	N/A	Booking ID - 1	Message box should appear with "The booking is now permanent"	Function worked as intended
Guest Form					
18	First name	Valid	Bob	Message box should appear with 'Success' and guest should be added to the data base	Function worked as intended
		Invalid	123	Message box should appear with 'Invalid guest Information'	Function worked as intended
19	Last name	Valid	Smith	Message box should appear with 'Success' and guest should be added to the data base	Function worked as intended

		Invalid	456	Message box should appear with 'Invalid guest Information'	Function worked as intended
20	Email	Valid	smith@ gmail.com	Message box should appear with 'Success' and guest should be added to the data base	Function worked as intended
		Invalid	invalidMail .com	Message box should appear with 'Invalid guest Information'	Function worked as intended
21	Phone number	Valid	566788123 12	Message box should appear with 'Success' and guest should be added to the data base	Function worked as intended
		Invalid	invalidNum	Message box should appear with 'Invalid guest Information'	Function worked as intended
22	Add button press	Invalid	No information	Message box should appear with 'Invalid guest Information'	Function worked as intended
		Invalid	Partial Information Bob Smith "" ""		
		Valid	Bob Smith smith@g mail.com 12345678910	Message box should appear with 'Success' new guest should be added to the database	
23	Delete guest dropdown displays correct values	N/A	N/A	All guests in the database should be displayed in this dropdown	Function worked as intended
24	Delete guest button click	N/A	Guest ID – BS225	Message box should appear with 'Are you sure?' 'Yes ' ' No'	Function worked as intended
			Yes	Message box should appear with 'Successful delete' and guest should be deleted form database	Function worked as intended

			No	Message box should appear with 'Delete cancelled' and guest should remain in database	Function worked as intended
25	Update button in DataGrid populates values	N/A	N/A	All guest information should be populated into corresponding textboxes from the selected record	Function worked as intended
26	Update button	N/A	N/A	Selected record should be updated with all the new information put in to the database	Function worked as intended
Pod Form					
27	Pod type dropdown	N/A	N/A	Pod type dropdown should display the two types of pods at Lakeside Escapes 'Standard' and 'Luxury'	Function worked as intended
28	Pod Capacity	Valid	4	Message box should appear with 'Success' and pod should be added to the data base	Function worked as intended
		Invalid	Four	Message box should appear with 'Invalid pod Information'	Function worked as intended
29	Pod cost/night	Valid	10	Message box should appear with 'Success' and pod should be added to the data base	Function worked as intended
		Invalid	Ten	Message box should appear with 'Invalid pod Information'	Function worked as intended
30	Add button press	Invalid Invalid Valid	No information	Message box should appear with 'Invalid pod Information'	Function worked as intended
			Partial Information Luxury "" ""	Message box should appear with 'Invalid pod Information'	Function worked as intended

			Luxury 4 100	Message box should appear with 'Success' new pod should be added to the database	Function worked as intended
31	Delete guest dropdown displays correct values	N/A	N/A	All pods in the database should be displayed in this dropdown	Function worked as intended
32	Delete pod button click	N/A	Pod ID – 4	Message box should appear with 'Are you sure?' 'Yes ' ' No'	Function worked as intended
			Yes	Message box should appear with 'Successful delete' and pod should be deleted form database	Function worked as intended
			No	Message box should appear with 'Delete cancelled' and pod should remain in database	Function worked as intended
33	Update button in DataGrid populates values	N/A	N/A	All pod information should be populated into corresponding textboxes and dropdown from the selected record	Function worked as intended
34	Update button	N/A	N/A	Selected record should be updated with all the new information put in to the database	Function worked as intended
Course Form					
35	Course name	Valid	Painting	Message box should appear with 'Success' and course should be added to the data base	Function worked as intended
		Invalid	12345	Message box should appear with 'Invalid course Information'	Function worked as intended
36	Course cost/person	Valid	10	Message box should appear with 'Success' and course should be added to the data base	Function worked as intended

		Invalid	Ten	Message box should appear with 'Invalid course Information'	Function worked as intended
37	Add button press	Invalid	No information:	Message box should appear with 'Invalid course Information'	Function worked as intended
		Invalid	Partial Information: Painting ""	Message box should appear with 'Invalid course Information'	Function worked as intended
		Valid	Painting 10	Message box should appear with 'Success' and course should be added to the data base	Function worked as intended
38	Delete course dropdown displays correct values	N/A	N/A	All pods in the database should be displayed in this dropdown	Function worked as intended
39	Delete course button click	N/A	Course ID – 3	Message box should appear with 'Are you sure?' 'Yes ' ' No'	Function worked as intended
			Yes	Message box should appear with 'Successful delete' and course should be deleted form database	The program crashed and the course was not deleted
			No	Message box should appear with 'Delete cancelled' and course should remain in database	Function worked as intended
40	Update button in DataGrid populates values	N/A	N/A	All course information should be populated into corresponding textboxes from the selected record	Function worked as intended
41	Update button	N/A	N/A	Selected record should be updated with all the new information put in to the database	Function worked as intended

Hub Form					
42	Hub Form is loaded	N/A	N/A	The three reports should be loaded into their respective report viewers reflecting the most up to date data in the database	Function worked as intended

Test Plan	**Link Testing -** The following table denotes the link tests carried out on the 'Lakeside Escapes Booking System' application.				
Booking Menu Form					
Test NO	**Test**	**Valid/Invalid Data**	**Test Data**	**Expected outcome**	**Actual Outcome**
1	Application loads	N/A	N/A	Booking Menu form loads	Function worked as intended
2	Guest button pressed	N/A	N/A	Main panel of menu form loads the guest form	Function worked as intended
3	Pod button pressed	N/A	N/A	Main panel of menu form loads the pod form	Pod Manager was not opened
4	Bookings button pressed	N/A	N/A	Main panel of menu form loads the Booking form	Function worked as intended
5	Courses button pressed	N/A	N/A	Main panel of menu form loads the Course form	Function worked as intended
6	Hub button pressed	N/A	N/A	Main panel of menu form loads the Hub form	Function worked as intended

Test Plan	**User Acceptance Testing -** The following table denotes the system tests carried out on the 'Lakeside Escapes Booking System' application.

Functional

Test NO	Test	Pass/Fail	Actual Outcome
1	Effective programming to minimise load times and system requirements	**Pass**	Function worked as intended
2	The application must have a main menu screen	**Pass**	Function worked as intended
3	The application must use an SQL based database to store data	**Pass**	Function worked as intended
4	The application must allow users to add, edit and delete bookings	**Fail**	**Booking could not be added**
5	The application must allow users to add, edit and delete pods	**Pass**	Function worked as intended
6	The application must allow users to add, edit and delete courses	**Fail**	**Courses could not be deleted**
7	The application must allow users to add, edit and delete guests	**Pass**	Function worked as intended
8	The database should store guest information, including contact details and create a unique guest ID	**Pass**	Function worked as intended
9	The database should store Pod information such as price and maximum occupancy	**Pass**	Function worked as intended
10	All bookings must be visible to staff so that they understand the true occupancy of the hotel	**Pass**	Function worked as intended
11	The application must allow customers to reserve pods for specific dates, indicating the start and end of their stay	**Pass**	Function worked as intended
12	The application must perform validation checks to ensure accurate data entry, preventing errors in customer information and reservations	**Pass**	Function worked as intended
13	The application must not be vulnerable to cyber-attacks (SQL injection)	**Pass**	Function worked as intended

14	The system should automatically apply eligible discounts to the total cost of the booking reservation	**Pass**	Function worked as intended
15	The system should update pod availability preventing double booking in the same pod	**Pass**	Function worked as intended
16	The user must be able to close the application from anywhere in the application (And not have to use the IDES Stop function)	**Pass**	Function worked as intended
17	The system must have a reporting system in order to create structured pod, courses and monthly revenue reports, that are based on certain criteria or parameters, that must update automatically	**Pass**	Function worked as intended
18	Normalised database to prevent data redundancy.	**Pass**	Function worked as intended
19	When a guest is deleted, all associated bookings must be deleted in order to maintain data integrity	**Pass**	Function worked as intended
20	When a Pod is deleted, all associated bookings must be deleted in order to maintain data integrity	**Pass**	Function worked as intended
21	When a Course is deleted, all associated bookings must be deleted in order to maintain data integrity	**Pass**	Function worked as intended
Non-Functional			
22	Aesthetic and intuitive user interface	**Pass**	Function worked as intended
23	The application should be stable and be able to handle numerous bookings per day without crashing	**Pass**	Function worked as intended

24	The application should be reliable and if a failure would occur no data loss should result	**Pass**	Function worked as intended
25	The system should be modular and well-documented to facilitate future enhancements and maintenance	**Pass**	Function worked as intended

Test Plan Results – **Data Entry**

Test NO.	Was the test Successful?	Corrective action
Booking form		
1	Yes	N/A
2	No	The combo boxes collection was updated to reflect the necessary data needed to add a booking
3	Yes	N/A
4	Yes	N/A
5	Yes	N/A
6	Yes	N/A
7	Yes	N/A
8	Yes	N/A
9	Yes	N/A
10	Yes	N/A
11	No	The stored procedure was updated to reflect changes to the SQL code that automatically generated booking id without one needing to be supplied
12	No	A method was implemented to load data into the Delete Booking Combo Box allowing the user to delete bookings
13	Yes	N/A
14	Yes	N/A
15	Yes	N/A
16	Yes	N/A
17	Yes	N/A
Pod Form		
18	Yes	N/A
19	Yes	N/A

20	Yes	N/A
21	Yes	N/A
22	Yes	N/A
23	Yes	N/A
24	Yes	N/A
25	Yes	N/A
26	Yes	N/A
Guest Form		
27	Yes	N/A
28	Yes	N/A
29	Yes	N/A
30	Yes	N/A
31	Yes	N/A
32	Yes	N/A
33	Yes	N/A
34	Yes	N/A
Course Form		
35	Yes	N/A
36	Yes	N/A
37	Yes	N/A
38	Yes	N/A
39	No	The SQL code was adjusted, specifically cascade delete functionality was added to allow for courses tied to bookings to be deleted
40	Yes	N/A
41	Yes	N/A
Hub Form		
42	Yes	N/A

Test Plan Results - LINK

Test NO.	Was the test Successful?	Corrective action
Booking form		
1	Yes	N/A
2	No	A method was added behind the click event of the Pod button facilitating user travel between the Main Menu and the Pod Manager
3	Yes	N/A
4	Yes	N/A
5	Yes	N/A
6	Yes	N/A

Test Plan Results – User Acceptance

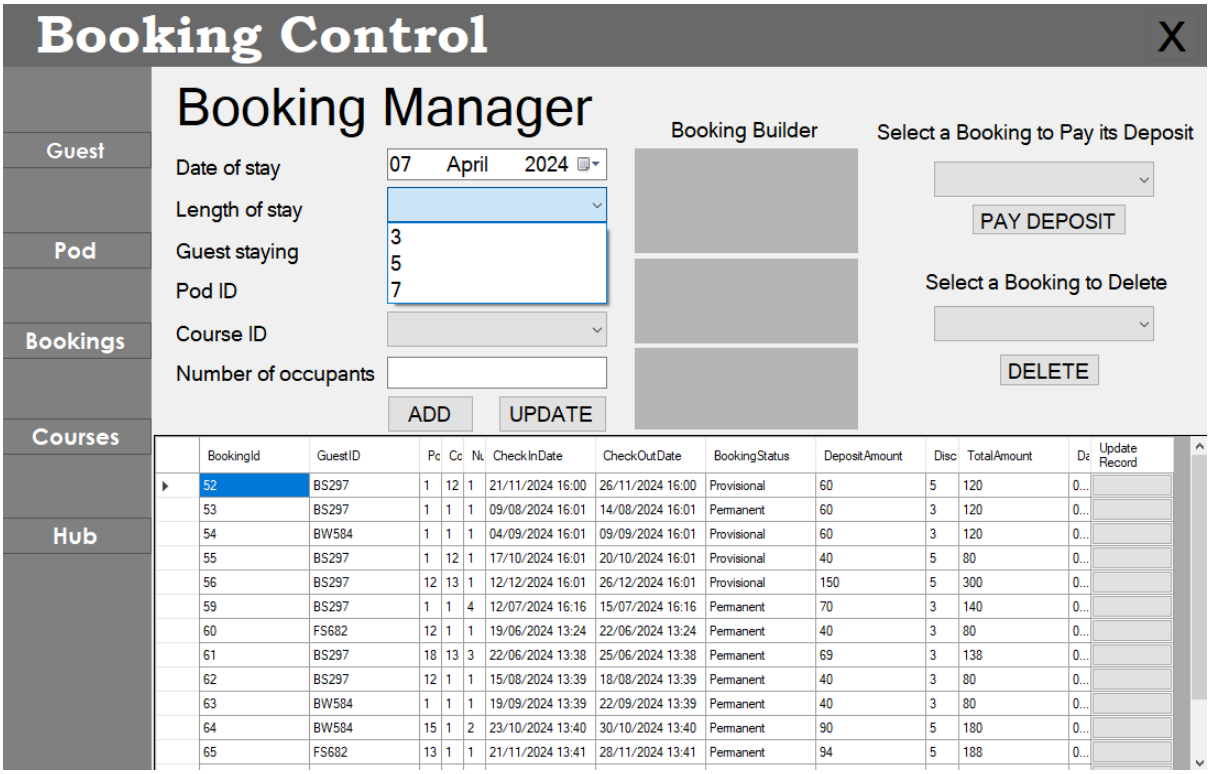
Test NO.	Was the test Successful?	Corrective action
Functional		
1	Yes	N/A
2	No	N/A
3	Yes	N/A
4	No	Action was taken to amend the booking form allowing the adding, editing and deleting of Bookings. Outlined above.
5	Yes	N/A
6	No	Action was taken to amend the booking form allowing the adding, editing and deleting of Courses. Outlined above.
7	Yes	N/A
8	Yes	N/A
9	Yes	N/A
10	Yes	N/A
11	No	N/A
12	No	N/A
13	Yes	N/A
14	Yes	N/A
15	Yes	N/A
16	Yes	N/A
17	Yes	N/A
18	Yes	N/A
19	Yes	N/A
20	Yes	N/A
21	Yes	N/A
Non-Functional		
22	Yes	N/A

23	Yes	N/A
24	Yes	N/A
25	Yes	N/A

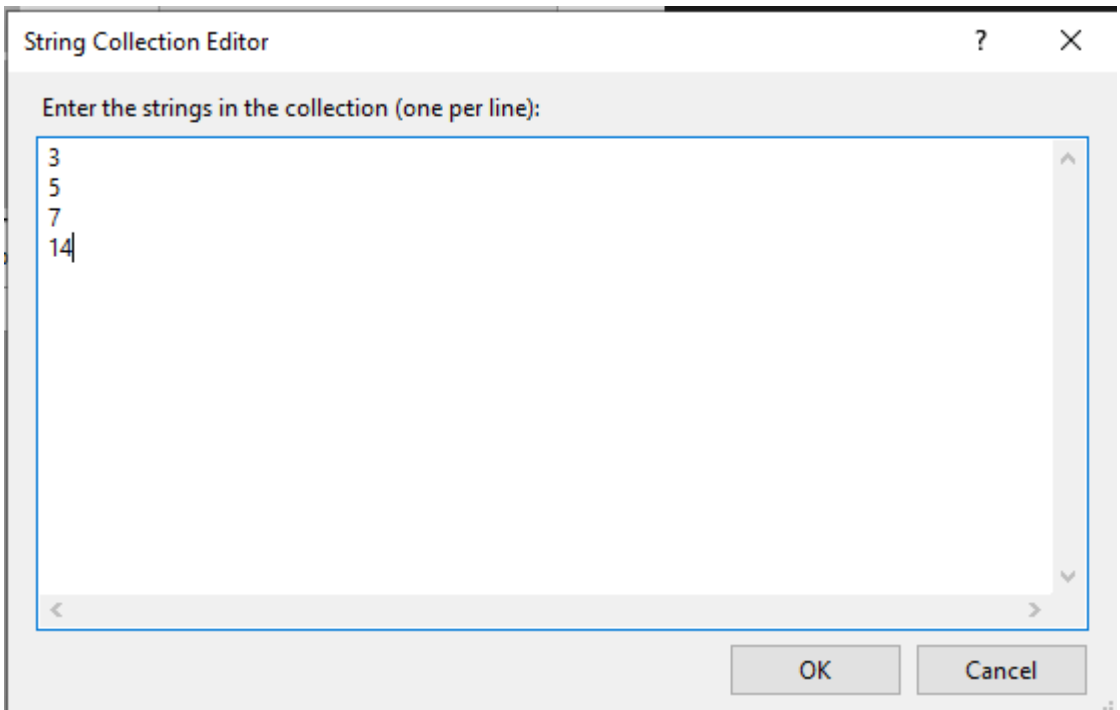
Testing Evidence

This section will include the evidence and corrective action of tests that have failed. The evidence will be displayed in the form of screen shots of the application during runtime. This information is based on the Test plan and the Test Plan results as shown above.

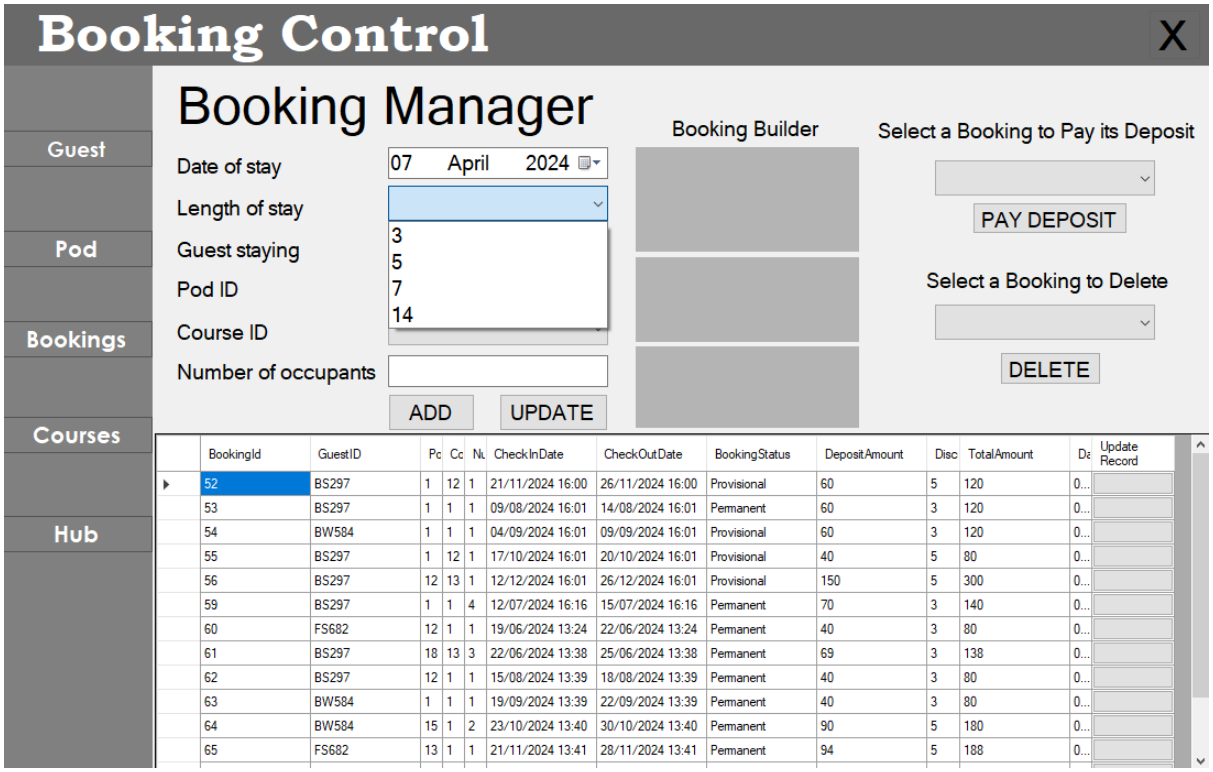
Test 2



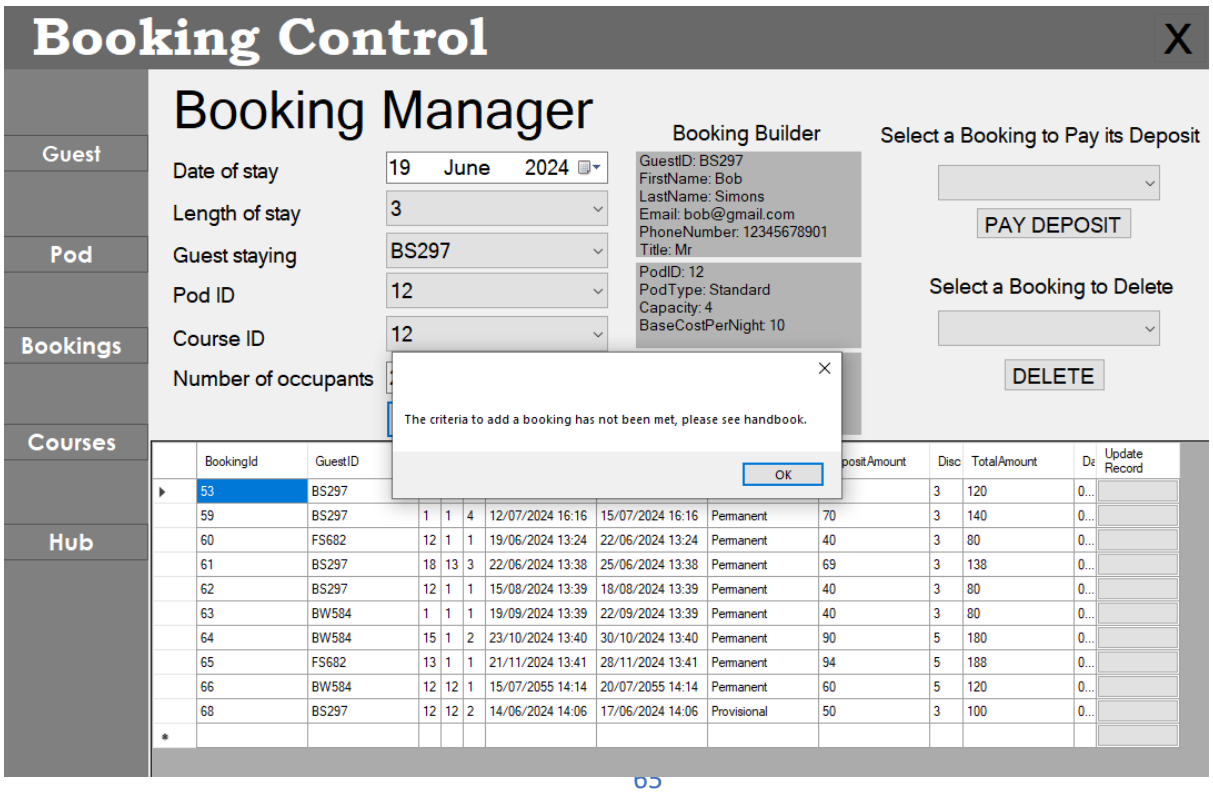
The corrective action for this error was to simply add the option of 14 into the combo boxes collection.



The corrective action has been taken and has fixed this error



Test 11



As shown above despite valid data being entered the booking was not added to the database

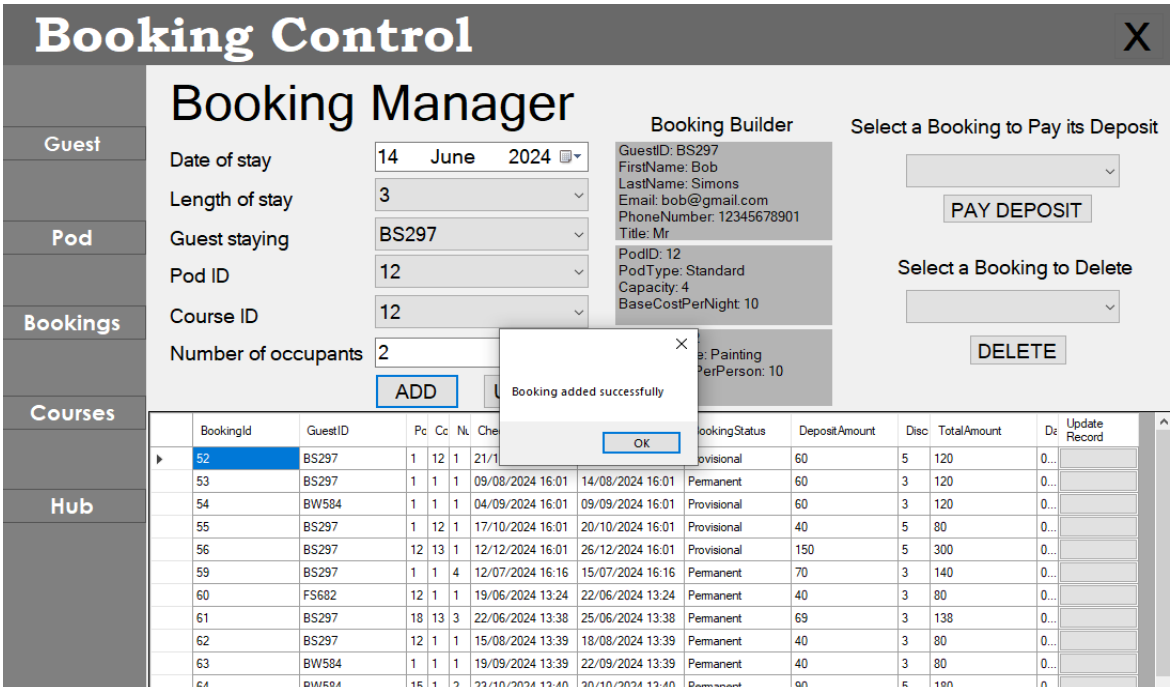
Upon further investigation it was realised that the stored procedure still had a booking id field despite the SQL auto generating this failed thus causing the error

```
Update
CREATE PROCEDURE [dbo].AddBooking
    @BookingID          INT,
    @GuestID            NVARCHAR (50),
    @PodID              NVARCHAR (50) ,
    @CourseID           NVARCHAR (50) ,
    @NumberOfOccupants  NVARCHAR (50) ,
    @CheckInDate        DATETIME      ,
    @CheckOutDate       DATETIME      ,
    @BookingStatus      NVARCHAR (50) ,
    @DepositAmount      INT           ,
    @DiscountPercentage INT           ,
    @TotalAmount        INT           ,
    @DateBooked         DATETIME
AS
     INSERT INTO Booking
     VALUES(@BookingID,@GuestID, @PodID, @CourseID, @NumberOfOccupants, @CheckInDate, @CheckOutDate, @BookingStatus, @DepositAmount, @DiscountPercentage, @TotalAmount, @DateBooked)
RETURN 0
```

The corrective action needed was to remove this booking ID field from the stored procedure

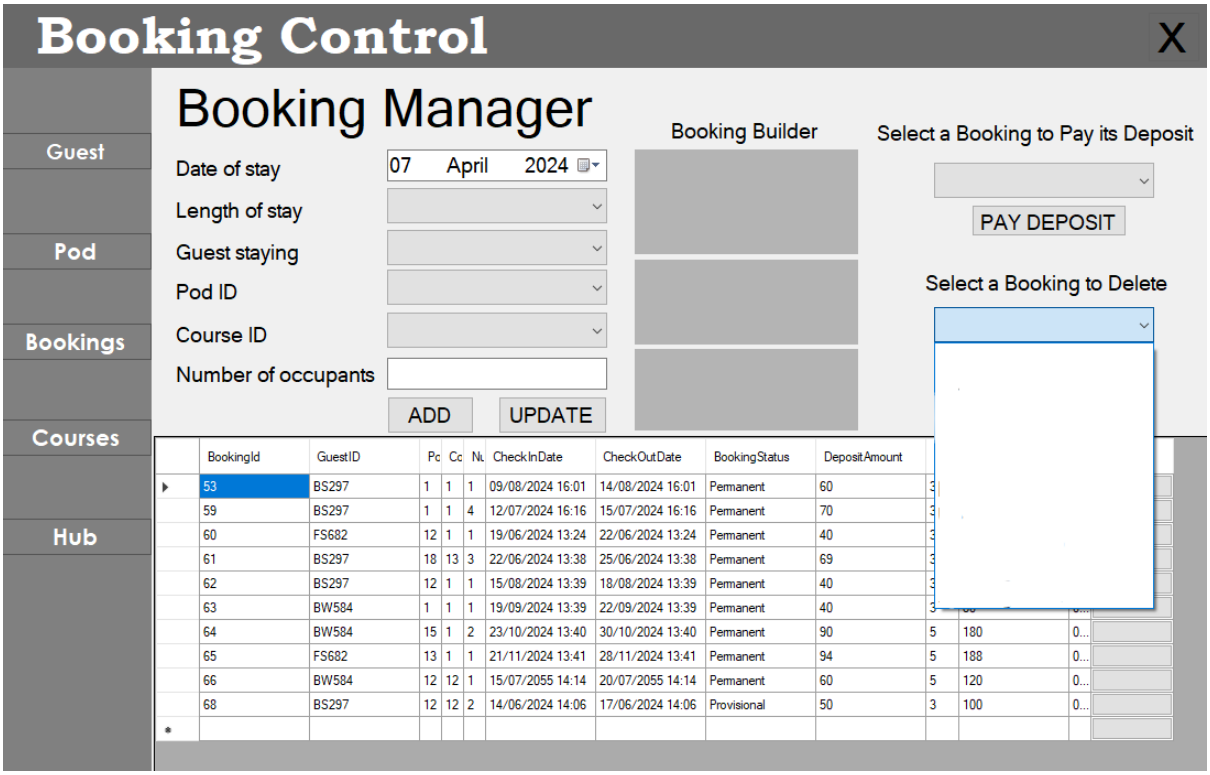
```
Update
CREATE PROCEDURE [dbo].AddBooking
    @GuestID            NVARCHAR (50),
    @PodID              NVARCHAR (50) ,
    @CourseID           NVARCHAR (50) ,
    @NumberOfOccupants  NVARCHAR (50) ,
    @CheckInDate        DATETIME      ,
    @CheckOutDate       DATETIME      ,
    @BookingStatus      NVARCHAR (50) ,
    @DepositAmount      INT           ,
    @DiscountPercentage INT           ,
    @TotalAmount        INT           ,
    @DateBooked         DATETIME
AS
     INSERT INTO Booking
     VALUES(@GuestID, @PodID, @CourseID, @NumberOfOccupants, @CheckInDate, @CheckOutDate, @BookingStatus, @DepositAmount, @DiscountPercentage, @TotalAmount, @DateBooked)
RETURN 0
```

The booking can now be added as expected



Test 12

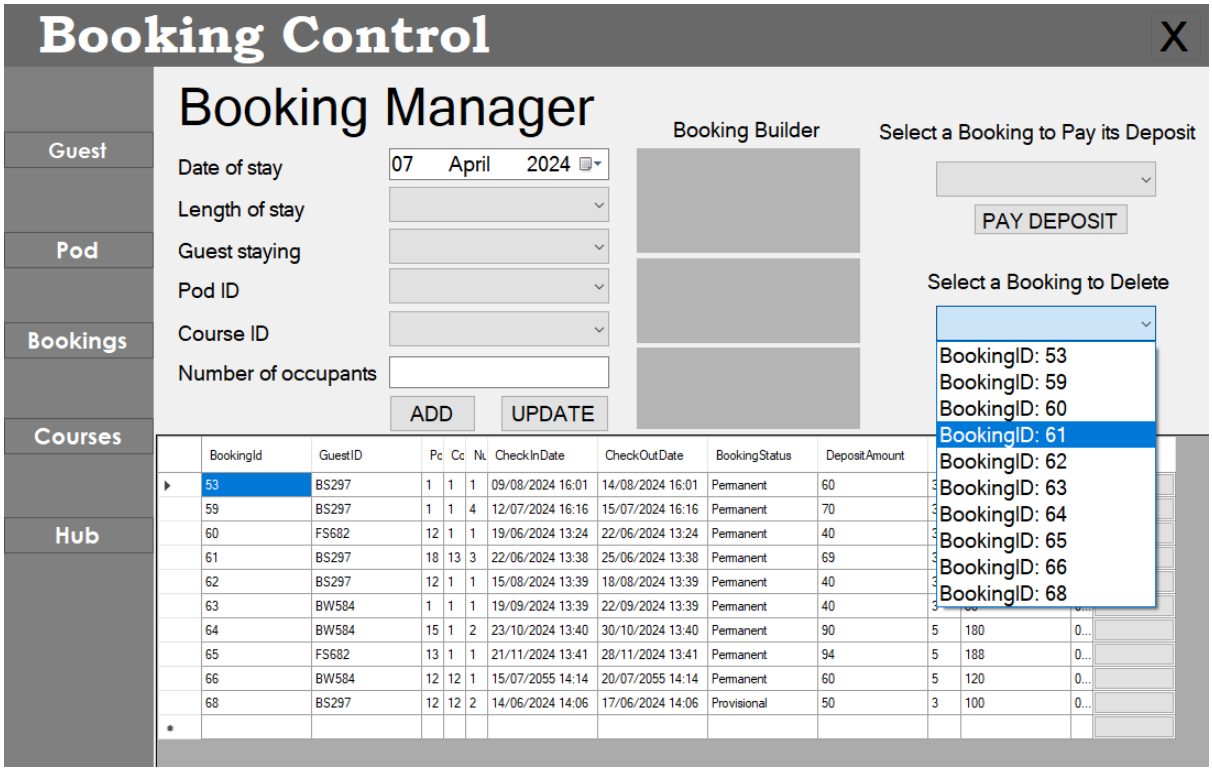
The delete booking combo box held no values which would in turn prevent a user from deleting any bookings



To fix this a method was added that would populate this combo box allowing for a user to delete a booking

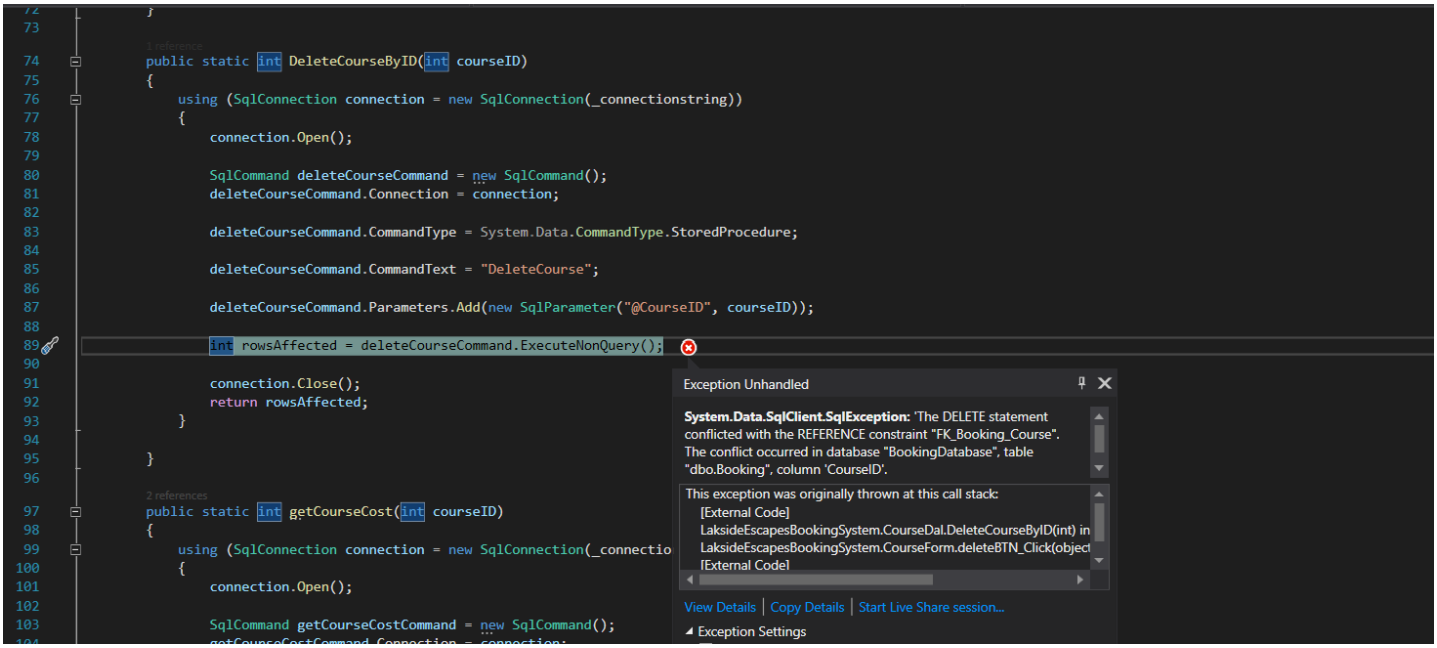
```
private void PopulateComboBoxBookingsForDelete()
{
    // Call the method to get the items
    List<string> items = BookingDal.GetBookingsIDs();

    // Clear existing items (optional)
    bookingIDsDeleteComboBox.Items.Clear();

    // Add items to the combo box
    foreach (string item in items)
    {
        bookingIDsDeleteComboBox.Items.Add(item);
    }

}
```

Delete Booking combo box after corrective action was taken



Test 39

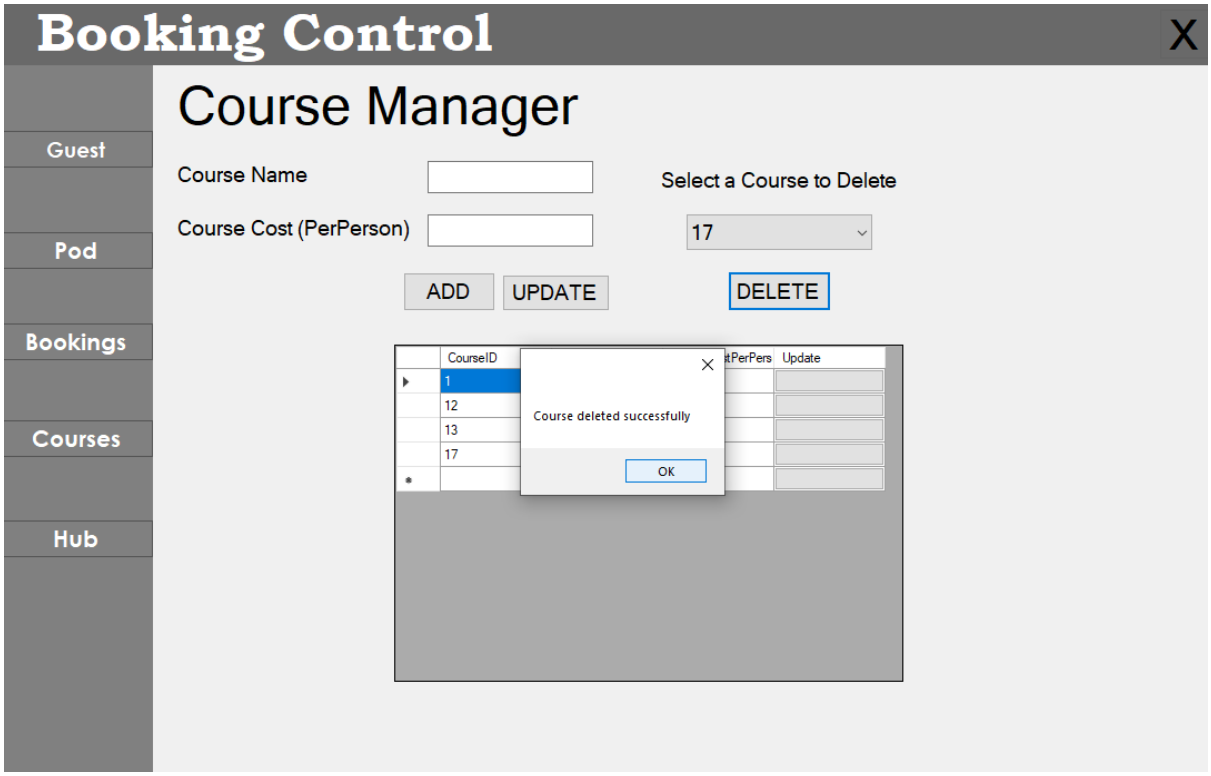
Upon attempting to delete a course that was included in any booking the user was faced with this error



Through investigation I discovered that the SQL code needed to be altered allowing for a course to cascade delete deleting all bookings tied to that course

```
PRIMARY KEY CLUSTERED ([BookingId] ASC),
CONSTRAINT [FK_Booking_Course] FOREIGN KEY ([CourseID]) REFERENCES [dbo].[Course] ([CourseID]) ON DELETE CASCADE,
CONSTRAINT [FK_Booking_Pod] FOREIGN KEY ([PodID]) REFERENCES [dbo].[Pod] ([PodID]) ON DELETE CASCADE,
CONSTRAINT [FK_Booking_Guest] FOREIGN KEY ([GuestID]) REFERENCES [dbo].[Guest] ([GuestID]) ON DELETE CASCADE
```

Course delete function working after corrective action was taken

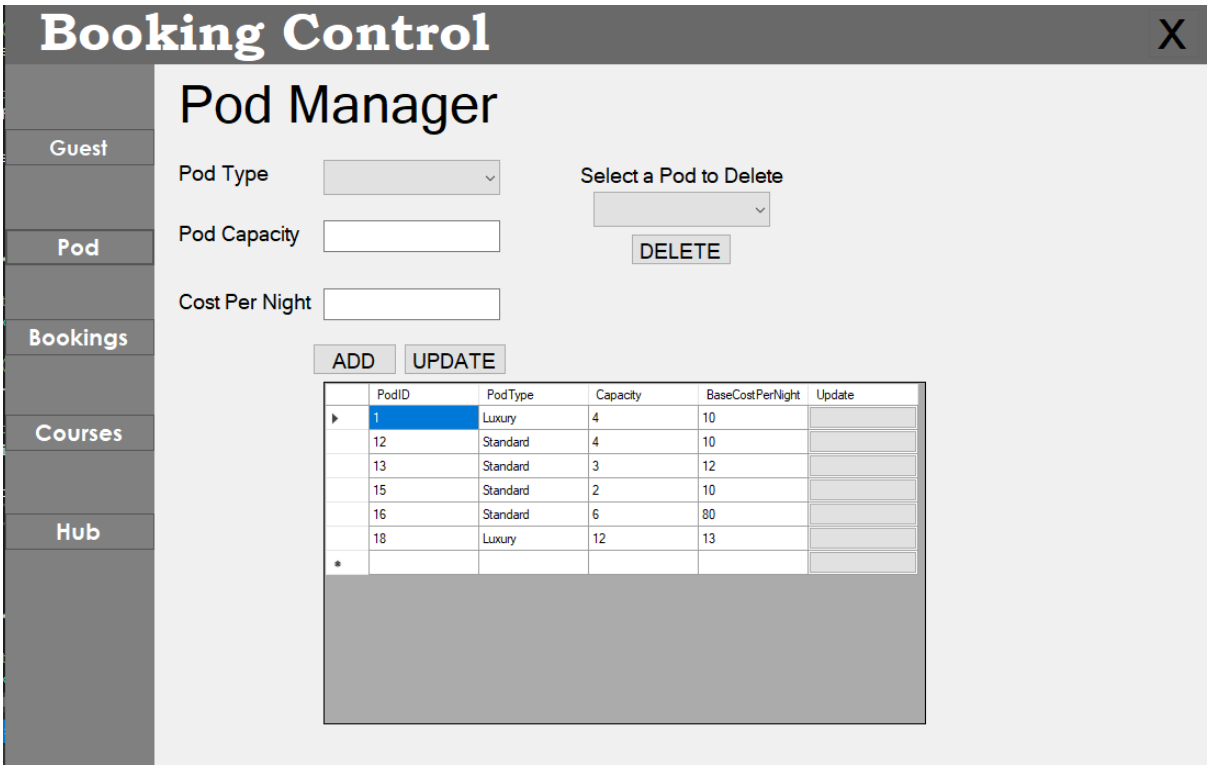


Test 2 LINK

It was discovered that in the main booking viewer menu upon pressing the pod button the program would not take the user to the Pod Manager


```
1 reference
private void podFormBTN_Click(object sender, EventArgs e)
{
    loadform(new PodForm());
}
```

The code above was entered into the click event of the Pod button



Picture above the Pod button now takes the user directly to the Pod Manager form

Pseudocode

Pseudocode is a non-technical breakdown of the primary functions and methods that I have used in my solution. This section will outline five methods including adding, updating and deleting a booking.

