**Hotel Room Booking System**

**Lerato Phetle**

[Overview 1](#_Toc1225295375)

[Introduction 1](#_Toc1157388204)

[The system will consist of the following validations: 1](#_Toc1709430322)

[System Users: 1](#_Toc860424199)

[Features that are required to make the hotel room booking system function are: 1](#_Toc1975032285)

[Product Backlog 1](#_Toc1530484237)

[Sprint Backlogs (in a tabular format) 2](#_Toc906318011)

[Sprint backlog - Week 5 .NET Angular 2](#_Toc1383759887)

[Sprint backlog - Week 4 .NET ASP.NET/Blazor 3](#_Toc1895655194)

[Sprint backlog - Week 3 .NET Web App 4](#_Toc1007161938)

[Sprint backlog - Week 2 .NET REST API 5](#_Toc1226695390)

[Sprint backlog - Week 1 .NET Data Access 6](#_Toc775610395)

[Sprint backlog – Week 4 OOD4 7](#_Toc1351206970)

[Sprint backlog – Week 3 OOD TDD Solo Project 8](#_Toc1550804077)

[Sprint backlog – Week 2 OOD2 9](#_Toc134585924)

[UML Diagrams: 10](#_Toc535646097)

[User Stories 10](#_Toc1160296668)

[Use Case Diagram 10](#_Toc1164413534)

[ERD 10](#_Toc477556608)

[Class Diagram 10](#_Toc2046699265)

[UI Flowchart 10](#_Toc103537498)

[Technical Architecture 10](#_Toc1809643890)

[Sample data for User Registration 10](#_Toc629218662)

[Sample data for User Login 10](#_Toc417487753)

[System logging (log file) and design 11](#_Toc441780908)

[Week4.ASPNET logging and design 11](#_Toc190417673)

[Week3.WebApp logging and design 13](#_Toc1570663861)

[List of REST API endpoints with description 15](#_Toc501955938)

[Assumptions and Dependencies 16](#_Toc2111162459)

[Lessons Learned 16](#_Toc1113298029)

[Lesson Learned – Week 5.ANGULAR 16](#_Toc358113729)

[Lesson Learned – Week 4.ASP.NET 16](#_Toc2042284061)

[Lesson Learned – Week 3.WEBAPP 16](#_Toc1369910754)

[Lesson Learned – Week 2 .NET REST API 16](#_Toc1320860477)

[Lesson Learned - Week 1 .NET Data Access 16](#_Toc1771264487)

[Lesson Learned - Week 4 OOD4 16](#_Toc414065262)

[Lesson Learned - Week 3 OOD3 16](#_Toc1698083402)

[Lesson Learned - Week 2 OOD2 16](#_Toc455490932)

[Lesson Learned – Week 1 OOD1 16](#_Toc1219917072)

# **Overview**

## Introduction

A hotel room booking system is a tool used to manage bookings, payments, guest details, room availability and overall hotel operations. This system is for Safari Hotel. The application is designed to streamline the booking process for customers looking to reserve hotel rooms. The system aims to provide a user-friendly interface for customers to search for available rooms, make reservations, view bookings details, and receive confirmation details seamlessly. The system will facilitate the reservation and optimization of hotel rooms and related services. It will solely be designed for guests/customers to search for available rooms, make bookings, manage reservations, and access information about hotel amenities, rates, and policies.

## The system will consist of the following validations:

**Data Validation:**

* **Input Validation:** Validate user inputs such as name, email, phone number, and payment information to ensure they are in the correct format (e.g., email validation for proper email format, phone number validation for digits only).
* **Date Validation:** Validate check-in and check-out dates to ensure they are valid dates and meet business rules (e.g., check-out date should be after the check-in date).
* **Price Validation:** Validate prices and total costs to ensure they are within acceptable ranges and calculations are accurate.
* **Room Availability:** Validate room availability based on selected dates and room types to prevent double bookings or bookings for unavailable rooms.

**Integration Validation:**

* **Third-Party Integrations:** Validate integrations with external APIs or services (e.g., mapping services, email/SMS notifications) to ensure seamless communication and functionality within the booking system.

**Testing and Quality Assurance:**

* **Unit Testing:** Conduct unit tests to validate individual components, functions, and modules within the booking system.
* **User Acceptance Testing (UAT):** Involve real users or testers in UAT to validate the system's functionality, usability, and adherence to business requirements from a customer's perspective.

By implementing these validation strategies, the hotel room booking system can ensure data integrity, security, compliance with business rules, and a positive user experience for customers throughout the booking process.

## System Users:

**Customers/Guests:**

Customers or guests are the main users of the hotel room booking system.

**How:**

* Access the Booking Website/App: Customers access the system through a user-friendly website or mobile application specifically designed for booking hotel rooms.
* Search for Rooms: Customers search for available rooms by entering their desired check-in and check-out dates, selecting room types (e.g., standard, deluxe), specifying the number of guests.
* Make Reservations: Customers can book rooms by selecting desired dates, entering personal information (name, contact details), and providing payment information (credit/debit card details).
* Manage Bookings: Customers manage their bookings by viewing upcoming reservations, modifying booking details, canceling bookings.
* Receive Confirmation: Customers receive booking confirmation via email or SMS, containing booking details, payment receipt, check-in/check-out instructions, and contact information for the hotel.

