Desing Patterns - Hotel Management System

Tahir Emre Semiz - 20200808058

Project Goals

At the beginning of this project, the following 3 objectives were set respectively:

- Requirements and solution phases in the hotel management system

- The most efficient and appropriate design patterns required for these solution phases

- These selected design patterns should be coded efficiently and have a flexible structure

When thinking about the first item, since this project is not a very comprehensive and detailed project, it was aimed to build the main structure first. These were the room structure and the customer structure. Then, it was aimed to design the functions that can be done about the rooms and customers and to provide functionality.

In order for the functions to be efficient and flexible to meet the requirements, the right patterns had to be selected. The following design patterns were targeted during the project for functions such as sending notifications to customers, room management, payment transactions, customer management:

-Singleton

-Factory

-Observer

-Strategy

-Decorator

While working with these design patterns, certain ideas were designed to have a flexible structure. For example, when designing a room, the hotel management can increase the items to be added to the room as much as they want. when sending notifications to the customer, the methods of sending notifications can be increased. The variety of payment methods can be increased. with updates to such existing features, it is aimed that the hotel functionality can always be expanded.

Project Purposes

Creating a Hotel Management System:

The main objective of the project is to develop a hotel management system. This system should be able to successfully manage hotel operations, especially room and customer related hotel operations, increase customer satisfaction and make hotel operations more effective and flexible.

Hotels often have to manage a number of complex transactions, so the system should cover basic functions such as bookings, room statuses, customer information and payment processing.

By providing a user-friendly interface, it is intended that hotel staff and customers can easily use the system.

Using Design Patterns:

The project should include design patterns in order to reduce complexity in the system by following best practices in software design and to create a sustainable and flexible code structure. The purposes of some design patterns will be explained in more detail later in the paper. Some of them are for example as follows:

- The Singleton pattern should be implemented with a single instance of HotelManager, ensuring that there is only one hotel manager in the system.

- The Factory pattern should enable the creation of different room types (e.g. Standard and Suite rooms).

- The Observer pattern should send notifications to customers about booking status and other important events.

Flexible Design:

The system should have a modular structure and updates should be easy to have. it is aimed to provide a smooth process when adding new features and updating existing features.

Items such as room types, payment strategies and additional features should be able to be added modularly.

The system should be tailored to suit the needs of different hotels.

Project Scope

The scope of the project was mentioned in the aims and objectives section above and its scope was mentioned in detail. in the rest of the article, it will be mentioned how these scopes are implemented. the scopes are as follows if it is necessary to specify the main headings and one sentence:

Room Types and Modifications:

- Management and reservations of various room types such as standard and suite rooms.

Customer Transactions:

- Recording and updating customer information, tracking customer bookings and related information.

Payment Processing and Strategies:

- A system that supports different payment methods and makes payment strategies selectable.

Notifications:

- Sending notifications to customers about bookings and other important changes in the system

Design Patterns

Each of the design patterns in this project serves a different purpose and role. These patterns are to improve the code quality of the system environment. Here is a list of these design patterns and the purposes and roles of each:

1. Singleton Pattern:

Explanation:

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance.

Role in the Project:

The **HotelManager** class is implemented as a Singleton to ensure that there is only one instance managing the hotel.

It ensures that there is a single point of control for hotel-related operations.

This pattern prevents the unnecessary creation of multiple hotel management instances, maintaining consistency in data and operations.

2. Factory Pattern:

Explanation:

The Factory pattern defines an interface for creating an object but lets subclasses alter the type of objects that will be created.

Role in the Project:

The **RoomFactory** interface declares the method **createRoom()** for creating different types of rooms.

Concrete implementations like **StandardRoomFactory** and **SuiteRoomFactory** provide specific implementations for creating standard and suite rooms, respectively.

The Factory pattern allows the system to create rooms dynamically based on the type of room requested, enhancing flexibility.

New room types can be easily added to the system by adding new room functions to the code.

3. Observer Pattern:

Explanation:

The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Role in the Project:

The **BookingObserver** interface declares the update() method to notify observers of changes.

The **Customer** class implements the observer and is notified when significant events, such as booking updates, occur.

The **BookingSystem** class acts as the subject, managing a list of observers (customers) and notifying them of relevant changes. This enhances communication between the system and customers.

4. Strategy Pattern:

Explanation:

The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Role in the Project:

The **PaymentStrategy** interface declares the **pay()** method for payment processing.

Concrete implementations like **CreditCardPayment** and **CashPayment** provide specific payment strategies.

