



UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Travel Management System

Submitted by:

T P Shriambhikesh PES1UG21CS659

Tanmay Praveen Udupa PES1UG21CS662

Srimitravinda Ponnada PES1UG21CS621

Suhas P Shroff PES1UG21CS634

6th Semester K Section

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

1. PROBLEM STATEMENT

In the rapidly evolving landscape of travel and tourism, there is a growing need for a comprehensive Travel Management System (TMS) that integrates various booking functionalities and administrative tools to streamline the travel planning process. Our project aims to develop a robust and user-friendly TMS that caters to the diverse needs of travelers while offering seamless administrative capabilities for managing bookings and resources.

- **Booking Functionalities:**

The TMS will offer a range of booking functionalities, including hotel reservations, car rentals, villa accommodations, and flight bookings. Each module will provide users with an intuitive interface to search, compare, and book services tailored to their preferences and requirements.

- **Package Booking:**

In addition to individual bookings, the TMS will facilitate the creation and customization of travel packages. Users will have the option to bundle various services into comprehensive packages, allowing for convenient and cost-effective trip planning.

- **User Management:**

The system will support user authentication and management features, enabling travelers to create accounts, manage bookings, and access personalized recommendations based on their travel history and preferences.

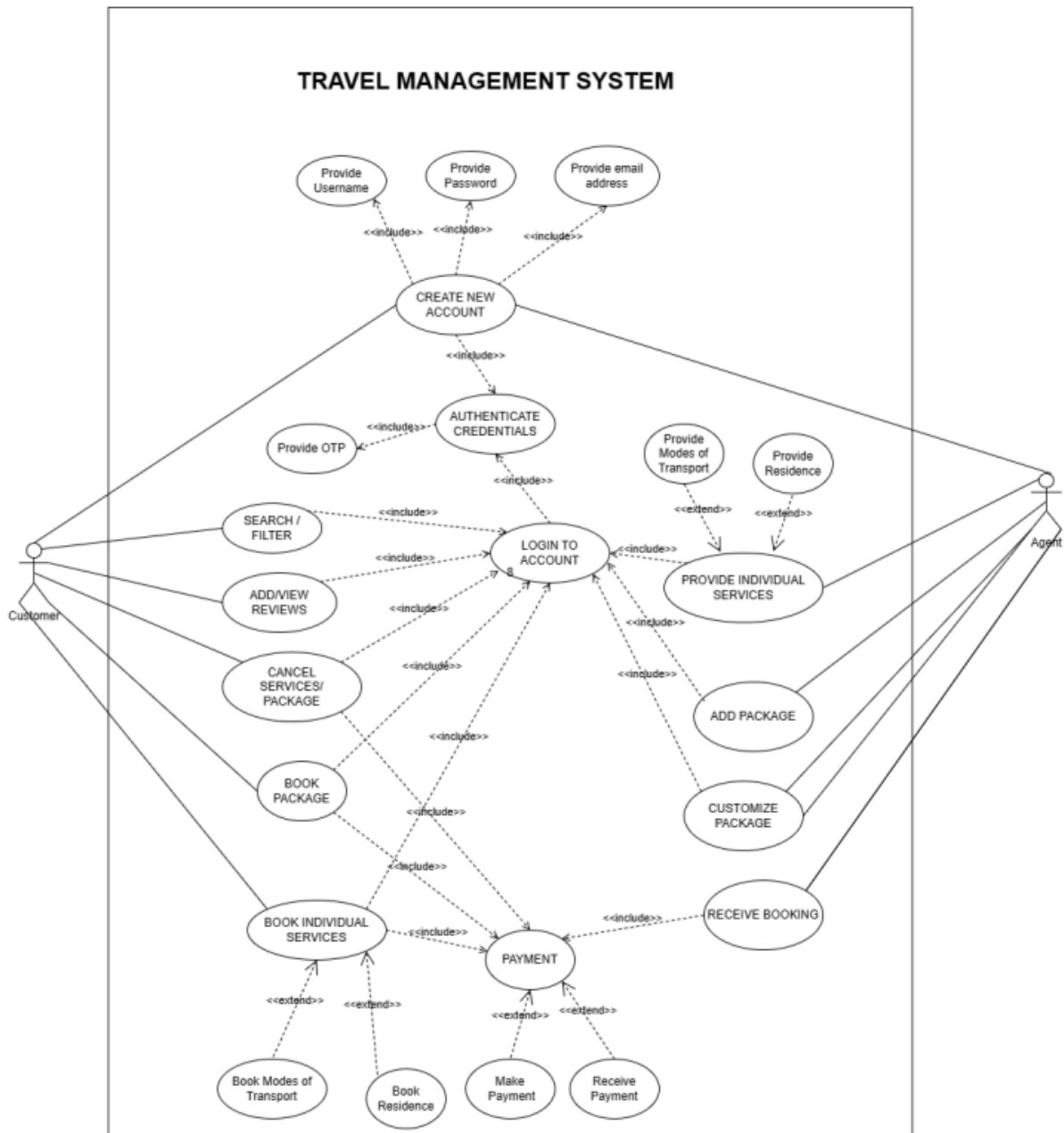
- **Admin Dashboard:**

Administrators will have access to a dedicated dashboard equipped with tools for managing bookings, inventory, and user accounts. The admin interface will provide real-time insights into booking trends, revenue generation, and resource utilization, empowering administrators to make informed decisions and optimize operations.