Features that are required to make the hotel room booking system function are:

**Search and Filter Rooms:**

* Provide a user-friendly interface for customers to search for available rooms based on check-in and check-out dates, room types, number of guests, and other preferences.
* Implement filters and sorting options to help customers narrow down their search results and find suitable rooms quickly.

**Room Information and Images:**

* Display detailed information about each room, including room type, amenities, bed configurations, view, occupancy limits, and pricing per night.
* Include high-quality images or virtual tours of rooms to give customers a visual representation of the accommodation options.

**Booking Process:**

* Enable customers to select desired check-in and check-out dates, enter guest information (names, ages, special requests), and specify payment details for booking confirmation.
* Calculate total booking costs based on selected dates, room rates, additional services, taxes, and fees.
* Provide options for customers to choose payment methods (credit/debit cards, online payment gateways) and process secure transactions.

**Real-Time Availability Updates:**

* Implement a system that updates room availability in real time to prevent double bookings and ensure accurate availability information for customers.

**Reservation Management:**

* + Allow customers to view their upcoming reservations, modify booking details and cancel bookings (within the cancellation policy).
  + Send automated email or SMS notifications to customers confirming their bookings, providing reservation details, check-in instructions, and contact information for the hotel.

**Customer Support and Helpdesk:**

* Offer customer support channels such as email support, or phone assistance to address customer inquiries, aid with bookings, and resolve issues promptly.

## Product Backlog

|  |  |  |  |
| --- | --- | --- | --- |
| **.NET Project Backlog** | | | |
| **Sprint** | **Product backlog items** | **Priority** | **Status** |
| 1 | OOD Week1 - Research Project | 1 | Done |
| 2 | OOD Week2 - Code-based Solo Project | 2 | Done |
| 3 | OOD Week3 - TDD Solo Project | 3 | Done |
| 4 | 1 .NET Data Access | 4 | Done |
| 5 | 2 .NET REST API | 5 | Done |
| 6 | 3 .NET Web Apps | 6 | Done |
| 7 | 4 .NET ASP.NET | 7 | Done |
| 8 | 5 .NET Angular/React | 8 | Done |

# **Sprint Backlogs (in a tabular format)**

## Sprint backlog - Week 5 .NET Angular

|  |  |  |  |
| --- | --- | --- | --- |
| ID | ITEM | PRIORITY | STATUS |
| 1 | Update Requirement Specification document | 1 | In Progress |
| 2 | Create the landing page with Login and Registration | 1 | Done |
| 3 | Angular (performing at least CRUD operations) | 2 | Done |
| 4 | Angular Forms and validation | 3 | Done |
| 5 | Angular Services | 3 | Done |
| 6 | Angular Routing and Navigation | 2 | Done |
| 71 | Angular with logging framework | 2 | Done |

## Sprint backlog - Week 4 .NET ASP.NET/Blazor

|  |  |  |  |
| --- | --- | --- | --- |
| ID | ITEM | PRIORITY | STATUS |
| 1 | Update Requirement Specification document | 1 | Done |
| 2 | Create the landing page with Login and Registration | 1 | Done |
| 3 | ASP.NET/Blazor (performing at least CRUD operations) | 2 | Done |
| 4 | ASP.NET/Blazor with Security (AAA) | 2 | Done |
| 5 | ASP.NET/Blazor with design patterns | 3 | Done |
| 6 | ASP.NET/Blazor with TDD | 3 | Not Started |
| 7 | ASP.NET/Blazor with logging framework | 2 | Done |
| 8 | Update Rest API | 1 | Done |

## Sprint backlog - Week 3 .NET Web App

|  |  |  |  |
| --- | --- | --- | --- |
| ID | ITEM | PRIORITY | STATUS |
| 1 | Update Requirement Specification document | 1 | Done |
| 2 | Create the landing page/home page | 1 | Done |
| 3 | Update Rest API | 1 | Done |
| 4 | Create login and registration pages | 1 | Done |

## Sprint backlog - Week 2 .NET REST API

|  |  |  |  |
| --- | --- | --- | --- |
| ID | ITEM | PRIORITY | STATUS |
| 1 | SQL Server Database and SSMS | 1 | Done |
| 2 | Microservices | 1 | Done |
| 3 | Entity Framework - Code First | 1 | Done |
| 4 | Data Access - ADO.NET Connected | 1 | Done |
| 5 | HTTP protocol 5 | 1 | Done |
| 6 | REST API's performing CRUD operations | 2 | Done |
| 7 | REST API with Security (AAA) | 2 | Done |
| 8 | REST API with design patterns | 2 | Done |
| 9 | REST API with TDD | 3 | Done |
| 10 | REST API with logging framework | 2 | Done |

## Sprint backlog - Week 1 .NET Data Access

|  |  |  |  |
| --- | --- | --- | --- |
| ID | ITEM | PRIORITY | STATUS |
| 1 | Database Script | 1 | Done |
| 2 | Database Schema - ADO.NET | 1 | Done |
| 3 | Database Schema – EF Code First | 1 | Done |
| 4 | Data Access - ADO.NET Connected | 1 | Done |
| 5 | Data Access - ADO.NET Disconnected | 1 | Done |
| 6 | Data Access - ADO.NET Disconnected Procedure | 1 | Done |
| 7 | Entity Framework Code First | 1 | Done |
| 8 | Repository with EF (TDD CRUD) | 1 | Done |
| 9 | Update ERD | 1 | Done |