The **PaymentStrategy** is interchangeable, allowing the system to dynamically switch between different payment strategies during runtime.

5. Decorator Pattern:

Explanation:

The Decorator pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

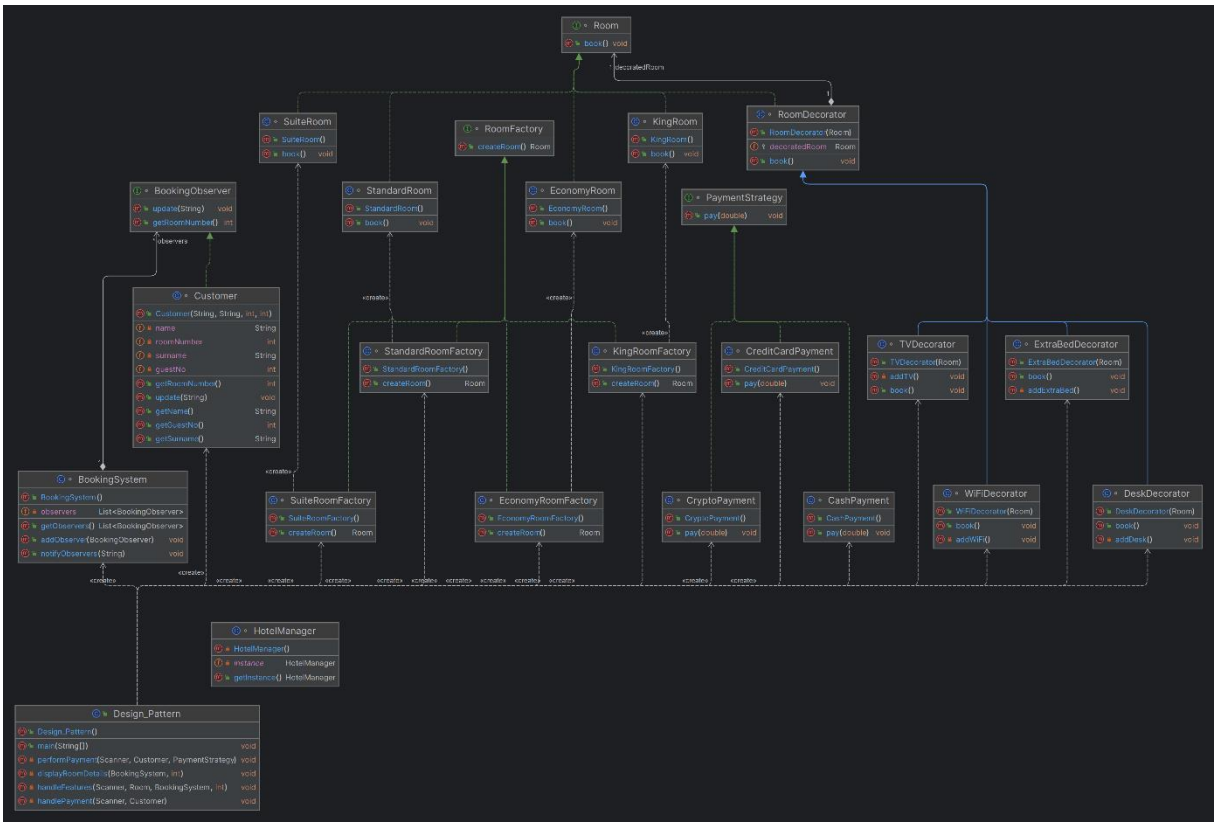
Role in the Project:

The **RoomDecorator** abstract class implements the Room interface and acts as the base decorator.

Concrete decorators like **WiFiDecorator** add specific features (e.g., WiFi) to the base Room.

The Decorator pattern allows the dynamic addition of features to rooms during runtime without modifying their core behavior.

UML Diagram



Relationships Between Design Patterns

1. Singleton and Factory Patterns:

Relationship:

The Singleton pattern ensures that there is only one instance of the **HotelManager** class responsible for managing the hotel.

The Factory pattern is used to create instances of different room types (**StandardRoom** and **SuiteRoom**).

Interaction:

The Singleton instance (**HotelManager**) is responsible for coordinating and managing the creation of different types of rooms through the Factory pattern.

The Singleton ensures a centralized point for room creation and management, promoting consistency.

2. Factory and Observer Patterns:

Relationship:

The Factory pattern is used to create instances of different room types (e.g. **StandardRoom** and **SuiteRoom**).