Displaying screens in the main menu

The below method is shows how a user's input is transformed into the program loading the desired screen. This method is found in the main menu of my solution (Booking Viewer Form).

```
openBookingScreenButtonClick
{
        mainPanel Form Close
        new Form object Booking Form
        mainPanel Data = Booking Form
        Booking Form.Show()
}
```

Adding a new booking

```
addBookingButtonClick
{
        if(BookingDataEntered is valid)
                BookingDal.AddBooking
                If(BookingAddedSuccessfully = True)
                Show Message "Booking added successfully"
                Else
                Show Message "Booking not added successfully"
}
BookingDal.AddBooking(BookingDataEntered)
{
        Open the connection to BookingDatabase
        Add Booking to database using AddBooking Stored Procedure
        Close the connection to BookingDatabase
}
```

Updating a booking

```
UpdateBookingButtonClick
{
        if(NewBookingDataEntered is valid)
                BookingDal.UpdateBooking
                If(BookingUpdatedSuccessfully = True)
                Show Message “Booking updated successfully”
                Else
                Show Message “Booking not updated successfully”
}
BookingDal.UpdateBooking(NewBookingDataEntered)
{
        Open the connection to BookingDatabase
        Update Booking to database using AddBooking Stored Procedure
        Booking is first deleted an then recreated with new information added
        Close the connection to BookingDatabase
}
```

Deleting a booking

```
DeleteBookingButtonClick
{
        if(BookingID for Delete Exists)
                BookingDal.DeleteBooking
                If(BookingDeletedSuccessfully = True)
                Show Message “Booking Deleted successfully”
                Else
                Show Message “Booking not Deleted successfully”
}
BookingDal.DeleteBooking(BookingID for Delete)
{
        Open the connection to BookingDatabase
        Find booking in the database
        Delete Booking in the database
        Close the connection to BookingDatabase
}
```

Creating a Booking Object

```
PopulateBookingInformation
{
        if(Booking information Exists)
                Information array = BookingDal.GetBookingInformation(BookingID)
                Booking object =BookingDal.CreateBookingObjectByBookingInformation (information array)
                If(Booking Object is not Null)
                Populate user controls with booking information from booking object
                Else
                Show Message “Something went wrong”
}
BookingDal.CreateBookingObjectByBookingInformation (information array)
{
        Open the connection to BookingDatabase
        Create a new instance of a booking model class

        For each(item in information array)
        {
                Add item to respective portion of booking object
        }
                Return booking object
        Close the connection to BookingDatabase
}
BookingDal.GetBookingInformation(Booking Information)
{
        Open the connection to BookingDatabase
        Find booking in the database
        Populate an array with the information
        Close the connection to BookingDatabase
}
```

Walkthrough

This section outlines the flow of program control to the new Lakeside escapes Pod Booking System. The table below shows each feature of the solution and guides the user through its functionality. For more detail see the video walkthrough.

Form	Feature	Description
Main menu (Booking Viewer)	Guest button	Takes the user to the Guest manager
	Pod button	Takes the user to the Pod manager
	Booking button	Takes the user to the Booking manager
	Course button	Takes the user to the Course manager
	Hub button	Takes the user to the Hub form
	Close solution button	Closes the solution
Guest manger menu	Data entry for guest details	Allows the user to enter guest information
	Add guest button	Adds a guest to the database
	Update guest button	Updates a guest in the database
	Delete guest button	Deletes a guest in the database
	Data grid view for guests	User can view all guests in the database
Pod manager menu	Data entry for pod details	Allows the pod to enter guest information
	Add pod button	Adds a pod to the database
	Update pod button	Updates a pod in the database
	Delete pod button	Deletes a pod in the database
	Data grid view for pods	User can view all pods in the database
Course manager menu	Data entry for course details	Allows the user to enter course information
	Add course button	Adds a course to the database
	Update course button	Updates a course in the database
	Delete course button	Deletes a course in the database

	Data grid view for course	User can view all courses in the database
Booking manager menu	Data entry for booking details	Allows the user to enter booking information
	Add booking button	Adds a booking to the database
	Update booking button	Updates a booking in the database
	Delete booking button	Deletes a booking in the database
	Pay bookings deposit button	Pays a bookings deposit making it permanent
	Data grid view for booking	User can view all bookings in the database
Hub form	Report of pods	Shows the Top 5 Most Popular Pods by Booking Count
	Report of courses	Shows the Most Popular courses by Booking Count
	Report of revenue	Shows a breakdown of revenue per month in 2024

Input/Process/Output

The Input/Process/Output model (IPO model) is a method of illustrating the structure in which information is processed within a particular system, in this case the LakeSide Escapes Booking System.

The input section is the data the system attains from the user, in the Booking System this is through text and combo boxes,. Process is when the users input data is used by the system to achieve output data. Output is the information displayed to the user as a result of the systems process.

The Booking System for LakeSide Escapes primary inputs will be guest, pod, course and booking data. With the process being the data access layers controlling the information sent to the database. The output is the reports such as the most popular courses.

The users entered data such as guest name or address.

Input
Guest details
Pod details
Course details
Booking details

The system then programmically converts that data to be able to interface in an SQL database, through Data Access Layers (DALS).

Process
Guest DAL
Pod DAL
Course DAL
Booking DAL

The user in this case a manager can view reports on the system and see which aspects of the company are preforming best.

Output
Top 5 Most Popular Pods by Booking Count - Report
Most Popular Courses by Booking Count - Report
Revenue per Month in 2024 (£) - Report

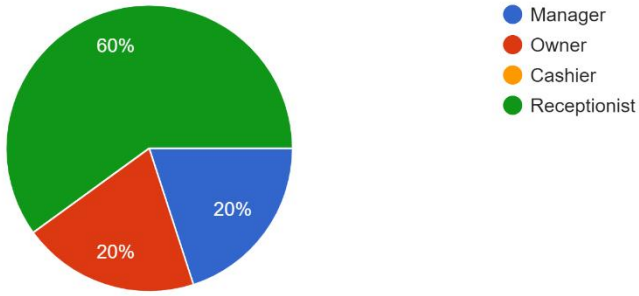
End-User Design Feedback

This section will outline the feedback of the end users on this project in this case the LakeSide Escapes staff. By gathering this information of their views and opinions of the system the developer can gain a better understanding of what may need to be done to improve the system further. The way I have chosen to gather this information is through the use of a Google Form as it is quick and easy to use.

Responses are shown below:

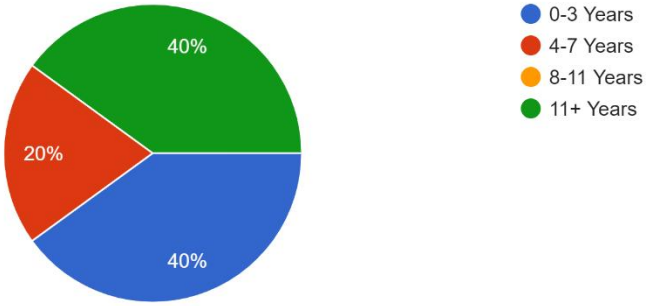
What is your position in this company?

5 responses



How long have you worked for LakeSide Escapes?

5 responses



How would you rate the current Booking System within the company?

5 responses

Have all the requirements for the system been identified within the user requirements?

5 responses

Please add any requirements you feel the system should meet below:

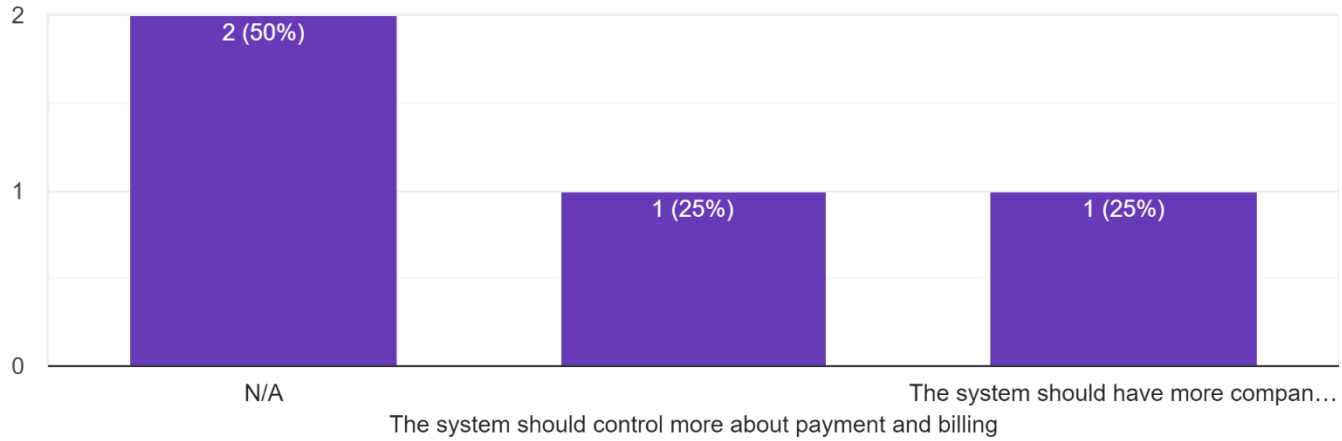
The system should control more about payment and billing

Please add any requirements you feel the system should meet below:

The system should have more company branding such as a logo

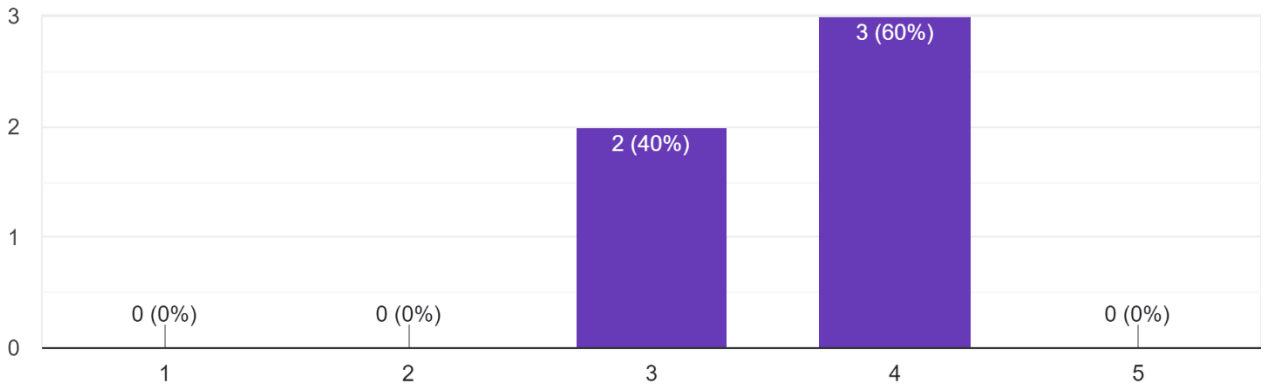
Please add any requirements you feel the system should meet below:

4 responses



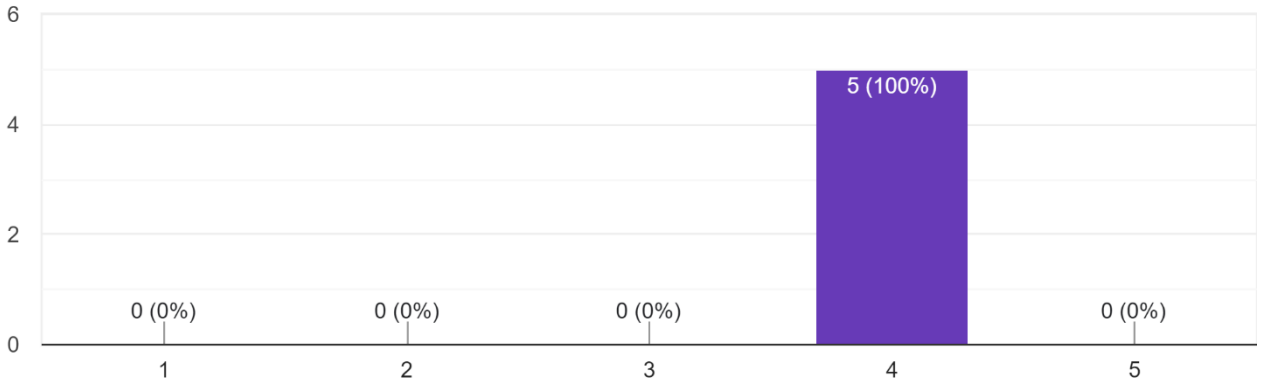
How well do you feel the User Interface is intuitive to use?

5 responses



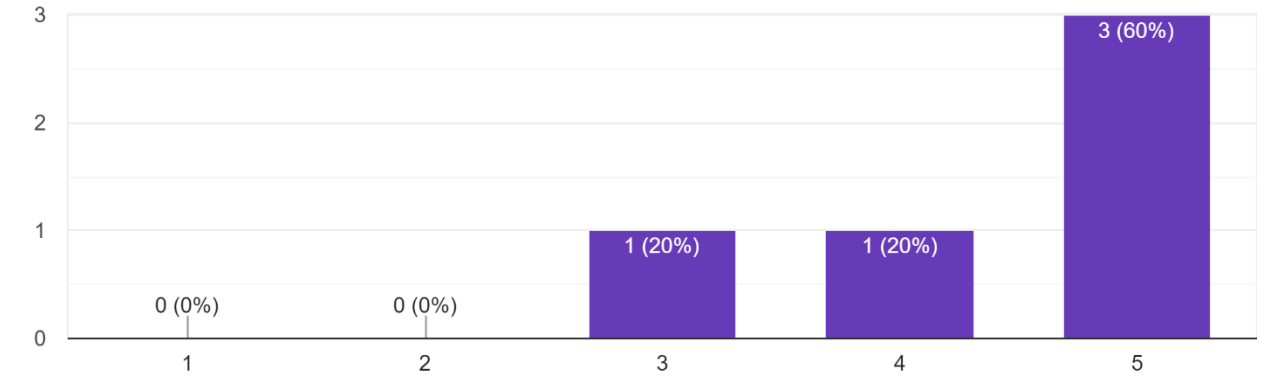
How well do you feel that the application is reliable (Minimal bugs and crashes)?

5 responses



How well do you feel that the application is optimized to run on LakeSide Escapes machines?

5 responses



How would you improve the Booking System?

5 responses

- Less cluttered screens
- Help messages would makes the system ease or to use
- A less busy booking screen would make the application much easier to use
- Better tab control for quicker adding of guests and bookings during busy times
- Increase company branding and add tooltips

If you are a manager at LakeSide Escapes, how would you improve the reporting system?

2 responses

- More reports on guests, such as guest who has made the most bookings
- Tooltips to show how to print and save the reports

Discussion of User-Feedback

This will be a summary of the above user feedback. Overall, the consensus was that all staff questioned within the LakeSide Escapes business no matter their station felt that the current system used to track and create bookings was poor. Most respondents did feal that the proposed Booking System would improve their booking capabilities, however a primary issue with the system was that the UI could be too cluttered.

The manager and owner respondents both felt like more reports could be added to the system. This feature is now planned for a future update of the Booking System.

LakeSide Escapes Booking System User Guide

This Booking System user guide will give an overview of how to navigate, control and manage the application. This guide will also contain an FAQ section paired with a video walkthrough providing further guidance on the system.

Installation

The installation process of the Booking System will be outlined in this section. While the installation process has been made as intuitive as possible to ensure proper installation follow these steps:

1. Download the provided Zipped Folder

2. Unzip the folder on your machines file explorer

3. Install the essential NuGet packages that are outlined within a text document inside the zipped folder

4. Open the file in Visual Studio

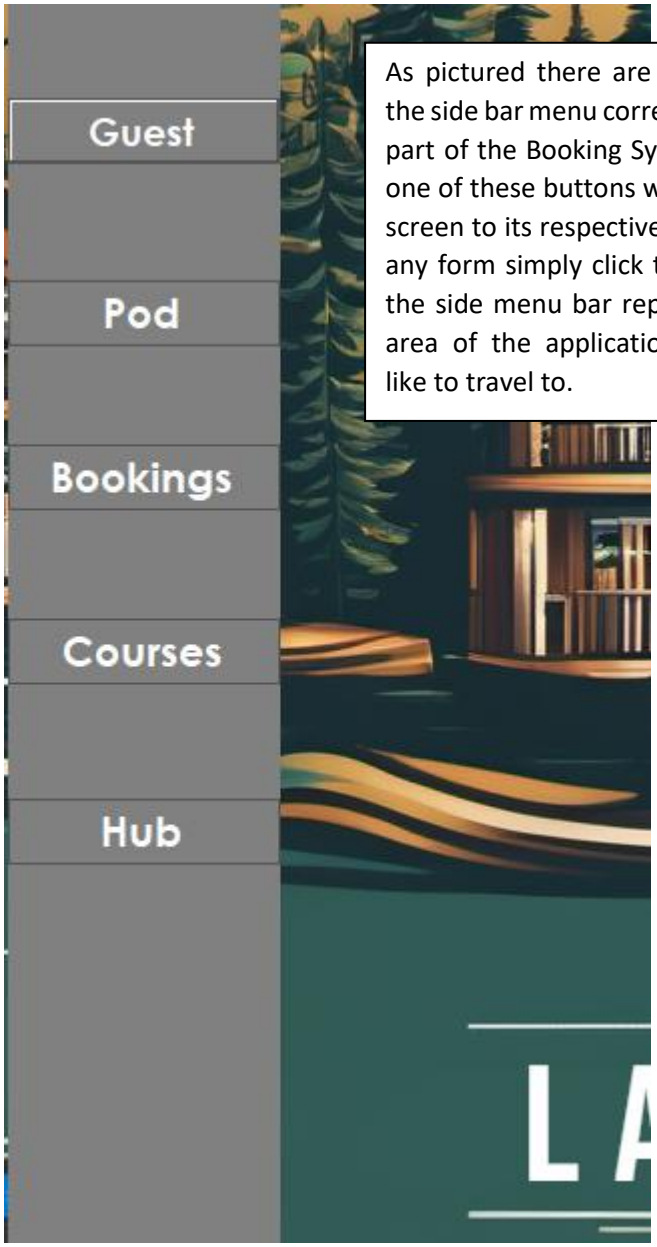
5. Click **Start** to run the Booking System

The Booking System should now be visible on your screen with full functionality in a fully operable state.

How to control the Booking System

To reach every section of the Booking System the side bar menu will be used. It is located on the left side of the screen.

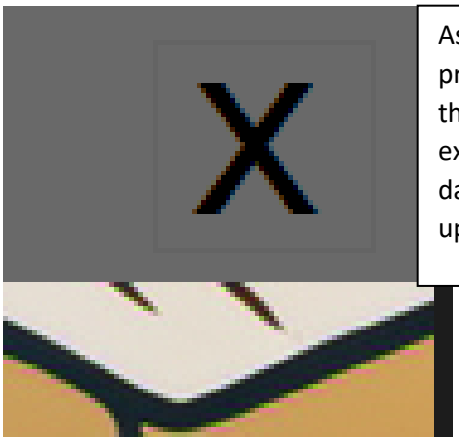
Side bar menu:



As pictured there are 5 buttons on the side bar menu correlating to each part of the Booking System. Clicking one of these buttons will change the screen to its respective form. To exit any form simply click the button on the side menu bar representing the area of the application you would like to travel to.

How to control the Booking System

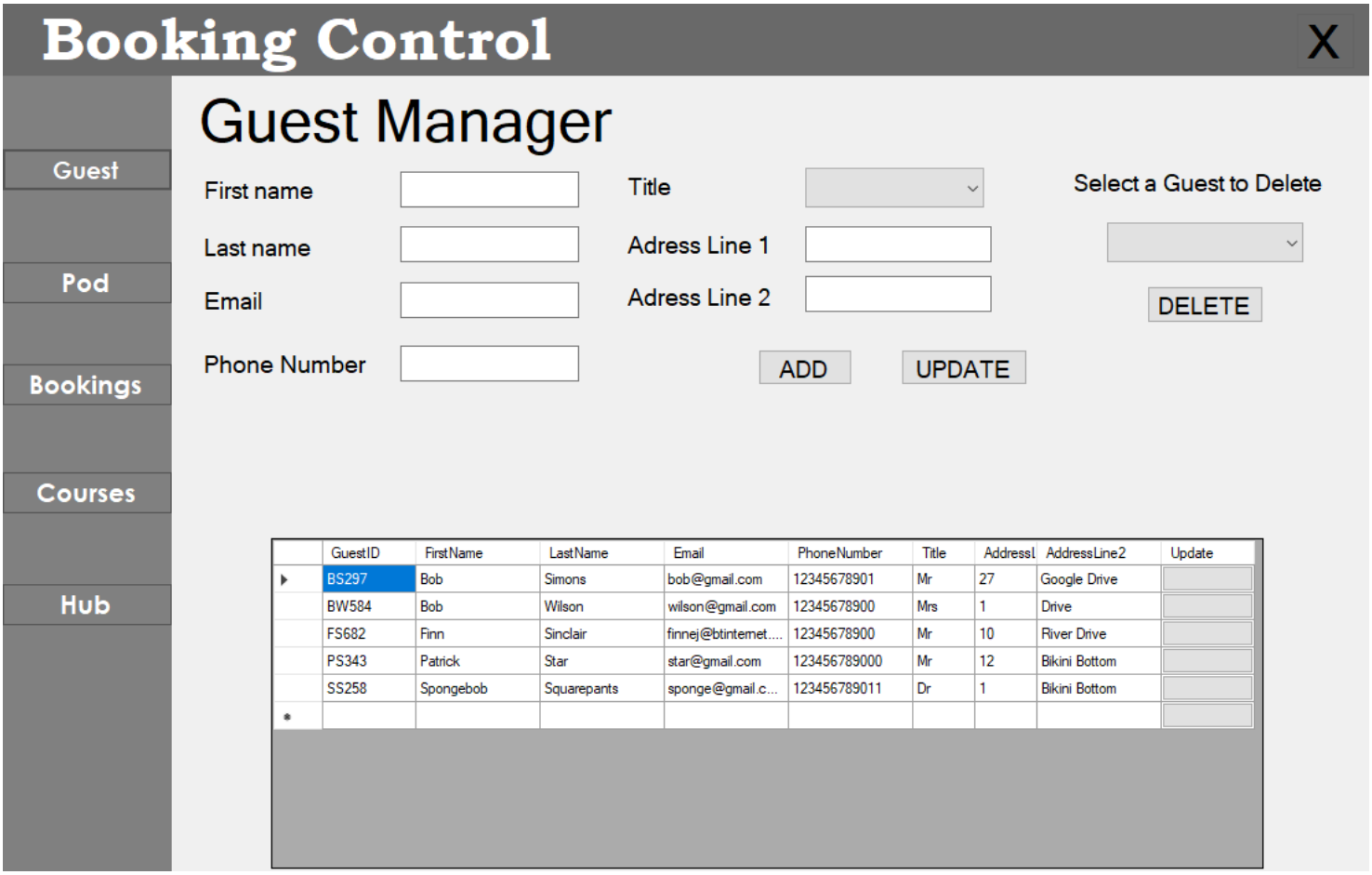
There are multiple ways to exit the Booking System. Firstly there is the built in exit button in the Visual Studio 2019 IDE, however due to issues on some displays to make the application as user friendly as possible an exit button has been incorporated into the Booking Control.



As pictured this button will exit the program. It is always on display to the user. No saving is required upon exiting the application as the database will be automatically updated prevent potential data lose.

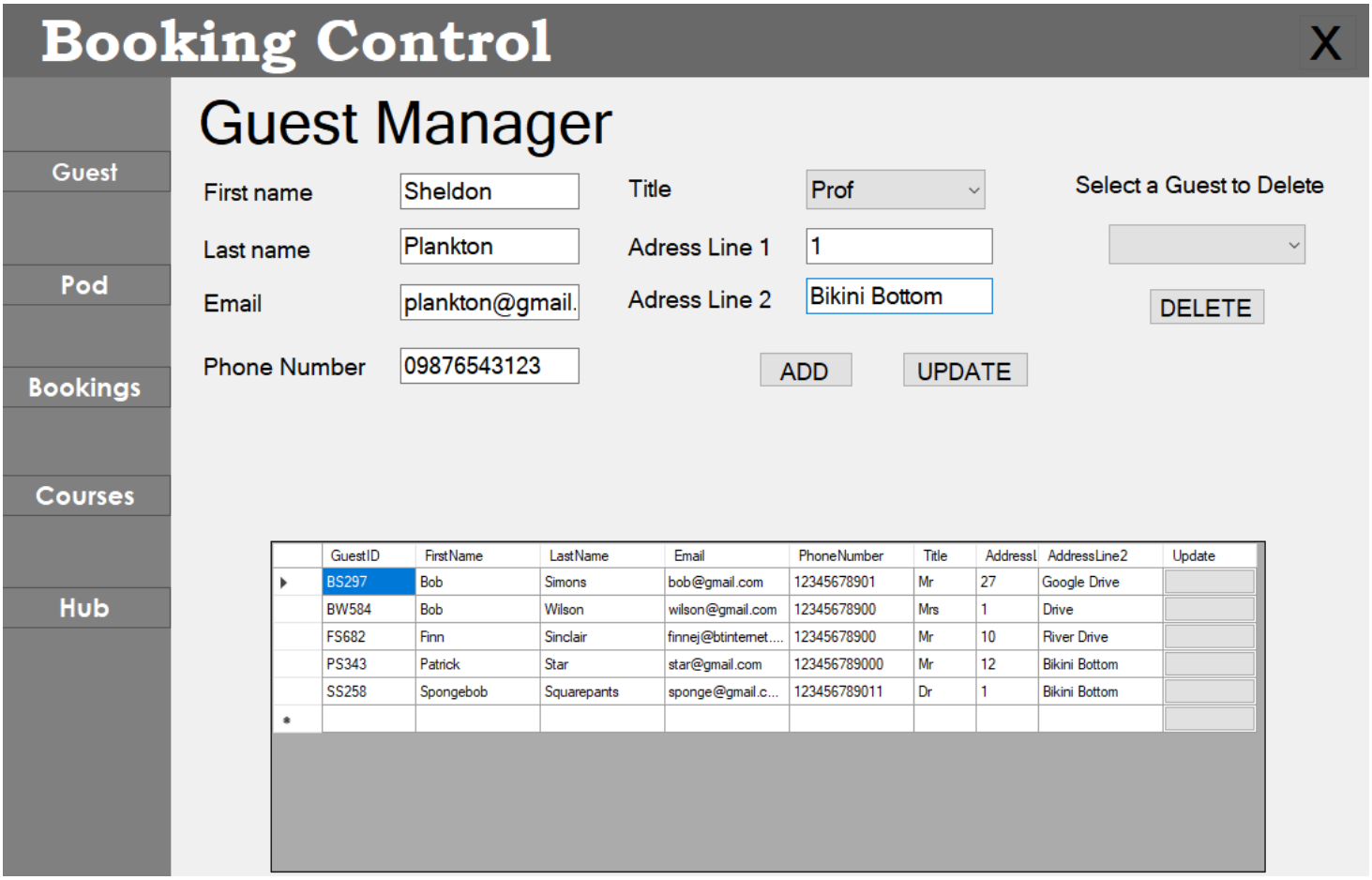
Guest Management

To utilise all functions all related to a guest the user would enter the **Guest Manager**. Pictured below:



Adding new Guests

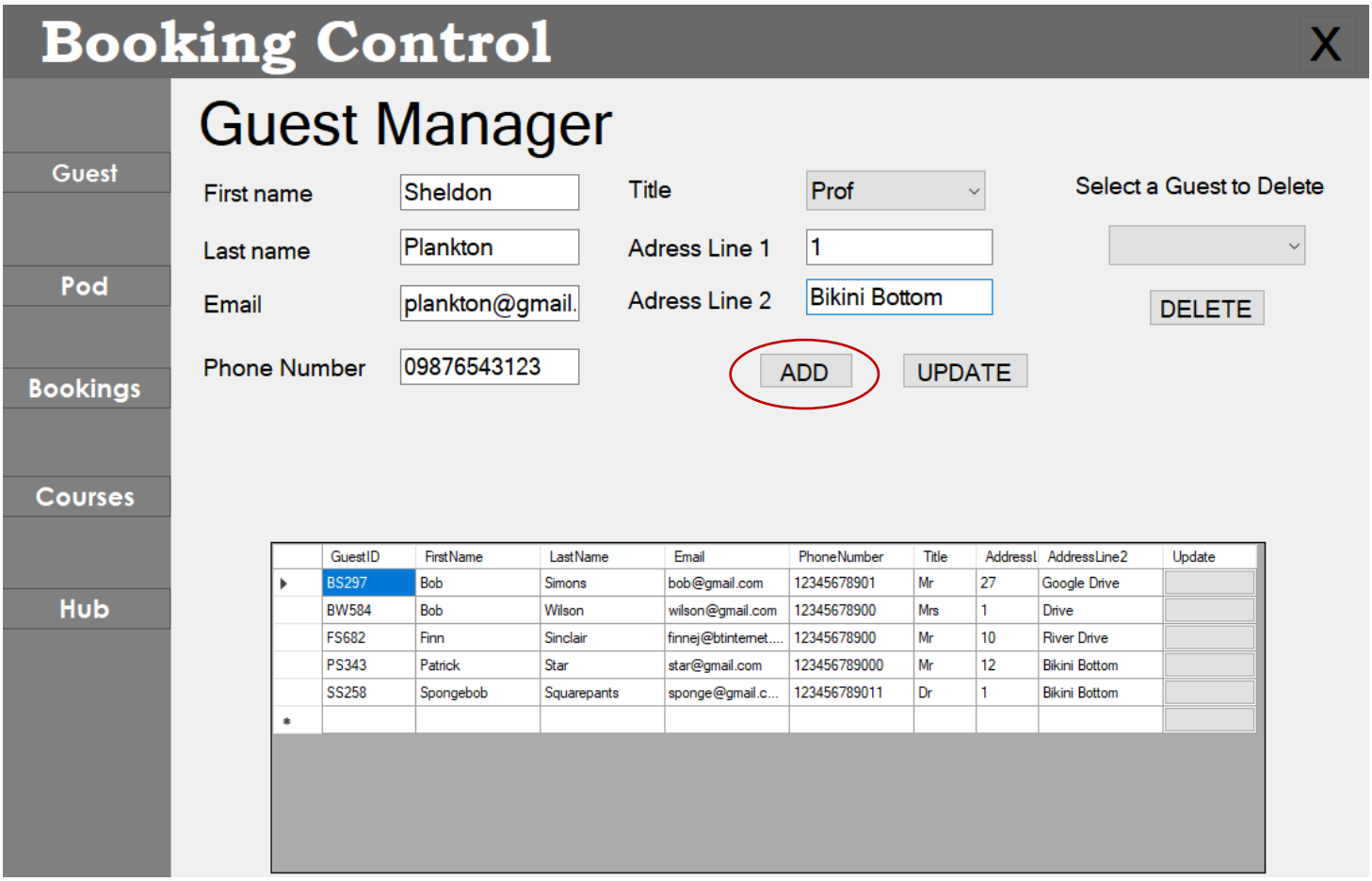
To add a new guest the user must first enter the guests information. For this example, the guest we are adding will be Plankton.



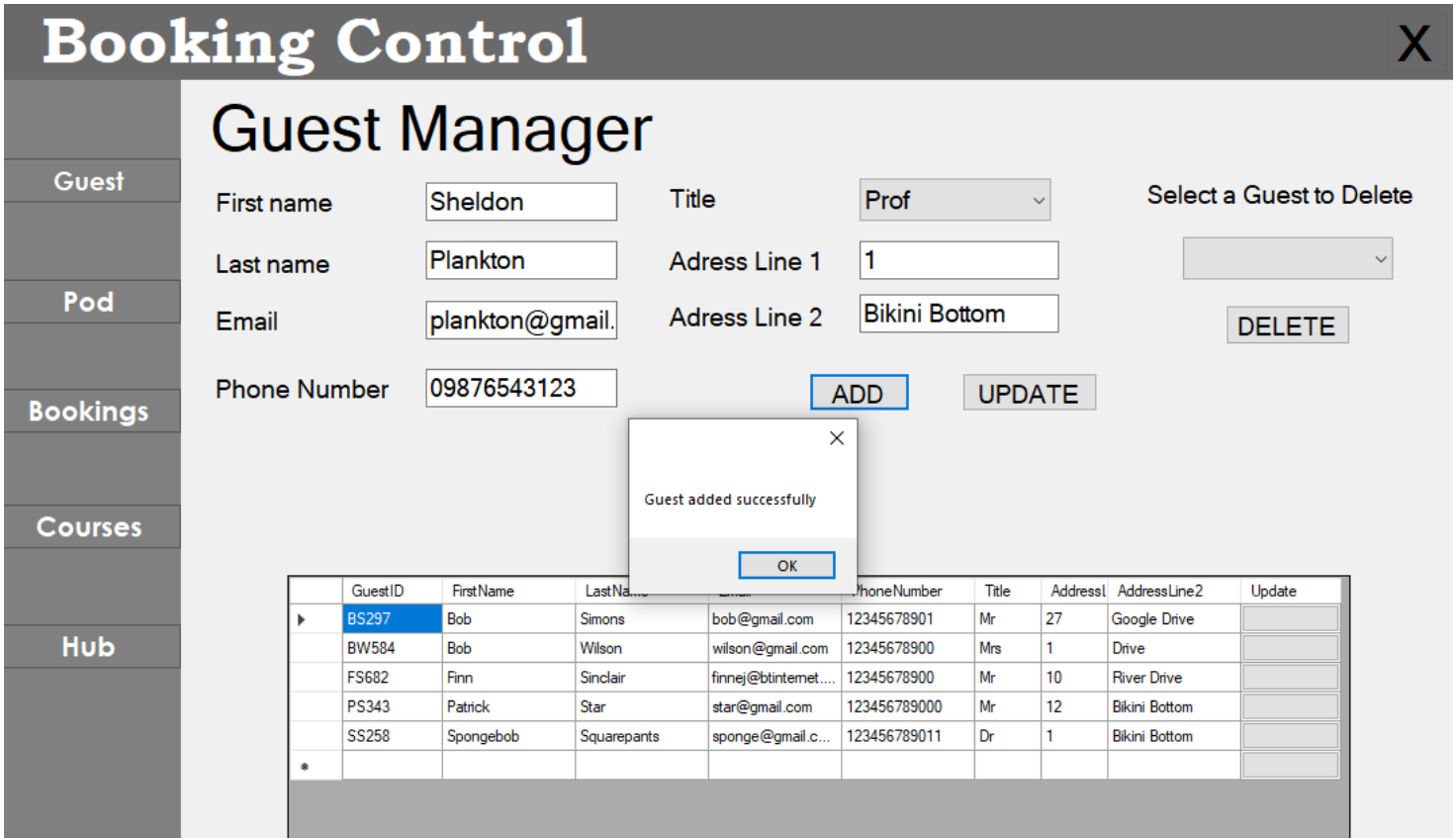
Stipulations when adding data (Guest Form)

A guest must have all of the following data – First name, Last name, email (With an @ symbol), Phone number (At least 11 digits long), a title, an address line 1 and an address line 2. Once all of this information is entered the user will then press the ADD button, pictured below:

ADD button for the guest form

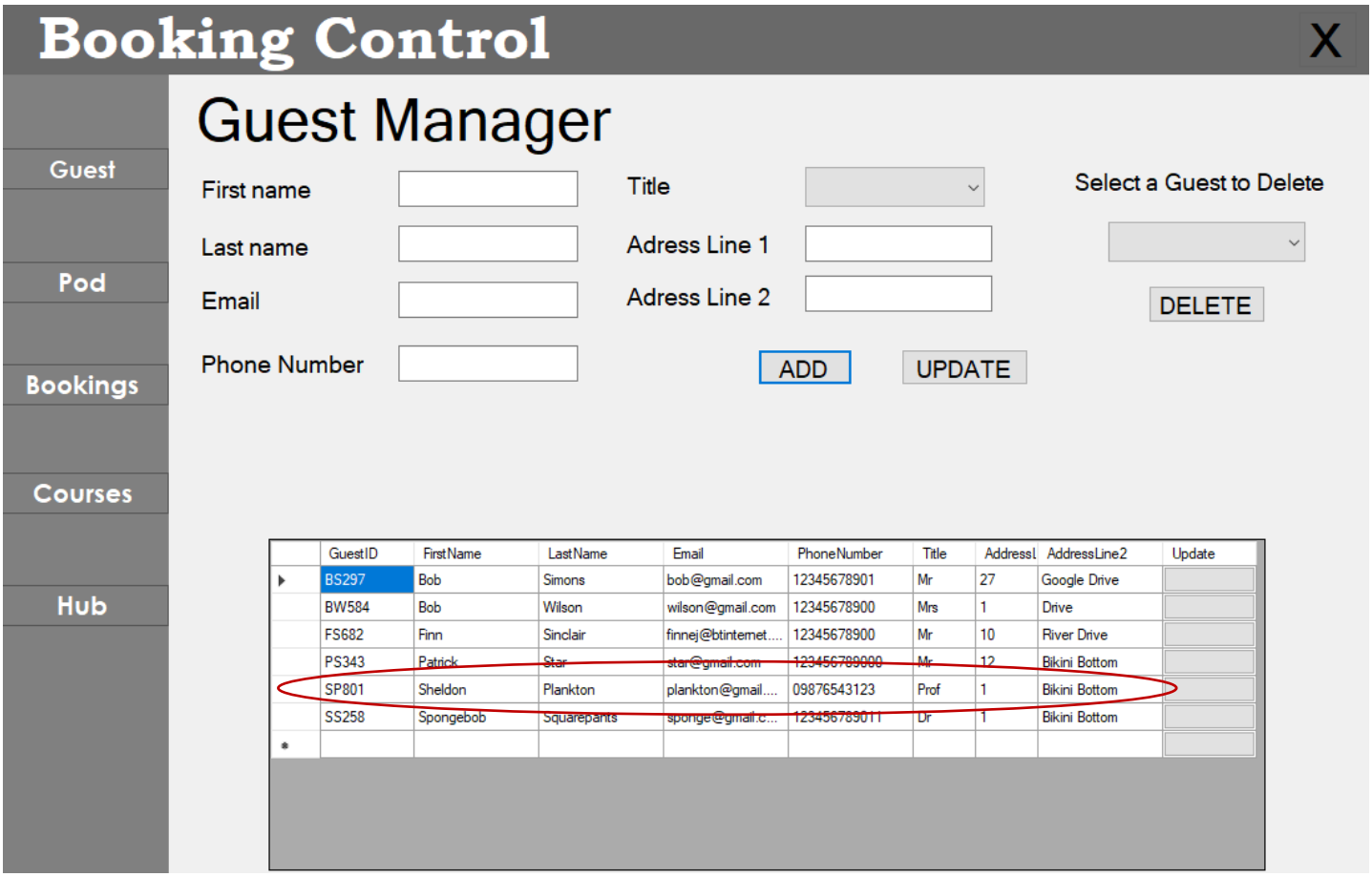


If all data entered is valid the guest will be added to the Booking System



Viewing Guests

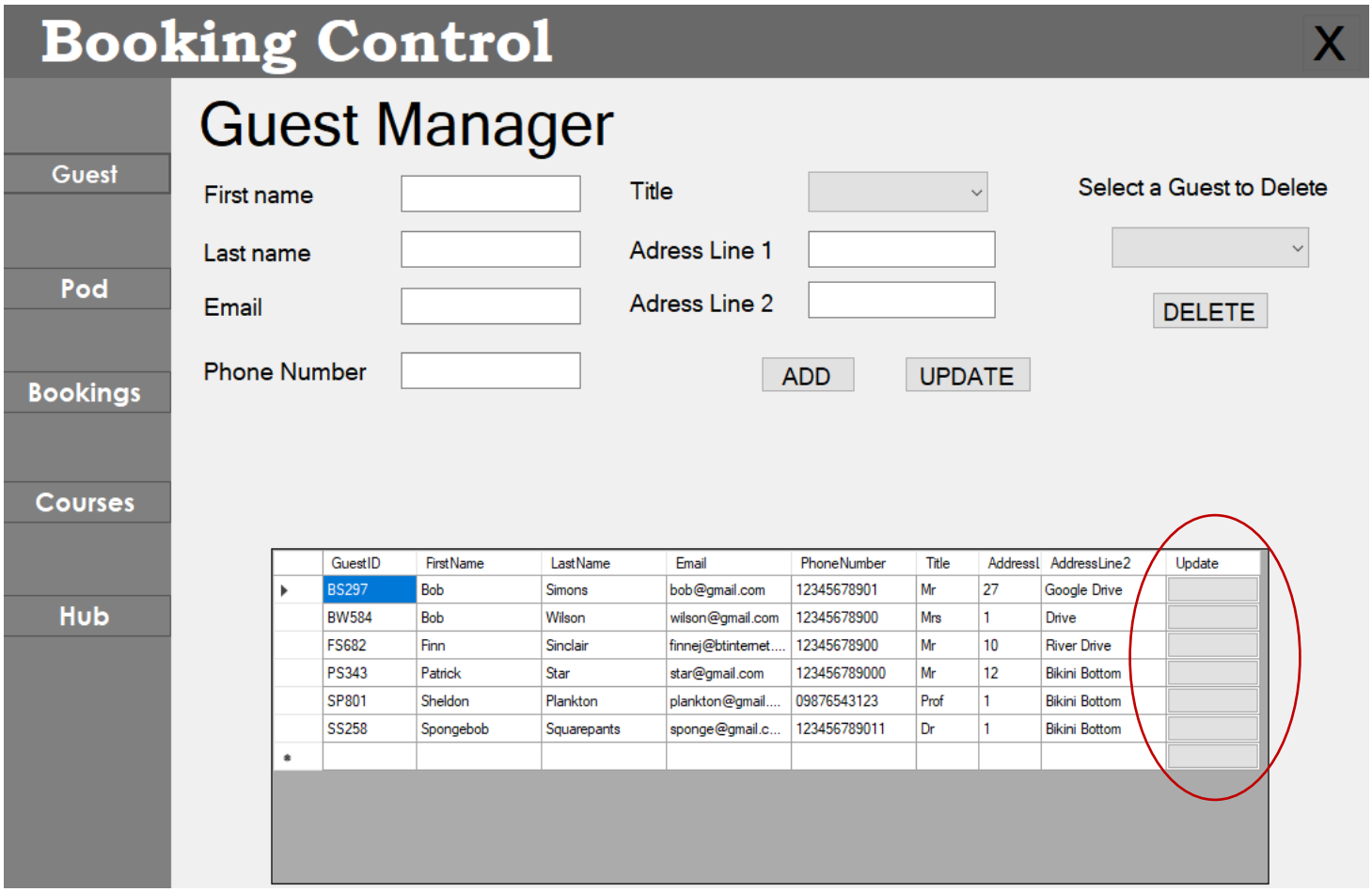
Guests that are in the system can be viewed by simply looking to the DataGrid View located at the bottom of the form.