## Sprint backlog – Week 4 OOD4

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Items** | **Priority** | **Status** |
| 1 | Create Use cases and User Stories | 1 | Done |
| 2 | Class/Entities: Board, buyable tile, property and special bonus tile | 2 | Done |
| 3 | Create all classes responsible for | 2 | Done |
| 4 | Write tests for all entities responsible for | 2 | Done |
| 5 | Setup Logging for the application to ensure errors are logged | 2 | Done |
| 6 | Make use of log4net where appropriate for entities responsible for | 4 | Done |
| 7 | Implementation of SOLID, Generics and design patterns | 2 | Done |
| 8 | Implementation of four pillars in entities responsible for | 1 | Done |
| 9 | Build project using Test Driven Development | 1 | Done |
| 10 | Write IBoard ITileWithAction interface |  | Done |
| 11 | Code documentation | 8 | Done |

## Sprint backlog – Week 3 OOD TDD Solo Project

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Items** | **Priority** | **Status** |
| 1 | Write documentation comments and generate XML output documentation file | 2 | Done |
| 2 | TDD project configuration | 2 | Done |
| 3 | Code coverage | 3 | Done |
| 4 | Use of Version/Source Control 10 | 1 | Done |
| 5 | Using NUnit | 2 | Done |
| 6 | Write the implementation code using TDD approach | 2 | Done |
| 7 | Write passing NUnit tests using TDD approach | 2 | Done |
| 8 | Use exception handling | 3 | Done |
| 9 | Use reference and value datatypes and collections | 1 | Done |
| 10 | Use flow control statements | 1 | Done |
| 11 | Use of advanced features of NUnit | 1 | Done |
| 12 | Write the implementation code using TDD approach | 1 | Done |
| 13 | Write passing MOQ tests using TDD approach | 1 | Done |
| 14 | Use exception handling | 1 | Done |
| 15 | Use reference and value datatypes and collections | 1 | Done |
| 16 | Use flow control statements | 1 | Done |
| 17 | Use of advanced features of MOQ | 1 | Done |
| 18 | Outline lessons learnt | 1 | Done |
| 19 | Clean code | 1 | Done |
| 20 | Code indented properly, no unnecessary comments | 1 | Done |
| 21 | Good choice of variable names | 1 | Done |
| 22 | Classes should do one thing; they should do it well. | 1 | Done |
| 23 | Methods should be as small as possible | 1 | Done |
| 24 | There should only be one return statement in a Method | 1 | Done |

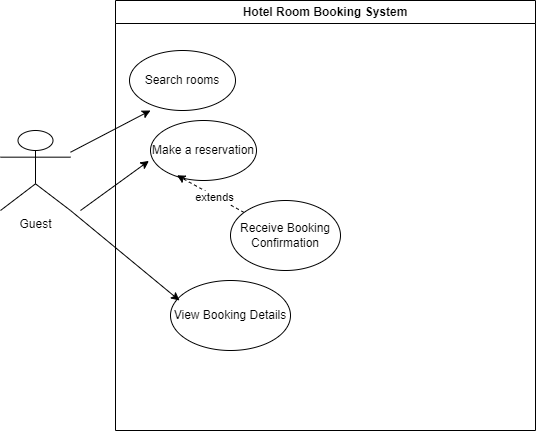
## Sprint backlog – Week 2 OOD2

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Items** | **Priority** | **Status** |
| 1 | View available rooms | 1 | Done |
| 2 | View My details | 1 | Done |
| 3 | View Reservations | 1 | Done |
| 4 | list guests | 2 | Done |