The Observer pattern is employed for notifying customers (**BookingObserver**) about changes, such as room bookings.

Interaction:

When a new room is created using the Factory pattern, the Observer pattern facilitates notifying customers interested in booking updates.

The Factory provides dynamic object creation, and the Observer ensures that relevant entities are notified of changes.

3. Decorator and Singleton Patterns:

Relationship:

The Decorator pattern is used to dynamically add features to rooms (**RoomDecorator**).

The Singleton pattern ensures a single instance of the **HotelManager** class.

Interaction:

The Singleton pattern provides a single point of control and coordination through the **HotelManager**.

The Decorator pattern enhances the features of rooms dynamically, enriching the hotel experience for customers.

4. Observer and Strategy Patterns:

Relationship:

The Observer pattern is used for notifying customers (**BookingObserver**) about changes, such as booking updates.

The Strategy pattern is applied to handle different payment strategies (**PaymentStrategy**).

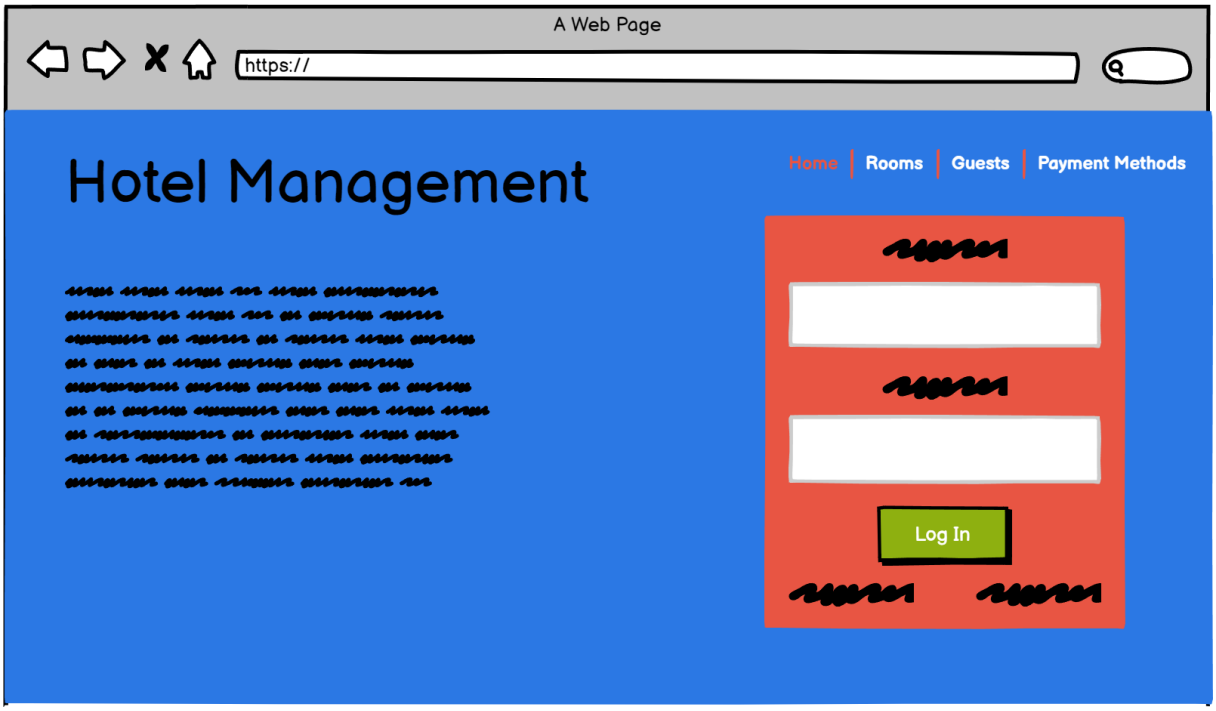
Interaction:

Customers, acting as observers, may receive notifications about booking updates and payment events, which are handled through the Strategy pattern.

The Observer and Strategy patterns work together to provide a dynamic and responsive system, where changes in booking or payment strategies trigger notifications.