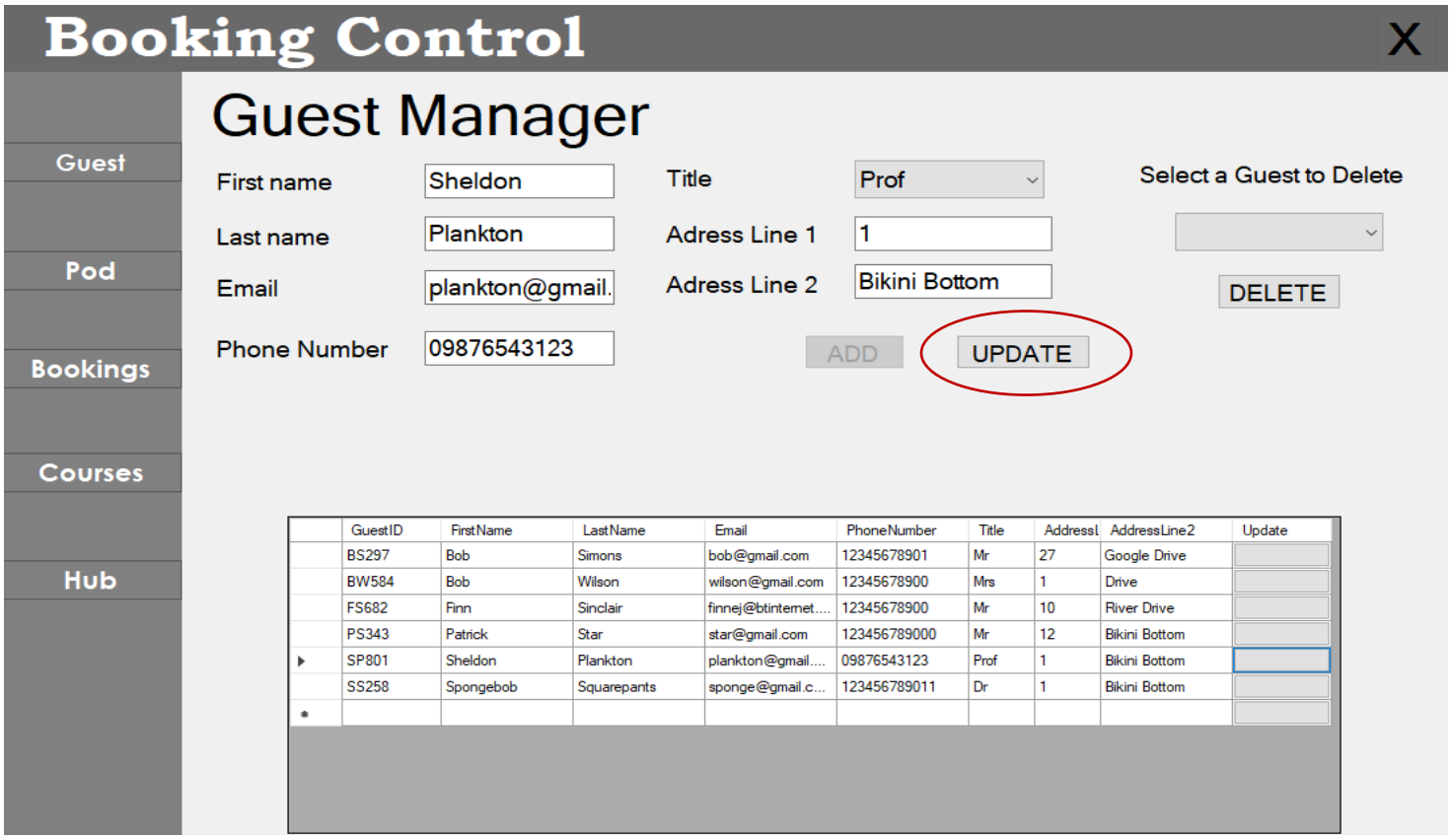
As pictured above Plankton has been added to the database and his information can be viewed by staff. The system also creates a unique GuestID for every guest for more intuitive guest identification.

Updating Guests

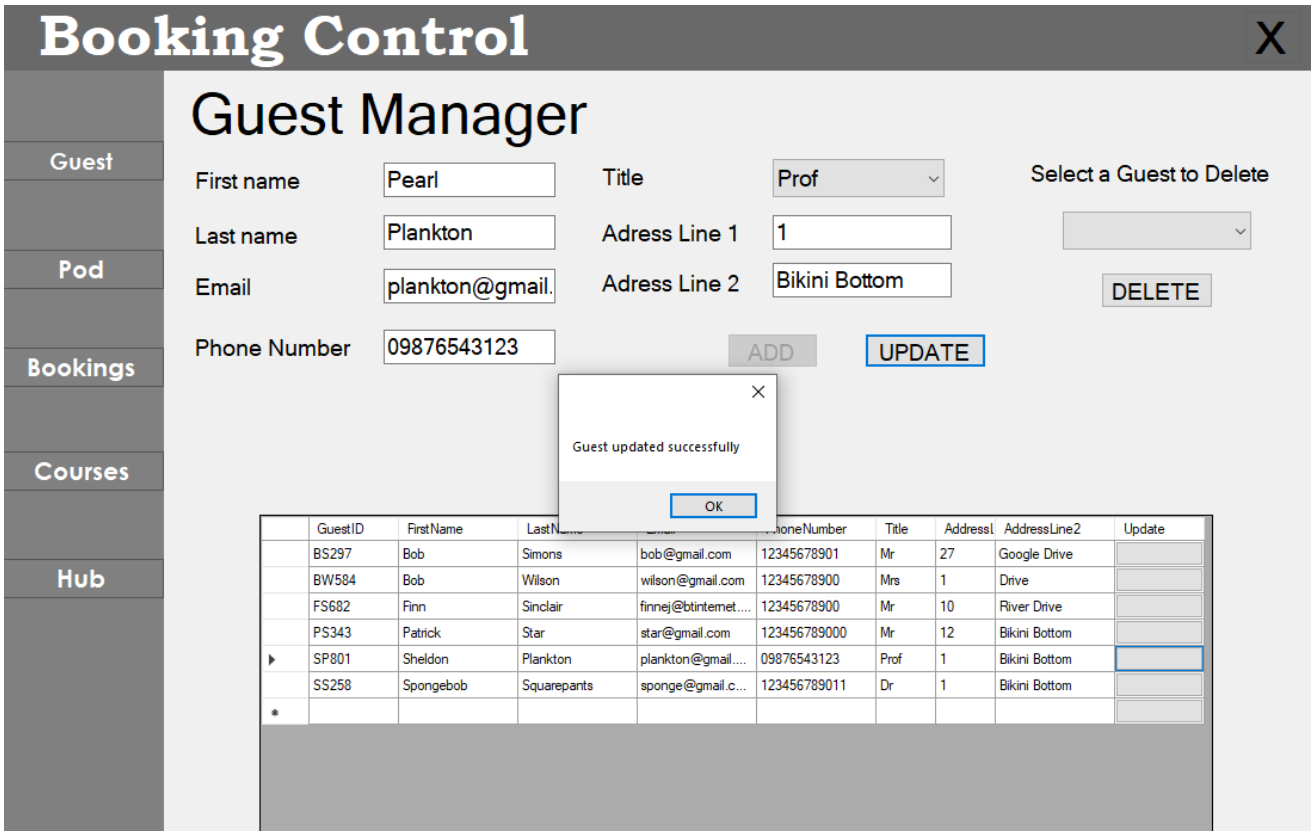
Guest information can be altered by using the update function that has been introduced to the Booking System.



The user first looks for the guest within the DataGrid View at the bottom of the screen and then presses the button titled update beside that guest. This will populate the guest information fields with that particular guests data pictured below.

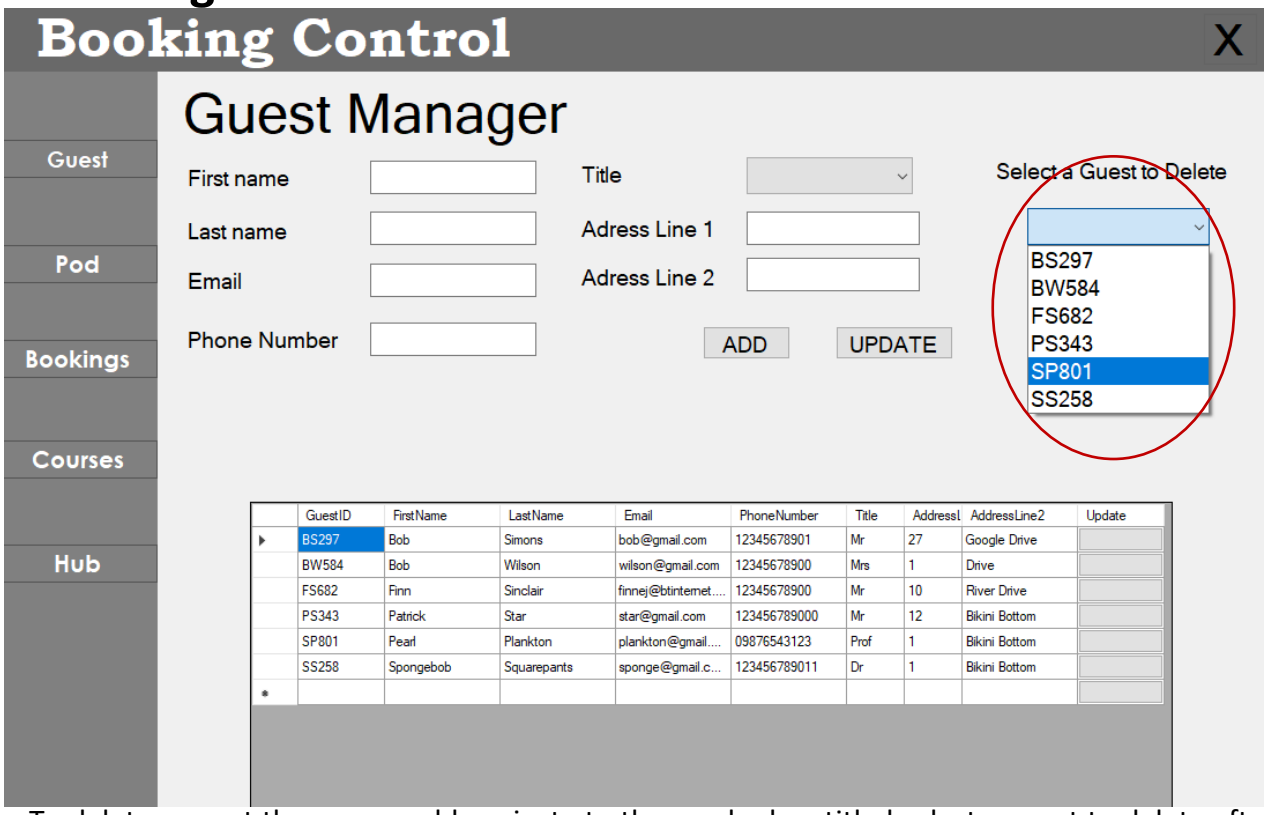


The guests information can now be edited and upon clicking on the button titled UPDATE in the centre of the screen that guests information will be changed.

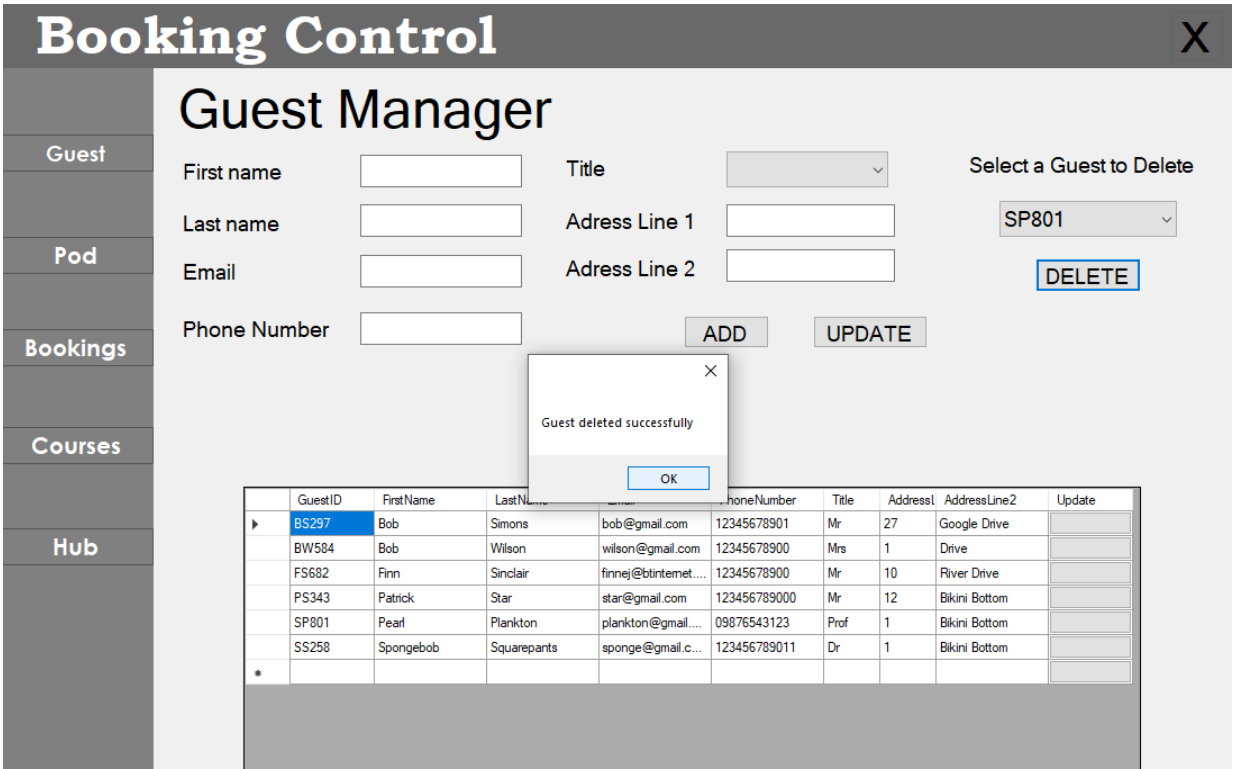


Planktons first name is now updated to pearl in the database, all data entry stipulations still apply even during the updating phase of a guest.

Deleting Guests



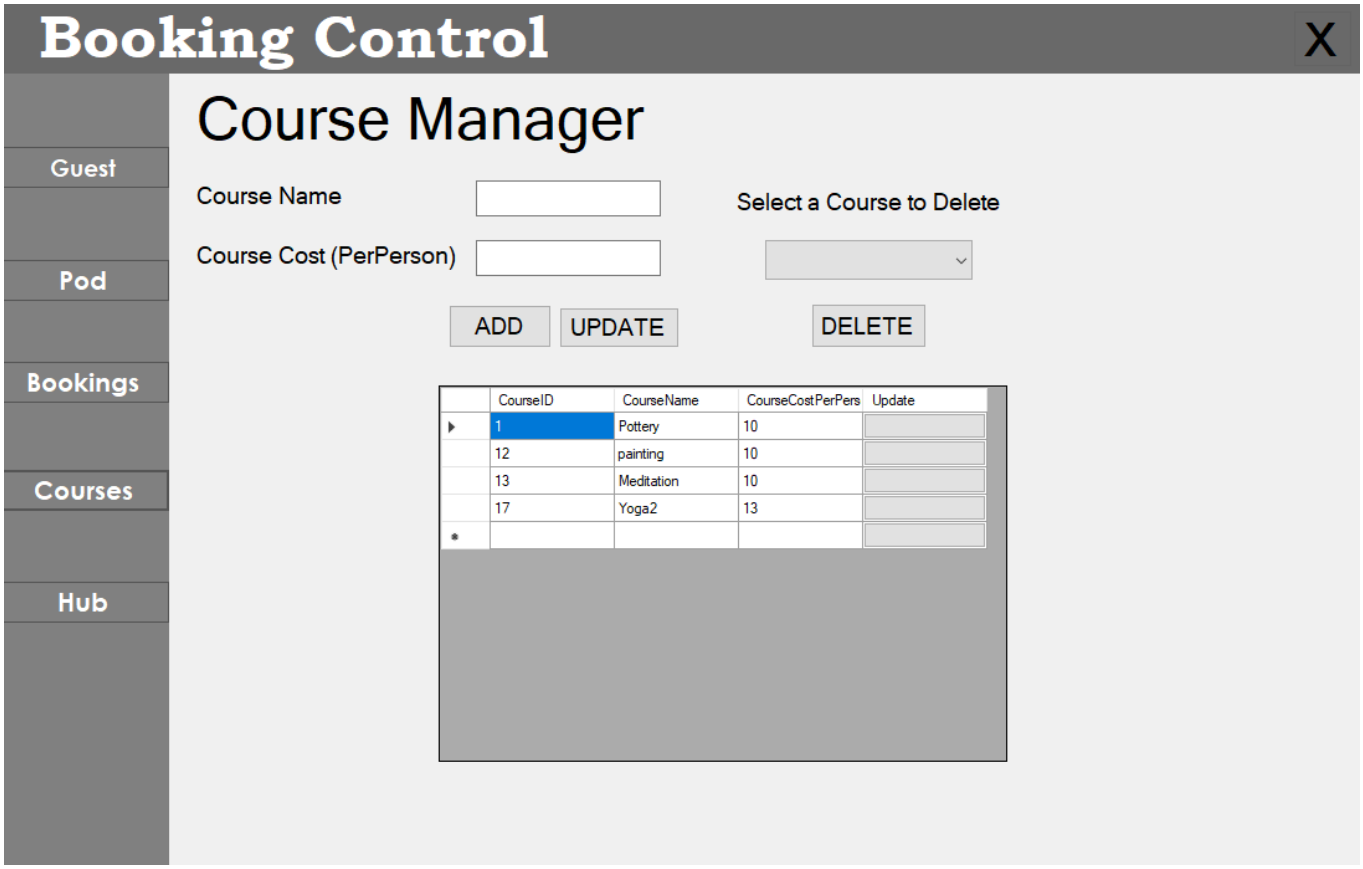
To delete a guest the user would navigate to the combo box titled select a guest to delete after choosing the guest they would like to delete they then press the DELETE button, this will permanently delete the guest.



Plankton is now deleted form the Booking System.

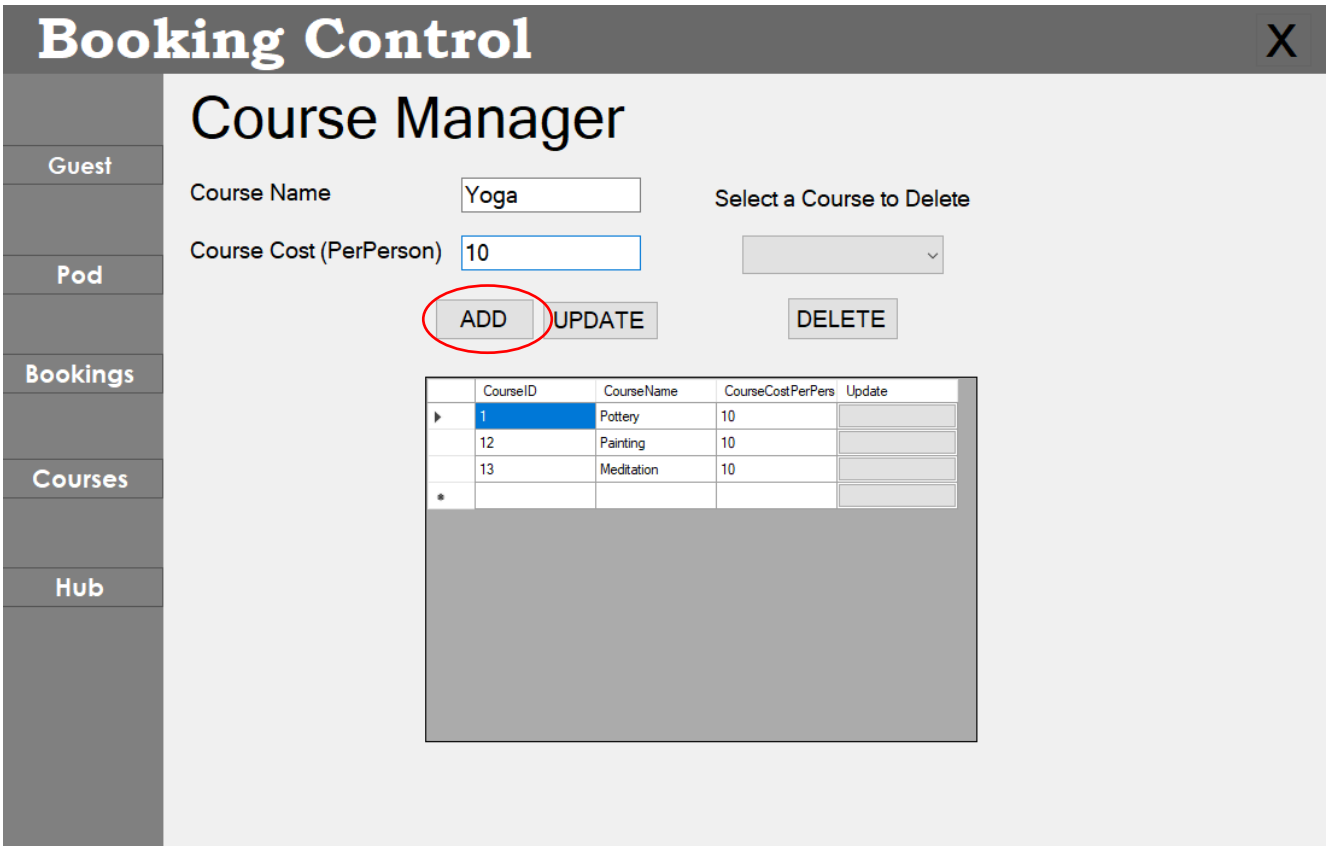
Course Management

To use all the functions related to courses the user would enter the **Course Manager**. Pictured below:



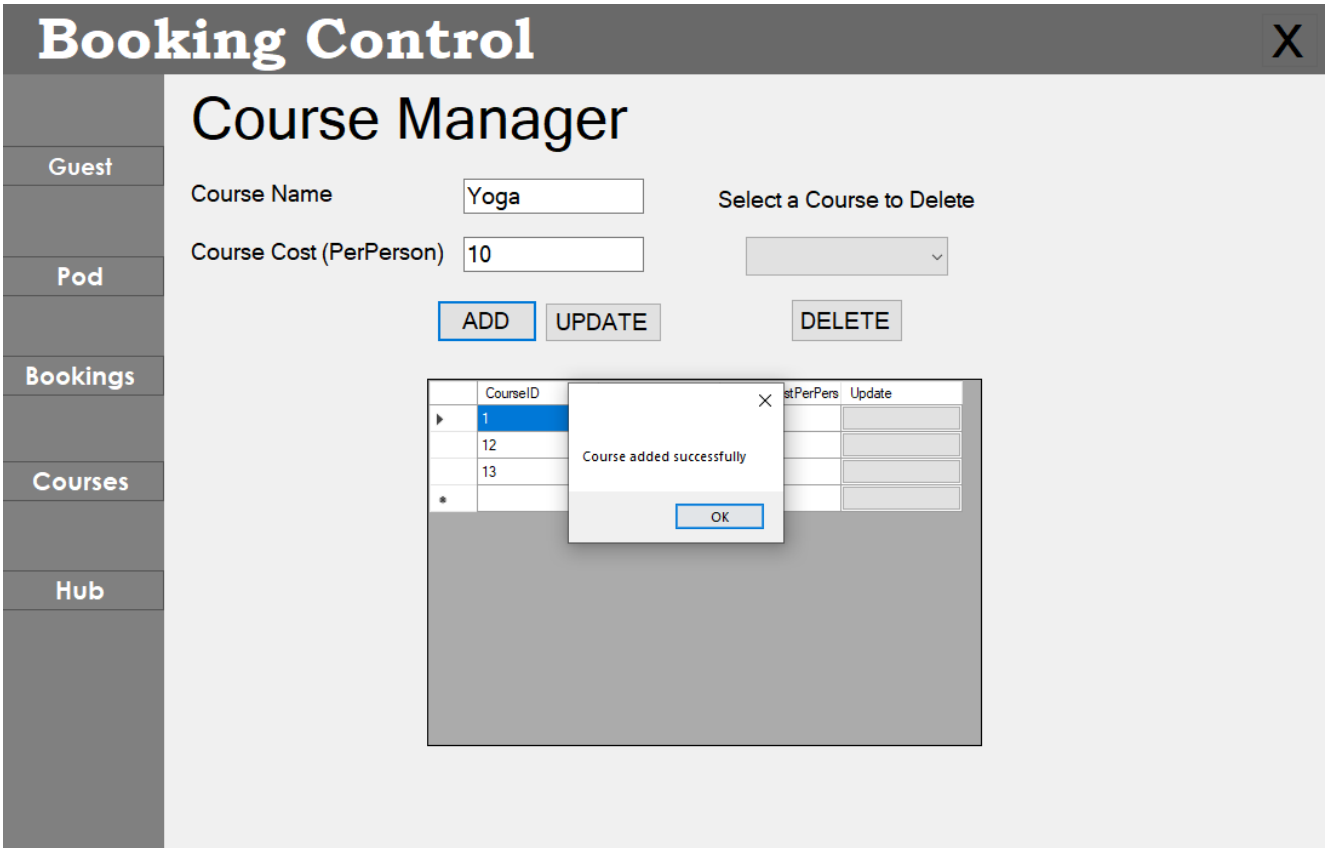
Adding a course

To add a new course the user must first enter all the corresponding data into the Course Manager's textboxes. Picture below:

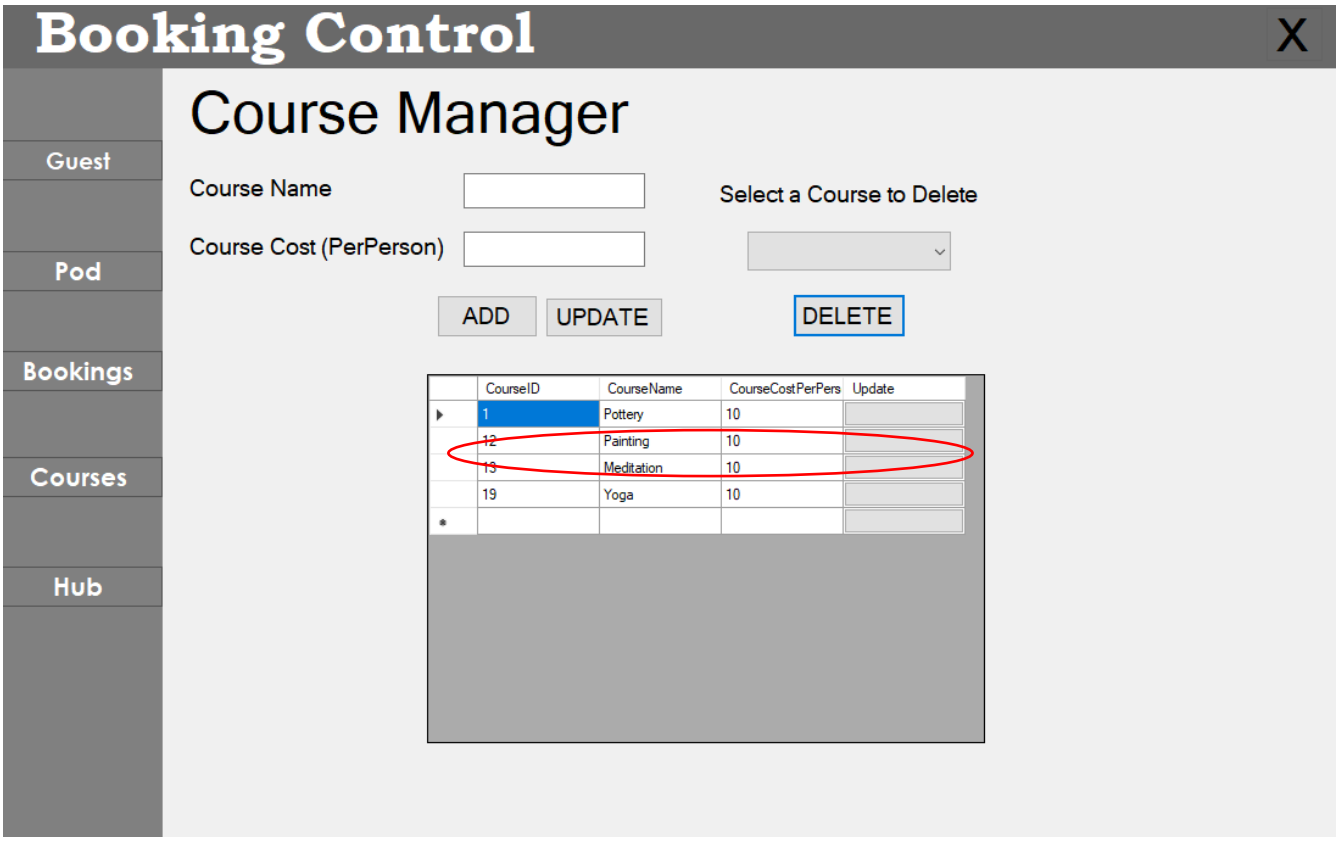


Data Entry Stipulations - Course Manager

There are stipulations when entering course data they are a course must have a name and a course must have a cost (The cost must not contain any characters that are not numeric). The user then presses the ADD button circled above. The course will then be added to the database.



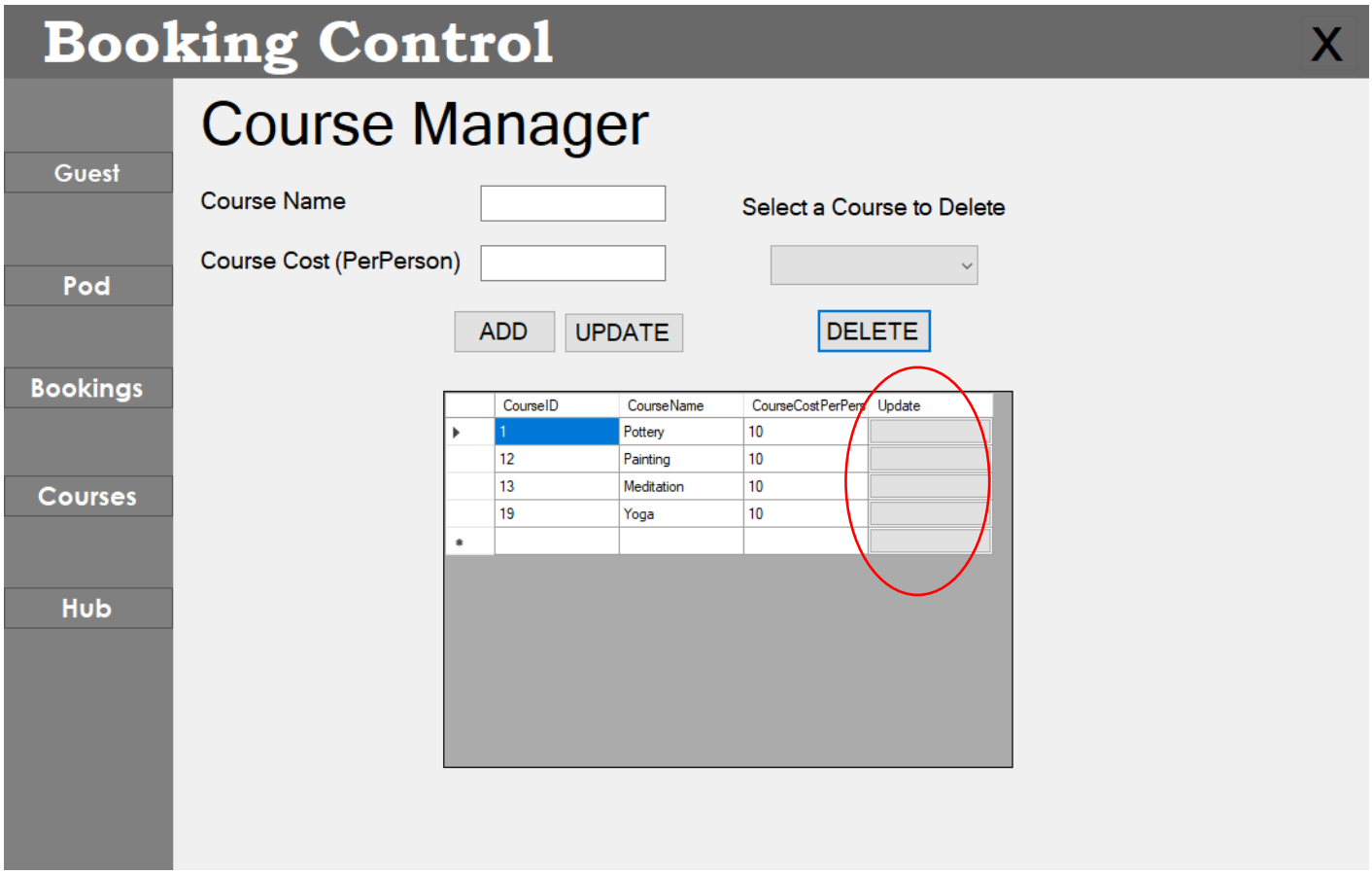
The course Yoga with a cost of 10 has now been successfully added to the database. To view the course the user would look to the DataGrid View at the bottom of their screen. Pictured below:



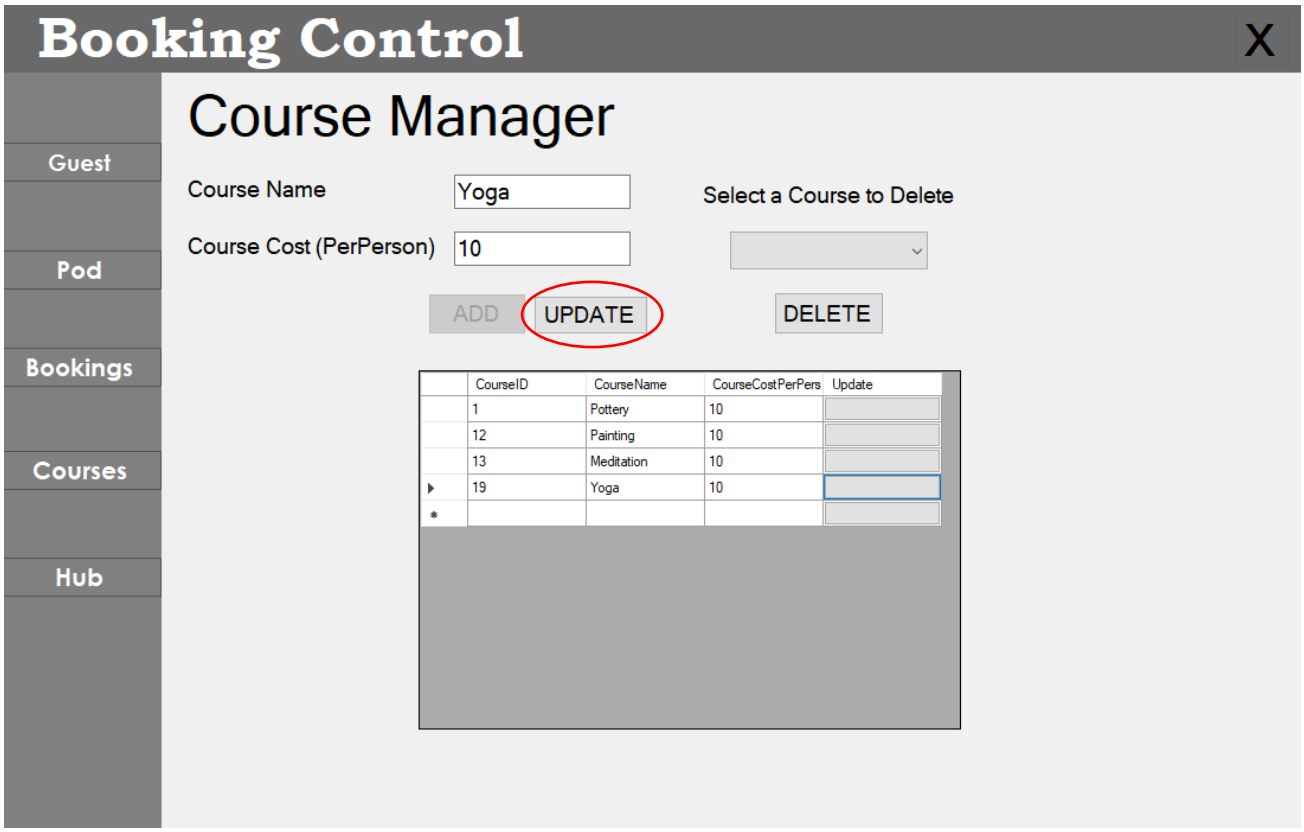
The course can be seen above

Updating a course

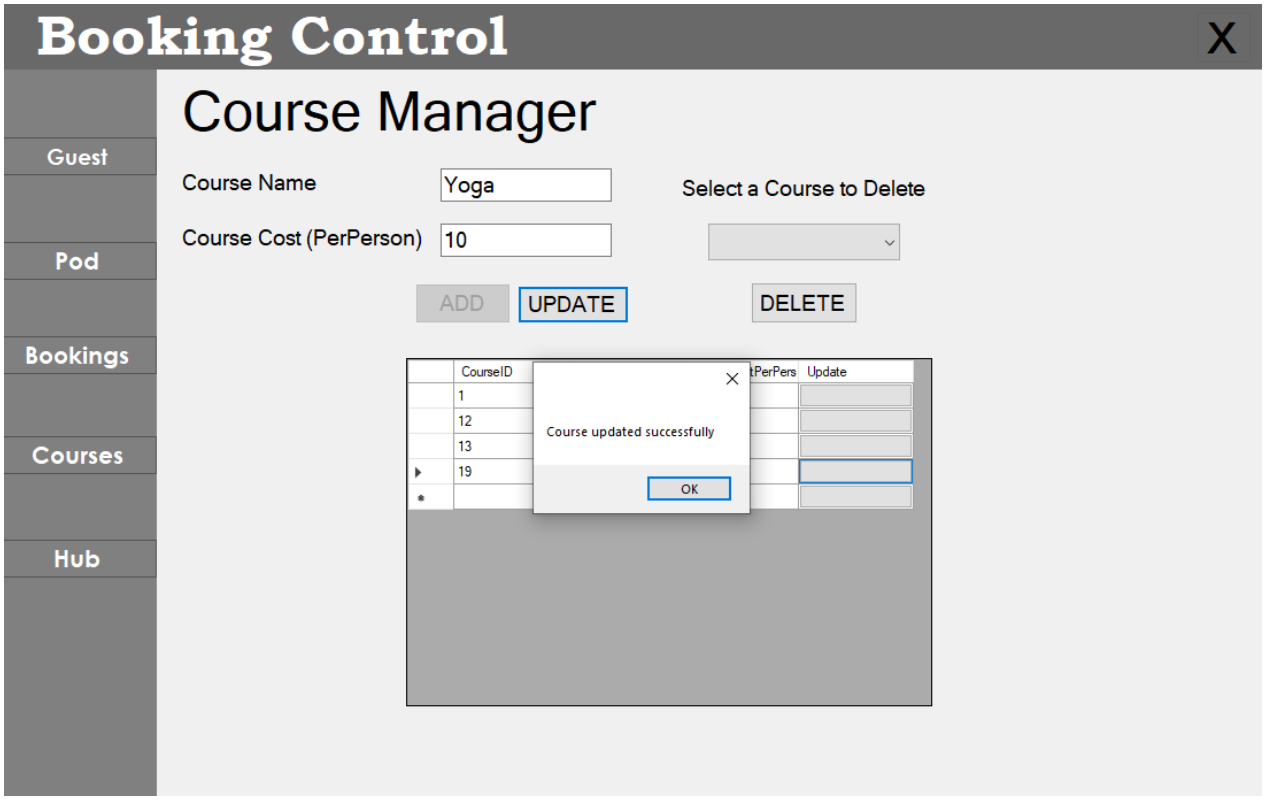
Course information can be altered by using the update function that has been introduced to the Booking System



The user first looks for the course within the DataGrid View at the bottom of the screen and then presses the button titled update beside that course. This will populate the course information fields with that particular courses data pictured below.