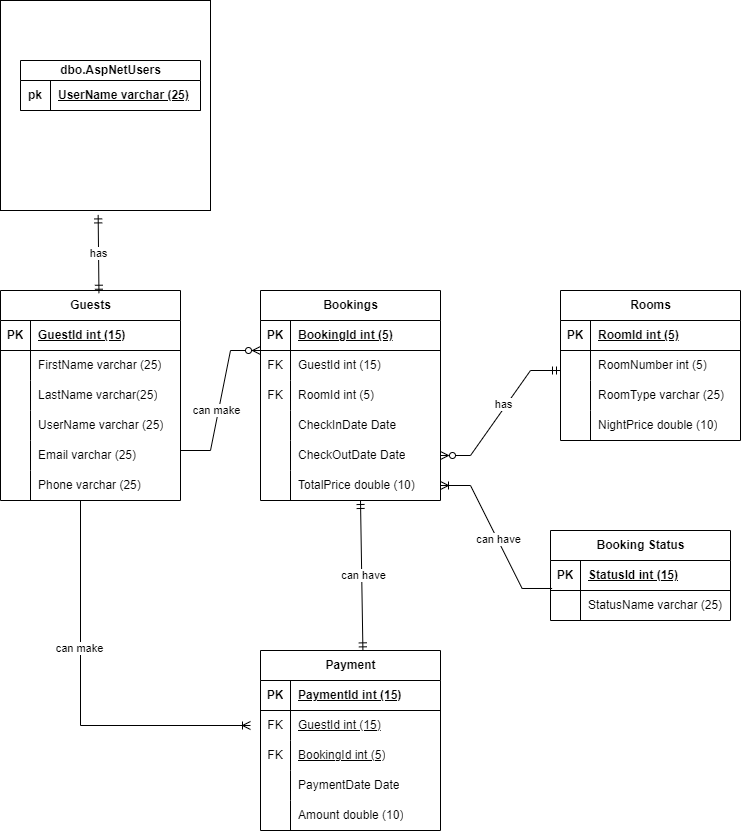
# **UML Diagrams:**

## User Stories

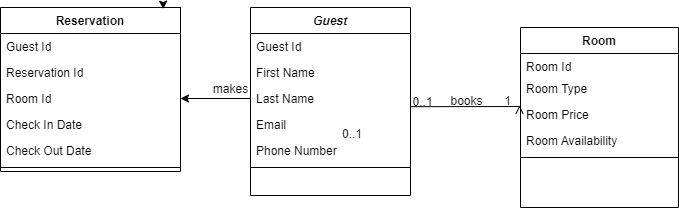
## Use Case Diagram



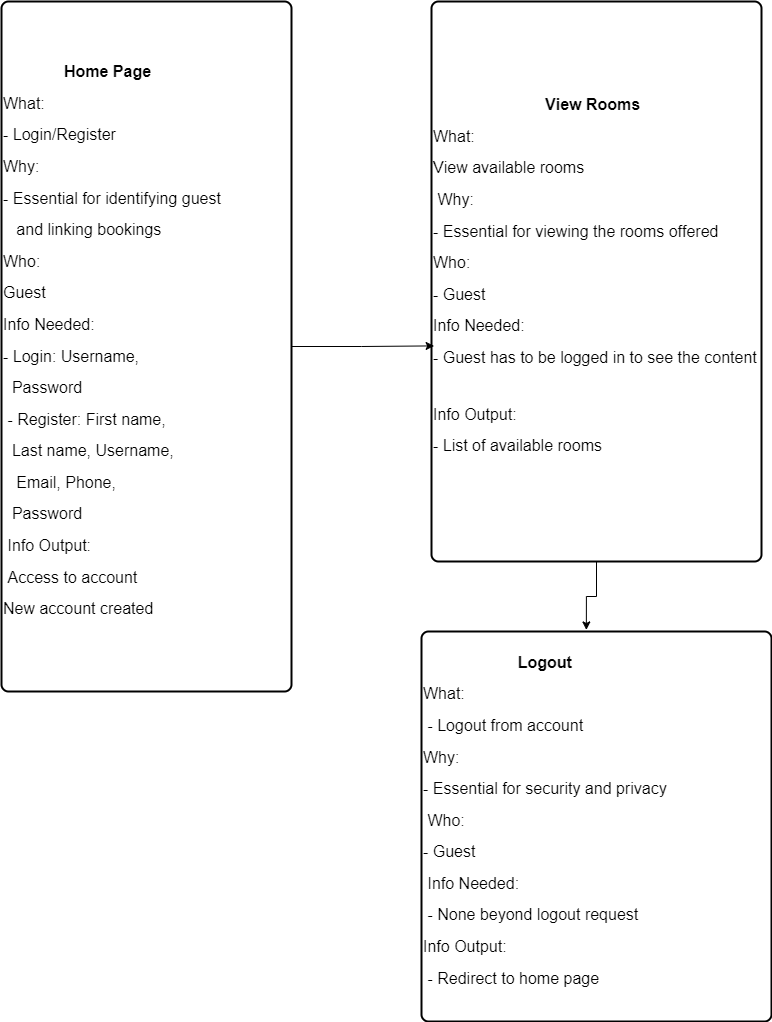
## ERD

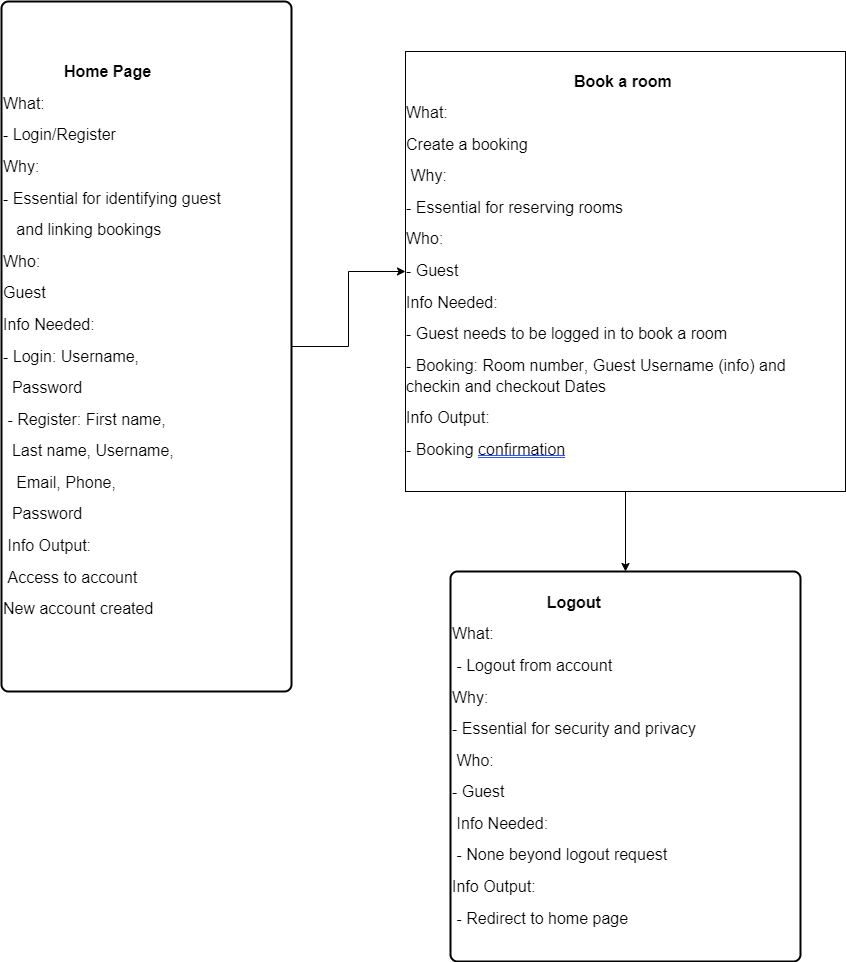


## Class Diagram



## UI Flowchart





# **Technical Architecture**

FDM Git Lab Project Name:

LP\_HotelRoomBooking\_WK5.ANGULAR

**Sample data for User Registration**

**Hotel Room Booking System users**

**Registration**

{  
 "userName": "EmilyBrown",  
 "email": "Emily@gmail.com",  
 "password": "EmilyB\*\*@2024",  
 "firstName": "Emily",  
 "lastName": "Brown",  
 "role": "Customer"  
}

{  
 "userName": "PietWhite",  
 "email": "Piet@gmail.com",  
 "password": "PietW\*\*@2024",  
 "firstName": "Piet",  
 "lastName": "White",  
 "role": "Administrator"  
}

{  
 "userName": "MercyLong",  
 "email": "Mercy@gmail.com",  
 "password": "MercyL\*\*@2024",  
 "firstName": "Mercy",  
 "lastName": "Long",  
 "role": "Receptionist"  
}

**ADMINISTRATOR**

{

"userName": "PeterCoxAdmin",

"email": "Admin1@gmail.com",

"password": "Peter\*@2024",

"firstName": "Peter",

"lastName": "Cox",

"role": "Administrator"

}

{

"userName": "PeterCoxAdmin",

"password": "Peter\*@2024"

}

**RECEPTIONIST**

{

"userName": "MariaBellRecept",

"email": "Receptionist@gmail.com",

"password": "MariaBel\*@2024",

"firstName": "Maria",

"lastName": "Bell",

"role": "Receptionist"

}

{

"userName": "MariaBellRecept",

"password": "MariaBel\*@2024"

}

**CUSTOMER 1**

{

"userName": "JohnSmith",

"email": "John@gmail.com",

"password": "John\*@2024",

"firstName": "John",

"lastName": "Smith",

"role": "Customer"

}

{

"userName": "JohnSmith",

"password": "John\*@2024"

}

## **Sample data for User Login**

|  |  |  |  |
| --- | --- | --- | --- |
| **Security - User Login** |  |  |  |
|  | Username | Password | Role |
| General user/Customer 1 | JohnSmith | John\*@2024 | Customer |
| Receptionist | MariaBellRecept | MariaBel\*@2024 | Receptionist |
| Administrator | PeterCoxAdmin | Peter\*@2024 | Administrator |

## **System logging (log file) and design**

## Week4.ASPNET logging and design

|  |  |
| --- | --- |
| **logging framework** |  |
| Name of Framework | log4net |
| Name (and path) of log file | HotelRoomBookingASPNETRolling.log |

|  |  |
| --- | --- |
| **Design Patterns** |  |
| Name of Design Patterns | **Template method** |
| Where/how it's used | * Update and Delete Details have a common pattern of checking user credentials (\_userLoginToken and \_userLoginRole)   And singleton in log4net. |