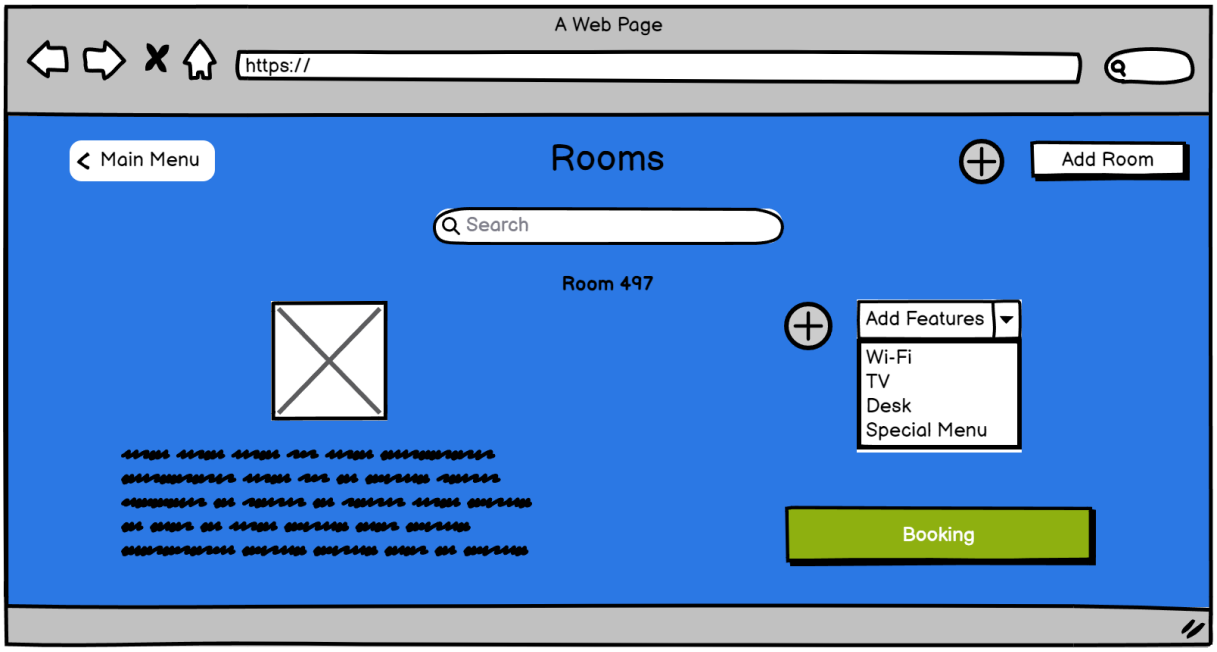
User Interface

Main Menu



In the main menu there is a navigation bar for switching to the room, customers and payment method tabs. The login page takes up most of the space. by using this login option, **Singleton Pattern** is activated and the system is logged in.