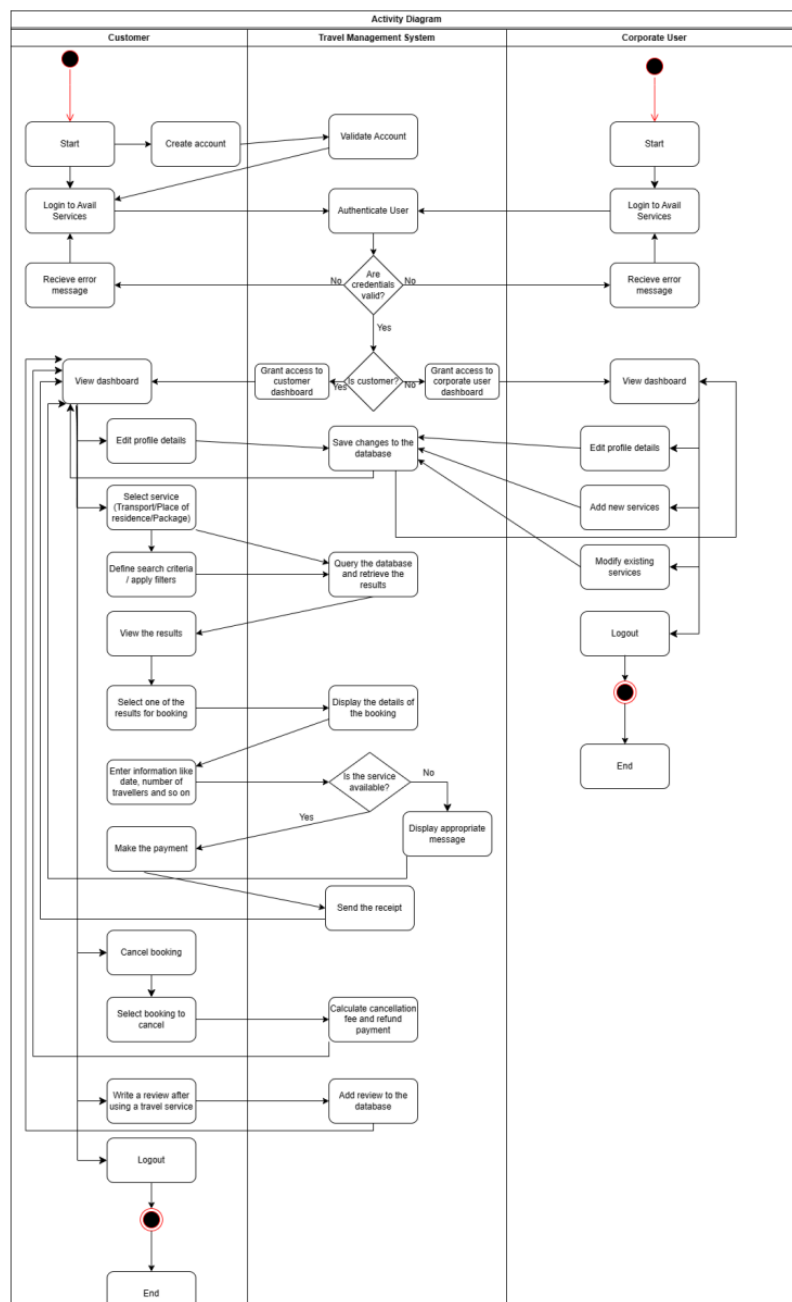
2. UML DIAGRAMS

Link to Diagrams: Travel Management System - Diagrams.pdf

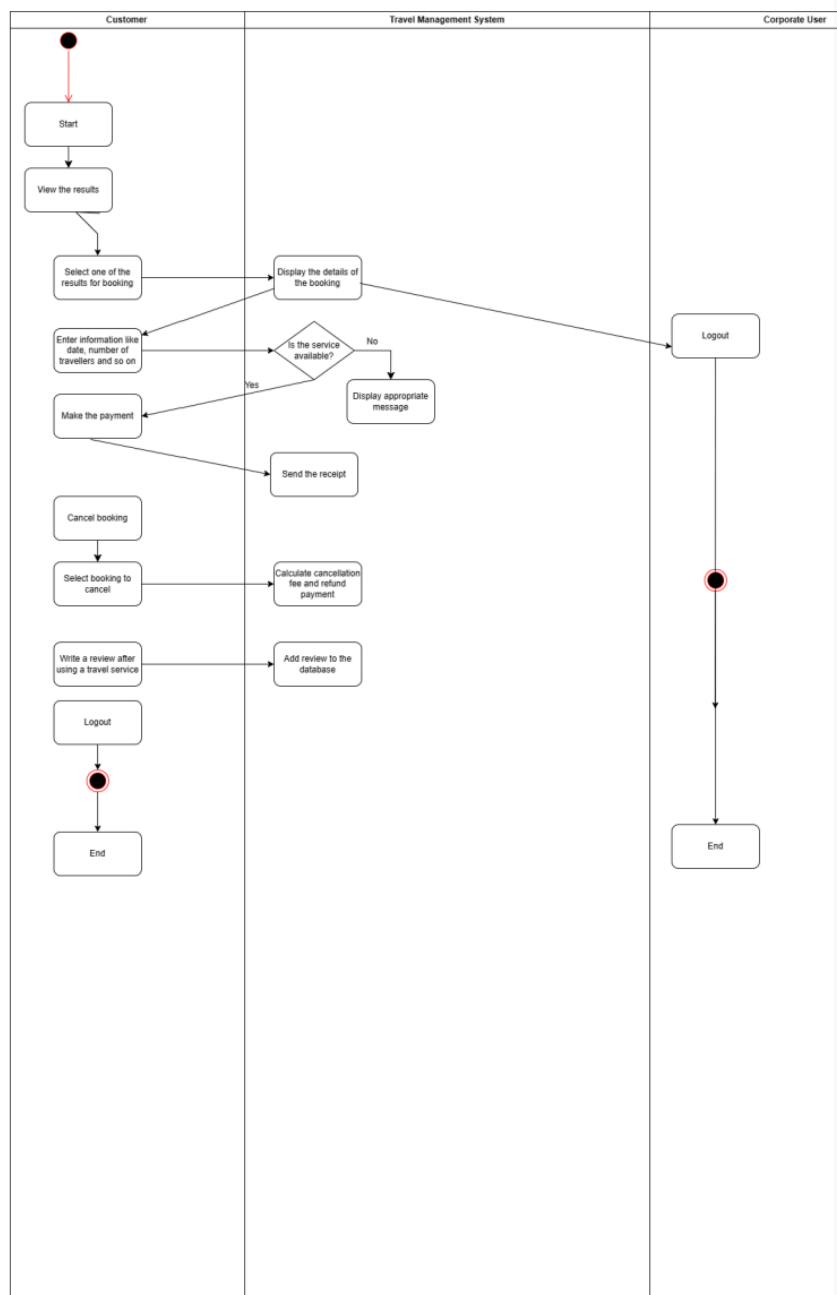
(a) Use Case Diagram



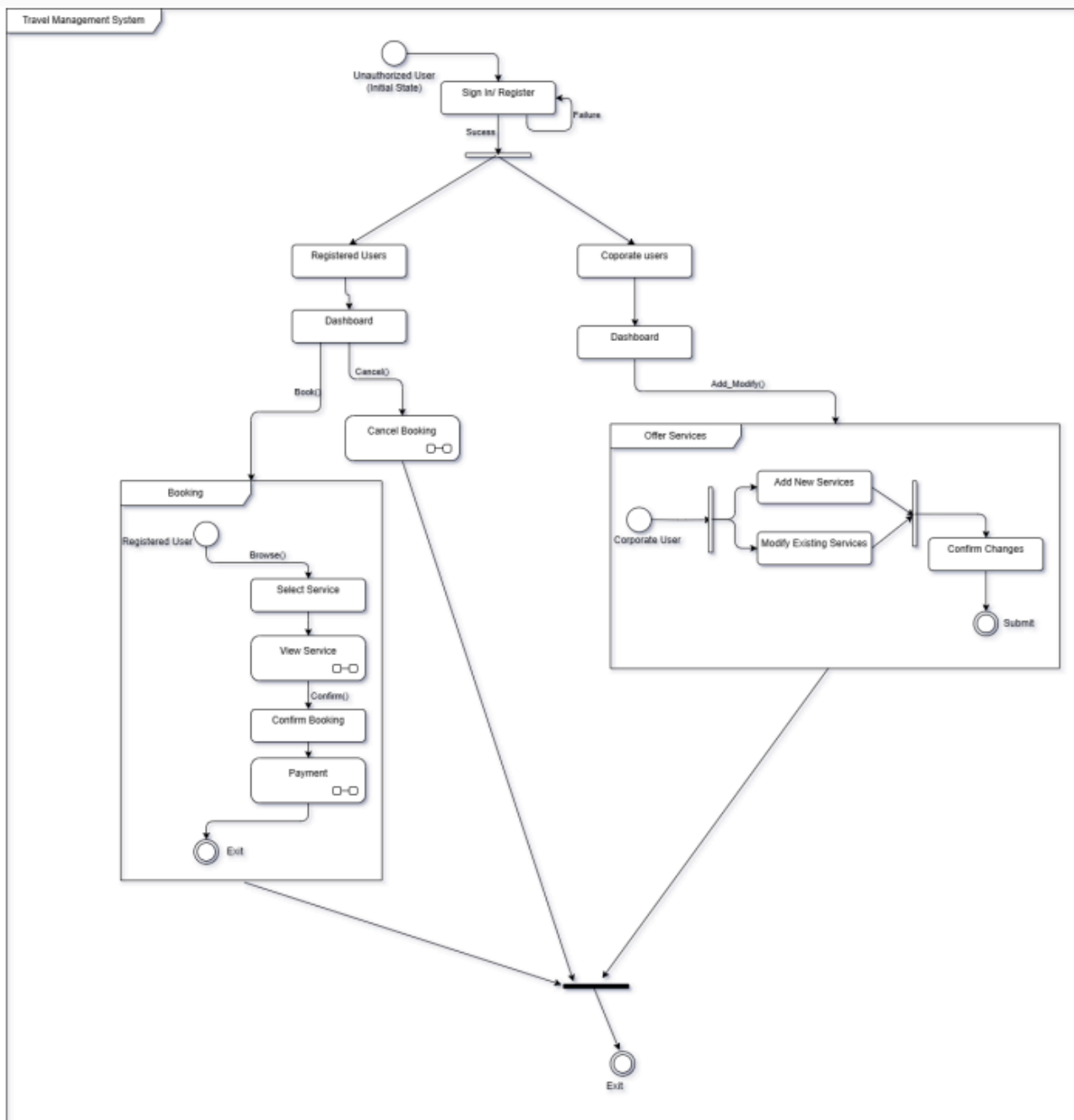
(b) Activity Diagram - 1



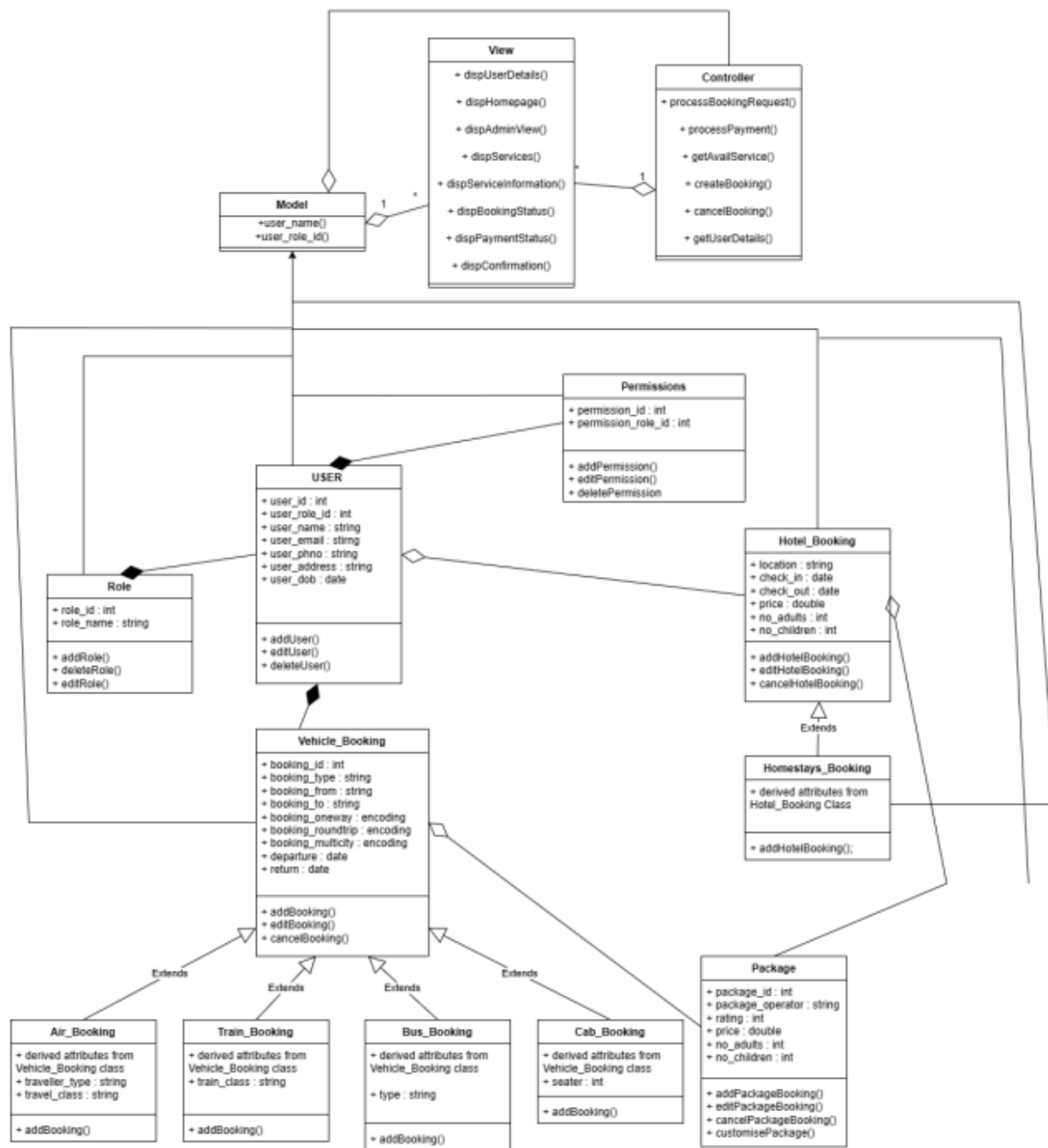
(c) Activity Diagram - 2 (Book Package)



(d) State Diagram



(e) Class Diagram



IMPLEMENTATION

Our project was developed using Vaadin framework

Our project does not violate SOLID principles

MVC architecture pattern is used in our project

- Model:

The Model component represents the data and business logic of the application. In our TMS, the Model includes classes that interact with the database to retrieve and manipulate travel-related data. For example, we have a FlightBooking model, HotelBooking Model classes