## Week3.WebApp logging and design

|  |  |
| --- | --- |
| **logging framework** |  |
| Name of Framework | log4net |
| Name (and path) of log file | HotelRoomBookingRestapiRolling.log |

|  |  |
| --- | --- |
| **Design Patterns** |  |
| Name of Design Patterns | Singleton pattern |
| Where/how it's used | It is used in the IdentityHelper class.   * The constructor is private, which prevents external classes from directly creating instances of IdentityHelper. * The class contains a private static member \_instance of type IdentityHelper, which is initialized to null. * The GetInstance method is used to ensure that only one instance of IdentityHelper is created and returned. * Within the GetInstance method, if \_instance is null, a new instance of IdentityHelper is created and assigned to \_instance. And also subsequent calls to GetInstance return the existing \_instance, ensuring that only one instance of IdentityHelper exists throughout the application. |

# **List of REST API endpoints with description**

|  |  |
| --- | --- |
| **Endpoint: Hotel Room Booking System** | **Description** |
| https://localhost:7168/api/Guests | Get all guests information, GET |
| https://localhost:7168/api/Guests | Create a guest, POST |
| https://localhost:7168/api/Guests/18 | Update guest by guest Id, PUT |
| https://localhost:7168/api/Guests/1 | Get guest info by guest Id |
| https://localhost:7168/api/Guests/18 | Delete guest by guest Id |
| https://localhost:7168/api/Guests/UserName/JohnSmith | Get guest information by username |
| https://localhost:7168/api/GuestsBookings/MyBookings?username=JohnSmith | Get bookings by guest username |
| https://localhost:7168/api/GuestsBookings | Get all bookings information |
| https://localhost:7168/api/GuestsBookings/gstbookings/1 | Get booking information by booking Id |
| https://localhost:7168/api/GuestsBookings/4 | Update booking by booking Id |
| https://localhost:7168/api/GuestsBookings/6 | Delete booking by booking Id |
| https://localhost:7168/api/Rooms | Read all rooms information |
| https://localhost:7168/api/Rooms | Create new room info |
| https://localhost:7168/api/Rooms/1 | Update rooms by room Id |
| https://localhost:7168/api/Rooms/2 | Delete room by room Id |
| https://localhost:7168/api/Rooms/3 | Get room info by room Id |
| https://localhost:7168/api/UserProfile | Get User Profile |
| https://localhost:7168/api/ApplicationUser/Register | User registration |
| https://localhost:7168/api/ApplicationUser/Login | User login |