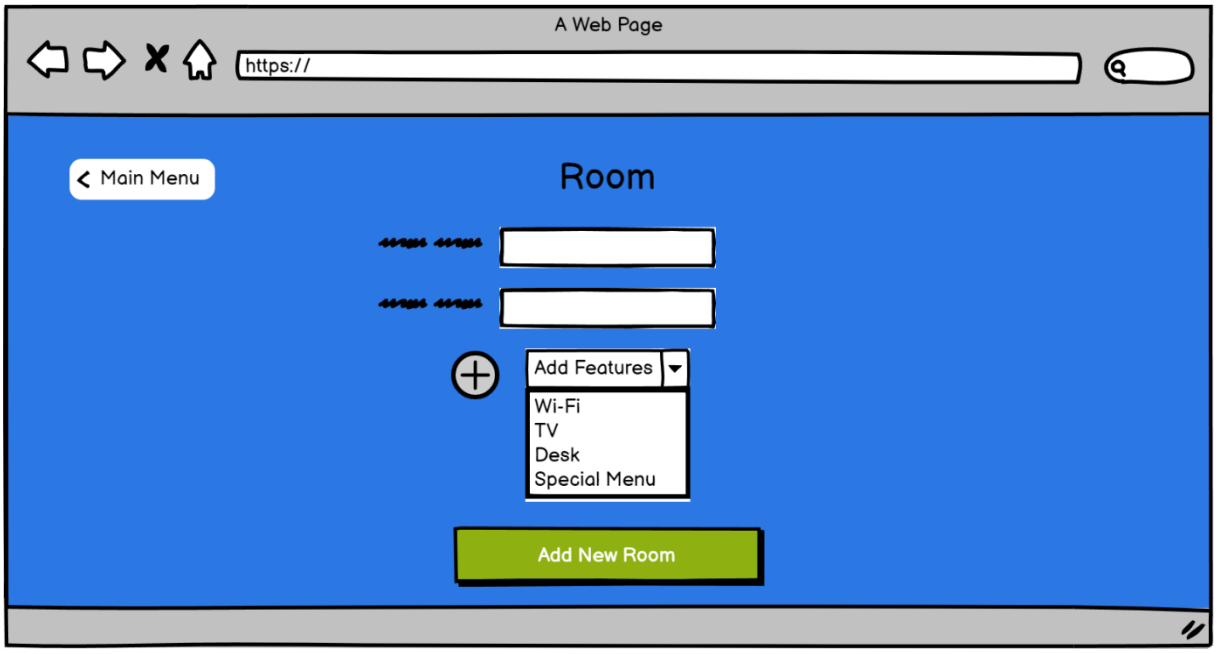
Room Page



This page contains the information of the searched room. For example, the features in the room, the customer staying in the room. also new features can be added to the rooms. A new room can be added to the system from the button on the top right.

New features are added to a already existing room with the **Decorator Pattern**.

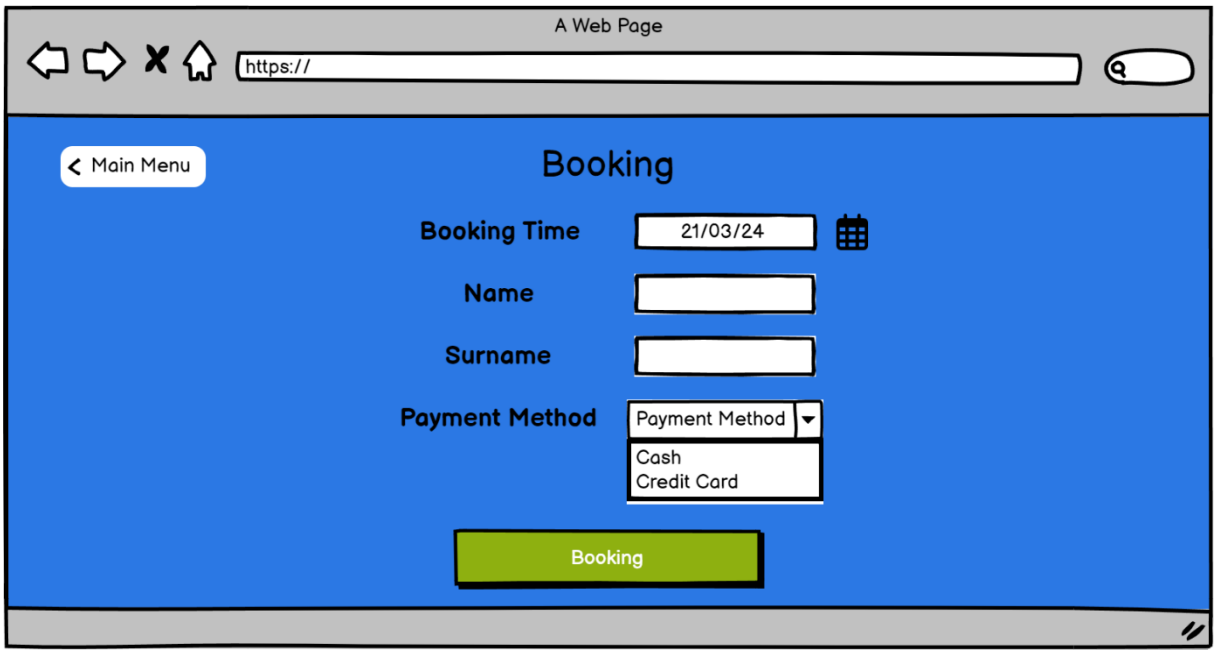
Adding New Room Page



Firstly, information such as room name, floor where the room is located is entered into the system. Then the features that should be included are added and all room information is added to the system by clicking the button.

Factory and Decorator Pattern work together on this screen. With the **Factory Pattern**, new room types and new room setup are provided. With the **Decorator Pattern**, new features are added to a new room.