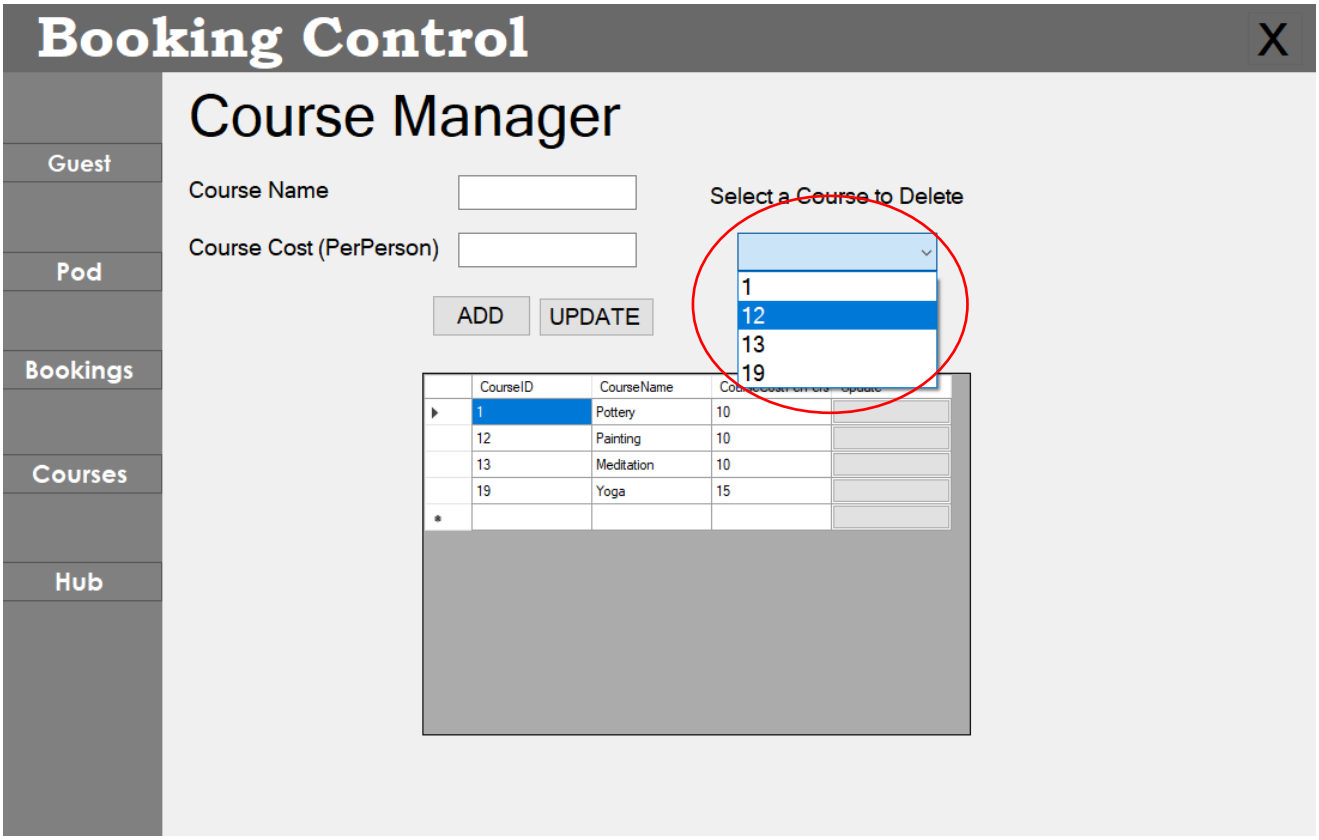


The courses information can now be edited and upon clicking on the button titled UPDATE in the centre of the screen that courses information will be changed.

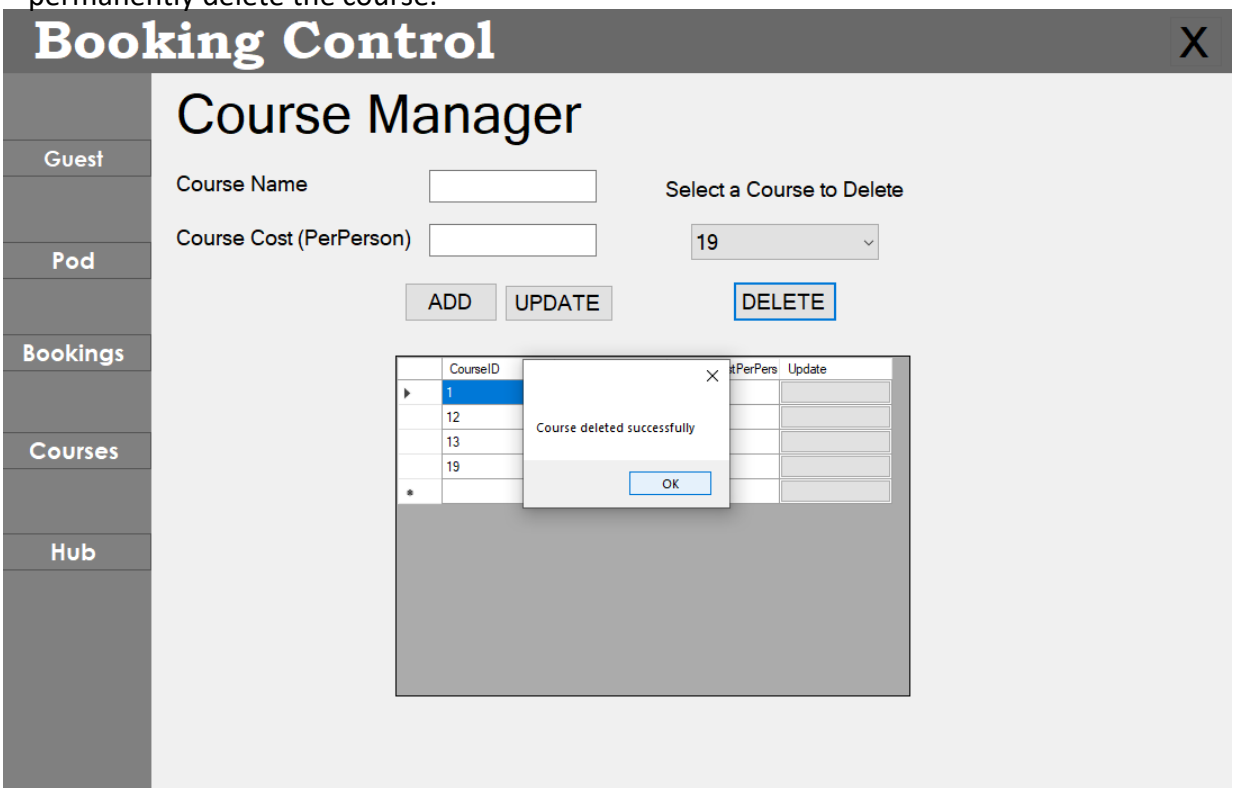


The courses cost has now been updated to 15 in the database, all data entry stipulations still apply even during the updating phase of a course.

Deleting Courses

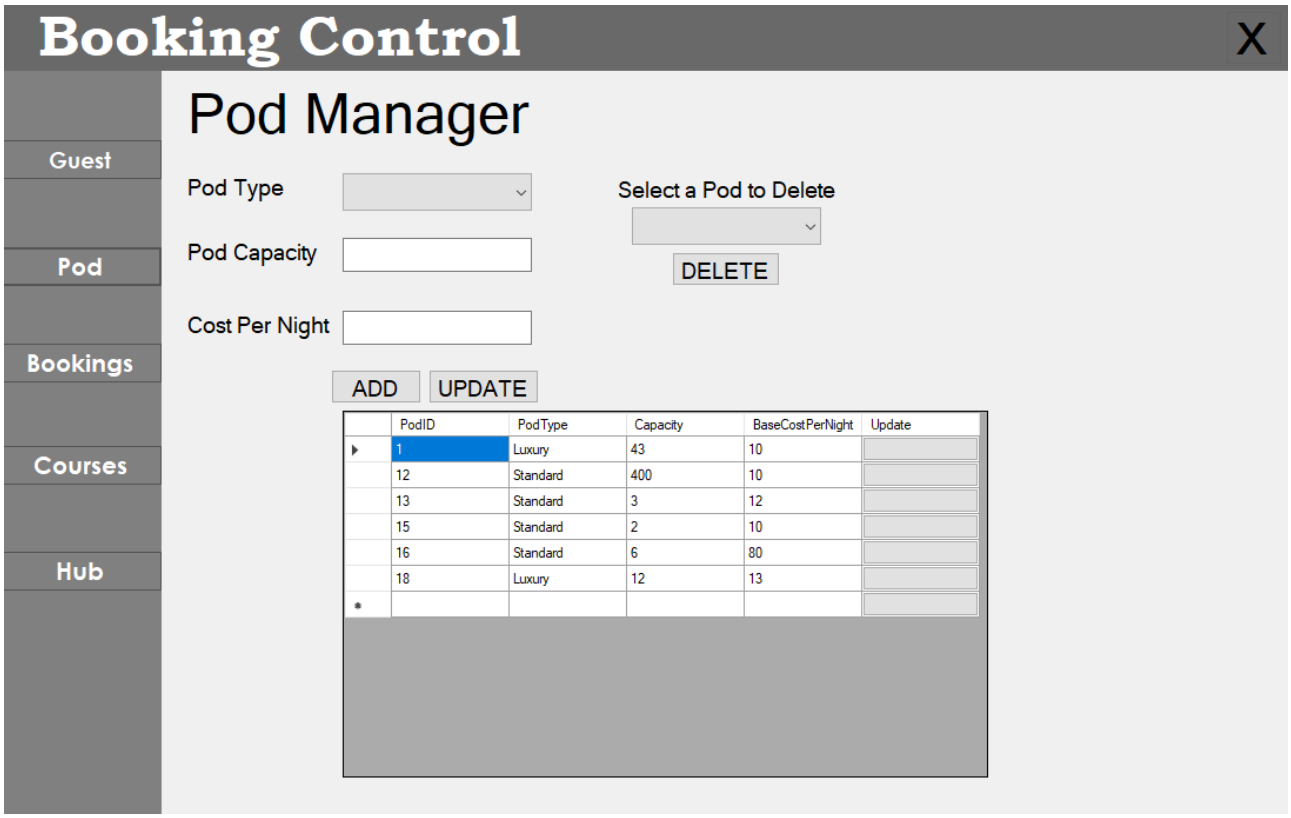


To delete a course the user would navigate to the combo box titled select a course to delete after choosing the course they would like to delete they then press the DELETE button, this will permanently delete the course.



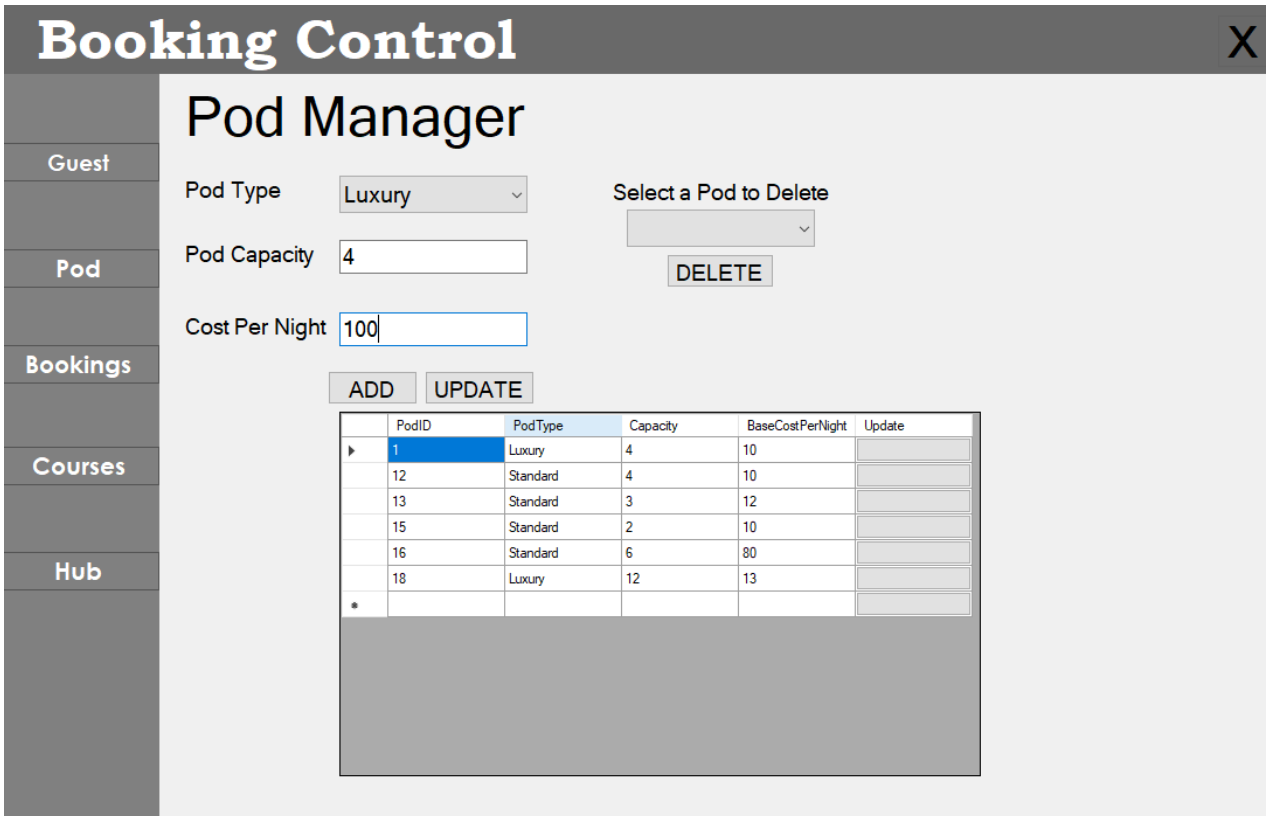
Pod Management

To utilise all functions all related to a pod the user would enter the **Pod Manager**. Pictured below:



Adding new Pods

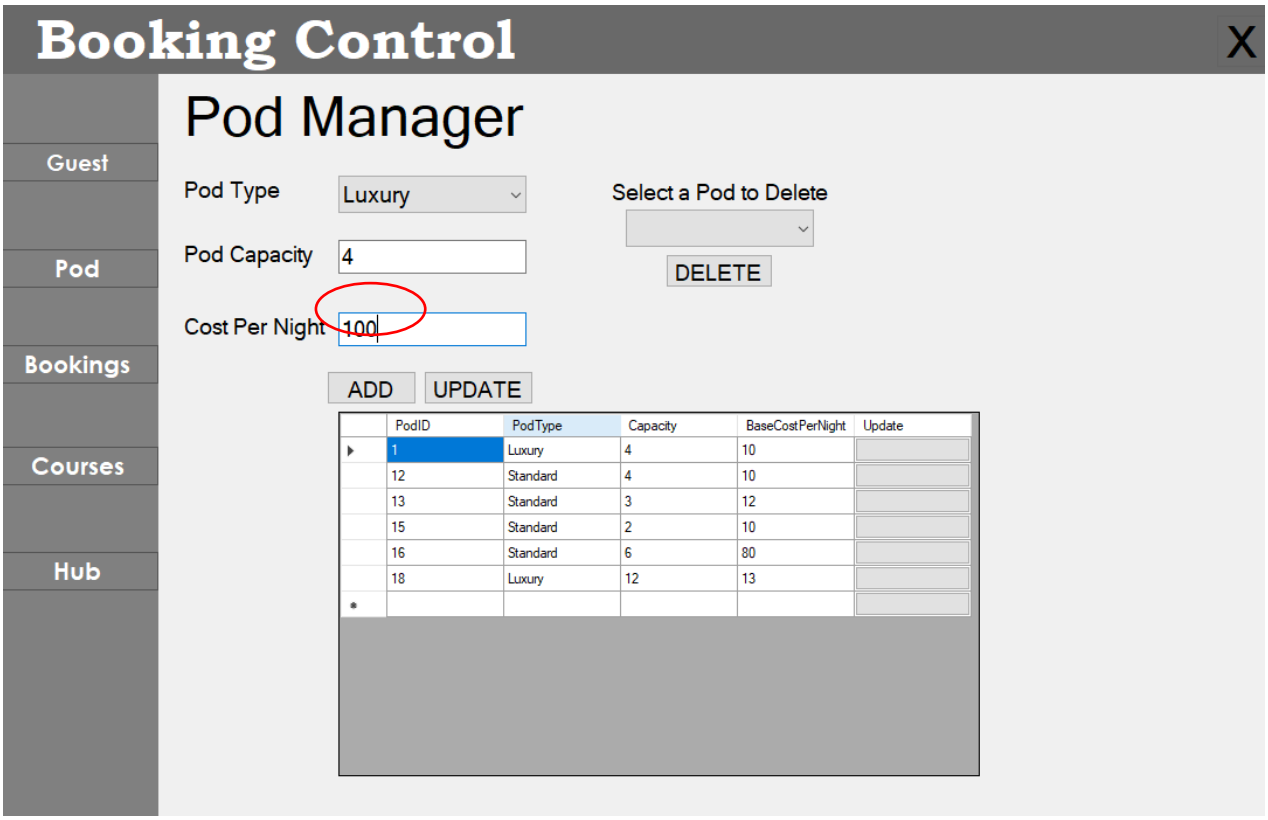
To add a new pod the user must first enter the pods information. For this example, the pod we are adding will be a luxury pod.



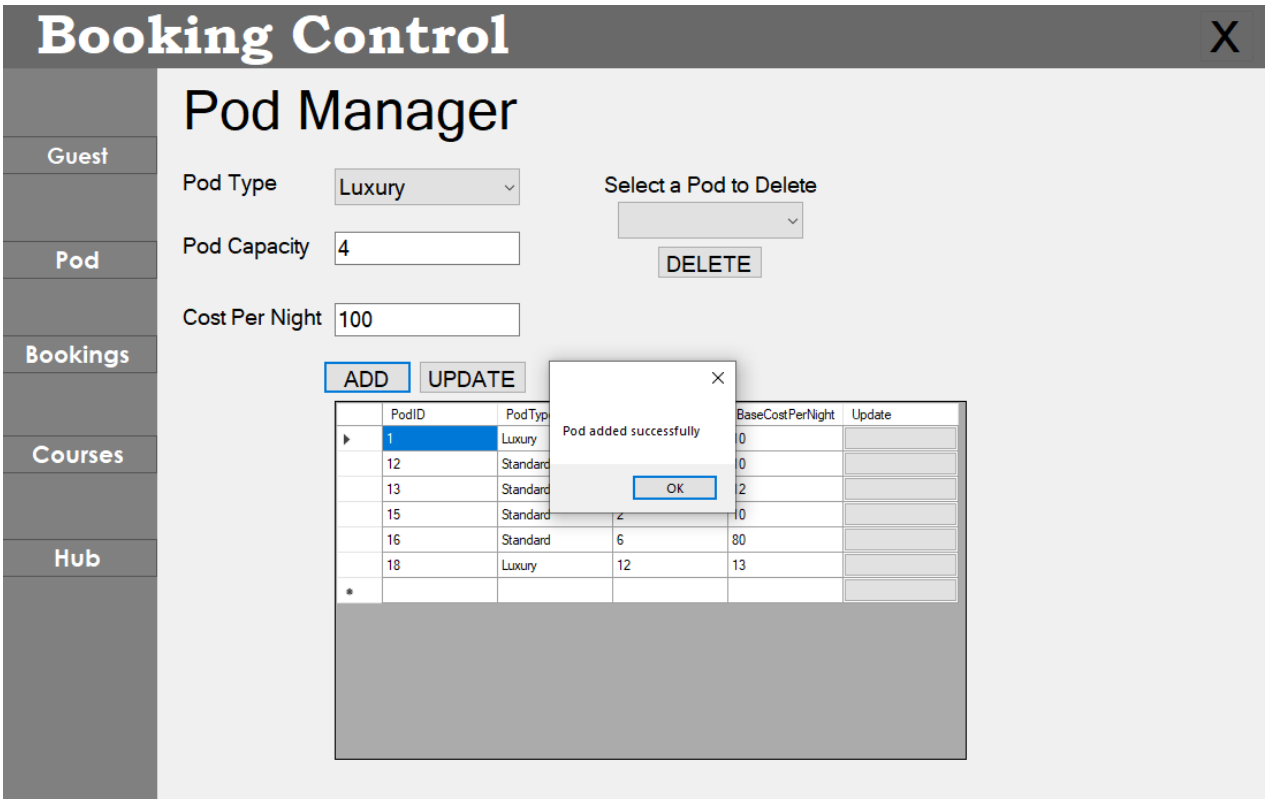
Data Entry stipulations – Pod Manager

A pod must have a pod type (Standard or Luxury), it must have a numeric pod capacity and it must have a numeric cost per night.

ADD button for Pod Manager

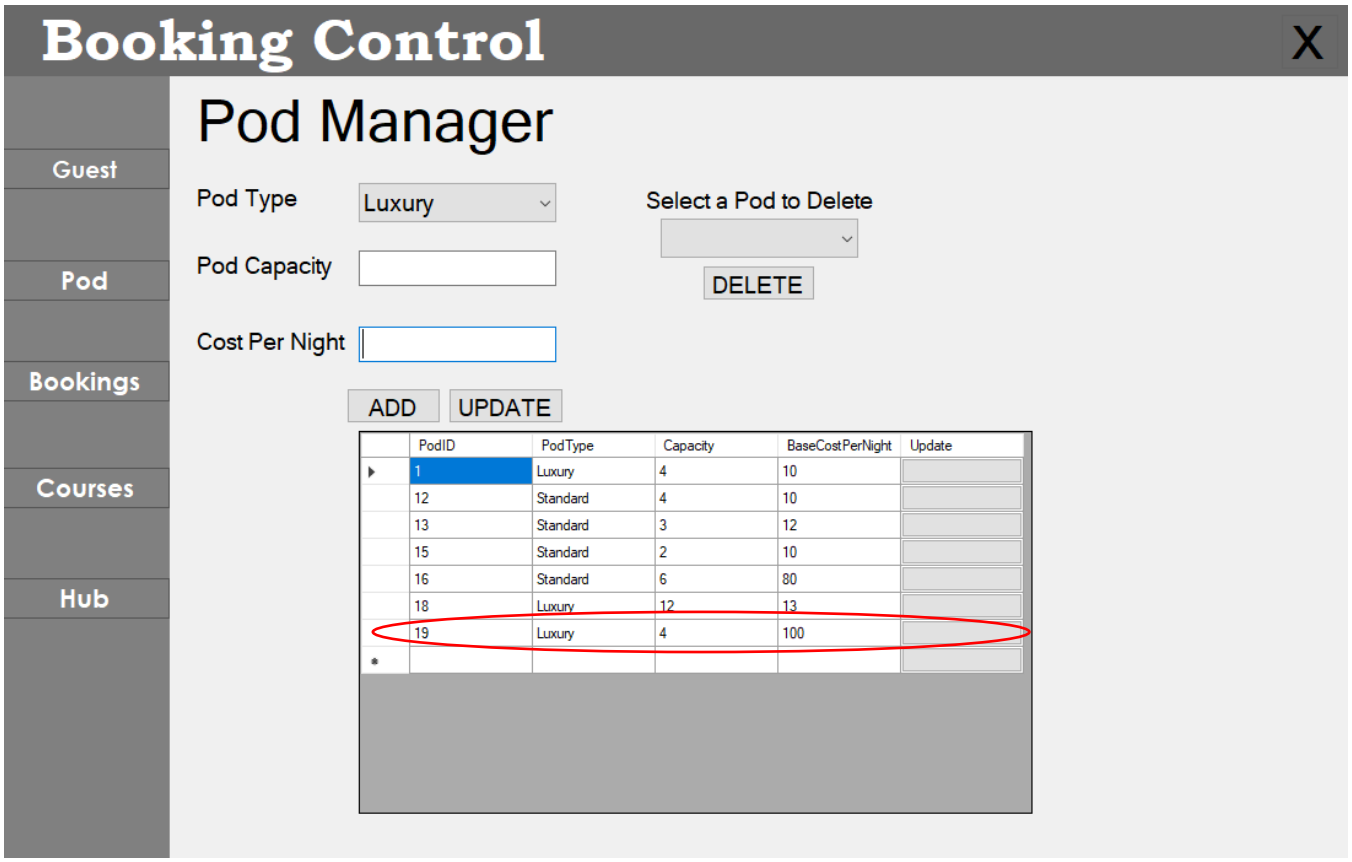


If all data entered is valid the pod will be add to the Booking System


Booking Control
X
Pod Manager
Guest
Pod
Bookings
Courses
Hub
Pod Type
Luxury
Pod Capacity
4
Cost Per Night
100
ADD
UPDATE
Select a Pod to Delete
DELETE
Pod added successfully
OK
PodID
PodTyp
BaseCostPerNight
Update
1
Luxury
12
Standard
13
Standard
15
Standard
16
Standard
6
80
18
Luxury
12
13

Viewing Pods

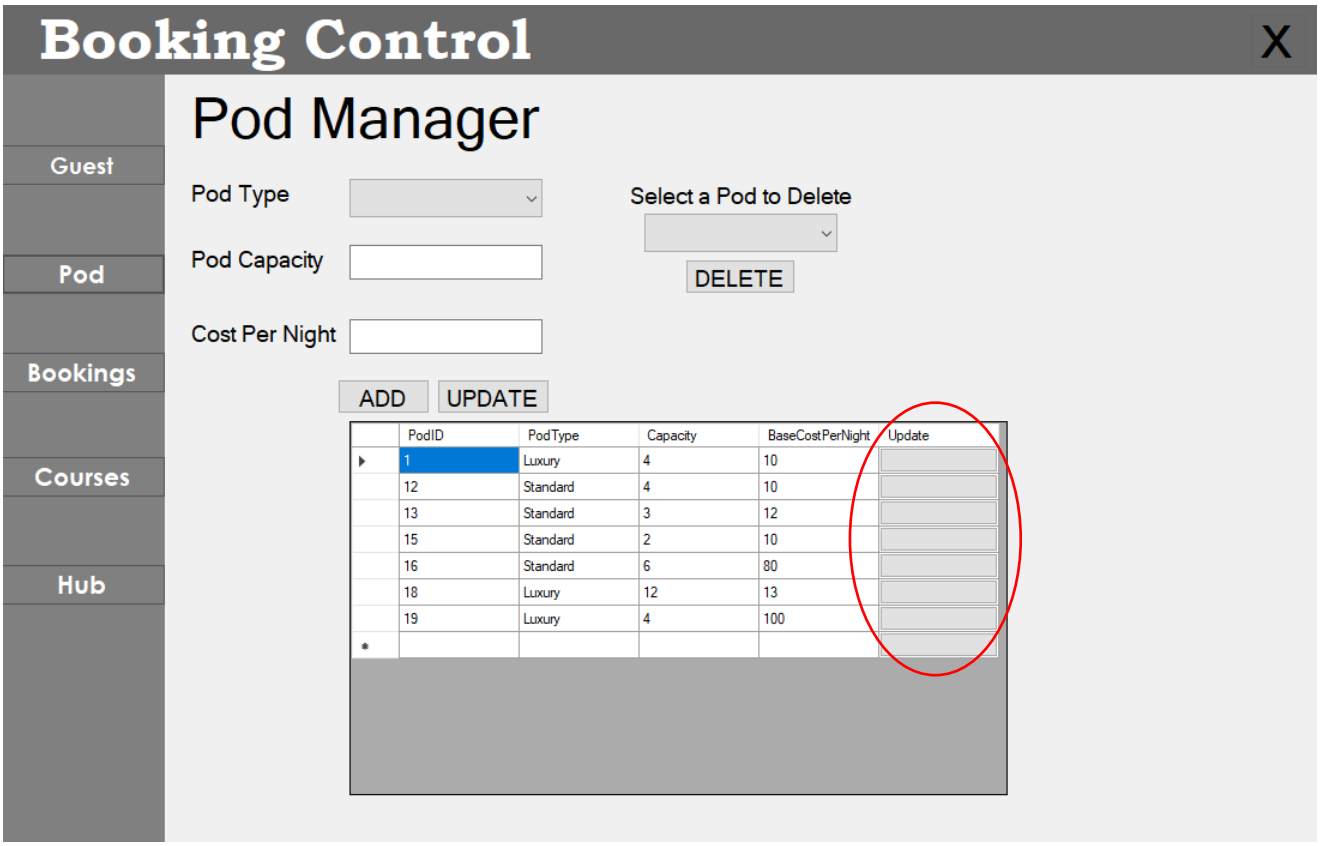
Pods that are in the system can be viewed by simply looking to the DataGrid View located at the bottom of the form.



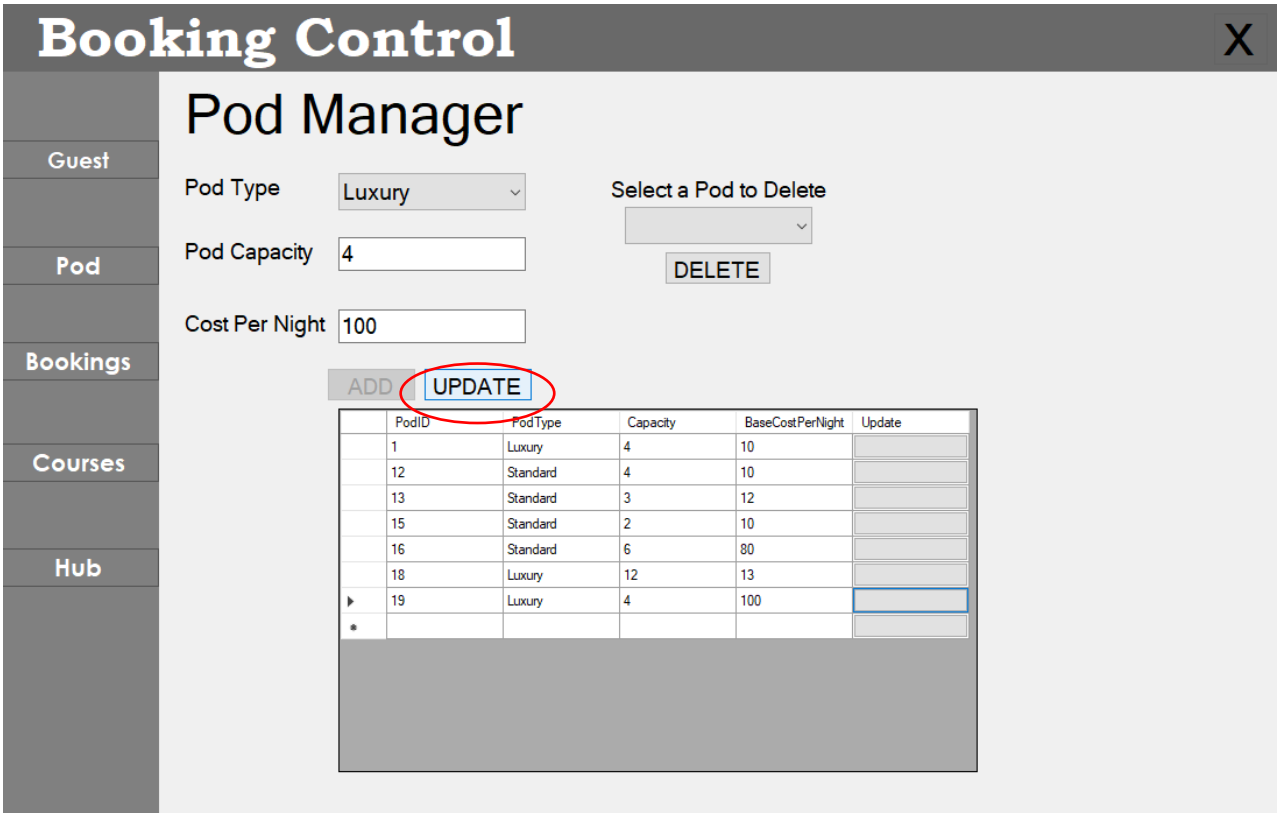
As pictured above the Yoga course that costs 100 has been added to the database with a unique course id applied to it to prevent confusion and mis identification.

Updating Pods

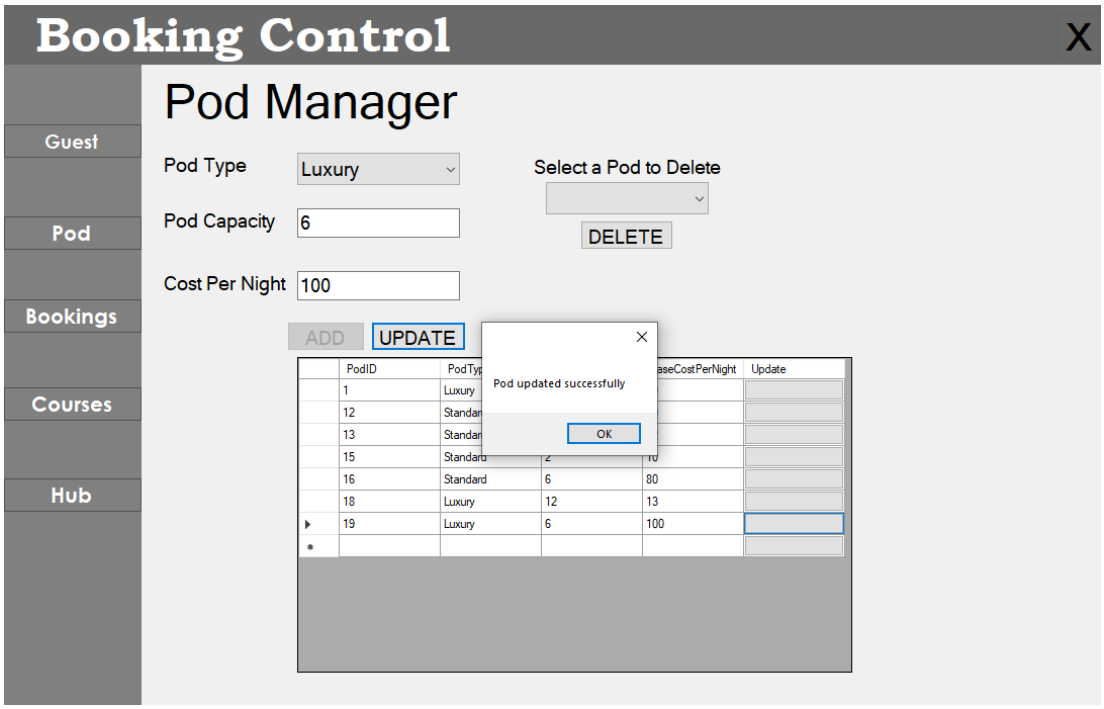
Pod information can be altered by using the update function that has been introduced to the Booking System.



The user first looks for the pod within the DataGrid View at the bottom of the screen and then presses the button titled update beside that pod. This will populate the pod information fields with that particular pods data pictured below.

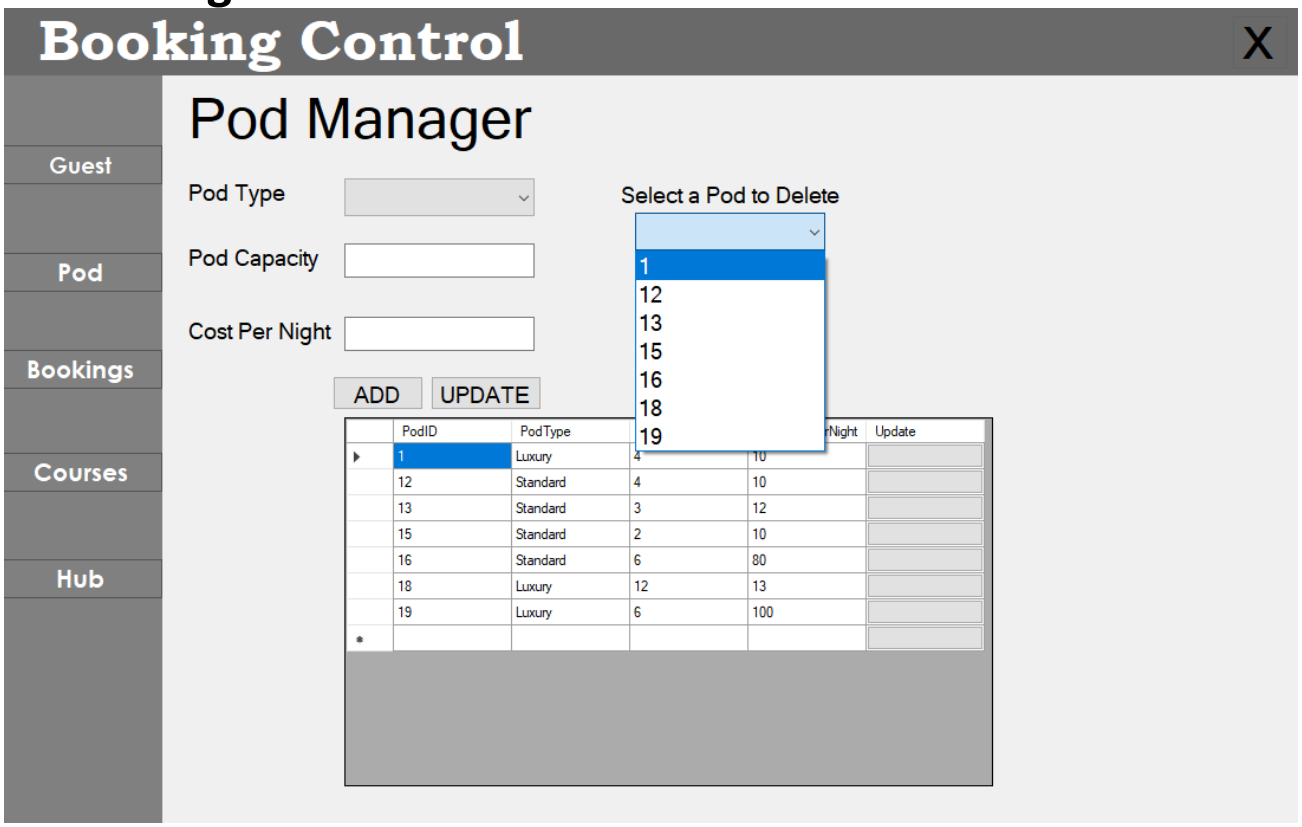


The pods information can now be edited and upon clicking on the button titled UPDATE in the centre of the screen that pods information will be changed.