**(e) UX – User Experience refers to the overall experience that a person has when interacting with a product, service, or system.**

**Key UX considerations for each in my application:**

* **Guest:**
* **User-Friendly Registration:** Ensure the guest registration process is user-friendly. Use clear and concise forms with validation to prevent errors.
* **Profile Management:** Allow guests to easily manage their profiles, update personal information, and view booking history.
* **Booking:**
  + **Intuitive Booking Process:** Design a step-by-step booking process that guides guests through selecting dates, room types.
  + **Real-Time Availability:** Show real-time availability of rooms to avoid overbooking and provide accurate information to guests.
* **Booking Status:**
  + **Status Updates:** Keep guests informed about their booking status. Provide a dashboard or section where guests can check the status of their bookings.
* **Payment:**
  + **Transparent Pricing:** Display transparent pricing, including taxes, fees, and any additional charges.
* **Room:**
  + **Detailed Room Information:** Provide detailed information about each room, including descriptions.
  + **Room Selection:** Allow guests to easily select rooms based on their preferences, such as room type and accessibility options.

  (f) Interaction Design Laws

* **Hick's Law:** Hick's Law states that the time it takes for a person to make a decision increases with the number of options they have. In the context the hotel room booking system, applying Hick's Law can help streamline the booking process and reduce decision fatigue for guests.
  + **Why Hick's Law:** By limiting the number of options presented to guests at each stage of the booking process, the system will simplify their decision-making and enhance the overall user experience. For example, when selecting a room type, offering a concise list of options (e.g., standard, deluxe, suite) rather than overwhelming them with numerous choices can lead to faster decision-making and reduced cognitive load.

  (g) Site Map  
   You are to describe and show which Site Map you wish to use for your application and why

* **Home**
  + Overview of the hotel and its services
  + Quick access to room booking options
* **Rooms**
  + View different room types available
  + Details about each room category
  + Option to book a specific room
* **Booking**
  + Check availability based on desired dates
  + Select room type and number of guests
  + Customize booking preferences
  + Proceed to payment
* **Payment**
  + Secure payment gateway for completing bookings (Not sure about this)
  + Confirmation of payment and booking details
* **Booking Status**
  + View booking status
  + Cancel or modify existing bookings
* **Guest Profile**
  + Manage guest profile information
  + View booking history
* **FAQs/Help**
  + Contact information for customer support or assistance

**Why this Site Map:**

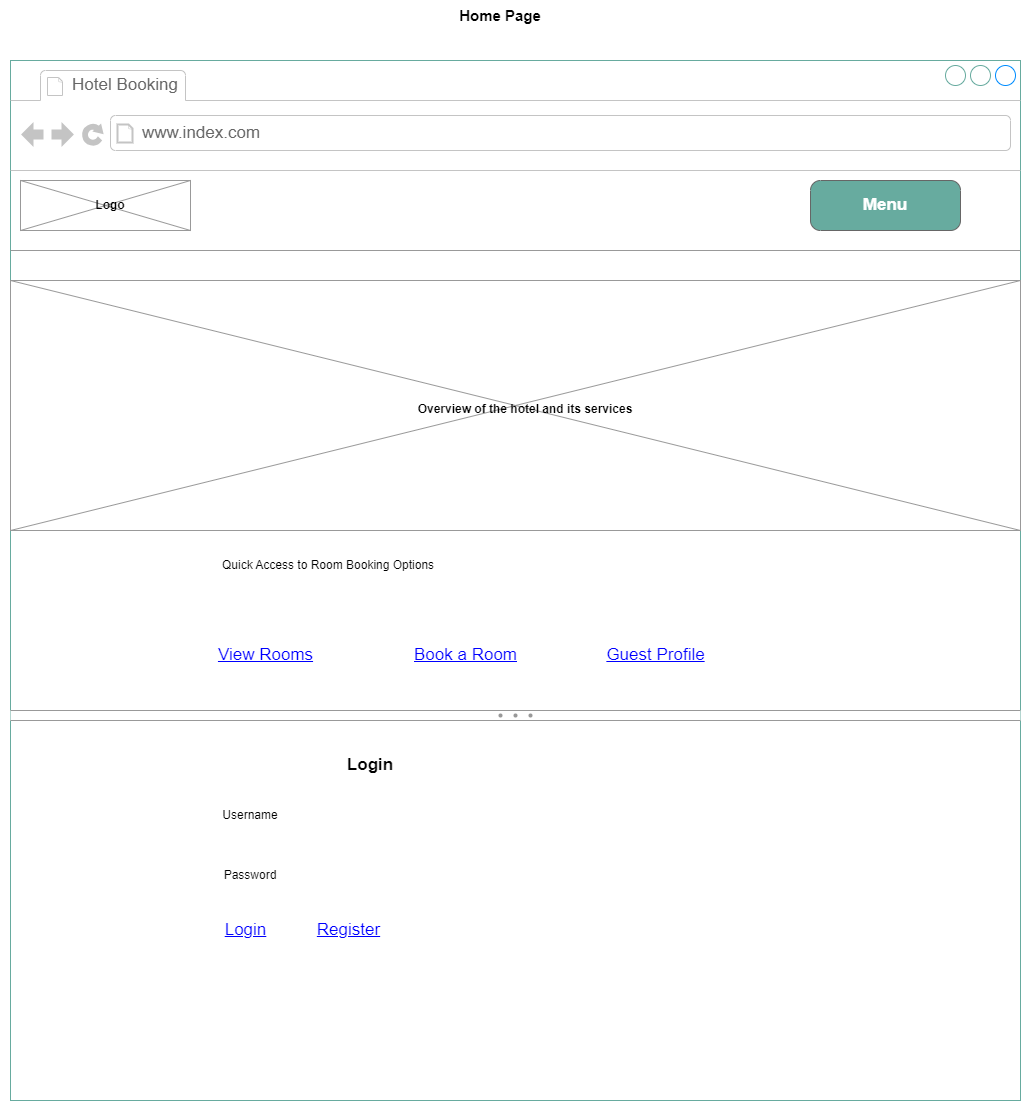
* **User-Centric Structure:** The site map is organized based on the typical user journey, starting from exploring room options to completing a booking, managing bookings, and accessing support if needed. This structure aligns with the primary goals of guests using the system.
* **Clear Hierarchy:** Each section in the site map has a clear hierarchy and flow, making it easy for guests to understand where they are in the process and navigate to the desired section without confusion.
* **Focus on Essential Entities:** The site map focuses on key entities such as guests, rooms, bookings, payments, and booking status, ensuring that guests can access and manage the core functionalities of the booking system efficiently.
* **User-Friendly Navigation:** The navigation paths are designed to be intuitive, with logical groupings of related features and easy access to important actions such as booking, payment, and managing bookings.
* **Accessibility and Support:** Including a dedicated section for FAQs and help ensures that guests can access support resources if they encounter any issues during the booking process.