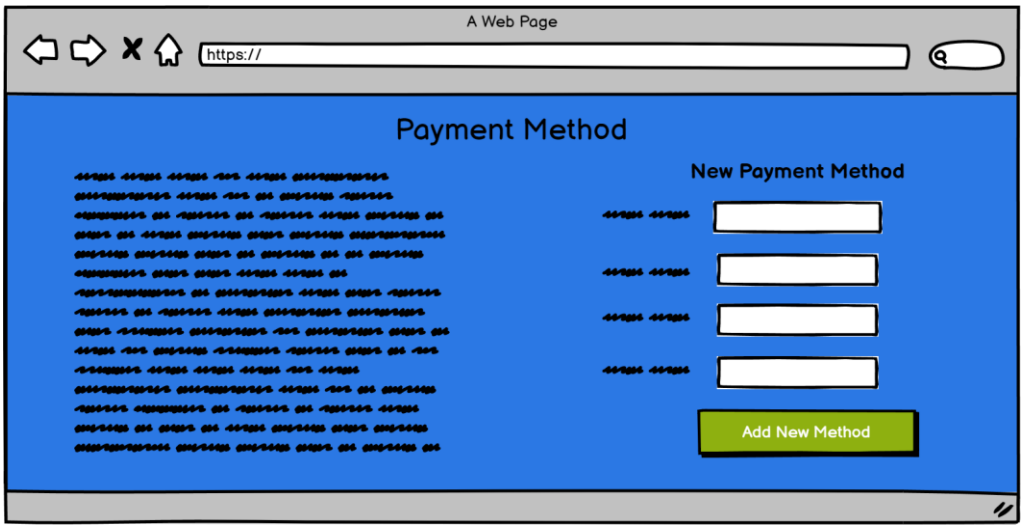
Booking Page



This screen is accessed by clicking the green button on the room page. After the dates of the selected room, guest information and payment method are determined, the reservation of the room is completed.

By uploading the guest information to the system, it provides both notifications and transferring the information of that guest to the room. On this screen, **Observer Pattern** is used in the guest section and **Strategy Pattern** is used in the payment method selection section.

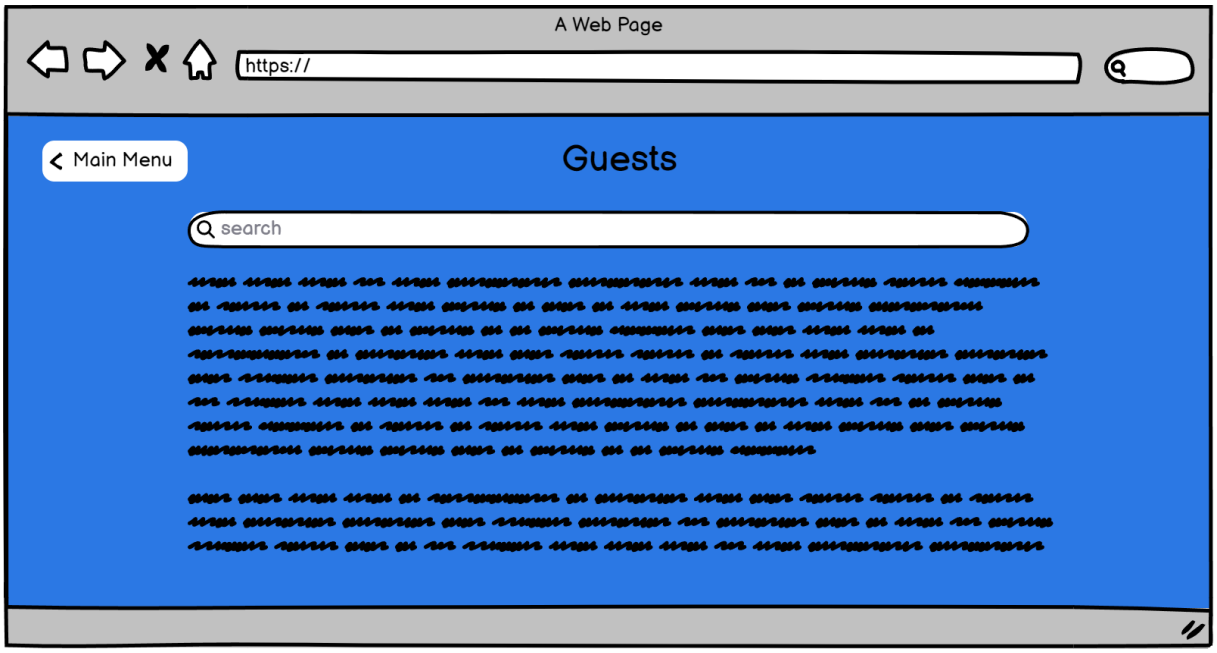
Payment Method Page



This screen is opened by selecting from the navigation bar in the main menu. On this screen, the payment methods supported by the hotel are written on the left side. On the right side, the details of the payment method to be added to the hotel system are added.

In this screen, new functions are added to the **Strategy Pattern**.

Guests Page



This page is also provided with the navigation bar on the main screen. Customer information is listed with the search bar here.

With the search made on this screen, the information stored with the **Observer Pattern** is accessed.