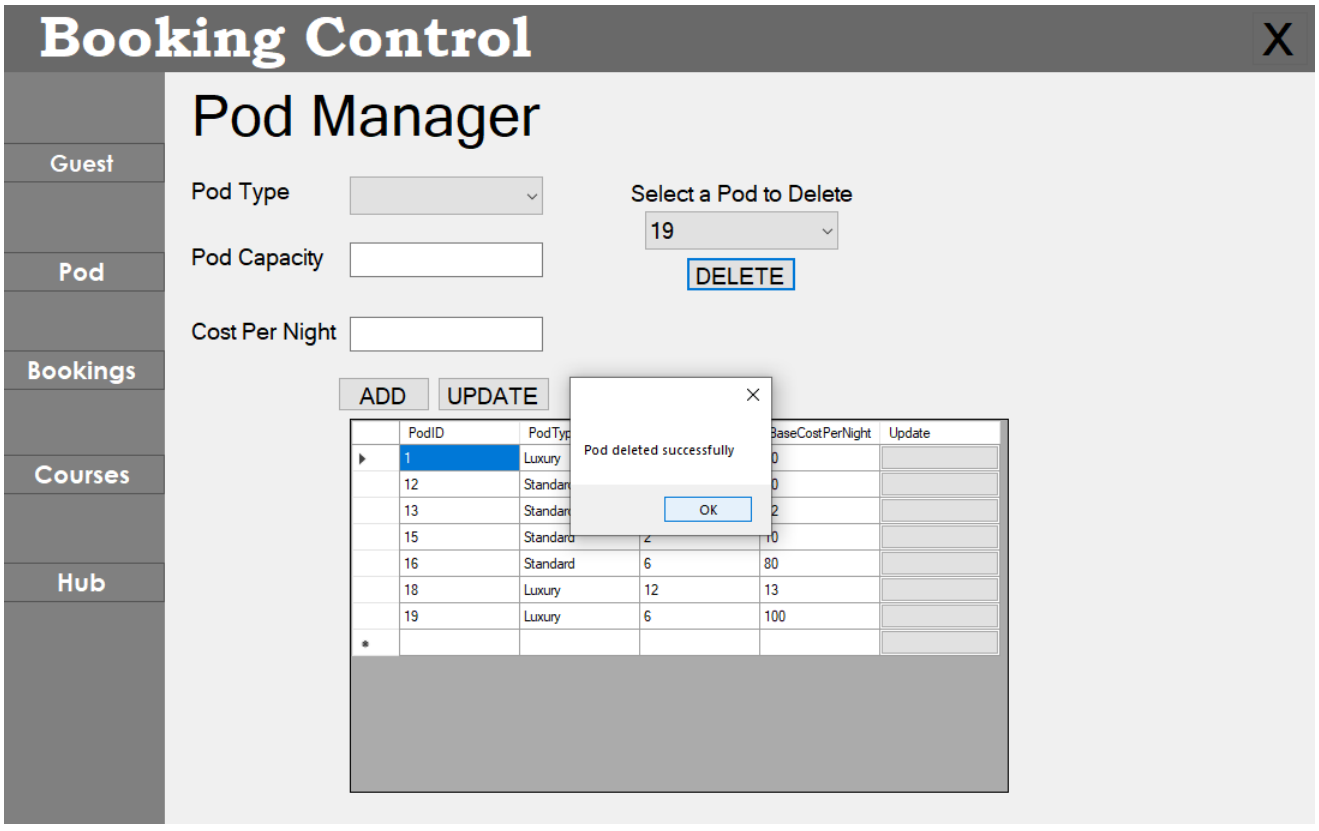


The pods capacity has been updated to 6 in the database, all data entry stipulations still apply even during the updating phase of a pod.

Deleting Pods

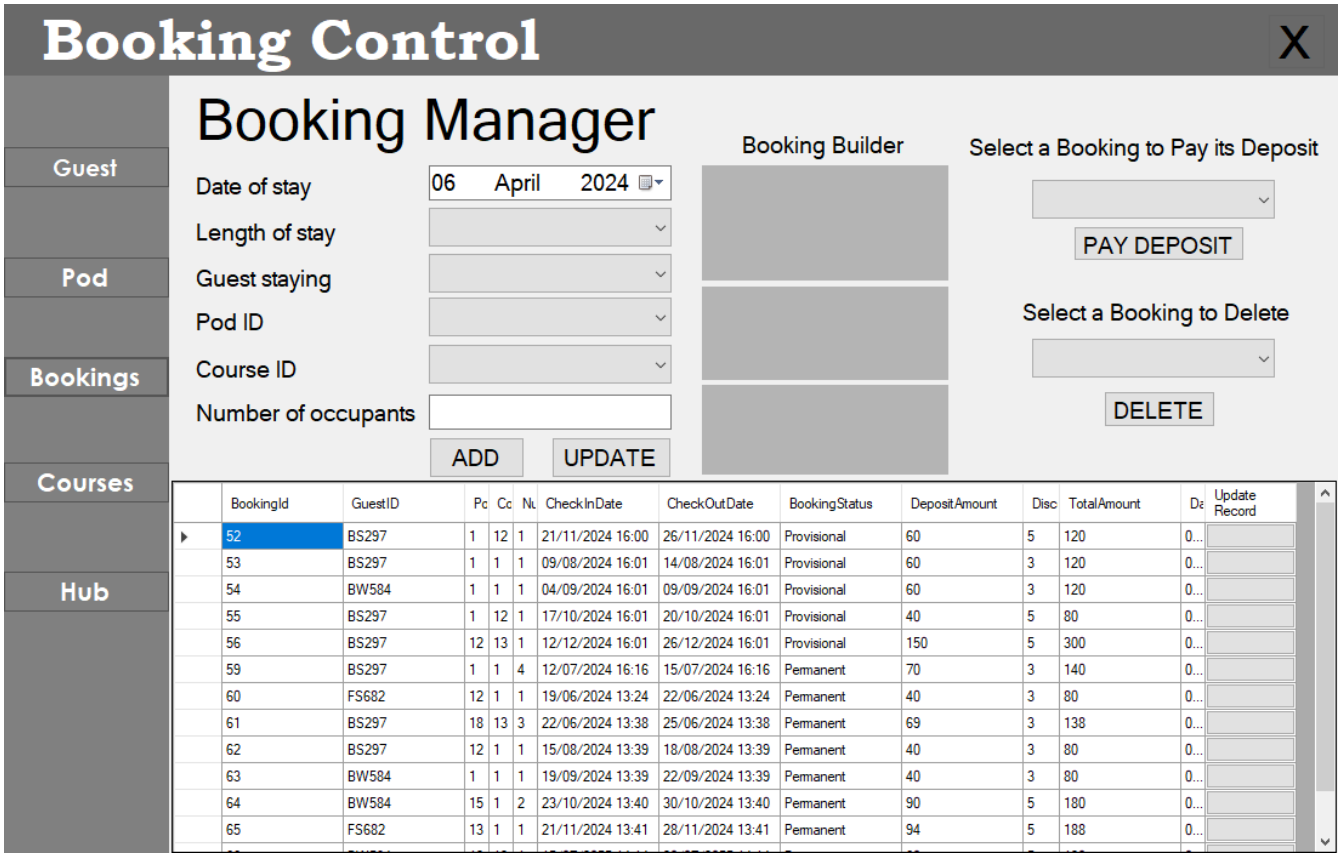


To delete a pod the user would navigate to the combo box titled select a pod to delete after choosing the pod they would like to delete they then press the DELETE button, this will permanently delete the pod.



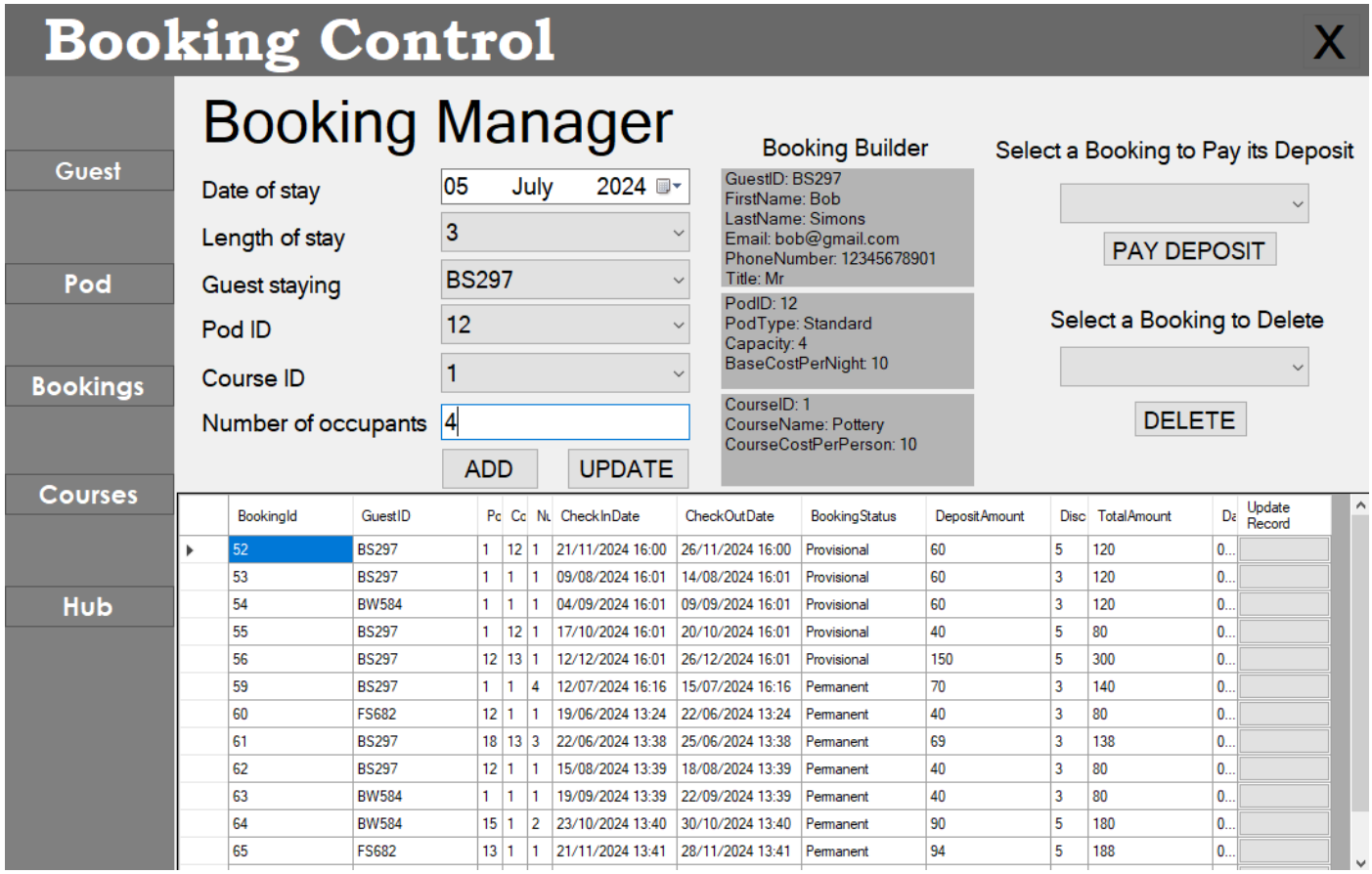
Booking Management

To utilise all functions all related to a booking the user would enter the **Booking Manager**. Pictured below:



Adding new Bookings

To add a new booking the user must first enter the bookings information.

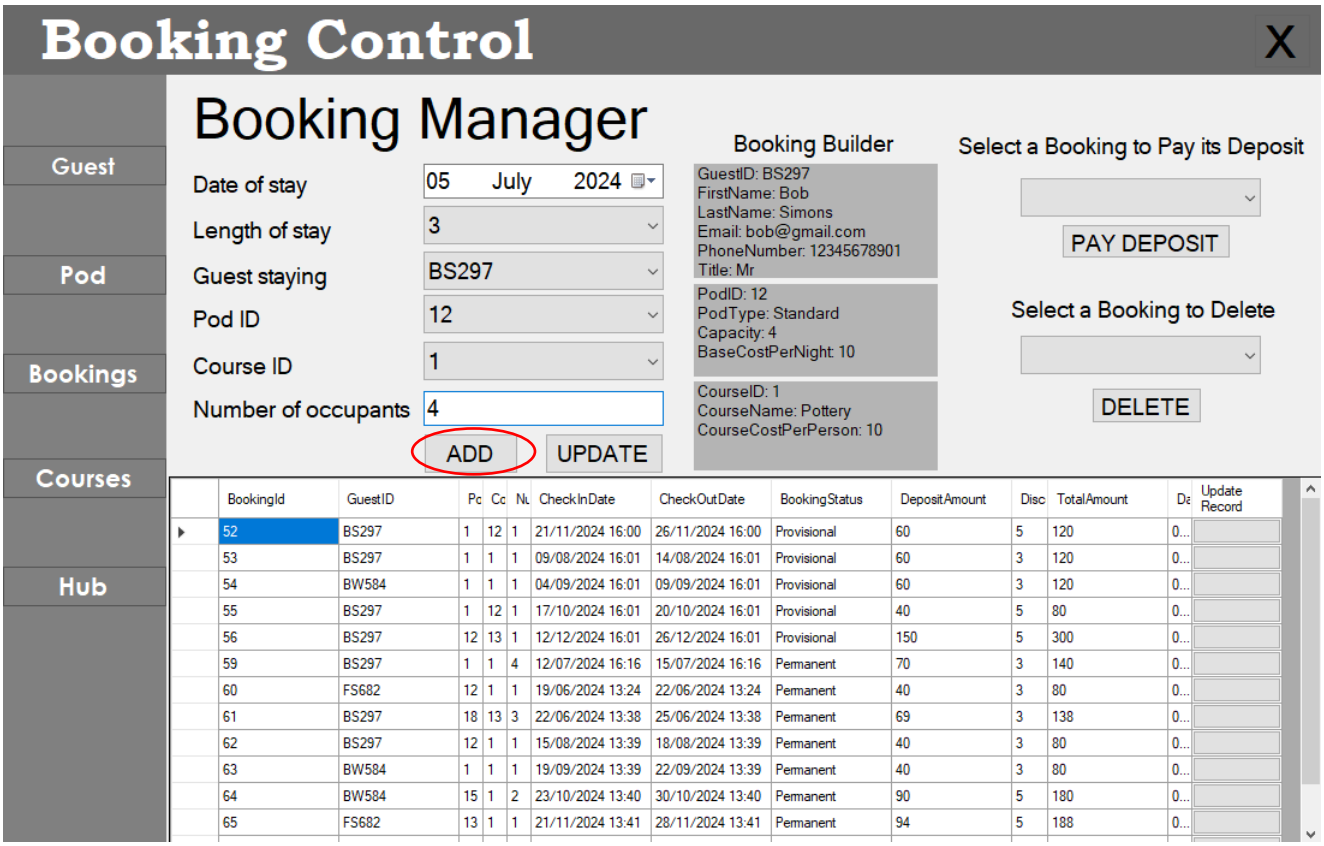


Data Entry Stipulations – Booking Manager

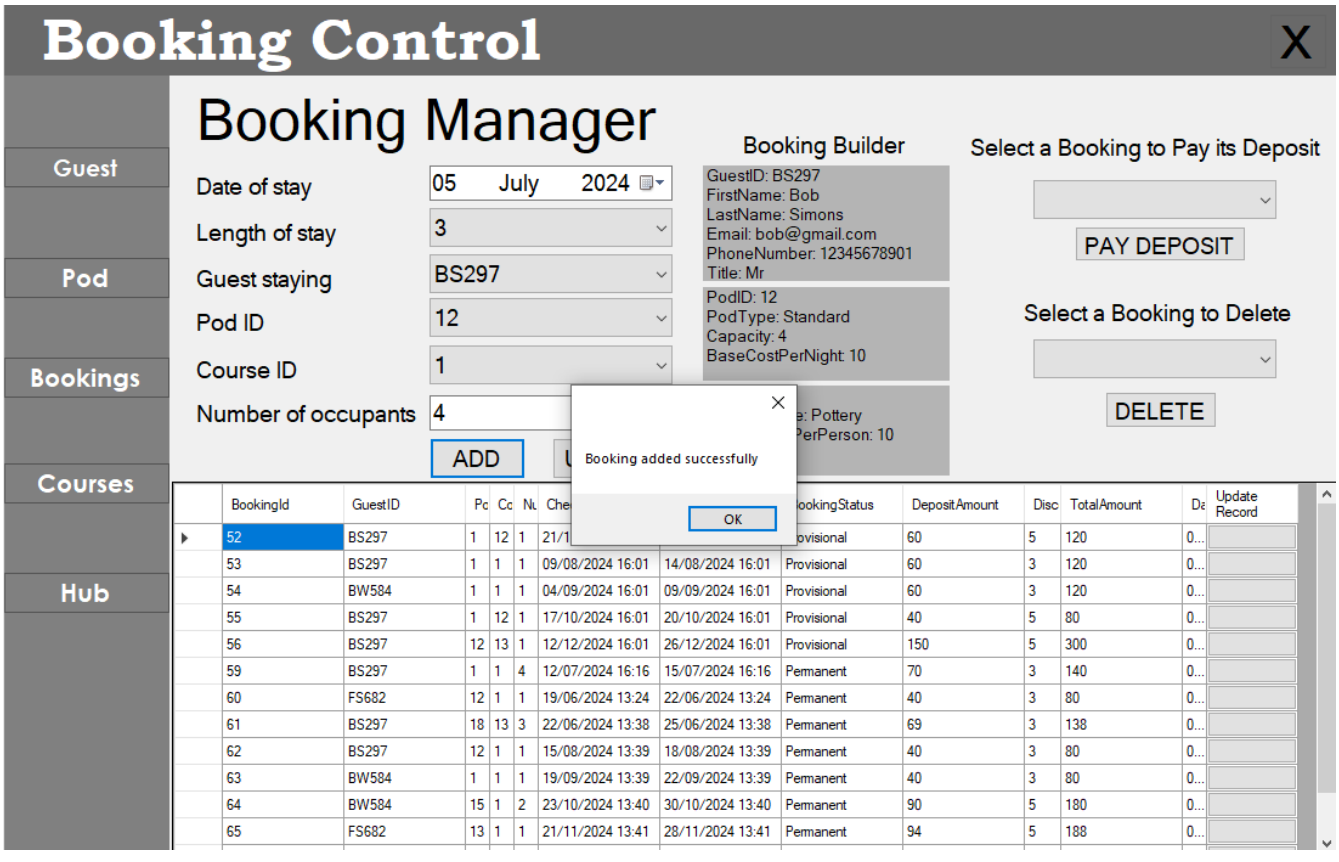
A booking must have all of the following data – a date of stay (At least 2 months in advance of the current date), a length of stay (The duration of stay must not overlap with any bookings or be in the no bookings period of 20th December to the 20th January), a guest staying, a pod ID, a course ID and a number of occupants (Must not exceed pod capacity).

As pictured above the Booking Systems booking builder helps give a visual representation of the current booking being created. This will not only make creating bookings more efficient but also prevent putting the wrong information into the system. Which has been a problem at LakeSide Escapes as seen in the messy booking diary.

ADD button for the booking form

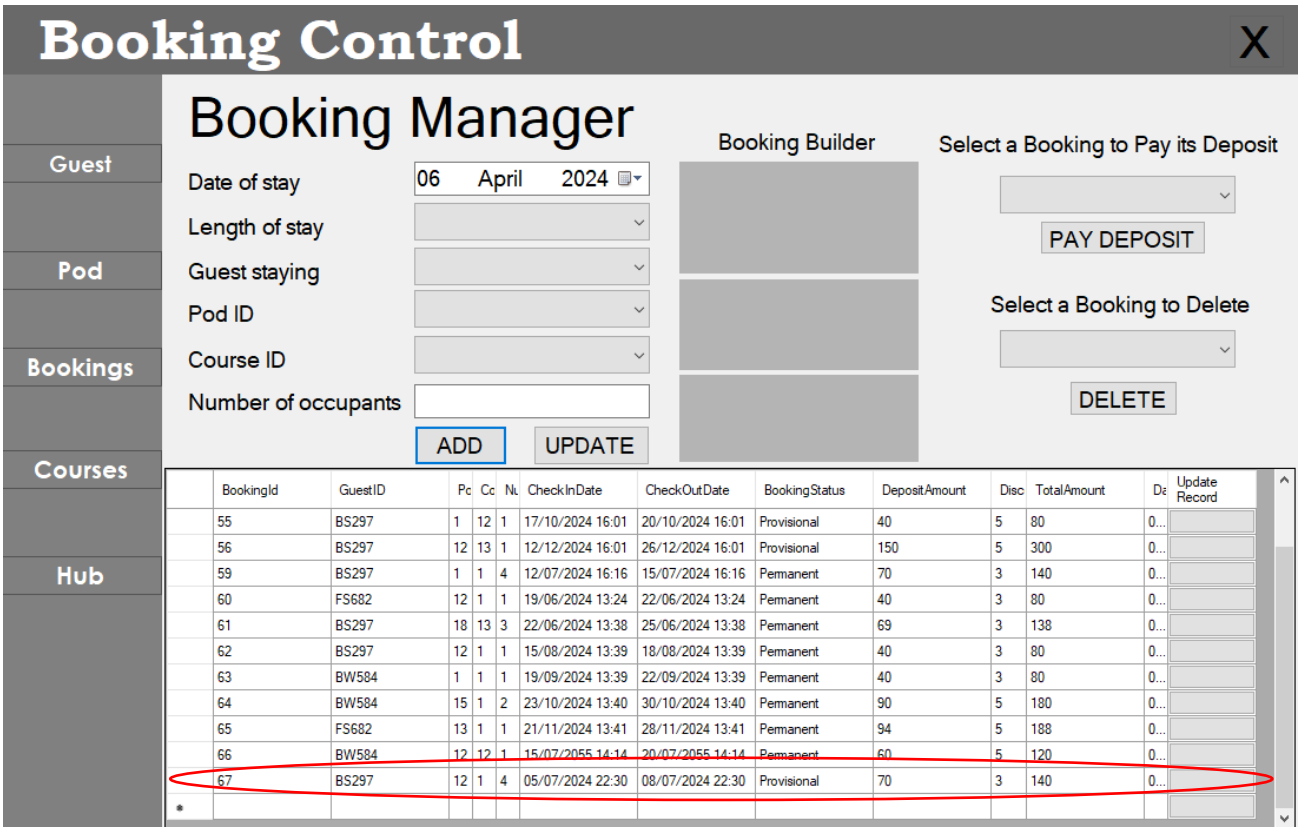


If all data entered is valid the booking will be added to the Booking System



Viewing Bookings

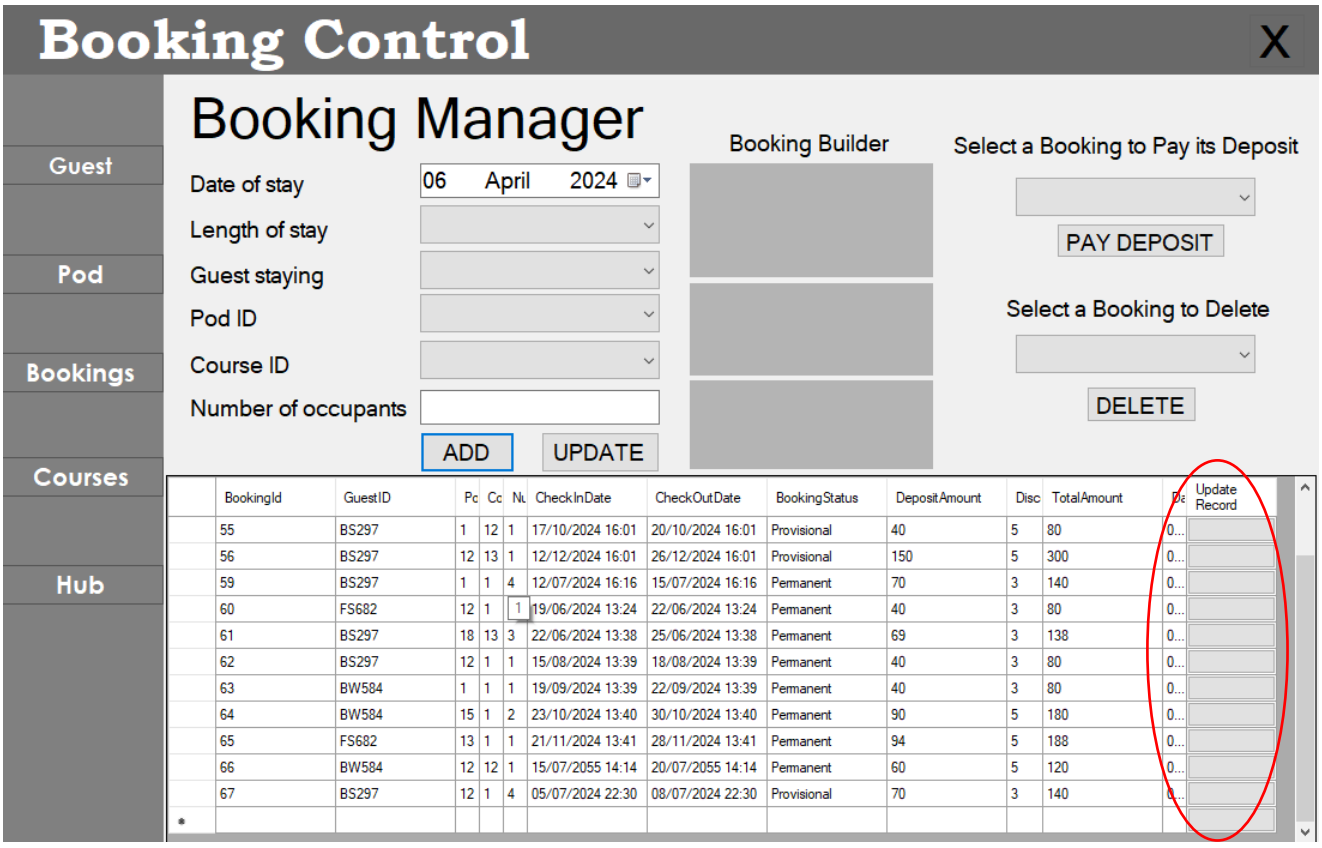
Bookings that are in the system can be viewed by simply looking to the DataGrid View located at the bottom of the screen.



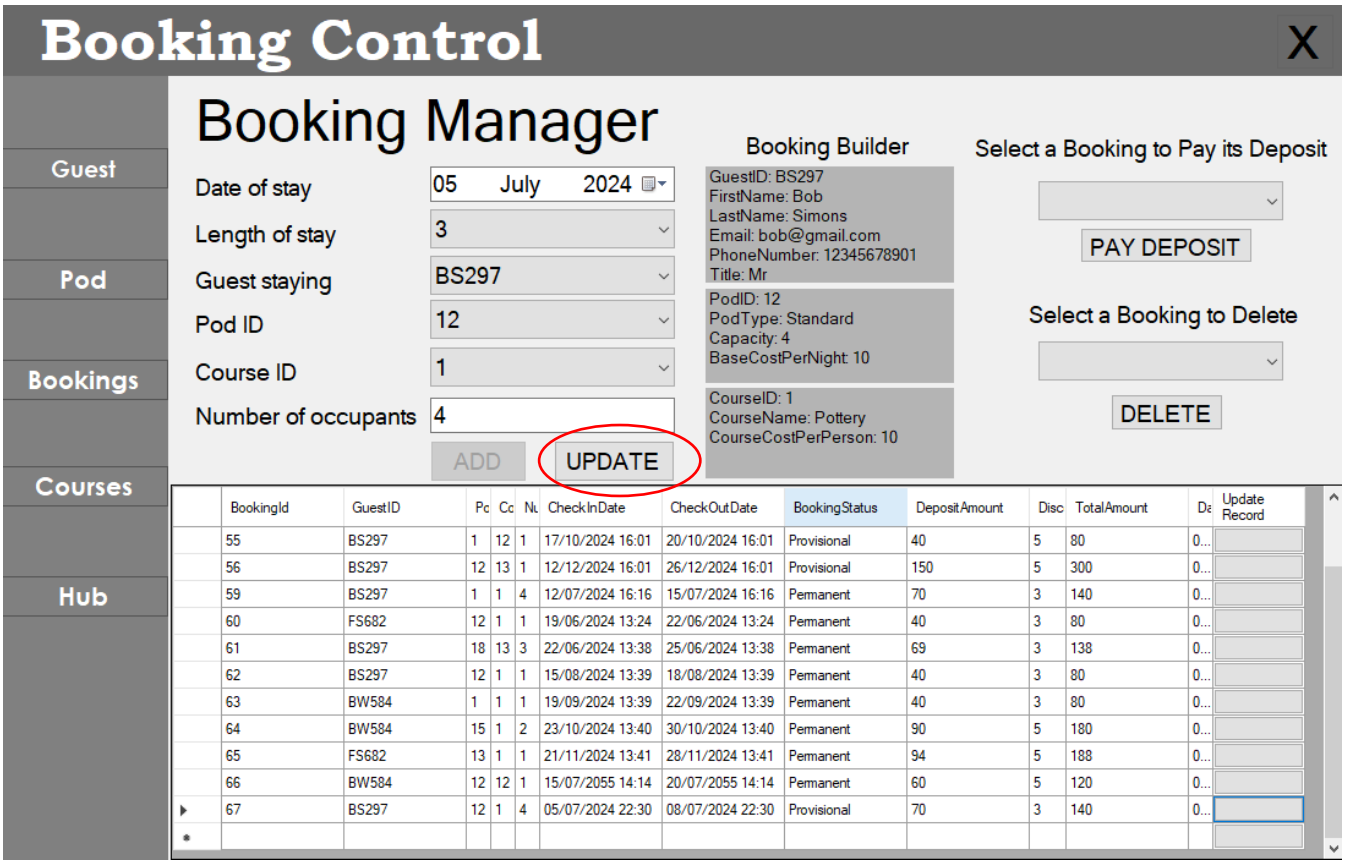
As pictured above the booking has been added to the database and the information can be viewed by staff. The system also creates a unique BookingID for every booking for more intuitive booking identification.

Updating Bookings

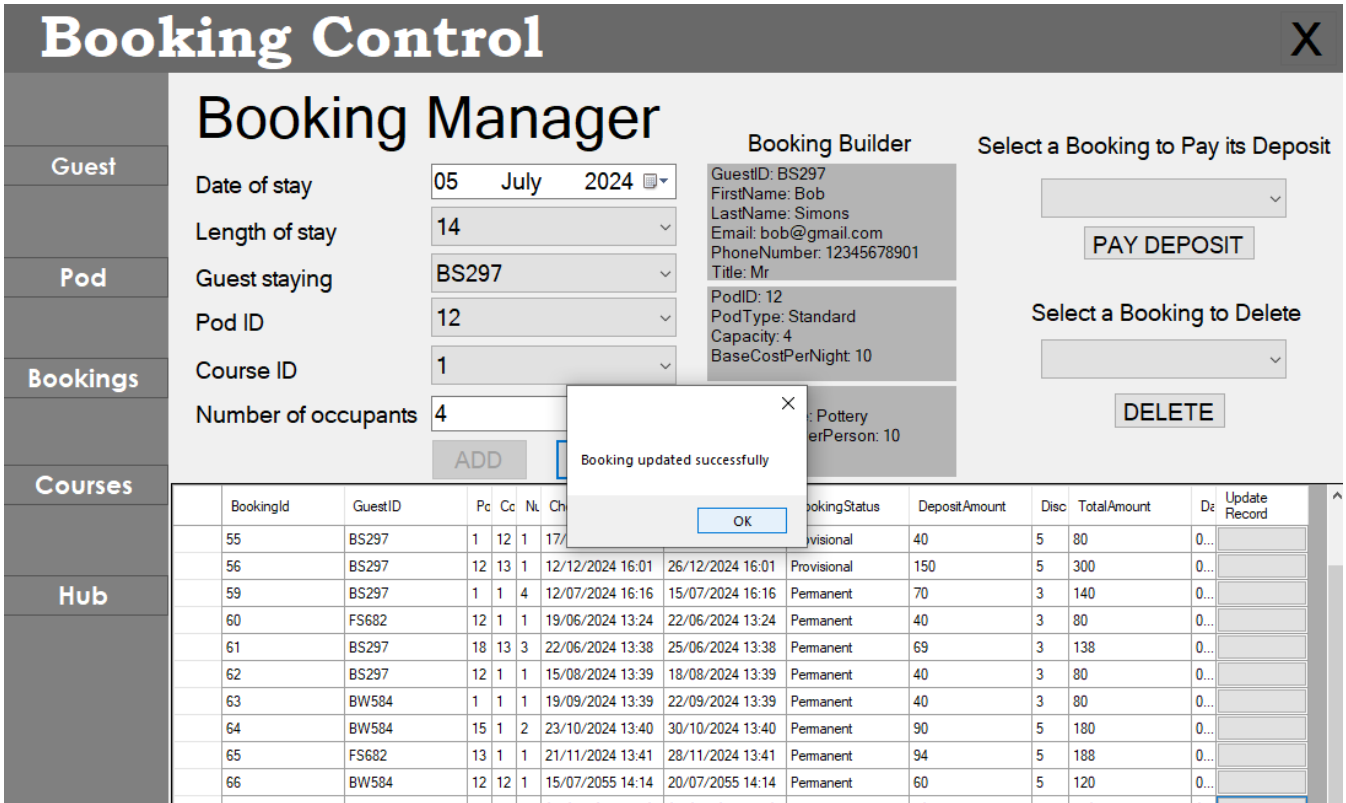
Booking information can be altered by using the update function that has been introduced to the Booking System.



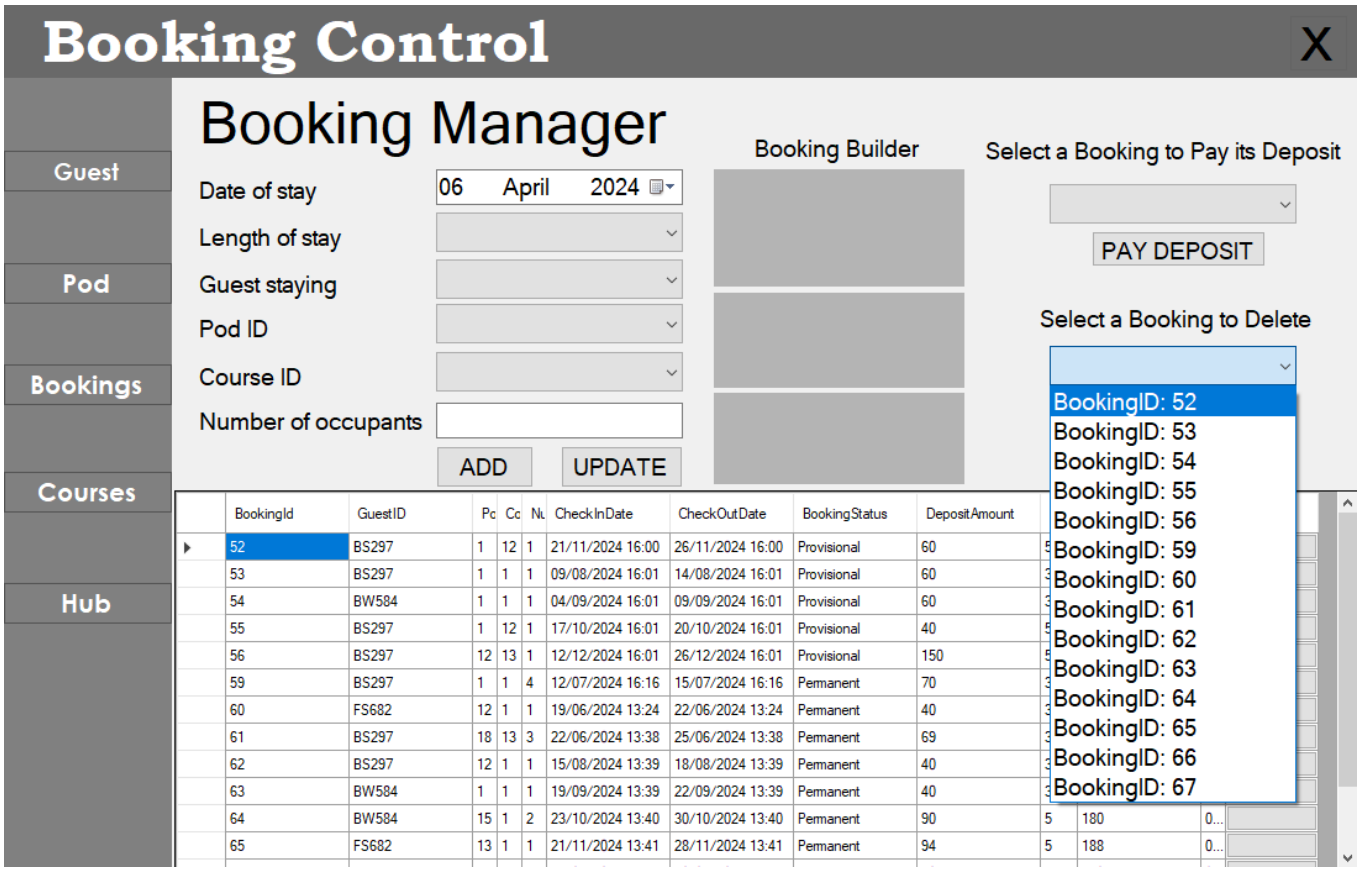
The user first looks for the guest within the DataGrid View at the bottom of the screen and then presses the button titled update beside that booking. This will populate the guest information fields with that particular guests data pictured below.



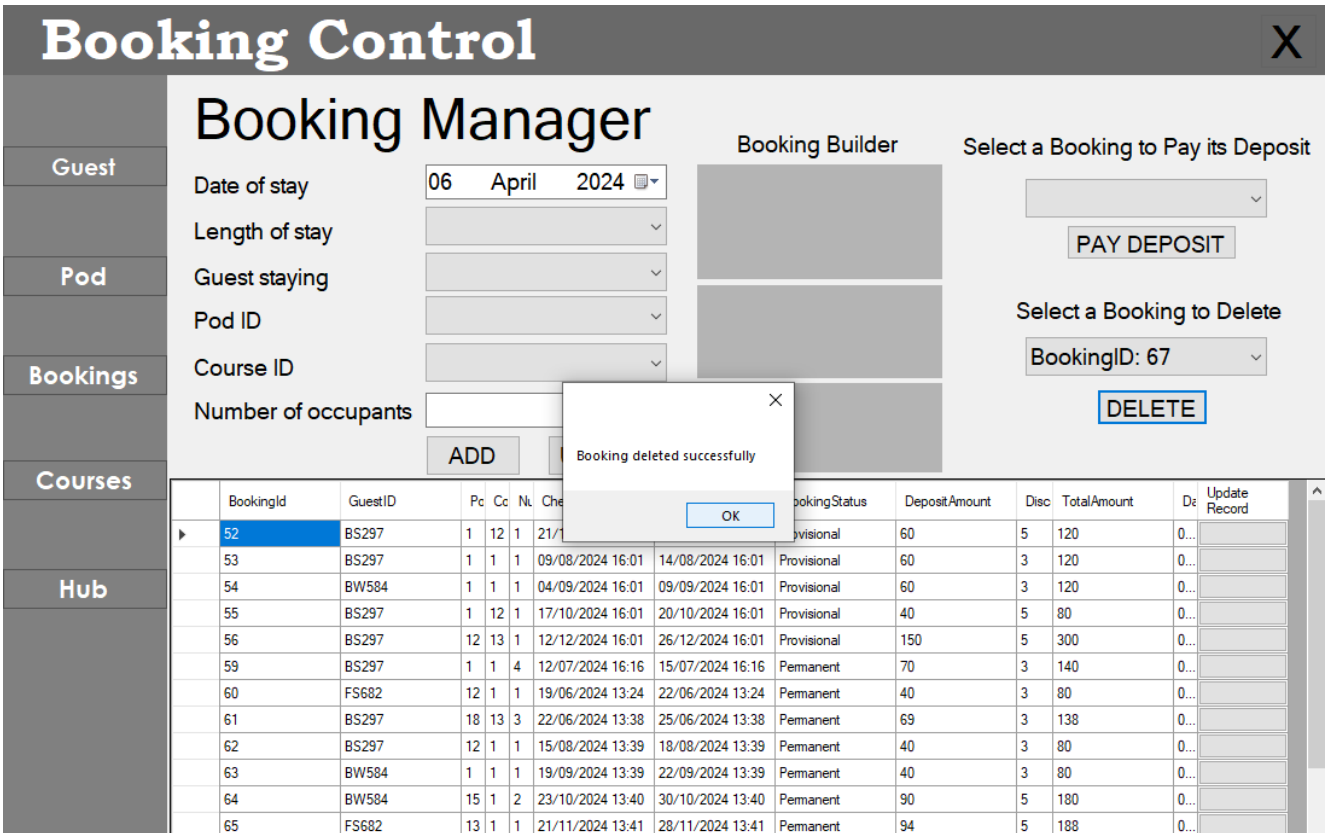
The bookings information can now be edited and upon clicking on the button titled UPDATE in the centre of the screen that bookings information will be changed.