  (h) Patterns of Visualisation: I'm considering two patterns which are:

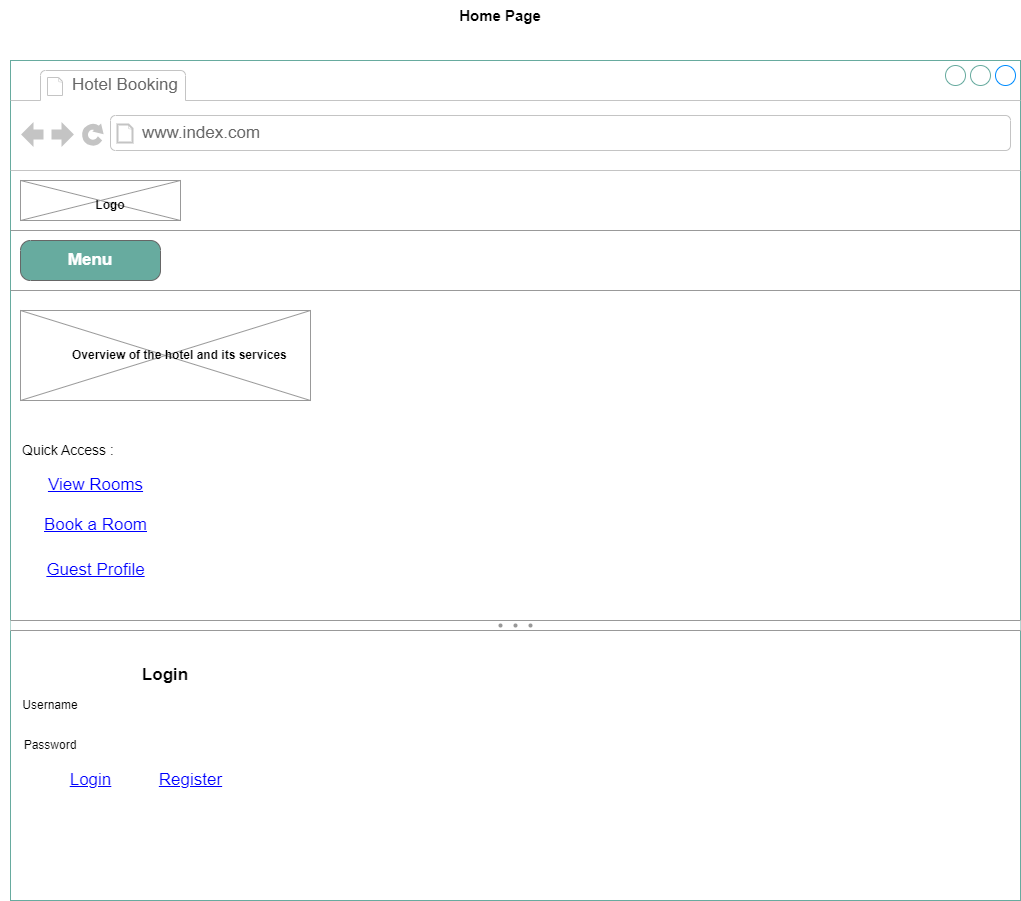
* **The F pattern:**
* Search results page: Guests may scan horizontally through room listings and then vertically for detailed information.
* Booking details: After selecting a room, guests may scan horizontally through booking details and then vertically for payment options and confirmation.
* **The Card-Based design pattern:**
  + **Entities:** Guest, Room, Booking, Payment, Booking Status
  + **Why:** Since the above design is effective for displaying discrete units of information in a visually appealing and organized manner. Each entity (guest, room, booking, payment, booking status) can be represented as a card with key details visible. For example:
    - Guest card: Display guest's name, contact information
    - Room card: Show room type, image and price.
    - Booking card: Include booking details like check-in/check-out dates, room type, and total price.
    - Payment card: Provide payment details such as amount.
    - Booking status card: Show current booking status.