Deleting Bookings

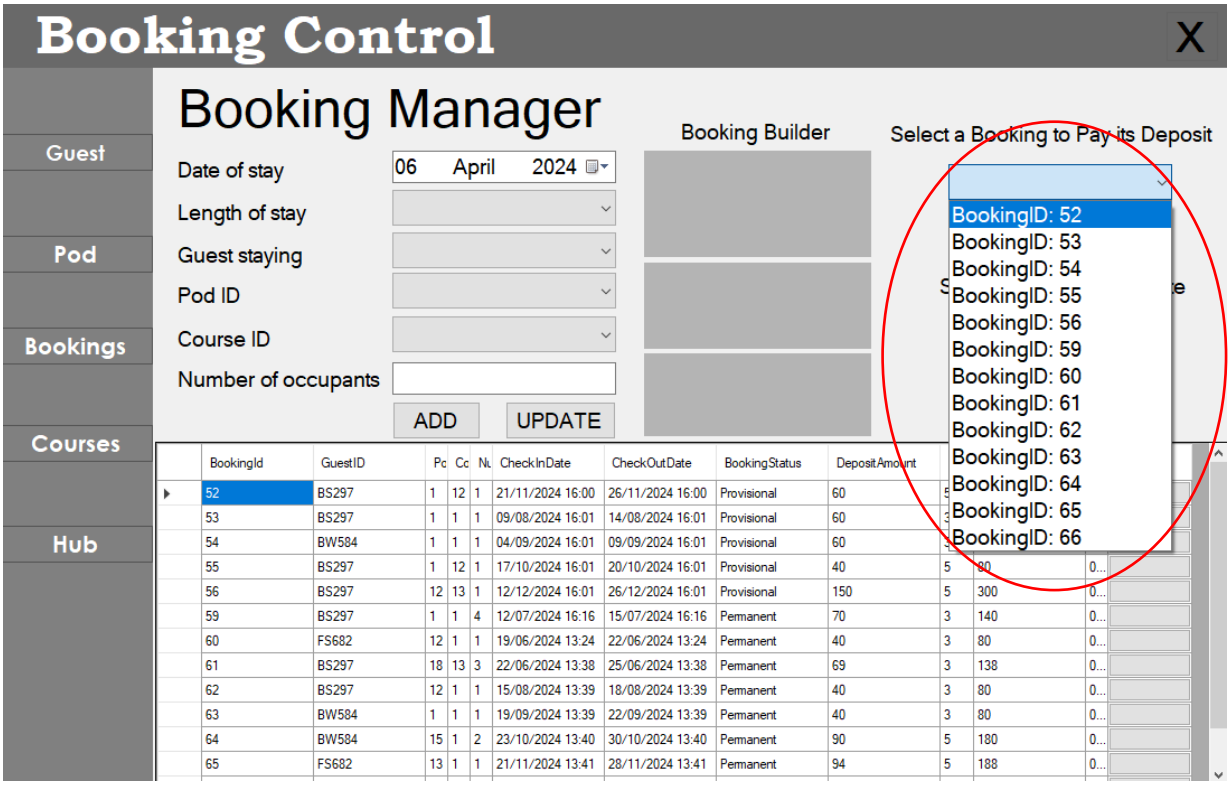


To delete a booking the user would navigate to the combo box titled select a booking to delete after choosing the booking they would like to delete they then press the DELETE button, this will permanently delete the booking.

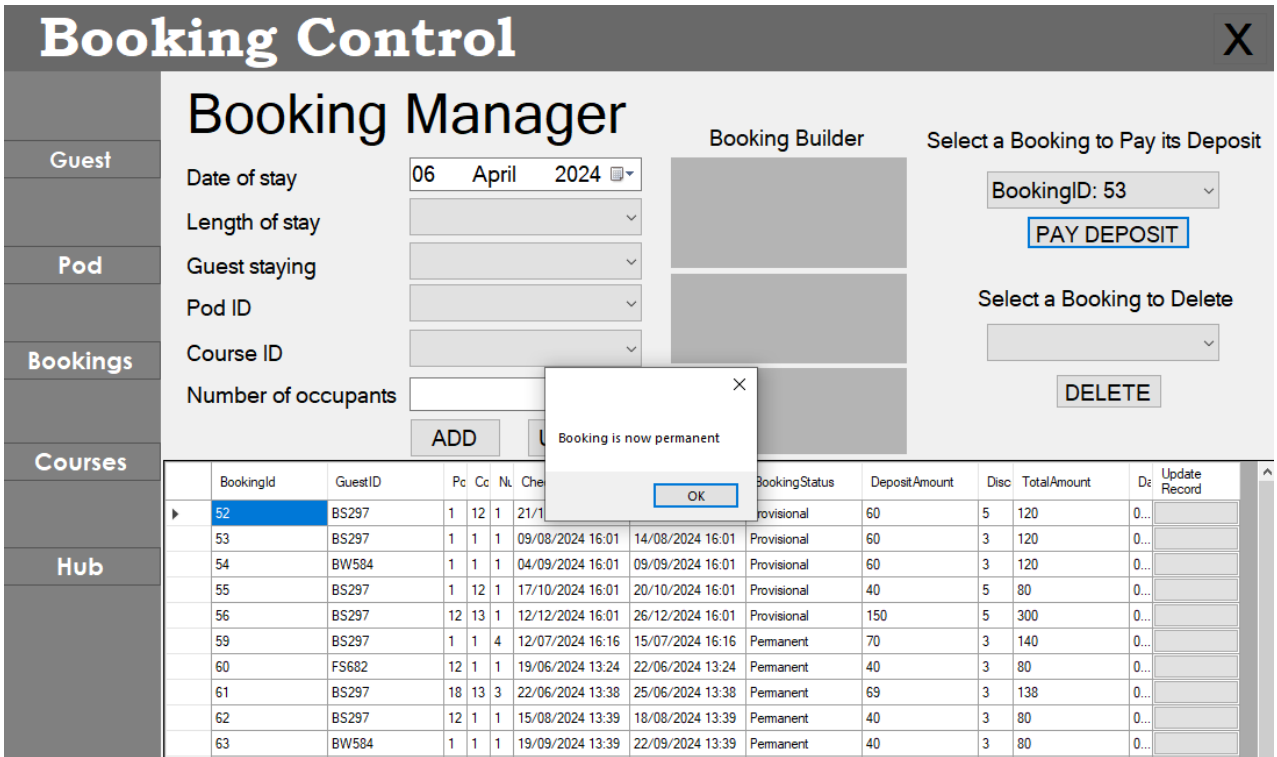


Paying a Bookings Deposit

The booking form has a function to pay a bookings deposit. This function will change the booking's status from provisional to permanent, If this action is not done on a booking within 3 days of the bookings creation the booking will be deleted form the system. The process to make a booking permanent is shown below:

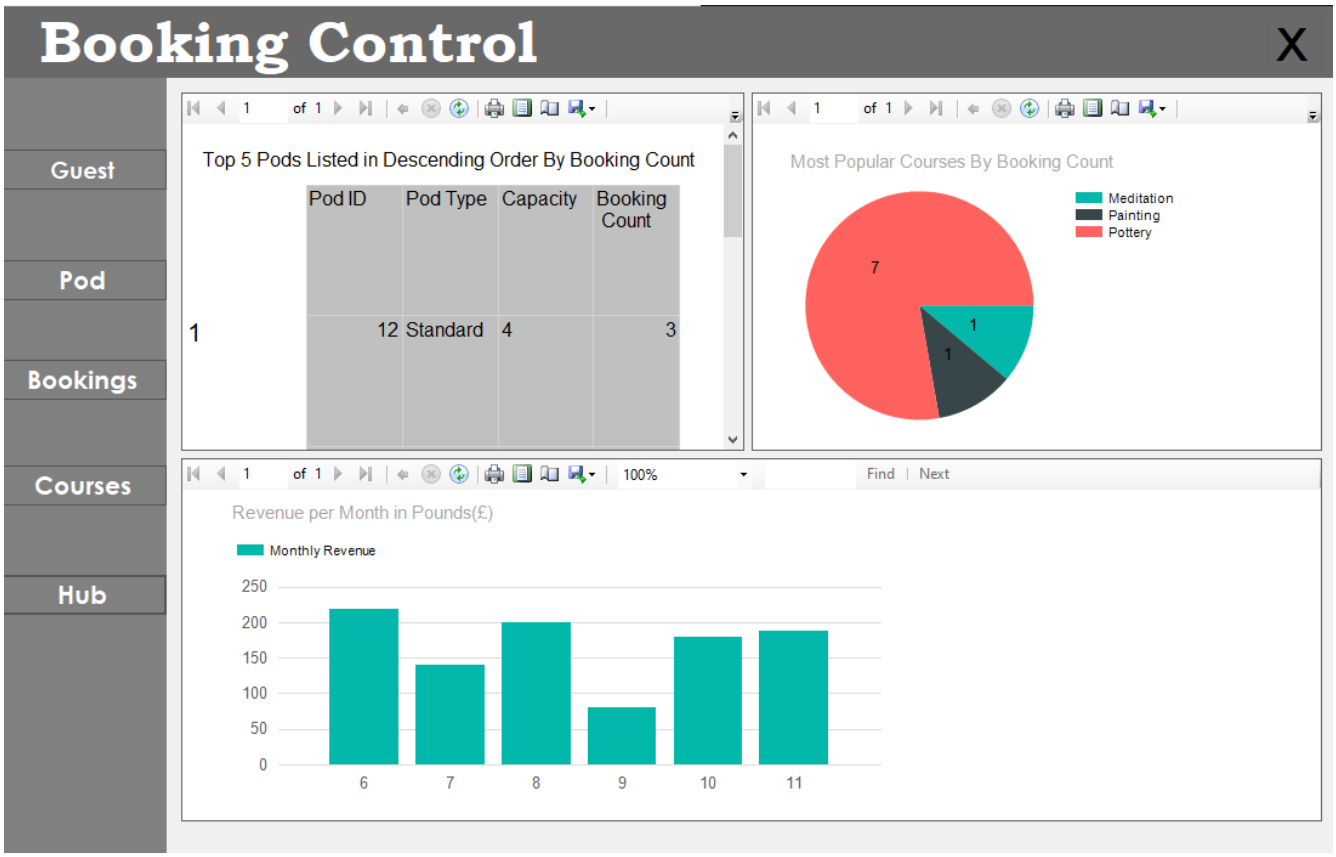


Like the delete booking function the user selects the booking they would like to pay the deposit of and then clicks the button titled PAY DEPOSIT. The booking below is now permanent



Hub Screen

The hub screen is where the management reports are displayed. There are three reports on the Hub Screen one displays The Top 5 Pods by Booking Count, another the Top Courses by Booking Count and the third displays the revenue generated by the business per month of 2024.



Each report has print, save and refresh features available for the user to use and each report is automatically updated to reflect the newest information in the database. **A reminder that only permanent bookings are included in reports.**

If a Guest is deleted will all their bookings be deleted?

Yes, if a guest is deleted from the system then all associated bookings for that guest will be removed from the database.

Can the application window be resized or viewed in full screen?

Currently resizing the window is prevented by the application due to performance and usability concerns. However, this feature may be included in a future update.

Can a Guests ID be changed when editing a guest?

No, when you edit a guest their ID cannot be modified this is to ensure data integrity as the system will create a completely unique ID for that guest preventing user input error.

Are there plans to develop other microsystems?

In the future there will be other microsystems implemented such as a stock control system, a course options system and a billing system. Each section will be modular and compatible with each other as this Booking System serves as a general blueprint.

Can a Booking be recovered if it’s deleted?

No, currently bookings cannot be recovered if they are deleted.

Can a Guest be recovered if their deleted?

No, currently guests cannot be recovered if they are deleted.

Video Walkthrough

The provided folder of the complete application includes a video walkthrough. It will give you a better understanding of how the system functions and it is recommended that you watch it in its entirety before using the application.

Back-up Options

It is important to perform regular backups of data to prevent data loss. This is especially important in a system that handles any form of payment as this Booking System does. If data lose were to occur than the LakeSide Escapes business would have to halt operations and manually renter all necessary information into the Booking System, this would be very time consuming. However, with a backup of the database this can be prevented and the data can be recovered instantly reducing any downtime in business operations.

With a back-up option not being outlined in the user requirements it was deemed appropriate for the end user, LakeSide Escapes, to handle the back up procedures of the system. The recommended option to back-up this Booking System is to copy the ‘‘BookingDatabase.mdf’’ file and upload it to a cloud based server such as google drive or one drive. It is recommended that this is carried out every few days and possibly daily in busier times of the year to prevent larger amounts of data loss. This will ensure that the Booking Systems data integrity is preserved.

It may also be noted that if the LakeSide Escapes company wants an automated service to upload the database, this can be introduced in further updates of the Booking System.

Support Options

This system has a reliable support team. The System has been programmed to be as error free as possible. However, if issues do arise there are a number of contact options listed below:

Phone – 07896721333

Email – LakeSidesupport@gmail.com

Website – www.lakesideescapes/support

FAQ – See above

The support team aims to reply to all emails in a timely manner provided they are sent during regular business hours. If urgent support is needed an emergency telephone number has been listed above

Evaluation

Range of Approaches

After thorough research into a range of approaches I felt that a C# system built with Windows Forms, including a locally hosted SQL database was the approach most fit to purpose.

This is because with the intended solutions language being C#, a well-documented programming language, I would better be able to troubleshoot and fix issues with the codebase. Another reason for C# is that it is an efficient language to program in as it harnesses performance optimization techniques, has an inbuilt garbage collector and is a type-safe language with dynamic capabilities. This becomes important from a business perspective as a more efficient program allow for less expensive machines to be needed to run them; saving LakeSide Escapes money.

The chosen IDE is Visual Studio as it holds Windows Forms, allowing for the creation of an intuitive menu driven system that would be user-friendly, hence relating to user requirement 22. Microsoft's Visual Studio IDE also offered the compatibility of an SQL database that was locally hosted. A locally hosted SQL Database gives the client full control over the data allowing them to change how they utilize the system. A locally hosted database also allows for better data security. Therefore, there will be less issues raised surrounding GDPR due to the data being more protected. However with the database being stored locally large amounts of data may be lost if the operating machine is damaged or if the database has not been backed up properly.

Retrospectively, the Booking System could have been made more fit for purpose if the solution of a Web-based application with a hosted oracle database was chosen. This is because despite the security concerns of guest's data being held online, the database could have updates remotely and automatically back its data up to the cloud preventing data loss. A Web-based application could also be used on any device with an internet connection thus the end user would have more freedom on how they use their system. However, with the LakeSide Escapes Hotel being in the countryside the online connection that's available may not be fast enough to keep up with a large workload, this may lead to data loss.

User Requirements

After conducting a thorough analysis and assessment of the LakeSide Escapes Hotel Case Study, I was able to create clear and concise user requirements for the Booking System. The user requirements were split into two sets with one being the functions of the system and the other being the more subjective non functional parts of the system. In summary, I believe that the requirements of the LakeSide Escapes Hotel were fulfilled. However, the more subjective non functional requirements where shown to have room for improvement in light of the End User Design Feedback Form.

While carrying out the requirements it was important to ensure that 'Requirement creep' did not occur or was reduced as much as possible. This became especially present with the more open ended requirements like "The application must have effective programming to minimise load times and system requirements". This is because after creating a functioning development time had to be spent refining it and making it as effective as possible.

Functional			
User Requirement	**Description**	**Fulfilled?**	**Evidence**
UR1	Effective programming to minimise load times and system requirements	YES	I have used efficient coding techniques throughout the application, to the best of my ability
UR2	The application must have a main menu screen	YES	The application has a menu driven system utilising panels throughout
UR3	The application must use an SQL based database to store data	YES	See BookingDatabase
UR4	The application must allow users to add, edit and delete bookings	YES	See Booking Manager
UR5	The application must allow users to add, edit and delete pods	YES	See Pod Manager

UR6	The application must allow users to add, edit and delete courses	YES	See Course Manager
UR7	The application must allow users to add, edit and delete guests	YES	See Guest Manager
UR8	The database should store guest information, including contact details and create a unique guest ID	YES	See BookingDatabase. A unique Guest ID is generated using guest initials and a random number.
UR9	The database should store Pod information such as price and maximum occupancy	YES	See BookingDatabase
UR10	All bookings must be visible to staff so that they understand the true occupancy of the hotel	YES	All bookings are visible to staff they are displayed in a DataGrid view in the booking screen
UR11	The application must allow customers to reserve pods for specific dates, indicating the start and end of their stay	YES	The application has a check in date and length of stay feature fulfilling this requirement
UR12	The application must perform validation checks to ensure accurate data entry, preventing errors in customer information and reservations	YES	The application safeguards against data entry errors by displaying error messages and preventing functions from taking place with erroneous data
UR13	The application must not be vulnerable to cyber-attacks (SQL injection)	YES	The system uses stored procedures to prevent SQL injection attacks and the system is offline preventing online attacks

UR14	The system should automatically apply eligible discounts to the total cost of the booking reservation	YES	The system applies discounts to guest totals automatically based on the input information
UR15	The system should update pod availability preventing double booking in the same pod	YES	The system has safeguards in place to prevent pod double bookings
UR16	The user must be able to close the application from anywhere in the application (And not have to use the IDES Stop function)	YES	A dedicated close button has been added to the application. See design storyboards
UR17	The system must have a reporting system in order to create structured pod, courses and monthly revenue reports, that are based on certain criteria or parameters, that must update automatically	YES	Reports have been add that create structured pod, courses and monthly revenue reports. These reports reflect the most up to date version of the database
UR18	Normalised database to prevent data redundancy.	YES	See BookingDatabase or normalisation outlined in writeup
UR19	When a guest is deleted, all associated bookings must be deleted in order to maintain data integrity	YES	A deleted guest will delete all bookings that had that guest on it increasing data integrity

UR20	When a Pod is deleted, all associated bookings must be deleted in order to maintain data integrity	YES	A deleted Pod will delete all bookings that had that course on it increasing Pod data integrity
UR21	When a Course is deleted, all associated bookings must be deleted in order to maintain data integrity	YES	A deleted course will delete all bookings that had that course on it increasing data integrity
Non Functional			
User Requirement	**Description**	**Fulfilled?**	**Evidence**
UR22	The application must have an aesthetic and intuitive user interface	No	After the results of the End User Design Feedback form it was evident that some screens were too cluttered
UR23	The application should be stable and be able to handle numerous bookings per day without crashing	YES	Multiple bookings can be made per day with no issues
UR24	The application should be reliable and if a failure would occur no data loss should result	YES	The application is as error free as possible with safeguards in place to prevent errors from occurring
UR25	The system should be modular and well-documented to facilitate future enhancements and maintenance	Yes	The systems write up and user guide are detailed enough to ensure that users can understand the system. The system is modular.

As shown above the requirement not met was "The application must have an aesthetic and intuitive user interface". This was in light of the End User Design Feedback Form results.

Testing

Extensive testing served as a cornerstone in ensuring the robustness and reliability of the Lakeside Escapes booking system. This encompassed meticulous examinations, including data capture testing, link testing, and comprehensive user acceptance testing. Recognizing the pivotal role of testing in refining the application, significant time and effort were dedicated to this phase. Testing not only facilitated the identification and rectification of errors but also ensured alignment with the prescribed user requirements, thereby enhancing the overall quality and functionality of the solution.

However, the solitary nature of the testing process introduced certain limitations, notably in terms of time constraints and the scope of testing. The sheer complexity of the application needed a more exhaustive testing regimen which is difficult to achieve single-handedly. In hindsight, leveraging third-party testing services specialized in applications akin to the booking system could have offered a more efficient and comprehensive testing approach. This would have afforded more time to focus on development, consequently enhancing the application's intuitiveness and feature set.

The testing strategy commenced with the formulation of a meticulous test plan following the V-Model framework, renowned for its systematic approach to software testing. This choice stemmed from its industry-standard status and its efficacy in identifying software discrepancies early in the development cycle. Despite its merits, the rigidity of the V-Model occasionally posed challenges, particularly in accommodating the agile nature of the development process. Iterative testing strategies were thus integrated throughout the development lifecycle to ensure ongoing alignment with user requirements and feature implementation.

In retrospect, adopting a more granular approach to testing by evaluating each function incrementally could have yielded more efficient results. This iterative testing approach, coupled with a focus on testing as features were developed and refined, would have minimized time spent revisiting and familiarizing oneself with previously coded segments. Furthermore, incorporating a more exhaustive testing regimen from the outset would have bolstered the overall effectiveness of the testing process, leading to a more robust and refined final product.

Chosen Design Methodology

The selection of the Rapid Application Development (RAD) methodology proves effective in addressing the diverse set of user requirements outlined for the Lakeside Escapes booking system. RAD's iterative nature aligns well with the need for effective programming to minimize load times and system requirements (UR1), as it allows for continuous refinement and optimization throughout the development process. Moreover, the emphasis on rapid prototyping enables the timely delivery of a main menu screen (UR2) and ensures the utilization of an SQL-based database (UR3) for efficient data storage and management.

In the absence of real end-users, RAD's flexibility becomes paramount in discovering how well the application meets user requirements. While direct user involvement may be limited in a fictional scenario, RAD's iterative cycles facilitate adjustments based on the development team's interpretation of user needs and feedback obtained during testing phases. This adaptability proves crucial in accommodating changes and ensuring the application's alignment with evolving user expectations, as highlighted by requirements such as the ability to add, edit, and delete bookings, pods, courses, and guests (UR4-UR7).

Despite the absence of real end-users, RAD remains an appropriate choice given the special circumstances of a fictional end-user. Its flexibility allows for midstream adjustments to be made quickly, ensuring that the application remains responsive to changing requirements and project dynamics. This is particularly evident in requirements such as the validation checks to prevent errors in customer information and reservations (UR12) and the system's resilience against cyber-attacks (UR13), where ongoing refinement and testing are essential to maintain robustness and security.

The impact of the chosen design methodology on the end product is significant, as RAD's iterative approach fosters a user-centric development process. By prioritizing active stakeholder involvement and continuous feedback loops, RAD ensures that the application not only meets but surpasses user requirements. For instance, the system's ability to automatically apply eligible discounts (UR14) and update pod availability to prevent double bookings (UR15) reflects the responsiveness and adaptability facilitated by RAD's iterative cycles.

The Scrum design methodology could have also been appropriate for the Lakeside Escapes booking system due to its emphasis on adaptability and collaboration. In the context of a fictional end-user, where requirements may change dynamically, Scrum's iterative approach allows for frequent reassessment and adjustment of priorities.

If I was to imagine a custom methodology tailored specifically for Lakeside Escapes, key aspects such as iterative development, user involvement, flexibility, collaboration, and documentation would be paramount. By integrating these principles, the development process would be optimized to deliver a solution that not only meets but exceeds user expectations, ensuring a seamless user experience and long-term viability for the booking system.

Project Management Techniques Used

During the development of the LakeSide Escapes Pod Booking System, time utilization was reasonably efficient. The Gantt chart provided a clear visual representation of project tasks and timelines, aiding in effective planning and resource allocation.

To enhance system efficiency, I could have conducted a thorough risk analysis at the outset to identify development pitfalls early on in the project lifecycle. While the Gantt chart provided a structured timeline, a comprehensive risk assessment could have identified potential bottlenecks, dependencies, and external factors that could impact project delivery. By proactively addressing these risks, I could have implemented contingency plans and mitigation strategies, thereby reducing the likelihood of delays and ensuring smoother project execution.

The delays in delivering the LakeSide Escapes Pod Booking System and its associated documentation stemmed from various factors. Firstly, the inherent complexity of tasks involved in developing a comprehensive booking system led to unanticipated hurdles and prolonged task completion. Additionally, the need for frequent updates to project management tools, such as the Gantt chart, consumed valuable time and resources as requirements evolved. Moreover, the iterative nature of software development, coupled with the adoption of agile methodologies, introduced challenges in maintaining project stability and predictability. Addressing these challenges requires proactive risk management, robust project planning, and effective communication to mitigate delays and ensure timely delivery of the project. Progress was monitored primarily through the Gantt chart, which provided a visual overview of completed tasks, current status, and upcoming milestones. Regular monitoring of this and the PERT chart ensured a timely delivery of the project.

In future projects, considering a blend of project management techniques could be advantageous. While the Gantt chart offered simplicity and clarity, incorporating elements of PERT charts for critical path analysis and time estimation could provide a more comprehensive view of project dynamics. Additionally, I could explore project management monitoring software such as Kanban or Jira. Kanban is a visual project management technique that emphasizes continuous delivery and workflow optimization. Tasks are represented as cards on a Kanban board, categorized into columns based on their status (e.g., to do, in progress, done). This would allow me to visualize my workflow, identify blockers, and reprioritize tasks accordingly.

My Booking System Solution

In terms of providing a solution for LakeSide Escape, I believe my implementation has surpassed expected functionality.

I believe my system meets the intended functionality for the Lakeside Escape Hotel for a number of reasons. the Lakeside Escapes Pod Booking System requirements outline the booking process and management procedures for the hotel's pod accommodations. My implementation ensured that Bookings are restricted between December 20th and January 20th, with options solely available for stays of three, five, seven, or fourteen days. This was enforced by the Validation class and predefined options in the combo box UI components.

Similarly Guests can make provisional bookings, which become permanent upon payment of a 50% deposit. This ability was facilitated by the BookingDal as it interfaced with the database to set bookings as permanent when the appropriate Boolean flag was checked (i.e. deposit was marked as paid in the UI). The deposit amount is determined by factors such as pod type, duration of stay, number of occupants, and selected courses, with discounts offered for repeat guests and early bookings. The validation class ensured this by extracting data from each data access layer and mathematically generating the booking costs and deposit in encapsulated algorithms. Similarly, if the deposit was not paid within the designated timeframe, this automatically triggered the delete query to release these dates, making them available for future bookings.

Moreover, the use of a data grid view ensured that all bookings, both provisional and permanent, were recorded in the Pod Booking Diary at least two months in advance to allow staff to manage course numbers and activities.

I also implemented data visualisation strategies that the existing Lakeside Escape solution did not provide to its staff. These were in the form of RDLC reports which documented the most popular pods, most popular courses and a breakdown of revenue generated per month. This efficient access to data allows staff to easily upsell services that the hotel provides and allows them to invest more money into elements of the business that are ranking highly from the outputted statistics.

In terms of usability, I believe my system is easy to navigate and with a UI and UX which is intuitive. The vibrant colour scheme but simple design allows the staff to engage with the system and efficiently carry out their processes. This was evident from the survey results ranking the usability on average as 4 out of 5. This is a great improvement from the current diary system which was described as disorganized and challenging to navigate, hindering visibility of bookings and availability. I believe that I have removed all of these

challenges in the new system, however, if I was to complete a future iteration of the system, I would consider creating more forms to separate some of the information up. Although I believe having all related information such as booking and payments on the same screen is very practical and easy to manoeuvre, it may be better from a visual perspective for each form to be less cluttered.

I definitely think that this system is scalable. By hosting the database on the cloud and distributing the performance over multiple PODS – this solution could easily support multiple user access. This would be a great benefit if there is a need for lots of staff members to be concurrently using the system. However, at this point, I do not think this is necessary and the cost of cloud deployment may outweigh this need as Lakeside Escape is such a small business. Although, this option could be considered if the business was to expand into more branches.

My Performance

Overall, as I enjoy programming and have a curiosity for the mechanics and automation behind software, I was enthusiastic to begin development and was feeling positive towards being able to curate my own system for the Lakeside escapes business. I also feel that I was able to preform well and meet milestones that I had set for myself effectively and thus developed an application that I am proud of.

I would rate my performance highly in terms of meeting and actually exceeding user requirements. Throughout the development process, I ensured to refer back to the provide case study on my microsystem. This allowed me to fully understand the needs of the system and therefore create a realistic draft of user requirements to follow. This template allowed me to iterate through each requirement and ensure it was fully met. However, while this approach allowed me to fully meet and in some cases exceed user expectations the large scope of some requirements led to delays in development due to additionally useability being added beyond the set user requirement. Although the majority of user requirements were met in full the application ease of use was questioned after the End User Design Feedback form was received as some staff thought screens were too cluttered.

Documenting the development process was a priority for me, and I believe I have been thorough in doing so. I maintained records of requirements, design decisions, implementation steps, and testing procedures. Additionally, I created comprehensive user manuals and technical documentation to aid in the understanding and maintenance of the application. Non-technical documentation was also included such as UML diagrams this would allow for more business minded members of the Lakeside Escapes team to better understand and use the application.

Meeting deadlines was a primary focus throughout the project, and I am pleased with my performance in this area. I created clear project milestones and timelines, while using time management tools such as Gantt charts. I regularly monitored progress to ensure my tasks were completed on time and on schedule. This lead to program deadlines being met on time effectively in some cases there was time to spare that allowed me further time to refine my code base and make it more efficient. The documentation portion of the project was the most time consuming and involved longer working sessions than I would have been preferred, however with effective schedules milestones were met effectively for this side of the project as well.

My understanding of the required software was thorough, allowing me to select and utilize the appropriate tools and technologies effectively. I conducted extensive research into various options, considering factors such as scalability, compatibility, and ease of use. This enabled me to implement the system fully while leveraging the capabilities of the chosen software. My knowledge of C# was tested heavily when developing the projects Data Access Layers as these were the most complex and time consuming areas to program. My knowledge of the software has improved through the development process and as a result of this knowledge I believe I was able to implement a application of a high standard.

I believe I demonstrated a strong awareness of the skills required to implement the system to the required standard. Drawing upon my knowledge of database management, software development methodologies, and project management techniques, I effectively took on challenges and optimized solutions to meet project objectives. Continuous learning and skill development were employed as I knew this was a factor that would allow me to release an application of the highest standard for the Lakeside Escapes business.

Looking back on this project, there are several areas where I would approach things differently in future projects. Firstly, I would prioritize collaboration with the end-users at Lakeside Escapes from the start to ensure a more iterative development process and create a more tailored application. Additionally, I would explore opportunities to automate certain tasks, such as documentation generation and testing procedures, to improve efficiency and streamline the solutions workload. Finally, I would invest more time in thoroughly researching and selecting software tools to ensure optimal compatibility and scalability as the project evolves.

Code Repository

BookingViewer.cs (Main Menu screen)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    public partial class BookingViewer : Form
    {
        public BookingViewer()
        {
            InitializeComponent();
        }

        public void loadform(object Form)
        {
            if (this.panelMain.Controls.Count > 0)
                this.panelMain.Controls.RemoveAt(0);
            Form f = Form as Form;
            f.TopLevel = false;
            f.Dock = DockStyle.Fill;
            this.panelMain.Controls.Add(f);
            this.panelMain.Tag = f;
            f.Show();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            loadform(new GuestForm());
        }

        private void podFormBTN_Click(object sender, EventArgs e)
        {
            loadform(new PodForm());
        }

        private void bookingFormBTN_Click(object sender, EventArgs e)
        {
            loadform(new BookingMenu());
        }

        private void courseFormBTN_Click(object sender, EventArgs e)
        {
            loadform(new CourseForm());
        }

        private void hubFormBTN_Click(object sender, EventArgs e)
        {
```

```
            loadform(new HubForm());
        }

        private void closeProgramLBL_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

CourseForm.cs (Course management screen)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    public partial class CourseForm : Form
    {
        public CourseForm()
        {
            InitializeComponent();
        }
        private void UpdateDatabase()
        {
            courseTableAdapter.Fill(courses.Course);
        }
        private void addBTN_Click(object sender, EventArgs e)
        {
            int courseCostPerPerson;
            if (!string.IsNullOrEmpty(courseNameTB.Text) &&
!string.IsNullOrEmpty(corseCostPerPersonTB.Text) &&
int.TryParse(corseCostPerPersonTB.Text, out courseCostPerPerson))
            {
                string[] courseDetails = new string[] { courseIDTB.Text,
courseNameTB.Text, courseCostPerPerson .ToString()};

                int rowsAffected = CourseDal.AddCourse(courseDetails);

                if (rowsAffected > 0)
                {
                    MessageBox.Show("Course added successfully");
                    UpdateDatabase();
                    PopulateComboBoxCourseIDs();
                    courseNameTB.Clear();
                    courseIDTB.Clear();
                    corseCostPerPersonTB.Clear();
                }
                else
                {
                    MessageBox.Show("Fail");
                }
            }
            else
            {
                MessageBox.Show("The criteria to add a course has not been met, please
see handbook.");
            }
```

```
        }
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        private void CourseForm_Load(object sender, EventArgs e)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
                // TODO: This line of code loads data into the 'bookingTable.Booking'
table. You can move, or remove it, as needed.
                this.courseTableAdapter.Fill(this.courses.Course);
                connection.Close();
            }
            PopulateComboBoxCourseIDs();
        }

        private void deleteBTN_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(deleteComboBox.Text))
            {
                if (MessageBox.Show("Are you sure?", "Delete Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    // user clicked yes

                    string IdValue = deleteComboBox.SelectedItem.ToString();
                    int rowsAffected =
CourseDal.DeleteCourseByID(Convert.ToInt32(IdValue));

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Course deleted successfully");
                        UpdateDatabase();
                        PopulateComboBoxCourseIDs();
                    }
                    else
                    {
                        MessageBox.Show("Failed delete");
                    }
                }
                else
                {
                    // user clicked no
                    MessageBox.Show("Delete cancelled");
                }
            }
            else
            {
                MessageBox.Show("Select a course to delete");
            }

        }

        private void PopulateComboBoxCourseIDs()
        {
            // Call the method to get the items
            List<string> items = CourseDal.GetCourseIDs();
```

```
            // Removing "GuestID: " prefix from each string
            var formattedCourseIds = items.Select(courseId => courseId.Replace("CourseID:
", ""));

            // Clear existing items (optional)
            deleteComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in formattedCourseIds)
            {
                deleteComboBox.Items.Add(item);
            }

        }

        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex >= 3)
            {
                object cellValue = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex -
3].Value;
                if (cellValue != null && int.TryParse(cellValue.ToString(), out int
courseID))
                {
                    Courses course = CourseDal.GetCourseObjectByID(courseID);
                    courseIDTB.Text = courseID.ToString();
                    courseNameTB.Text = course.CourseName.ToString();
                    corseCostPerPersonTB.Text = course.CourseCostPerPerson.ToString();
                    addBTN.Enabled = false;
                }
                else
                {
                    MessageBox.Show("The cell does not contain a valid integer value.");
                }
            }
        }

        private void updateBTN_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(courseIDTB.Text))
            {
                if (!string.IsNullOrEmpty(courseNameTB.Text) &&
!string.IsNullOrEmpty(corseCostPerPersonTB.Text))
                {
                    string[] courseDetails = new string[] { courseIDTB.Text,
courseNameTB.Text, corseCostPerPersonTB.Text };

                    int rowsAffected = CourseDal.UpdateCourse(courseDetails);

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Course updated successfully");
                        UpdateDatabase();
                        PopulateComboBoxCourseIDs();
                        courseNameTB.Clear();
                        courseIDTB.Clear();
```

```
                    corseCostPerPersonTB.Clear();
                    addBTN.Enabled = true;
                }
                else
                {
                    MessageBox.Show("The record could not be found");
                    UpdateDatabase();
                    PopulateComboBoxCourseIDs();
                    courseNameTB.Clear();
                    courseIDTB.Clear();
                    corseCostPerPersonTB.Clear();
                    addBTN.Enabled = true;
                }
            }
            else
            {
                MessageBox.Show("The criteria to alter a courses information has not
been met, please see handbook");
            }

        }
        else
        {
            MessageBox.Show("Select a course to update");
        }

    }
}
}
```

GuestForm.cs (Guest management screen)

```
using LaksideEscapesBookingSystem.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    public partial class GuestForm : Form
    {
        public GuestForm()
        {
            InitializeComponent();
        }

        private void addGuestBTN_Click(object sender, EventArgs e)
        {
            string fName = firstNameTB.Text;
            string sName = lastNameTB.Text;
            string email = emailTB.Text;
            string phoneNumber = phoneNumberTB.Text;
            string title = titleComboBox.Text;
            string addressLine1 = addressLine1TB.Text;
            string addressLine2 = addressLine2TB.Text;
            string previouslyBooked = "NO";
            int rowsAffected = 0;

            if (!string.IsNullOrEmpty(fName) && !string.IsNullOrEmpty(sName) &&
ValidationClass.validateEmail(email) && ValidationClass.validatePhoneNumber(phoneNumber)
&& !string.IsNullOrEmpty(title) && !string.IsNullOrEmpty(addressLine1) &&
!string.IsNullOrEmpty(addressLine2))
            {
                string generatedGuestID = ValidationClass.GenerateGuestID(fName, sName);
                while (!GuestDAL.IsIdUnique(generatedGuestID))
                {
                    generatedGuestID = ValidationClass.GenerateGuestID(fName, sName);
                }
                string[] guestDetails = new string[] { generatedGuestID, fName, sName,
email, phoneNumber, title, addressLine1, addressLine2, previouslyBooked };
                rowsAffected = GuestDAL.AddGuest(guestDetails);

            }
            else
            {

                MessageBox.Show("The criteria to add a guest has not been met, please see
handbook.");
                rowsAffected = 0;
            }
```

```
            if (rowsAffected > 0)
            {
                MessageBox.Show("Guest added successfully");
                UpdateDatabase();
                PopulateComboBoxGuestIDs();
                firstNameTB.Clear();
                lastNameTB.Clear();
                emailTB.Clear();
                phoneNumberTB.Clear();
                titleComboBox.Items.Clear();
                PopulateComboBox();
                addressLine1TB.Clear();
                addressLine2TB.Clear();
            }
            else
            {
                MessageBox.Show("Fail");
            }
        }


        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        private void GuestForm_Load(object sender, EventArgs e)
        {

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                this.guestTableAdapter2.Fill(this.guestTableDGV2.Guest);

                connection.Close();
            }


            UpdateDatabase();
            PopulateComboBoxGuestIDs();
        }
        private void PopulateComboBoxGuestIDs()
        {
            // Call the method to get the items
            List<string> items = GuestDAL.GetGuestIDs();

            // Removing "GuestID: " prefix from each string
            var formattedGuestIds = items.Select(guestId => guestId.Replace("GuestID: ",
""));

            // Clear existing items (optional)
            guestDeleteComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in formattedGuestIds)
            {
                guestDeleteComboBox.Items.Add(item);
            }
```

```
        }

        private void UpdateDatabase()
        {
            guestTable.Clear();
            guestTableAdapter2.Fill(guestTableDGV2.Guest);
        }
        private void deleteBTN_Click(object sender, EventArgs e)
        {
            if(!string.IsNullOrEmpty(guestDeleteComboBox.Text))
            {
                if (MessageBox.Show("Are you sure?", "Delete Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    // user clicked yes

                    string IdValue = guestDeleteComboBox.SelectedItem.ToString();
                    int rowsAffected = GuestDAL.DeleteGuestByID(IdValue);

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Guest deleted successfully");
                        UpdateDatabase();
                        PopulateComboBoxGuestIDs();
                    }
                    else
                    {
                        MessageBox.Show("Failed delete");
                    }
                }
                else
                {
                    // user clicked no
                    MessageBox.Show("Delete cancelled");
                }
            }
            else
            {
                MessageBox.Show("Select a guest to delete");
            }
        }
        private string guestID2;
        private void updateBTN_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(guestID2))
            {
                if(!string.IsNullOrEmpty(firstNameTB.Text) &&
!string.IsNullOrEmpty(lastNameTB.Text) && ValidationClass.validateEmail(emailTB.Text) &&
ValidationClass.validatePhoneNumber(phoneNumberTB.Text) &&
!string.IsNullOrEmpty(titleComboBox.Text) && !string.IsNullOrEmpty(addressLine1TB.Text)
&& !string.IsNullOrEmpty(addressLine2TB.Text))
                {
                    string[] guestDetails = new string[] { guestID2, firstNameTB.Text,
lastNameTB.Text, emailTB.Text, phoneNumberTB.Text, titleComboBox.Text,
addressLine1TB.Text, addressLine2TB.Text };

                    int rowsAffected = GuestDAL.UpdateGuest(guestDetails);
```

```
                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Guest updated successfully");
                        UpdateDatabase();
                        firstNameTB.Clear();
                        lastNameTB.Clear();
                        emailTB.Clear();
                        phoneNumberTB.Clear();
                        PopulateComboBox();
                        addressLine1TB.Clear();
                        addressLine2TB.Clear();
                        guestID2 = "";
                        addGuestBTN.Enabled = true;
                    }
                    else
                    {
                        MessageBox.Show("The record could not be found");
                        UpdateDatabase();
                        firstNameTB.Clear();
                        lastNameTB.Clear();
                        emailTB.Clear();
                        phoneNumberTB.Clear();
                        PopulateComboBox();
                        addressLine1TB.Clear();
                        addressLine2TB.Clear();
                        guestID2 = "";
                        addGuestBTN.Enabled = true;
                    }
                }
                else
                {
                    MessageBox.Show("The criteria to alter a guests information has not
been met, please see handbook");
                }

            }
            else
            {
                MessageBox.Show("Select a guest to be updated");
            }

        }
        //OLD DGV
        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {

        }

        private void dataGridView2_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex >= 8)
            {
                object cellValue = dataGridView2.Rows[e.RowIndex].Cells[e.ColumnIndex -
8].Value;
                if (cellValue != null && cellValue is string guestID)
```

```
                {
                    Guests guest = GuestDAL.GetGuestObjectByID(guestID);
                    guestID2 = guestID;
                    firstNameTB.Text = guest.FirstName;
                    lastNameTB.Text = guest.LastName;
                    emailTB.Text = guest.Email;
                    phoneNumberTB.Text = guest.PhoneNumber;
                    titleComboBox.Text = guest.Title;
                    addressLine1TB.Text = guest.AddressLine1;
                    addressLine2TB.Text = guest.AddressLine2;
                    addGuestBTN.Enabled = false;
                }
                else
                {
                    MessageBox.Show("The cell does not contain a valid value.");
                }
            }
        }
        private void PopulateComboBox()
        {
            List<string> items = new List<string> { "Mr", "Mrs", "Ms", "Dr", "Prof",
"Sir", "Lady" };

            // Clear existing items (optional)
            titleComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in items)
            {
                titleComboBox.Items.Add(item);
            }
        }

        private void phonenumberLBL_Click(object sender, EventArgs e)
        {

        }
    }
}
```

PodForm.cs (Pod Management Screen)

```
using LaksideEscapesBookingSystem.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    public partial class PodForm : Form
    {
        public PodForm()
        {
            InitializeComponent();
        }

        private void addBTN_Click(object sender, EventArgs e)
        {
            int baseCostPerNight;
            int podCapacity;

            if (!string.IsNullOrEmpty(podTypeComboBox.Text) &&
!string.IsNullOrEmpty(podCapacityTB.Text) &&
!string.IsNullOrEmpty(BaseCostPerNightTB.Text) && int.TryParse(BaseCostPerNightTB.Text,
out baseCostPerNight) && int.TryParse(podCapacityTB.Text, out podCapacity))
            {
                string[] podDetails = new string[] { podIDTB.Text, podTypeComboBox.Text,
podCapacity.ToString(), baseCostPerNight.ToString() };
                int rowsAffected = PodDal.AddPod(podDetails);

                if (rowsAffected > 0)
                {
                    MessageBox.Show("Pod added successfully");
                    UpdateDatabase();
                    PopulateComboBoxPodIDs();
                    podIDTB.Clear();
                    podCapacityTB.Clear();
                    podTypeTB.Clear();
                    BaseCostPerNightTB.Clear();
                }
                else
                {
                    MessageBox.Show("Fail");
                }
            }
            else
            {
                MessageBox.Show("The criteria to add a pod has not been met, please see
handbook");
            }
```

```
        }
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        private void PodForm_Load(object sender, EventArgs e)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
                // TODO: This line of code loads data into the 'bookingTable.Booking'
table. You can move, or remove it, as needed.
                this.podTableAdapter.Fill(this.pods.Pod);
                connection.Close();
            }

            PopulateComboBoxPodIDs();
        }
        private void UpdateDatabase()
        {
            podTableAdapter.Fill(pods.Pod);
        }
        private void deleteButton_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(deleteComboBox.Text))
            {
                if (MessageBox.Show("Are you sure?", "Delete Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    // user clicked yes

                    string IdValue = deleteComboBox.SelectedItem.ToString();
                    int rowsAffected = PodDal.DeletePodByID(Convert.ToInt32(IdValue));

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Pod deleted successfully");
                        UpdateDatabase();
                        PopulateComboBoxPodIDs();
                    }
                    else
                    {
                        MessageBox.Show("Failed delete");
                    }
                }
                else
                {
                    // user clicked no
                    MessageBox.Show("Delete cancelled");
                }
            }
            else
            {
                MessageBox.Show("select a pod to delete");
            }

        }

        private void PopulateComboBoxPodIDs()
```

```
        {
            // Call the method to get the items
            List<string> items = PodDal.GetPodIDs();

            // Removing "GuestID: " prefix from each string
            var formattedPodIds = items.Select(podId => podId.Replace("PodID: ", ""));

            // Clear existing items (optional)
            deleteComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in formattedPodIds)
            {
                deleteComboBox.Items.Add(item);
            }


        }

        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex >= 4)
            {
                object cellValue = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex -
4].Value;
                if (cellValue != null && int.TryParse(cellValue.ToString(), out int
podID))
                {
                    Pods pod = PodDal.GetPodObjectByID(podID);
                    podIDTB.Text = podID.ToString();
                    podTypeComboBox.Text = pod.PodType;
                    podCapacityTB.Text = pod.Capacity;
                    BaseCostPerNightTB.Text = pod.BaseCostPerNight.ToString();
                    addBTN.Enabled = false;
                }
                else
                {
                    MessageBox.Show("The cell does not contain a valid integer value.");
                }
            }
        }

        private void updateBTN_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(podIDTB.Text))
            {
                if (!string.IsNullOrEmpty(podTypeComboBox.Text) &&
!string.IsNullOrEmpty(podCapacityTB.Text) &&
!string.IsNullOrEmpty(BaseCostPerNightTB.Text))
                {
                    string[] podDetails = new string[] { podIDTB.Text,
podTypeComboBox.Text, podCapacityTB.Text, BaseCostPerNightTB.Text };

                    int rowsAffected = PodDal.UpdatePod(podDetails);

                    if (rowsAffected > 0)
                    {
```

```
                        MessageBox.Show("Pod updated successfully");

                        UpdateDatabase();
                        PopulateComboBoxPodType();
                        podIDTB.Text = "";
                        podCapacityTB.Text = "";
                        BaseCostPerNightTB.Text = "";
                        addBTN.Enabled = true;
                    }
                    else
                    {
                        MessageBox.Show("The record could not be found");
                        UpdateDatabase();
                        PopulateComboBoxPodType();
                        podIDTB.Text = "";
                        podCapacityTB.Text = "";
                        BaseCostPerNightTB.Text = "";
                        addBTN.Enabled = true;
                    }
                }
                else
                {
                    MessageBox.Show("The criteria to alter a pods information has not
been met, please see handbook");
                }
            }
            else
            {
                MessageBox.Show("Select a pod to update");
            }

        }

        private void PopulateComboBoxPodType()
        {
            List<string> items = new List<string> { "Standard", "Luxury" };

            // Clear existing items (optional)
            podTypeComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in items)
            {
                podTypeComboBox.Items.Add(item);
            }
        }
    }
}
```

Form1.cs (Booking Management Screen)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Text.RegularExpressions;
using System.Configuration;
using LaksideEscapesBookingSystem.Models;
using System.Net.Mail;
using System.Net;

namespace LaksideEscapesBookingSystem
{
    public partial class BookingMenu : Form
    {
        public BookingMenu()
        {
            InitializeComponent();


        }

        private void AddBookingBTN_Click(object sender, EventArgs e)
        {
            string[] bookingInfo = new string[12];

            string errorMessage =
ValidationClass.errorBuilderBookingForm(guestIDComboBox.Text, podIDComboBox.Text,
courseIDComboBox.Text, noOfOccupantsTB.Text);

            if(errorMessage.Contains("not"))
            {
                MessageBox.Show(errorMessage);
            }
            else
            {
                if (ValidationClass.bookedInAdvance(bookingDateTimePicker.Value,
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem))) &&
BookingDal.DoesBookingOverlap(podIDComboBox.Text, bookingDateTimePicker.Value,
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem)), 0)
&& !BookingDal.DoesNoOccsExceedCapacity(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(noOfOccupantsTB.Text)))
                {
                    //No validation required can be any selected value
                    bookingInfo[0] = "";
                    bookingInfo[1] = guestIDComboBox.Text;
                    bookingInfo[2] = podIDComboBox.Text;
                    bookingInfo[3] = courseIDComboBox.Text;
```

```
                    //Must not exceed pod capacity (Handled in above if statement)
                    bookingInfo[4] = noOfOccupantsTB.Text;

                    //Has to be booked in advance and cant overlap (Handled in if
statement)
                    bookingInfo[5] = bookingDateTimePicker.Value.ToString();
                    bookingInfo[6] =
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem)).ToStr
ing();

                    //No validation needed default value
                    bookingInfo[7] = "Provisional";

                    //Needed to calculate deposit
                    int selectedNumberOfDays =
Convert.ToInt32(noOfDaysComboBox.SelectedItem);

                    //Deposit calculation
                    bookingInfo[8] =
ValidationClass.calculateDeposit(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(courseIDComboBox.Text), Convert.ToInt32(noOfOccupantsTB.Text),
selectedNumberOfDays);

                    //Discount calculation
                    bookingInfo[9] =
ValidationClass.calculateDiscount(guestIDComboBox.Text.Substring(0, 5),
bookingDateTimePicker.Value);

                    //Total claculation
                    bookingInfo[10] =
ValidationClass.calculateTotal(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(courseIDComboBox.Text), Convert.ToInt32(noOfOccupantsTB.Text),
selectedNumberOfDays);

                    //No validation needed is default value of today
                    bookingInfo[11] = DateTime.Now.ToString();

                    int rowsAffected = BookingDal.AddBooking(bookingInfo);

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Booking added successfully");
                        UpdateDatabase();
                        PopulateComboBoxBookingsForDelete();
                        PopulateComboBoxGuestIDs();
                        PopulateComboBoxBookings();
                        PopulateComboBoxNoOfDays();
                        PopulateComboBoxCourse();
                        PopulateComboBoxPods();
                        noOfOccupantsTB.Text = "";
                        bookingDateTimePicker.Value = DateTime.Now;
                        previewBookinglbl.Text = "";
                        previewBookinglbl2.Text = "";
                        previewBookinglbl3.Text = "";
                        podIDTB.Text = "";
                    }
                    else
                    {
```

```
                        MessageBox.Show("Fail");
                        UpdateDatabase();
                        PopulateComboBoxBookingsForDelete();
                        PopulateComboBoxGuestIDs();
                        PopulateComboBoxBookings();
                        PopulateComboBoxNoOfDays();
                        PopulateComboBoxCourse();
                        PopulateComboBoxPods();
                        noOfOccupantsTB.Text = "";
                        bookingDateTimePicker.Value = DateTime.Now;
                        previewBookinglbl.Text = "";
                        previewBookinglbl2.Text = "";
                        previewBookinglbl3.Text = "";
                        podIDTB.Text = "";
                    }
                }
                else
                {
                    MessageBox.Show("The criteria to add a booking has not been met,
please see handbook.");
                }
            }
        }


        static string ExtractNumber(string input)
        {
            string substringToRemove = "BookingID: ";
            string result = input.Replace(substringToRemove, "");
            Convert.ToInt32(result);

            return result;
        }
        //Pay deposit btn
        private void button1_Click(object sender, EventArgs e)
        {
            //Pay selected guests deposit making their booking permanent

            if (!string.IsNullOrEmpty(bookingIDComboBox.Text))
            {
                string IdValue = bookingIDComboBox.SelectedItem.ToString();
                string numberString = ExtractNumber(IdValue);
                int id = Convert.ToInt32(numberString);


                DateTime dateBooked = BookingDal.getDateBooked(id);


                if (ValidationClass.canBookingBeMadePermanent(dateBooked))
                {
                    BookingDal.UpdateBookingStatusToPermanent(id);
                    MessageBox.Show("Booking is now permanent");
                    UpdateDatabase();
                    PopulateComboBoxBookings();
                }
                else
                {
```

```
                    MessageBox.Show("Booking is not availbe to be made permanent or is
already permanent");
                }
            }
            else
            {
                MessageBox.Show("Select a booking to be made permanent");
            }


            /*
            //OLD VIEW GUEST AS A LIST

            //format string to just a number
            //BookingID: 1

            string IdValue = bookingIDComboBox.SelectedItem.ToString();
            string numberString = ExtractNumber(IdValue);

            List<string> bookings =
BookingDal.GetBookingsByBookingID(Convert.ToInt32(numberString));

            if (bookings.Count >= 1)
            {
                StringBuilder sb = new StringBuilder();

                foreach(string booking in bookings)
                {
                    sb.AppendLine(booking);
                }

                MessageBox.Show(sb.ToString());
            }
            else
            {
                MessageBox.Show("There are no bookings by that id");
            }

            */
        }

        private void bookingDateTimePicker_ValueChanged(object sender, EventArgs e)
        {
        }
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        private void BookingMenu_Load(object sender, EventArgs e)
        {
            //Method called to delete bookings that have not been made permanent in time
            BookingDal.DeleteExpiredBookings();

            //Populates DGV
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
                // TODO: This line of code loads data into the
'updatedBookingTable.Booking' table. You can move, or remove it, as needed.
                this.bookingTableAdapter1.Fill(this.updatedBookingTable.Booking);
```

```
                connection.Close();
            }


            //Populates combo boxes
            PopulateComboBoxBookings();
            PopulateComboBoxCourse();
            PopulateComboBoxPods();
            PopulateComboBoxGuestIDs();
            PopulateComboBoxBookingsForDelete();
        }
        private void UpdateDatabase()
        {
            bookingTableAdapter1.Fill(updatedBookingTable.Booking);
        }
        private void PopulateComboBoxGuestIDs()
        {
            // Call the method to get the items
            List<string> items = GuestDAL.GetGuestIDs();

            // Removing "PodID: " prefix from each string
            var formattedGuestIds = items.Select(guestId => guestId.Replace("GuestID: ",
"""));

            // Clear existing items (optional)
            guestIDComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in formattedGuestIds)
            {
                guestIDComboBox.Items.Add(item);
            }

        }
        private void PopulateComboBoxBookingsForDelete()
        {
            // Call the method to get the items
            List<string> items = BookingDal.GetBookingsIDs();

            // Clear existing items (optional)
            bookingIDsDeleteComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in items)
            {
                bookingIDsDeleteComboBox.Items.Add(item);
            }


        }
        private void PopulateComboBoxBookings()
        {
            // Call the method to get the items
            List<string> items = BookingDal.GetBookingsIDs();

            // Clear existing items (optional)
            bookingIDComboBox.Items.Clear();
```

```
            // Add items to the combo box
            foreach (string item in items)
            {
                bookingIDComboBox.Items.Add(item);
            }


        }
        private void PopulateComboBoxCourse()
        {
            List<string> items2 = CourseDal.GetCourseIDs();

            // Clear existing items (optional)
            courseIDComboBox.Items.Clear();

            // Removing "PodID: " prefix from each string
            var formattedCourseIds = items2.Select(courseId =>
courseId.Replace("CourseID: ", ""));

            // Add items to the combo box
            foreach (string item in formattedCourseIds)
            {
                courseIDComboBox.Items.Add(item);
            }


        }

        private void PopulateComboBoxPods()
        {
            List<string> items3 = PodDal.GetPodIDs();

            // Clear existing items (optional)
            podIDComboBox.Items.Clear();

            // Removing "PodID: " prefix from each string
            var formattedPodIds = items3.Select(podId => podId.Replace("PodID: ", ""));

            // Add items to the combo box
            foreach (string item in formattedPodIds)
            {
                podIDComboBox.Items.Add(item);
            }


        }

        private void PopulateComboBoxNoOfDays()
        {
            List<string> items = new List<string> { "3", "5", "7", "14" };

            // Clear existing items (optional)
            noOfDaysComboBox.Items.Clear();

            // Add items to the combo box
            foreach (string item in items)
            {
                noOfDaysComboBox.Items.Add(item);
```

```
            }
        }
        private void bookingIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void dataGridView1_Click(object sender, EventArgs e)
        {

        }

        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex >= 12)
            {
                object cellValue = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex -
12].Value;
                if (cellValue != null && int.TryParse(cellValue.ToString(), out int
bookingID))
                {
                    Bookings booking = BookingDal.GetBookingObjectByID(bookingID);
                    podIDTB.Text = bookingID.ToString();
                    noOfDaysComboBox.Text =
ValidationClass.guestLengthOfStay(booking.CheckInDate, booking.CheckOutDate).ToString();
                    bookingDateTimePicker.Value = booking.CheckInDate;
                    guestIDComboBox.Text = booking.GuestID;
                    podIDComboBox.Text = booking.PodID.ToString();
                    courseIDComboBox.Text = booking.CourseID.ToString();
                    noOfOccupantsTB.Text = booking.NumberOfOccupants.ToString();
                    AddBookingBTN.Enabled = false;
                }
                else
                {
                    MessageBox.Show("The cell does not contain a valid integer value.");
                }
            }
        }
        //delete button
        private void button2_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(bookingIDsDeleteComboBox.Text))
            {
                if (MessageBox.Show("Are you sure?", "Delete Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    // user clicked yes
                    string IdValue = bookingIDsDeleteComboBox.SelectedItem.ToString();
                    string numberString = ExtractNumber(IdValue);

                    int rowsAffected =
BookingDal.DeleteBookingByID(Convert.ToInt32(numberString));

                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Booking deleted successfully");
                        UpdateDatabase();
```

```
                    PopulateComboBoxBookingsForDelete();
                    PopulateComboBoxBookings();
                }
                else
                {
                    MessageBox.Show("Failed delete");
                }
            }
            else
            {
                // user clicked no
                MessageBox.Show("Delete cancelled");
            }
        }
        else
        {
            MessageBox.Show("Select a booking to delete");
        }
    }

    private void guestIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        previewBookinglbl.Text = "";
        var items =
GuestDAL.GetGuestByGuestID(guestIDComboBox.SelectedItem.ToString());
        for (int i = 0; i < items.Count; i++)
        {
            previewBookinglbl.Text += items.ElementAt(i);
        }

    }

    private void podIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        previewBookinglbl3.Text = "";
        var items =
PodDal.GetPodsByPodID(Convert.ToInt32(podIDComboBox.SelectedItem));
        for (int i = 0; i < items.Count; i++)
        {
            previewBookinglbl3.Text += items.ElementAt(i);
        }
    }

    private void courseIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        previewBookinglbl2.Text = "";
        var items =
CourseDal.GetCoursesByCourseID(Convert.ToInt32(courseIDComboBox.SelectedItem));
        for (int i = 0; i < items.Count; i++)
        {
            previewBookinglbl2.Text += items.ElementAt(i);
        }
    }

    private void updateBTN_Click(object sender, EventArgs e)
    {
        if (!string.IsNullOrEmpty(podIDTB.Text))
        {
```

```
                string[] bookingInfo = new string[12];

                if (ValidationClass.bookedInAdvance(bookingDateTimePicker.Value,
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem))) &&
BookingDal.DoesBookingOverlap(podIDComboBox.Text, bookingDateTimePicker.Value,
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem)),
Convert.ToInt32(podIDTB.Text)) &&
!BookingDal.DoesNoOccsExceedCapacity(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(noOfOccupantsTB.Text)))
                {
                    //Validation not needed can be any selected value
                    bookingInfo[0] = podIDTB.Text;
                    bookingInfo[1] = guestIDComboBox.Text;
                    bookingInfo[2] = podIDComboBox.Text;
                    bookingInfo[3] = courseIDComboBox.Text;

                    //Must not exceed pod capacity (Handled in above if statement)
                    bookingInfo[4] = noOfOccupantsTB.Text;

                    //Has to be booked in advance and cant overlap (Handled in if
statement)
                    bookingInfo[5] = bookingDateTimePicker.Value.ToString();
                    bookingInfo[6] =
bookingDateTimePicker.Value.AddDays(Convert.ToInt32(noOfDaysComboBox.SelectedItem)).ToStr
ing();

                    //No validation needed default value
                    bookingInfo[7] = "Provisional";

                    //Needed to calculate deposit
                    int selectedNumberOfDays =
Convert.ToInt32(noOfDaysComboBox.SelectedItem);

                    //Deposit calculation
                    bookingInfo[8] =
ValidationClass.calculateDeposit(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(courseIDComboBox.Text), Convert.ToInt32(noOfOccupantsTB.Text),
selectedNumberOfDays);

                    //Discount calculation
                    bookingInfo[9] =
ValidationClass.calculateDiscount(guestIDComboBox.Text.Substring(0, 5),
bookingDateTimePicker.Value);

                    //Total claculation
                    bookingInfo[10] =
ValidationClass.calculateTotal(Convert.ToInt32(podIDComboBox.Text),
Convert.ToInt32(courseIDComboBox.Text), Convert.ToInt32(noOfOccupantsTB.Text),
selectedNumberOfDays);
                    bookingInfo[11] = DateTime.Now.ToString();
                    int rowsAffected = BookingDal.UpdateBooking(bookingInfo);
                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Booking updated successfully");
                        UpdateDatabase();
                        PopulateComboBoxNoOfDays();
                        PopulateComboBoxGuestIDs();
                        PopulateComboBoxCourse();
```

```
                        PopulateComboBoxPods();
                        noOfOccupantsTB.Text = "";
                        bookingDateTimePicker.Value = DateTime.Now;
                        previewBookinglbl.Text = "";
                        previewBookinglbl2.Text = "";
                        previewBookinglbl3.Text = "";
                        podIDTB.Text = "";
                        AddBookingBTN.Enabled = true;
                    }
                    else
                    {
                        MessageBox.Show("The record could not be found");
                        UpdateDatabase();
                        PopulateComboBoxNoOfDays();
                        PopulateComboBoxGuestIDs();
                        PopulateComboBoxCourse();
                        PopulateComboBoxPods();
                        noOfOccupantsTB.Text = "";
                        bookingDateTimePicker.Value = DateTime.Now;
                        previewBookinglbl.Text = "";
                        previewBookinglbl2.Text = "";
                        previewBookinglbl3.Text = "";
                        podIDTB.Text = "";
                        AddBookingBTN.Enabled = true;
                    }
                }
                else
                {
                    MessageBox.Show("The criteria to update a booking has not been met,
please see handbook");
                }
            }
            else
            {
                MessageBox.Show("Select a booking to update");
            }


        }

        private void previewBookinglbl2_Click(object sender, EventArgs e)
        {

        }

        private void bookingIDsDeleteComboBox_SelectedIndexChanged(object sender,
EventArgs e)
        {

        }
    }
}
```

HubForm.cs (Report Visualisation screen)

```
using Microsoft.Reporting.WinForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    public partial class HubForm : Form
    {
        public HubForm()
        {
            InitializeComponent();
        }
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        private void HubForm_Load(object sender, EventArgs e)
        {

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
                // TODO: This line of code loads data into the
'DataSetMostPopCourses.GetMostPopularCourses' table. You can move, or remove it, as
needed.

this.GetMostPopularCoursesTableAdapter.Fill(this.DataSetMostPopCourses.GetMostPopularCour
ses);
                // TODO: This line of code loads data into the
'DataSetMostPopPods.GetMostPopularPods' table. You can move, or remove it, as needed.

this.GetMostPopularPodsTableAdapter.Fill(this.DataSetMostPopPods.GetMostPopularPods);
                // TODO: This line of code loads data into the 'MostPopularPod.Booking'
table. You can move, or remove it, as needed.
                this.BookingTableAdapter.Fill(this.DataSetMostPopularPod.Booking);
                //TODO

this.CalculateMonthlyRevenue_2024TableAdapter.Fill(this.DataSetMR.CalculateMonthlyRevenue
_2024);
                connection.Close();
            }
```

```
            this.reportViewer2.RefreshReport();
            this.reportViewer3.RefreshReport();
            this.monthlyRevenueViewer.RefreshReport();
        }

        private void reportViewer1_Load(object sender, EventArgs e)
        {

        }

        private void reportViewer3_Load(object sender, EventArgs e)
        {

        }
    }
}
```

GuestDAL.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Configuration;
using System.Data.SqlClient;
using System.Data;
using LaksideEscapesBookingSystem.Models;

namespace LaksideEscapesBookingSystem
{
    class GuestDAL
    {
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;

        public static int AddGuest(string[] guestDetails)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand insertGuestCommand = new SqlCommand();
                insertGuestCommand.Connection = connection;

                insertGuestCommand.CommandType = System.Data.CommandType.StoredProcedure;

                insertGuestCommand.CommandText = "AddGuest";

                insertGuestCommand.Parameters.Add(new SqlParameter("@GuestId",
guestDetails[0]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@FirstName",
guestDetails[1]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@LastName",
guestDetails[2]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@Email",
guestDetails[3]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@PhoneNumber",
guestDetails[4]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@Title",
guestDetails[5]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@AddressLine1",
guestDetails[6]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@AddressLine2",
guestDetails[7]));
                insertGuestCommand.Parameters.Add(new SqlParameter("@PreviouslyBooked",
guestDetails[8]));

                int rowsAffected = insertGuestCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
```

```
        }

        public static List<string> GetGuestIDs()
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> bookings = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT GuestID FROM Guest");

                SqlCommand getBookingsByBookingIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getBookingsByBookingIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all information about the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}";
                    }
                    bookings.Add(bookingInfo);
                }

                connection.Close();

                return bookings;
            }
        }
        public static int DeleteGuestByID(string guestID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand deleteGuestCommand = new SqlCommand();
                deleteGuestCommand.Connection = connection;

                deleteGuestCommand.CommandType = System.Data.CommandType.StoredProcedure;

                deleteGuestCommand.CommandText = "DeleteGuest";

                deleteGuestCommand.Parameters.Add(new SqlParameter("@GuestID", guestID));

                int rowsAffected = deleteGuestCommand.ExecuteNonQuery();

                connection.Close();
                return rowsAffected;
            }

        }

        public static List<string> GetGuestByGuestID(string guestID)
```

```
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> guest = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT * FROM Guest WHERE GuestID =
'{0}'", guestID.ToString());

                SqlCommand getGuestByGuestIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader = getGuestByGuestIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all the information of the booking
                    string guestInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        guestInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}\n";
                    }
                    guest.Add(guestInfo);
                }

                connection.Close();

                return guest;
            }
        }

        public static Guests GetGuestObjectByID(string guestID)
        {
            Guests guest = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                string sqlQuery = "SELECT * FROM Guest WHERE GuestID = @GuestID";

                SqlCommand getGuestByIDCommand = new SqlCommand(sqlQuery, connection);
                getGuestByIDCommand.Parameters.AddWithValue("@GuestID", guestID);

                SqlDataReader reader = getGuestByIDCommand.ExecuteReader();

                if (reader.Read())
                {
                    // Populate a Guests object with data from the database
                    guest = new Guests
                    {
                        GuestID = reader["GuestID"].ToString(),
                        FirstName = reader["FirstName"].ToString(),
                        LastName = reader["LastName"].ToString(),
                        Email = reader["Email"].ToString(),
                        PhoneNumber = reader["PhoneNumber"].ToString(),
                        Title = reader["Title"].ToString(),
```

```
                    AddressLine1 = reader["AddressLine1"].ToString(),
                    AddressLine2 = reader["AddressLine2"].ToString()
                };
            }

            connection.Close();
        }

        return guest;
    }

    public static int UpdateGuest(string[] guestDetails)
    {
        using (SqlConnection connection = new SqlConnection(_connectionstring))
        {
            connection.Open();

            SqlCommand updateGuestCommand = new SqlCommand();
            updateGuestCommand.Connection = connection;
            updateGuestCommand.CommandType = System.Data.CommandType.StoredProcedure;
            updateGuestCommand.CommandText = "UpdateGuest";

            updateGuestCommand.Parameters.Add(new SqlParameter("@GuestID",
guestDetails[0]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@FirstName",
guestDetails[1]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@LastName",
guestDetails[2]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@Email",
guestDetails[3]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@PhoneNumber",
guestDetails[4]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@Title",
guestDetails[5]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@AddressLine1",
guestDetails[6]));
            updateGuestCommand.Parameters.Add(new SqlParameter("@AddressLine2",
guestDetails[7]));


            int rowsAffected = updateGuestCommand.ExecuteNonQuery();

            connection.Close();

            return rowsAffected;
        }
    }

    public static bool IsIdUnique(string id)
    {
        using (SqlConnection connection = new SqlConnection(_connectionstring))
        {
            connection.Open();
            string query = "SELECT COUNT(*) FROM Guest WHERE GuestID = @Id";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Id", id);
            int count = (int)command.ExecuteScalar();
            return count == 0;
```

```
            }
        }

        public static string[] GetGuestNameAndEmailByID(string guestID)
        {
            string[] guestInfo = new string[3]; // Array to store first name, last name,
and email

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                string sqlQuery = "SELECT FirstName, LastName, Email FROM Guest WHERE
GuestID = @GuestID";

                SqlCommand getGuestByIDCommand = new SqlCommand(sqlQuery, connection);
                getGuestByIDCommand.Parameters.AddWithValue("@GuestID", guestID);

                SqlDataReader reader = getGuestByIDCommand.ExecuteReader();

                if (reader.Read())
                {
                    // Retrieve first name, last name, and email from the database
                    guestInfo[0] = reader["FirstName"].ToString();
                    guestInfo[1] = reader["LastName"].ToString();
                    guestInfo[2] = reader["Email"].ToString();
                }

                connection.Close();
            }

            return guestInfo;
        }
    }
}
```

CourseDal.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Configuration;
using System.Data.SqlClient;
using System.Data;
using LaksideEscapesBookingSystem.Models;

namespace LaksideEscapesBookingSystem
{
    class CourseDal
    {

        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;

        public static int AddCourse(string[] courseDetails)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand insertCourseCommand = new SqlCommand();
                insertCourseCommand.Connection = connection;

                insertCourseCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                insertCourseCommand.CommandText = "AddCourse";

                //insertCourseCommand.Parameters.Add(new SqlParameter("@CourseID",
courseDetails[0]));
                insertCourseCommand.Parameters.Add(new SqlParameter("@CourseName",
courseDetails[1]));
                insertCourseCommand.Parameters.Add(new
SqlParameter("@CoursecostPerPerson", courseDetails[2]));


                int rowsAffected = insertCourseCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
        }
        public static List<string> GetCourseIDs()
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> bookings = new List<string>();

                connection.Open();
```

```
                string sqlQuery = string.Format("SELECT CourseID FROM Course");

                SqlCommand getBookingsByBookingIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getBookingsByBookingIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all information about the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}";
                    }
                    bookings.Add(bookingInfo);
                }

                connection.Close();

                return bookings;
            }
        }

        public static int DeleteCourseByID(int courseID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand deleteCourseCommand = new SqlCommand();
                deleteCourseCommand.Connection = connection;

                deleteCourseCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                deleteCourseCommand.CommandText = "DeleteCourse";

                deleteCourseCommand.Parameters.Add(new SqlParameter("@CourseID",
courseID));

                int rowsAffected = deleteCourseCommand.ExecuteNonQuery();

                connection.Close();
                return rowsAffected;
            }

        }

        public static int getCourseCost(int courseID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand getCourseCostCommand = new SqlCommand();
                getCourseCostCommand.Connection = connection;
```

```
                getCourseCostCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                getCourseCostCommand.CommandText = "CourseCost";

                getCourseCostCommand.Parameters.Add(new SqlParameter("@CourseID",
courseID));

                var rowsAffected = getCourseCostCommand.ExecuteScalar();

                connection.Close();
                return (int)rowsAffected;
            }
        }

        public static List<string> GetCoursesByCourseID(int courseID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> courses = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT * FROM Course WHERE CourseID =
'{0}'", courseID.ToString());

                SqlCommand getCoursesByCourseIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getCoursesByCourseIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all the information of the booking
                    string courseInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        courseInfo += $"{sqlDataReader.GetName(i)}:
{sqlDataReader[i]}\n";
                    }
                    courses.Add(courseInfo);
                }

                connection.Close();

                return courses;
            }
        }

        public static Courses GetCourseObjectByID(int courseID)
        {
            Courses course = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
```

```
                string sqlQuery = "SELECT * FROM Course WHERE CourseID = @CourseID";

                SqlCommand getCourseByIDCommand = new SqlCommand(sqlQuery, connection);
                getCourseByIDCommand.Parameters.AddWithValue("@CourseID", courseID);

                SqlDataReader reader = getCourseByIDCommand.ExecuteReader();

                if (reader.Read())
                {
                    // Populate a Courses object with data from the database
                    course = new Courses
                    {
                        CourseID = (int)reader["CourseID"],
                        CourseName = reader["CourseName"].ToString(),
                        CourseCostPerPerson = (int)reader["CourseCostPerPerson"],
                    };
                }

                connection.Close();
            }

            return course;
        }

        public static int UpdateCourse(string[] courseDetails)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand updateCourseCommand = new SqlCommand();
                updateCourseCommand.Connection = connection;
                updateCourseCommand.CommandType =
System.Data.CommandType.StoredProcedure;
                updateCourseCommand.CommandText = "UpdateCourse";

                updateCourseCommand.Parameters.Add(new SqlParameter("@CourseID",
courseDetails[0]));
                updateCourseCommand.Parameters.Add(new SqlParameter("@CourseName",
courseDetails[1]));
                updateCourseCommand.Parameters.Add(new
SqlParameter("@CourseCostPerPerson", courseDetails[2]));

                int rowsAffected = updateCourseCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
        }
        public static string GetCourseNameByCourseID(int courseID)
        {
            string courseName = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
```

```
                SqlCommand command = new SqlCommand();
                command.Connection = connection;
                command.CommandType = CommandType.StoredProcedure;
                command.CommandText = "GetCourseNameByCourseID";

                // Add parameters
                command.Parameters.AddWithValue("@CourseID", courseID);

                // Execute the command to retrieve the course name
                object result = command.ExecuteScalar();

                // Check if the result is not null and convert it to string
                if (result != null)
                {
                    courseName = result.ToString();
                }

                connection.Close();
            }

            return courseName;
        }

    }
}
```

PodDal.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Configuration;
using System.Data.SqlClient;
using System.Data;
using LaksideEscapesBookingSystem.Models;

namespace LaksideEscapesBookingSystem
{
    class PodDal
    {

        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;

        public static int AddPod(string[] podDetails)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand insertPodCommand = new SqlCommand();
                insertPodCommand.Connection = connection;

                insertPodCommand.CommandType = System.Data.CommandType.StoredProcedure;

                insertPodCommand.CommandText = "AddPod";


                //insertPodCommand.Parameters.Add(new SqlParameter("@PodId",
podDetails[0]));
                insertPodCommand.Parameters.Add(new SqlParameter("@PodType",
podDetails[1]));
                insertPodCommand.Parameters.Add(new SqlParameter("@Capacity",
podDetails[2]));
                insertPodCommand.Parameters.Add(new SqlParameter("@BaseCostPerNight",
podDetails[3]));


                int rowsAffected = insertPodCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }

        }

        public static List<string> GetPodIDs()
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
```

```
                List<string> bookings = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT PodID FROM Pod");

                SqlCommand getBookingsByBookingIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getBookingsByBookingIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all information about the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}";
                    }
                    bookings.Add(bookingInfo);
                }

                connection.Close();

                return bookings;
            }

        }

        public static int DeletePodByID(int podID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand deletePodCommand = new SqlCommand();
                deletePodCommand.Connection = connection;

                deletePodCommand.CommandType = System.Data.CommandType.StoredProcedure;

                deletePodCommand.CommandText = "DeletePod";

                deletePodCommand.Parameters.Add(new SqlParameter("@PodID", podID));

                int rowsAffected = deletePodCommand.ExecuteNonQuery();

                connection.Close();
                return rowsAffected;
            }


        }
        public static int getPodCost(int podID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
```

```
                SqlCommand getCostCommand = new SqlCommand();
                getCostCommand.Connection = connection;

                getCostCommand.CommandType = System.Data.CommandType.StoredProcedure;

                getCostCommand.CommandText = "PodCost";

                getCostCommand.Parameters.Add(new SqlParameter("@PodID", podID));

                var podCost = getCostCommand.ExecuteScalar();

                connection.Close();
                return (int)podCost;
            }
        }

        public static List<string> GetPodsByPodID(int podID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> pods = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT * FROM Pod WHERE PodID = '{0}'",
podID.ToString());

                SqlCommand getPodsByPodIDCommand = new SqlCommand(sqlQuery, connection);

                SqlDataReader sqlDataReader = getPodsByPodIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all the information of the booking
                    string podInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        podInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}\n";
                    }
                    pods.Add(podInfo);
                }

                connection.Close();

                return pods;
            }
        }

        public static Pods GetPodObjectByID(int podID)
        {
            Pods pod = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                string sqlQuery = "SELECT * FROM Pod WHERE PodID = @PodID";
```

```
                SqlCommand getPodByIDCommand = new SqlCommand(sqlQuery, connection);
                getPodByIDCommand.Parameters.AddWithValue("@PodID", podID);

                SqlDataReader reader = getPodByIDCommand.ExecuteReader();

                if (reader.Read())
                {
                    // Populate a Courses object with data from the database
                    pod = new Pods
                    {
                        PodID  = (int)reader["PodID"],
                        PodType = reader["PodType"].ToString(),
                        Capacity = reader["Capacity"].ToString(),
                        BaseCostPerNight = (int)reader["BaseCostPerNight"]
                    };
                }

                connection.Close();
            }

            return pod;
        }

        public static int UpdatePod(string[] podDetails)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand updatePodCommand = new SqlCommand();
                updatePodCommand.Connection = connection;
                updatePodCommand.CommandType = System.Data.CommandType.StoredProcedure;
                updatePodCommand.CommandText = "UpdatePod";

                updatePodCommand.Parameters.Add(new SqlParameter("@PodID",
podDetails[0]));
                updatePodCommand.Parameters.Add(new SqlParameter("@PodType",
podDetails[1]));
                updatePodCommand.Parameters.Add(new SqlParameter("@Capacity",
podDetails[2]));
                updatePodCommand.Parameters.Add(new SqlParameter("@BaseCostPerNight",
podDetails[3]));


                int rowsAffected = updatePodCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
        }

        public static int getPodCapacity(int podID)
        {
            int capacity = 0;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
```

```
            {
                connection.Open();

                string sqlQuery = "SELECT Capacity FROM Pod WHERE PodID = @PodID";

                SqlCommand command = new SqlCommand(sqlQuery, connection);
                command.Parameters.AddWithValue("@PodID", podID);

                SqlDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    capacity = Convert.ToInt32(reader["Capacity"]);
                }

                connection.Close();
            }

            return capacity;
        }

        public static string GetPodTypeByPodID(int podID)
        {
            string podType = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand command = new SqlCommand();
                command.Connection = connection;
                command.CommandType = CommandType.StoredProcedure;
                command.CommandText = "GetPodTypeByPodID";

                // Add parameters
                command.Parameters.AddWithValue("@PodID", podID);

                // Execute the command to retrieve the pod type
                object result = command.ExecuteScalar();

                // Check if the result is not null and convert it to string
                if (result != null)
                {
                    podType = result.ToString();
                }

                connection.Close();
            }

            return podType;
        }



    }
}
```

BookingDal.cs

```
using LaksideEscapesBookingSystem.Models;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace LaksideEscapesBookingSystem
{
    class BookingDal
    {
        //"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\finne\SSDNot1Drive\Year14\SSD\Lak
sideEscapesBookingSystem\LaksideEscapesBookingSystem\BookingDatbase.mdf;Integrated
Security=True"
        private static string _connectionstring =
ConfigurationManager.ConnectionStrings["BookingSystemConnectionString"].ConnectionString;
        //"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Users\finne\\SSDNot1Drive\\Year14\\SS
D\\LaksideEscapesBookingSystem\\LaksideEscapesBookingSystem\\BookingDatbase.mdf;Integrate
d Security=True";



//C:\Users\finne\SSDNot1Drive\Year14\SSD\LaksideEscapesBookingSystem\LaksideEscapesBookin
gSystem\BookingDatabase.mdf
        //"Data Source=(LocalDB)\MSSQLLocalDB;Initial Catalog=BookingDatabase.mdf;
Integrated Security=True;"

        public static int AddBooking(string[] bookingInfo)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                //string sqlQuery = string.Format("INSERT INTO Booking VALUES(@BookingID,
@GuestID, @PodID, @CourseID, @NumberOfOccupants, @CheckInDate, @CheckOutDate,
@BookingStatus, @DepositAmount, @DiscountPercentage, @TotalAmount)");

                SqlCommand insertBookingCommand = new SqlCommand();
                insertBookingCommand.Connection = connection;

                insertBookingCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                insertBookingCommand.CommandText = "AddBooking";


                //insertProjectCommand.Parameters.AddWithValue("@CheckInDate",
SqlDbType.DateTime).Value = DateTime.Now;
                //insertProjectCommand.Parameters.AddWithValue("@CheckOutDate",
SqlDbType.DateTime).Value = DateTime.Now.AddDays(1);
```

```
                // Retrieve the datetime string from the array
                string datetimeString = bookingInfo[5];
                string datetimeString2 = bookingInfo[6];
                string dateBookedString = bookingInfo[11];