(i) Wireframes

  **PC wireframe**



Mobile wireframe



**Microservices and HTTP protocol from requirement specification document**

* Microservices architecture is a design approach where a complex application is broken down into smaller, independent services that are loosely coupled and can be developed, deployed, and scaled independently. Each microservice focuses on a specific business function and communicates with other services through well-defined APIs.
* The HTTP protocol (Hypertext Transfer Protocol) is the foundation of communication on the World Wide Web. It defines how clients (such as web browsers) and servers communicate by exchanging requests and responses.

**GET:**

* Use Case: The GET method is used to request data from a specified resource. It retrieves data without modifying the server's state or causing side effects.
* Example: When a user visits a web page, the browser sends a GET request to the server to fetch the HTML, CSS, JavaScript, images, and other resources needed to render the page.

**POST:**

* Use Case: The POST method is used to submit data to be processed to a specified resource. It can create new resources or update existing ones on the server.
* Example: When a user submits a form (e.g., a login form or a registration form) on a website, the data entered in the form is sent to the server using a POST request for processing.

**PUT:**

* Use Case: The PUT method is used to update or replace an existing resource on the server with the provided data. It is idempotent, meaning multiple identical requests have the same effect as a single request.
* Example: An API client might use a PUT request to update the details of a user profile by sending the updated user data to the server.

**DELETE:**

* Use Case: The DELETE method is used to request the removal of a specified resource from the server. It deletes the resource if it exists.
* Example: A user might send a DELETE request to delete a specific item from their shopping cart on an e-commerce website.

# **Assumptions and Dependencies**

**Assumptions:**

* Availability: The system assumes that there are available rooms for booking, and it can display the available dates for reservation based on room availability.
* User Interface: Assumed that the system provides a user-friendly interface for customers to search for rooms, view details, make reservations, and manage bookings.
* Room Types and Features: Assumed that the system categorizes rooms into different types (e.g., standard, deluxe, suite) and provides information about room features, amenities, and pricing.
* Email Notifications: Assumed that the system sends confirmation emails, booking details, and reminders to customers.
* Data Security: Assumed that the system follows industry standards and best practices for data security, including encryption of sensitive information.

**Dependencies:**

* Database: The system depends on a database to store and manage information such as customer details, room availability, bookings, payment transactions, etc.
* Internet Connectivity: The system relies on stable internet connectivity for customers to access the booking platform and complete transactions.
* Compliance: Dependency on regulatory compliance and legal requirements related to data protection, online payments, customer privacy, etc.
* User Support: Dependency on customer support services to handle inquiries, issues, and assistance for customers using the booking system.

# **Lessons Learned**

## Lesson Learned – Week 5.ANGULAR

I learned how to create the front end using angular, learned about services, components and how they connect with my rest API.

## Lesson Learned – Week 4.ASP.NET

I learned about Model-View-Controller design pattern that divides an application into three main components: Model, View, and Controller. I got to understand how the ASPNET interacts with my rest API.

## Lesson Learned – Week 3.WEBAPP

Learned Html and java script for the fist time. I’ll learn how to read code better so that my trainer and I can understand each other.

## Lesson Learned – Week 2 .NET REST API

I learned how to create and utilise ASP.NET Identity Management, API for user Registration and Login. I learned how to generate and verify JWT Token Add CRUD end points, Add User Authentication, Authorization , and Accounting/Auditing. TDD API Controller using In Memory database.

## Lesson Learned - Week 1 .NET Data Access

I understand the difference between .NET Standard, .NET Core and .NET Framework, I learned how to use SSMS. I can use ADO.NET Connected, Disconnected and Disconnected Procedure models to access data, I understand database connectivity

## Lesson Learned - Week 4 OOD4

I learned the gang of four design patterns, implementing logging in my project, and communicating better when working in a group. I also have a better understanding of the importance of implementing appropriate logic to control the flow of software operations. What I would do differently in future is start my project work early to avoid last minutes changes and sleepless nights.

## Lesson Learned - Week 3 OOD3

I learned how to read and write neat code, use data types and methods considering the 4 pillars and Solid principles. Write code documentation comments and generate XML output documentation file. TDD project configuration. I learned how to use NUnit to write the implementation code using TDD approach, write passing NUnit tests using TDD approach, use exception handling and MOQ.

## Lesson Learned - Week 2 OOD2

I learned how to read and write neat code, use data types and methods considering the 4 pillars and Solid principles. I also learned how to use source control effectively.

## Lesson Learned – Week 1 OOD1

I learned C-Sharp language syntax, modifiers, data types and operators. I learned how to use constructors and debug my code , I also learned about flow chart.