                // Convert the string to a DateTime object
                DateTime bookingDateTime;
                DateTime bookingDateTime2;
                DateTime dateBooked;



                if (DateTime.TryParse(datetimeString, out bookingDateTime) &&
DateTime.TryParse(datetimeString2, out bookingDateTime2) &&
DateTime.TryParse(dateBookedString, out dateBooked))
                {
                    //insertBookingCommand.Parameters.Add(new SqlParameter("@BookingID",
bookingInfo[0]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@GuestID",
bookingInfo[1]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@PodID",
bookingInfo[2]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@CourseID",
bookingInfo[3]));
                    insertBookingCommand.Parameters.Add(new
SqlParameter("@NumberOfOccupants", bookingInfo[4]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@CheckInDate",
bookingDateTime));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@CheckOutDate",
bookingDateTime2));
                    insertBookingCommand.Parameters.Add(new
SqlParameter("@BookingStatus", bookingInfo[7]));
                    insertBookingCommand.Parameters.Add(new
SqlParameter("@DepositAmount", bookingInfo[8]));
                    insertBookingCommand.Parameters.Add(new
SqlParameter("@DiscountPercentage", bookingInfo[9]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@TotalAmount",
bookingInfo[10]));
                    insertBookingCommand.Parameters.Add(new SqlParameter("@DateBooked",
dateBooked));
                }
                else
                {
                    Console.WriteLine("Invalid DateTime format");
                }


                int rowsAffected = insertBookingCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
        }

        public static List<string> GetBookingsByBookingID(int bookingID)
        {
```

```
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> bookings = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT * FROM Booking WHERE BookingID =
'{0}'", bookingID.ToString());

                SqlCommand getBookingsByBookingIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getBookingsByBookingIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all the information of the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}:
{sqlDataReader[i]}\n";
                    }
                    bookings.Add(bookingInfo);
                }

                connection.Close();

                return bookings;
            }
        }

        public static List<string> GetBookingsIDs()
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                List<string> bookings = new List<string>();

                connection.Open();

                string sqlQuery = string.Format("SELECT BookingID FROM Booking");

                SqlCommand getBookingsByBookingIDCommand = new SqlCommand(sqlQuery,
connection);

                SqlDataReader sqlDataReader =
getBookingsByBookingIDCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all information about the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}";
                    }
                    bookings.Add(bookingInfo);
```

```
                }

                connection.Close();

                return bookings;
            }
        }
        public static int DeleteBookingByID(int bookingID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand deleteBookingCommand = new SqlCommand();
                deleteBookingCommand.Connection = connection;

                deleteBookingCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                deleteBookingCommand.CommandText = "DeleteBooking";

                deleteBookingCommand.Parameters.Add(new SqlParameter("@BookingID",
bookingID));

                int rowsAffected = deleteBookingCommand.ExecuteNonQuery();

                connection.Close();
                return rowsAffected;
            }

        }

        public static int getNoOfOccs(int bookingID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand readNoOfOccCommand = new SqlCommand();
                readNoOfOccCommand.Connection = connection;

                readNoOfOccCommand.CommandType = System.Data.CommandType.StoredProcedure;

                readNoOfOccCommand.CommandText = "BookingOccupants";

                readNoOfOccCommand.Parameters.Add(new SqlParameter("@BookingID",
bookingID));

                var items = readNoOfOccCommand.ExecuteScalar();

                connection.Close();
                return (int)items;
            }
        }

        public static List<string> getStartAndEndDate(int bookingID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
```

```
            {
                List<string> bookings = new List<string>();

                connection.Open();

                SqlCommand getDateCommand = new SqlCommand();
                getDateCommand.Connection = connection;

                getDateCommand.CommandType = System.Data.CommandType.StoredProcedure;

                getDateCommand.CommandText = "BookingLengthOfStay";

                getDateCommand.Parameters.Add(new SqlParameter("@BookingID", bookingID));
                SqlDataReader sqlDataReader = getDateCommand.ExecuteReader();

                while (sqlDataReader.Read())
                {
                    // Construct a string containing all information about the booking
                    string bookingInfo = "";
                    for (int i = 0; i < sqlDataReader.FieldCount; i++)
                    {
                        bookingInfo += $"{sqlDataReader.GetName(i)}: {sqlDataReader[i]}";
                    }
                    bookings.Add(bookingInfo);
                }


                connection.Close();
                return bookings;
            }
        }

        public static bool GuestBookedBefore(string guestID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand bookedBeforeCommand = new SqlCommand();
                bookedBeforeCommand.Connection = connection;

                bookedBeforeCommand.CommandType =
System.Data.CommandType.StoredProcedure;

                bookedBeforeCommand.CommandText = "CheckGuestBooking";

                bookedBeforeCommand.Parameters.Add(new SqlParameter("@GuestID",
guestID));

                int rowsAffected = bookedBeforeCommand.ExecuteNonQuery();

                connection.Close();
                return true;
            }

        }

        public static Bookings GetBookingObjectByID(int bookingID)
```

```
        {
            Bookings booking = null;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                string sqlQuery = "SELECT * FROM Booking WHERE BookingID = @BookingID";

                SqlCommand getBookingByIDCommand = new SqlCommand(sqlQuery, connection);
                getBookingByIDCommand.Parameters.AddWithValue("@BookingID", bookingID);

                SqlDataReader reader = getBookingByIDCommand.ExecuteReader();

                if (reader.Read())
                {
                    // Populate a Bookings object with data from the database
                    booking = new Bookings
                    {
                        BookingID = (int)reader["BookingID"],
                        GuestID = reader["GuestID"].ToString(),
                        PodID = (int)reader["PodID"],
                        CourseID = (int)reader["CourseID"],
                        NumberOfOccupants = (int)reader["NumberOfOccupants"],
                        CheckInDate = (DateTime)reader["CheckInDate"],
                        CheckOutDate = (DateTime)reader["CheckOutDate"],
                        BookingStatus = reader["BookingStatus"].ToString(),
                        DepositAmount = (int)reader["DepositAmount"],
                        DiscountPercentage = (int)reader["DiscountPercentage"],
                        TotalAmount = (int)reader["TotalAmount"]
                    };
                }

                connection.Close();
            }

            return booking;
        }

        public static int UpdateBooking(string[] bookingInfo)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand updateBookingCommand = new SqlCommand();
                updateBookingCommand.Connection = connection;
                updateBookingCommand.CommandType =
System.Data.CommandType.StoredProcedure;
                updateBookingCommand.CommandText = "UpdateBooking";

                // Retrieve the datetime string from the array
                string datetimeString = bookingInfo[5];
                string datetimeString2 = bookingInfo[6];
                string dateBookedString = bookingInfo[11];

                // Convert the string to a DateTime object
                DateTime bookingDateTime;
```

```
                DateTime bookingDateTime2;
                DateTime dateBooked;

                if (DateTime.TryParse(datetimeString, out bookingDateTime) &&
DateTime.TryParse(datetimeString2, out bookingDateTime2) &&
DateTime.TryParse(dateBookedString, out dateBooked))
                {
                    updateBookingCommand.Parameters.Add(new SqlParameter("@BookingID",
bookingInfo[0]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@GuestID",
bookingInfo[1]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@PodID",
bookingInfo[2]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@CourseID",
bookingInfo[3]));
                    updateBookingCommand.Parameters.Add(new
SqlParameter("@NumberOfOccupants", bookingInfo[4]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@CheckInDate",
bookingDateTime));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@CheckOutDate",
bookingDateTime2));
                    updateBookingCommand.Parameters.Add(new
SqlParameter("@BookingStatus", bookingInfo[7]));
                    updateBookingCommand.Parameters.Add(new
SqlParameter("@DepositAmount", bookingInfo[8]));
                    updateBookingCommand.Parameters.Add(new
SqlParameter("@DiscountPercentage", bookingInfo[9]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@TotalAmount",
bookingInfo[10]));
                    updateBookingCommand.Parameters.Add(new SqlParameter("@DateBooked",
dateBooked));
                }
                else
                {
                    Console.WriteLine("Invalid DateTime format");
                    return 0; // Or handle the error as appropriate for your application
                }

                int rowsAffected = updateBookingCommand.ExecuteNonQuery();

                connection.Close();

                return rowsAffected;
            }
        }

        public static bool DoesBookingOverlap(string podId, DateTime checkInDate,
DateTime checkOutDate, int bookingID)
        {
            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();
                string query = "SELECT COUNT(*) FROM Booking WHERE PodId = @PodId AND
CheckOutDate > @CheckInDate AND CheckInDate < @CheckOutDate";
                if (bookingID > 0)
                {
                    // If a bookingID is passed, exclude it from the overlapping bookings
check
```

```
                    query += " AND BookingID <> @BookingID";
                }
                SqlCommand command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@PodId", podId);
                command.Parameters.AddWithValue("@CheckInDate", checkInDate);
                command.Parameters.AddWithValue("@CheckOutDate", checkOutDate);
                if (bookingID > 0)
                {
                    // If a bookingID is passed, set its value as a parameter
                    command.Parameters.AddWithValue("@BookingID", bookingID);
                }
                int overlappingBookingsCount = (int)command.ExecuteScalar();

                // If overlapping bookings count is 0, then there are no overlapping
bookings
                return overlappingBookingsCount == 0;
            }
        }


        public static bool DoesNoOccsExceedCapacity(int podID, int numberOfOccupants)
        {
            int podCapacity = PodDal.getPodCapacity(podID);
            if (podCapacity >= numberOfOccupants)
            {
                return false; // Occupants do not exceed pod capacity
            }
            else
            {
                return true; // Occupants exceed pod capacity
            }
        }

        public static int UpdateBookingStatusToPermanent(int bookingID)
        {
            int rowsAffected = 0;

            using (SqlConnection connection = new SqlConnection(_connectionstring))
            {
                connection.Open();

                SqlCommand updateStatusCommand = new SqlCommand();
                updateStatusCommand.Connection = connection;
                updateStatusCommand.CommandType = CommandType.StoredProcedure;
                updateStatusCommand.CommandText = "UpdateBookingStatusToPermanent";

                updateStatusCommand.Parameters.Add(new SqlParameter("@BookingID",
bookingID));

                rowsAffected = updateStatusCommand.ExecuteNonQuery();

                connection.Close();
            }

            return rowsAffected;
        }

        public static DateTime getDateBooked(int bookingId)
```

```
{
    DateTime dateBooked = DateTime.MinValue;

    using (SqlConnection connection = new SqlConnection(_connectionstring))
    {
        connection.Open();

        SqlCommand getDateBookedCommand = new SqlCommand();
        getDateBookedCommand.Connection = connection;
        getDateBookedCommand.CommandType = CommandType.StoredProcedure;
        getDateBookedCommand.CommandText = "GetDateBooked";

        getDateBookedCommand.Parameters.AddWithValue("@BookingID", bookingId);

        // Execute the command to retrieve the date booked
        object result = getDateBookedCommand.ExecuteScalar();

        // Check if the result is not null and convert it to string
        if (result != null)
        {
            dateBooked = (DateTime)result;
        }

        connection.Close();
    }

    return dateBooked;
}


public static int DeleteExpiredBookings()
{
        int rowsAffected = 0;

     using (SqlConnection connection = new SqlConnection(_connectionstring))
     {
         connection.Open();

         // Calculate the cutoff date (3 days older than current date time)
         DateTime cutoffDate = DateTime.Now.AddDays(-3);

         // Prepare the SQL command to delete bookings
         SqlCommand deleteCommand = new SqlCommand();
         deleteCommand.Connection = connection;
         deleteCommand.CommandType = CommandType.StoredProcedure;
         deleteCommand.CommandText = "DeleteExpiredBookings";
         deleteCommand.Parameters.AddWithValue("@CutoffDate", cutoffDate);

         // Execute the command and get the number of rows affected
         rowsAffected = deleteCommand.ExecuteNonQuery();

         connection.Close();
     }

     return rowsAffected;
}

public static string GetGuestIDByBookingID(int bookingID)
```

```
{
    string guestID = null;

    using (SqlConnection connection = new SqlConnection(_connectionstring))
    {
        connection.Open();

        SqlCommand command = new SqlCommand();
        command.Connection = connection;
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = "GetGuestIDByBookingID";

        // Add parameters
        command.Parameters.AddWithValue("@BookingID", bookingID);

        // Execute the command to retrieve the guest ID
        object result = command.ExecuteScalar();

        // Check if the result is not null and convert it to string
        if (result != null)
        {
            guestID = result.ToString();
        }

        connection.Close();
    }

    return guestID;
}

public static int GetPodIDByBookingID(int bookingID)
{
    int podID = -1; // Initialize to an invalid value

    using (SqlConnection connection = new SqlConnection(_connectionstring))
    {
        connection.Open();

        SqlCommand command = new SqlCommand();
        command.Connection = connection;
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = "GetPodIDByBookingID";

        // Add parameters
        command.Parameters.AddWithValue("@BookingID", bookingID);

        // Execute the command to retrieve the pod ID
        object result = command.ExecuteScalar();

        // Check if the result is not null and convert it to an integer
        if (result != null && result != DBNull.Value)
        {
            podID = Convert.ToInt32(result);
        }

        connection.Close();
    }
```

```
    return podID;
}

public static int GetCourseIDByBookingID(int bookingID)
{
    int courseID = -1; // Initialize to an invalid value

    using (SqlConnection connection = new SqlConnection(_connectionstring))
    {
        connection.Open();

        SqlCommand command = new SqlCommand();
        command.Connection = connection;
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = "GetCourseIDByBookingID";

        // Add parameters
        command.Parameters.AddWithValue("@BookingID", bookingID);

        // Execute the command to retrieve the course ID
        object result = command.ExecuteScalar();

        // Check if the result is not null and convert it to an integer
        if (result != null && result != DBNull.Value)
        {
            courseID = Convert.ToInt32(result);
        }

        connection.Close();
    }

    return courseID;
}

public static string[] GetBookingDetailsByBookingID(int bookingID)
{
    string[] bookingDetails = new string[5];

    using (SqlConnection connection = new SqlConnection(_connectionstring))
    {
        connection.Open();

        SqlCommand command = new SqlCommand();
        command.Connection = connection;
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = "GetBookingDetailsByBookingID";

        // Add parameters
        command.Parameters.AddWithValue("@BookingID", bookingID);

        SqlDataReader reader = command.ExecuteReader();

        if (reader.Read())
        {
            // Retrieve the values from the reader and store them in the array
            bookingDetails[0] = reader["CheckInDate"].ToString();
            bookingDetails[1] = reader["CheckOutDate"].ToString();
            bookingDetails[2] = reader["NumberOfOccupants"].ToString();
```

```
                    bookingDetails[3] = reader["DateBooked"].ToString();
                    // Read and store the total amount
                    bookingDetails[4] = reader["TotalAmount"].ToString();
                }

                connection.Close();
            }

            return bookingDetails;
        }

    }
}
```

ValidationClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LaksideEscapesBookingSystem
{
    class ValidationClass
    {

        public static bool bookedInAdvance(DateTime dateWanted, DateTime dateWanted2)
        {
            DateTime currentDate = DateTime.Now;

            // Check if the date is within the prohibited period (20th December to 20th
January)
            if (((dateWanted.Month == 12 && dateWanted.Day >= 20) ||
                 (dateWanted.Month == 1 && dateWanted.Day <= 20)) &&
                 ((dateWanted2.Month == 12 && dateWanted2.Day >= 20) ||
                 (dateWanted2.Month == 1 && dateWanted2.Day <= 20)))
            {
                return false; // Booking not allowed during this period
            }
            if (dateWanted >= currentDate.AddMonths(2))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        public static bool bookedInSixMonthsAdvance(DateTime dateWanted)
        {
            DateTime currentDate = DateTime.Now;

            // Check if the date is within the prohibited period (20th December to 20th
January)
            if ((dateWanted.Month == 12 && dateWanted.Day >= 20) || (dateWanted.Month ==
1 && dateWanted.Day <= 20))
            {
                return false; // Booking not allowed during this period
            }
            if (dateWanted >= currentDate.AddMonths(6))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
```

```
        public static string calculateDeposit(int PodID, int CourseID, int noOccs, int
noOfDays)
        {
            //initialize deposit cost
            int depositCost = -1;

            //Gets courseCostPerPerson and base costPerNight
            int podCost = PodDal.getPodCost(PodID);
            int courseCost = CourseDal.getCourseCost(CourseID);

            //times cost per person by numOfOccs times cost per night by nights staying
            podCost = podCost * noOfDays;
            courseCost = courseCost * noOccs;

            //add values
            depositCost = podCost + courseCost;

            //return value as a string
            return depositCost.ToString();
        }

        public static string calculateTotal(int PodID, int CourseID, int noOccs, int
noOfDays)
        {
            //initialize deposit cost
            int depositCost = -1;

            //Gets courseCostPerPerson and base costPerNight
            int podCost = PodDal.getPodCost(PodID);
            int courseCost = CourseDal.getCourseCost(CourseID);

            //times cost per person by numOfOccs times cost per night by nights staying
            podCost = podCost * noOfDays;
            courseCost = courseCost * noOccs;

            //add values
            depositCost = podCost + courseCost;

            //Total is 2x deposit
            depositCost =  depositCost * 2;

            //return value as a string
            return depositCost.ToString();
        }

        public static string calculateDiscount(string guestID, DateTime dateBooked)
        {
            if (bookedInSixMonthsAdvance(dateBooked) &&
BookingDal.GuestBookedBefore(guestID))
            {
                return "5";
            }
            else if (bookedInSixMonthsAdvance(dateBooked))
            {
                return "2";
            }
            else if (BookingDal.GuestBookedBefore(guestID))
```

```
            {
                return "3";
            }
            else
            {
                return "0";
            }
        }

        public static bool canBookingBeMadePermanent(DateTime dateBooked)
        {
            DateTime now = DateTime.Now;
            TimeSpan difference = now - dateBooked;
            TimeSpan threeDays = TimeSpan.FromDays(3);

            // If the difference is greater than or equal to three days, booking can not
be made permanent
            if (difference >= threeDays)
            {
                return false;
            }
            else
            {
                return true;
            }
        }


        public static bool validateEmail(string email)
        {
            if (!String.IsNullOrEmpty(email))
            {
                int countSymbol = 0;
                for (int i = 0; i < email.Length; i++)
                {
                    if (email[i] == '@')
                    {
                        countSymbol++;
                    }
                }

                if (countSymbol == 1)
                {
                    return true;
                }
            }
            return false;
        }

        public static bool validatePhoneNumber(string phoneNumber)
        {
            // Check if phone number is not null or empty
            if (!string.IsNullOrEmpty(phoneNumber))
            {
                // Check if phone number has at least 11 characters
                if (phoneNumber.Length >= 11)
                {
                    // Check each character of the phone number
```

```
                    foreach (char c in phoneNumber)
                    {
                        // If any character is not a digit, return false
                        if (!char.IsDigit(c))
                        {
                            return false;
                        }
                    }
                    // All characters are digits
                    return true;
                }
            }
            // Null, empty, or less than 11 characters
            return false;
        }

        public static int guestLengthOfStay(DateTime start, DateTime end)
        {
            TimeSpan difference = end.Subtract(start);
            return difference.Days;
        }

        public static string errorBuilderBookingForm(string guestID, string podID, string
courseID, string noOfOccupants)
        {
            string errorMessage = "THE FOLLOWING ERROS HAVE OCCURRED:";
            if (string.IsNullOrEmpty(guestID))
            {
                errorMessage += "----A GuestID has not been selected----";
            }
            if (string.IsNullOrEmpty(podID))
            {
                errorMessage += "----A PodID has not been selected----";
            }
            if (string.IsNullOrEmpty(courseID))
            {
                errorMessage += "----A CourseID has not been selected----";
            }
            if (string.IsNullOrEmpty(noOfOccupants))
            {
                errorMessage += "----A Number of occupants has not been selected----";
            }
            return errorMessage;
        }

        public static string GenerateGuestID(string firstName, string lastName)
        {
            // Take the first letter of the first name and last name
            string initials = firstName.Substring(0, 1) + lastName.Substring(0, 1);

            // Generate a random 3-digit number
            Random rnd = new Random();
            int randomNumber = rnd.Next(100, 1000); // generates a random number between
100 and 999

            // Concatenate the initials and the random number to create the guest ID
            string guestID = initials + randomNumber.ToString();
```

```
            return guestID;
        }
    }
}
```

Booking.cs (Booking Model)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LaksideEscapesBookingSystem.Models
{
    class Bookings
    {
        public int BookingID { get; set; }
        public string GuestID { get; set; }
        public int PodID { get; set; }
        public int CourseID { get; set; }
        public int NumberOfOccupants { get; set; }
        public DateTime CheckInDate { get; set; }
        public DateTime CheckOutDate { get; set; }
        public string BookingStatus { get; set; }
        public int DepositAmount { get; set; }
        public int DiscountPercentage { get; set; }
        public int TotalAmount { get; set; }
        public DateTime DateBooked { get; set; }

        public Bookings(string guestID, int podID, int courseID, int numberOfOccupants,
DateTime checkInDate, DateTime checkOutDate, string bookingStatus, int depositAmount, int
discountPercentage, int totalAmount, DateTime dateBooked)
        {
            GuestID = guestID;
            PodID = podID;
            CourseID = courseID;
            NumberOfOccupants = numberOfOccupants;
            CheckInDate = checkInDate;
            CheckOutDate = checkOutDate;
            BookingStatus = bookingStatus;
            DepositAmount = depositAmount;
            DiscountPercentage = discountPercentage;
            TotalAmount = totalAmount;
            DateBooked = dateBooked;
        }
        public Bookings()
        {

        }
    }
}
```

Courses.cs (Course Model)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LaksideEscapesBookingSystem.Models
{
    class Courses
    {
        public int CourseID { get; set; }
        public string CourseName { get; set; }
        public int CourseCostPerPerson { get; set; }

        public Courses(int courseID, string courseName, int courseCostPerPerson)
        {
            CourseID = courseID;
            CourseName = courseName;
            CourseCostPerPerson = courseCostPerPerson;
        }

        public Courses()
        {

        }

    }
}
```

Guests.cs (Guest Model)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LaksideEscapesBookingSystem.Models
{
    class Guests
    {
        public string GuestID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }
        public string Title { get; set; }
        public string AddressLine1 { get; set; }
        public string AddressLine2 { get; set; }
        public string PreviouslyBooked { get; set; }

        public Guests(string guestID, string firstName, string lastName, string email,
string phoneNumber, string title, string addressLine1, string addressLine2, string
previouslyBooked)
        {
            GuestID = guestID;
            FirstName = firstName;
            LastName = lastName;
            Email = email;
            PhoneNumber = phoneNumber;
            Title = title;
            AddressLine1 = addressLine1;
            AddressLine2 = addressLine2;
            PreviouslyBooked = previouslyBooked;
        }

        public Guests()
        {

        }

    }
}
```

Pods.cs (Pod Model)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LaksideEscapesBookingSystem.Models
{
    class Pods
    {
        public int PodID { get; set; }
        public string PodType { get; set; }
        public string Capacity { get; set; }
        public int BaseCostPerNight { get; set; }

        public Pods(int podID, string podType, string capacity, int baseCostPerNight)
        {
            PodID = podID;
            PodType = podType;
            Capacity = capacity;
            BaseCostPerNight = baseCostPerNight;
        }

        public Pods()
        {

        }

    }
}
```

App.config (Configuration file)

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
  </configSections>
  <connectionStrings>
    <add name="BookingSystemConnectionString" connectionString="Data
Source=(LocalDB)\MSSQLLocalDb;AttachDbFilename=|DataDirectory|\BookingDatbase.mdf;Initial
Catalog=BookingDatabase;Integrated Security=True" providerName="System.Data.SqlClient" />
    <add name="BookingDatbaseEntities"
connectionString="metadata=res://*/Model2.csdl|res://*/Model2.ssdl|res://*/Model2.msl;pro
vider=System.Data.SqlClient;provider connection string=&quot;data
source=(LocalDB)\MSSQLLocalDB;attachdbfilename=|DataDirectory|\BookingDatbase.mdf;integra
ted security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.ReportViewer.WinForms"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-12.0.0.0" newVersion="12.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.SqlServer.Types"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-14.0.0.0" newVersion="14.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.ReportViewer.Common"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-12.0.0.0" newVersion="12.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.ReportViewer.ProcessingObjectModel"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-15.0.0.0" newVersion="15.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.ReportViewer.DataVisualization"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-15.0.0.0" newVersion="15.0.0.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
  <entityFramework>
```

```
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="mssqllocaldb" />
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
</configuration>
```

Program.cs (The main entry point for the application)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LaksideEscapesBookingSystem
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            SetupDataDirectoryPath();

            Application.Run(new BookingViewer());


        }

        private static void SetupDataDirectoryPath()
        {
            string debugPath =
System.IO.Path.GetDirectoryName(Environment.CurrentDirectory);
            string dataDirectoryPath = System.IO.Path.GetDirectoryName(debugPath);
            AppDomain.CurrentDomain.SetData("DataDirectory", dataDirectoryPath);
        }
    }
}
```

SQL Repository (Database Schema and Stored Procedures)

Booking Table

```
CREATE TABLE [dbo].[Booking] (
    [BookingId]          INT           IDENTITY (1, 1) NOT NULL,
    [GuestID]            NVARCHAR (50) NULL,
    [PodID]              INT           NULL,
    [CourseID]           INT           NULL,
    [NumberOfOccupants]  INT           NULL,
    [CheckInDate]        DATETIME      NULL,
    [CheckOutDate]       DATETIME      NULL,
    [BookingStatus]      NVARCHAR (50) NULL,
    [DepositAmount]      INT           NULL,
    [DiscountPercentage] INT           NULL,
    [TotalAmount]        INT           NULL,
    [DateBooked]         DATETIME      NULL,
    PRIMARY KEY CLUSTERED ([BookingId] ASC),
    CONSTRAINT [FK_Booking_Course] FOREIGN KEY ([CourseID]) REFERENCES
[dbo].[Course] ([CourseID]) ON DELETE CASCADE,
    CONSTRAINT [FK_Booking_Pod] FOREIGN KEY ([PodID]) REFERENCES [dbo].[Pod]
([PodID]) ON DELETE CASCADE,
    CONSTRAINT [FK_Booking_Guest] FOREIGN KEY ([GuestID]) REFERENCES
[dbo].[Guest] ([GuestID]) ON DELETE CASCADE
);
```

Course Table

```
CREATE TABLE [dbo].[Course] (
    [CourseID]            INT           IDENTITY (1, 1) NOT NULL,
    [CourseName]          NVARCHAR (50) NULL,
    [CourseCostPerPerson] INT           NULL,
    PRIMARY KEY CLUSTERED ([CourseID] ASC)
);
```

Guest Table

```
CREATE TABLE [dbo].[Guest] (
    [GuestID]          NVARCHAR (50) NOT NULL,
    [FirstName]        NVARCHAR (50) NULL,
    [LastName]         NVARCHAR (50) NULL,
    [Email]            NVARCHAR (50) NULL,
    [PhoneNumber]      NVARCHAR (50) NULL,
    [Title]            NVARCHAR (50) NULL,
    [AddressLine1]     NVARCHAR (50) NULL,
    [AddressLine2]     NVARCHAR (50) NULL,
    [PreviouslyBooked] NVARCHAR (50) NULL,
    PRIMARY KEY CLUSTERED ([GuestID] ASC)
);
```

Pod Table

```
CREATE TABLE [dbo].[Pod] (
    [PodID]            INT           IDENTITY (1, 1) NOT NULL,
    [PodType]          NVARCHAR (50) NULL,
    [Capacity]         NVARCHAR (50) NULL,
    [BaseCostPerNight] INT           NULL,
    PRIMARY KEY CLUSTERED ([PodID] ASC)
);
```

AddBooking (Stored Procedure)

```
CREATE PROCEDURE [dbo].AddBooking
    @GuestID            NVARCHAR (50),
    @PodID              NVARCHAR (50) ,
    @CourseID           NVARCHAR (50) ,
    @NumberOfOccupants  NVARCHAR (50) ,
    @CheckInDate        DATETIME      ,
    @CheckOutDate       DATETIME      ,
    @BookingStatus      NVARCHAR (50) ,
    @DepositAmount      INT           ,
    @DiscountPercentage INT           ,
    @TotalAmount        INT                 ,
      @DateBooked             DATETIME
AS
       INSERT INTO Booking
       VALUES(@GuestID, @PodID, @CourseID, @NumberOfOccupants, @CheckInDate,
@CheckOutDate, @BookingStatus, @DepositAmount, @DiscountPercentage,
@TotalAmount, @DateBooked)
RETURN 0
```

AddCourse (Stored Procedure)

```
CREATE PROCEDURE [dbo].AddCourse
    @CourseName            NVARCHAR (50),
    @CourseCostPerPerson           int
AS
      INSERT INTO Course
      Values( @CourseName, @CourseCostPerPerson)
RETURN 0
```

AddGuest (Stored Procedure)

```
CREATE PROCEDURE [dbo].AddGuest
      @GuestID                NVARCHAR (50) ,
    @FirstName                NVARCHAR (50),
    @LastName           NVARCHAR (50) ,
    @Email                    NVARCHAR (50) ,
    @PhoneNumber        NVARCHAR (50),
      @Title                        NVARCHAR (50),
      @AddressLine1           NVARCHAR (50),
      @AddressLine2           NVARCHAR (50),
      @PreviouslyBooked NVARCHAR (50)
AS
      INSERT INTO Guest
      Values(@GuestID, @FirstName, @LastName, @Email, @PhoneNumber, @Title,
@AddressLine1, @AddressLine2, @PreviouslyBooked)
RETURN 0
```

AddPod (Stored Procedure)

```
CREATE PROCEDURE [dbo].AddPod
    @PodType            NVARCHAR (50),
    @Capacity              NVARCHAR (50) ,
    @BaseCostPerNight           int
AS
      INSERT INTO Pod
      Values( @PodType, @Capacity, @BaseCostPerNight)
RETURN 0
```

BookingLengthOfStay (SP)

```
CREATE PROCEDURE [dbo].BookingLengthOfStay
      @BookingID INT
AS
      DECLARE @StartDate DATETIME;
      DECLARE @EndDate DATETIME;

      SELECT @StartDate = CheckInDate, @EndDate = CheckInDate
    FROM Booking
    WHERE BookingID = @BookingID;
RETURN 0
```

BookingOccupants (SP)

```
CREATE PROCEDURE [dbo].BookingOccupants
    @BookingID INT
AS
BEGIN
    DECLARE @NoOccs INT;

    -- Initialize @NoOccs to a default value
    SET @NoOccs = -1;

    -- Retrieve the NumberOfOccupants for the given BookingID
    SELECT @NoOccs = NumberOfOccupants
    FROM Booking
    WHERE BookingID = @BookingID;

    -- Check if BookingID exists
    IF @NoOccs IS NULL
    BEGIN
        -- If BookingID doesn't exist, set @NoOccs to -1
        SET @NoOccs = -1;
    END

    -- Return the value of @NoOccs
    SELECT @NoOccs;
END;
```

CalculateMonthlyRevenue (SP)

```sql
CREATE PROCEDURE CalculateMonthlyRevenue_2024
AS
BEGIN
    DECLARE @MonthlyRevenue TABLE (
        Month INT,
        MonthlyRevenue DECIMAL(18, 2)
    )

    INSERT INTO @MonthlyRevenue (Month, MonthlyRevenue)
    SELECT
        DATEPART(MONTH, b.CheckInDate) AS Month,
        SUM(b.TotalAmount) AS MonthlyRevenue
    FROM
        Booking b
    WHERE
        YEAR(b.CheckInDate) = 2024
        AND b.BookingStatus = 'Permanent'
    GROUP BY
        DATEPART(MONTH, b.CheckInDate)
    ORDER BY
        Month;

    -- Return the monthly revenue
    SELECT * FROM @MonthlyRevenue;
END;
```

CheckGuestBooking (SP)

```
CREATE PROCEDURE CheckGuestBooking
    @GuestID NVARCHAR (50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @BookingCount INT;

    -- Check if the GuestID exists in the Booking table
    SELECT @BookingCount = COUNT(*)
    FROM Booking
    WHERE GuestID = @GuestID;

    -- Return 1 if the GuestID exists, otherwise return 0
    IF @BookingCount > 0
        SELECT 1 AS 'Exists';
    ELSE
        SELECT 0 AS 'Exists';
END;
```

CourseCost (SP)

```
CREATE PROCEDURE [dbo].CourseCost
    @CourseID INT
AS
BEGIN
    DECLARE @CourseCost INT;

    -- Initialize @CourseCost to a default value
    SET @CourseCost = -1;

    -- Retrieve the CourseCost for the given CourseID
    SELECT @CourseCost = CourseCostPerPerson
    FROM Course
    WHERE CourseID = @CourseID;

    -- Check if CourseID exists
    IF @CourseCost IS NULL
    BEGIN
        -- If CourseID doesn't exist, set @CourseCost to -1
        SET @CourseCost = -1;
    END

    -- Return the value of @CourseCost
    SELECT @CourseCost;
END;
```

DeleteBooking (SP)

```
CREATE PROCEDURE [dbo].DeleteBooking
      @BookingId int
AS
      DELETE FROM Booking
      WHERE BookingId=@BookingId
RETURN 0
```

DeleteCourse (SP)

```
CREATE PROCEDURE [dbo].DeleteCourse
      @CourseID NVARCHAR (50)
AS
      DELETE FROM Course
      WHERE CourseID=@CourseID
RETURN 0
```

DeleteExpiredBookings (SP)

```
CREATE PROCEDURE DeleteExpiredBookings
    @CutoffDate DATETIME
AS
BEGIN
    -- Delete bookings where the booking status is not permanent and
the date booked is older than the cutoff date
    DELETE FROM Booking
    WHERE BookingStatus != 'Permanent' AND DateBooked <= @CutoffDate;
END
```

DeleteGuest (SP)

```
CREATE PROCEDURE [dbo].DeleteGuest
     @GuestID NVARCHAR (50)
AS
     DELETE FROM Guest
     WHERE GuestID=@GuestID
RETURN 0
```

DeletePod (SP)

```
CREATE PROCEDURE [dbo].DeletePod
      @PodID NVARCHAR (50)
AS
      DELETE FROM Pod
      WHERE PodID=@PodID
RETURN 0
```

GetBookingDetailsByBookingID (SP)

```
CREATE PROCEDURE GetBookingDetailsByBookingID
    @BookingID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT CheckInDate, CheckOutDate, NumberOfOccupants, DateBooked,
TotalAmount
    FROM Booking
    WHERE BookingID = @BookingID;
END
```

GetCourseIDByBookingID (SP)

```
CREATE PROCEDURE GetCourseIDByBookingID
    @BookingID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT CourseID
    FROM Booking
    WHERE BookingID = @BookingID;
END
```

GetCourseNameByCourseID (SP)

```
CREATE PROCEDURE GetCourseNameByCourseID
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT CourseName
    FROM Course
    WHERE CourseID = @CourseID;
END
```

GetDateBooked (SP)

```
CREATE PROCEDURE GetDateBooked
    @BookingId INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Declare a variable to store the date booked
    DECLARE @DateBooked DATETIME

    -- Retrieve the date booked for the specified BookingID
    SELECT @DateBooked = DateBooked
    FROM Booking
    WHERE BookingId = @BookingId

    -- Return the date booked
    SELECT @DateBooked AS DateBooked
END
```

GetGuestIDByBookingID (SP)

```
CREATE PROCEDURE GetGuestIDByBookingID
    @BookingID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT GuestID
    FROM Booking
    WHERE BookingID = @BookingID;
END
```

GetMostPopularCourses (SP)

```
CREATE PROCEDURE GetMostPopularCourses
AS
BEGIN
    DECLARE @MostPopularCourses TABLE (
        CourseID INT,
        CourseName NVARCHAR(50),
        CourseCostPerPerson INT,
        BookingCount INT
    )

    INSERT INTO @MostPopularCourses (CourseID, CourseName,
CourseCostPerPerson, BookingCount)
    SELECT TOP 5 c.CourseID, c.CourseName,
c.CourseCostPerPerson, COUNT(b.BookingID) AS BookingCount
    FROM Course c
    INNER JOIN Booking b ON c.CourseID = b.CourseID
    WHERE b.BookingStatus = 'Permanent' -- Consider only
permanent bookings
    GROUP BY c.CourseID, c.CourseName, c.CourseCostPerPerson
    ORDER BY COUNT(b.BookingID) DESC;

    -- Return the list of most popular courses
    SELECT * FROM @MostPopularCourses;
END;
```

GetMostPopularPods (SP)

```
CREATE PROCEDURE GetMostPopularPods
AS
BEGIN
    DECLARE @MostPopularPods TABLE (
        PodID INT,
        PodType NVARCHAR(50),
        Capacity NVARCHAR(50),
        BookingCount INT
    )

    INSERT INTO @MostPopularPods (PodID, PodType, Capacity,
BookingCount)
    SELECT TOP 5 p.PodID, p.PodType, p.Capacity,
COUNT(b.BookingID) AS BookingCount
    FROM Pod p
    INNER JOIN Booking b ON p.PodID = b.PodID
    WHERE b.BookingStatus = 'Permanent' -- Consider only
permanent bookings
    GROUP BY p.PodID, p.PodType, p.Capacity
    ORDER BY COUNT(b.BookingID) DESC;

    -- Return the list of most popular pods
    SELECT * FROM @MostPopularPods;
END;
```

GetPodByBookingID (SP)

```
CREATE PROCEDURE GetPodIDByBookingID
    @BookingID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT PodID
    FROM Booking
    WHERE BookingID = @BookingID;
END
```

GetPodTypeByPodID (SP)

```
CREATE PROCEDURE GetPodTypeByPodID
    @PodID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT PodType
    FROM Pod
    WHERE PodID = @PodID;
END
```

PodCost (SP)

```
CREATE PROCEDURE [dbo].PodCost
    @PodID INT
AS
BEGIN
    DECLARE @PodCost INT;

    -- Initialize @PodCost to a default value
    SET @PodCost = -1;

    -- Retrieve the PodCost for the given PodID
    SELECT @PodCost = BaseCostPerNight
    FROM Pod
    WHERE PodID = @PodID;

    -- Check if PodID exists
    IF @PodCost IS NULL
    BEGIN
        -- If PodID doesn't exist, set @PodCost to -1
        SET @PodCost = -1;
    END

    -- Return the value of @PodCost
    SELECT @PodCost;
END;
```

UpdateBooking (SP)

```
CREATE PROCEDURE [dbo].UpdateBooking
    @BookingID          INT,
    @GuestID            NVARCHAR(50),
    @PodID              NVARCHAR(50),
    @CourseID           NVARCHAR(50),
    @NumberOfOccupants  NVARCHAR(50),
    @CheckInDate        DATETIME,
    @CheckOutDate       DATETIME,
    @BookingStatus      NVARCHAR(50),
    @DepositAmount      INT,
    @DiscountPercentage INT,
    @TotalAmount        INT,
     @DateBooked            DATETIME
AS
BEGIN
    UPDATE Booking
    SET
        GuestID = @GuestID,
        PodID = @PodID,
        CourseID = @CourseID,
        NumberOfOccupants = @NumberOfOccupants,
        CheckInDate = @CheckInDate,
        CheckOutDate = @CheckOutDate,
        BookingStatus = @BookingStatus,
        DepositAmount = @DepositAmount,
        DiscountPercentage = @DiscountPercentage,
        TotalAmount = @TotalAmount,
         DateBooked = @DateBooked
    WHERE
        BookingID = @BookingID;

    RETURN 0;
END
```

UpdateBookingStatusToPermanent (SP)

```
CREATE PROCEDURE UpdateBookingStatusToPermanent
    @BookingId INT
AS
BEGIN
    UPDATE Booking
    SET BookingStatus = 'Permanent'
    WHERE BookingId = @BookingId;
END
```

UpdateCourse (SP)

```
CREATE PROCEDURE [dbo].UpdateCourse
    @CourseID                INT,
    @CourseName            NVARCHAR(50),
    @CourseCostPerPerson   INT
AS
BEGIN
    UPDATE Course
    SET
        CourseName = @CourseName,
        CourseCostPerPerson = @CourseCostPerPerson
    WHERE
        CourseID = @CourseID;

    RETURN 0;
END
```

UpdateGuest (SP)

```
CREATE PROCEDURE [dbo].UpdateGuest
     @GuestID            NVARCHAR(50),
     @FirstName               NVARCHAR(50),
       @LastName         NVARCHAR(50),
    @Email                   NVARCHAR(50),
    @PhoneNumber         NVARCHAR(50),
     @Title                   NVARCHAR (50),
     @AddressLine1       NVARCHAR (50),
     @AddressLine2       NVARCHAR (50)
AS
BEGIN
    UPDATE Guest
    SET
        FirstName = @FirstName,
        LastName = @LastName,
        Email = @Email,
        PhoneNumber = @PhoneNumber,
         Title = @Title,
         AddressLine1 = @AddressLine1,
         AddressLine2 = @AddressLine2
    WHERE
        GuestID = @GuestID;

    RETURN 0;
END
```

UpdatePod (SP)

```
CREATE PROCEDURE [dbo].UpdatePod
     @PodID                      INT,
    @PodType             NVARCHAR(50),
     @Capacity            NVARCHAR(50),
    @BaseCostPerNight    INT
AS
BEGIN
    UPDATE Pod
    SET
        PodType = @PodType,
        Capacity = @Capacity,
           BaseCostPerNight = @BaseCostPerNight
    WHERE
         PodID = @PodID;

    RETURN 0;
END
```