responsible for managing bookings of flights, hotels respectively. These classes handle database operations such as querying for available flights, fetching hotel details and booking them.

- View

The View component represents the user interface of the application. In our TMS, the View includes the different pages and components that users interact with to search for and book travel services

- Controller

The Controller component acts as an intermediary between the Model and the View. It receives user inputs from the View, interacts with the Model to process requests, and updates the View with the results. For example, we have a Hotel Controller class that handles requests to book hotels. This class invokes methods on the corresponding Model classes to perform booking operations and renders appropriate View templates to display results or error messages.

3. DESIGN PATTERNS

- BUILDER PATTERN

The Builder pattern is utilized in the PackageBuilder class to simplify the creation of Package objects with optional components such as flights, trains, hotels, and villas. By encapsulating the construction logic, it offers a flexible and readable approach, allowing clients to specify only the necessary components for a Package. This promotes code clarity and maintainability, especially in scenarios where the object's structure may vary or evolve over time.

```
package com.example.application.builders;

import com.example.application.models.Package;
import com.example.application.models.Flight;
import com.example.application.models.Train;
import com.example.application.models.Hotel;
import com.example.application.models.Villa;

import java.util.List;

public class PackageBuilder {
    private Package packageInstance;

    public PackageBuilder() {
        packageInstance = null;
    }

    public PackageBuilder startNewPackage(String packageName) {
        packageInstance = new Package(packageName);
        return this;
    }

    public PackageBuilder setPackageName(String packageName) {
        packageInstance.setPackageName(packageName);
        return this;
    }

    public PackageBuilder setTotalPrice(double totalPrice) {
        packageInstance.setTotalPrice(totalPrice);
        return this;
    }

    public PackageBuilder addFlight(Flight flight) {
        packageInstance.addFlight(flight);
        return this;
    }

    public PackageBuilder addTrain(Train train) {
        packageInstance.addTrain(train);
        return this;
    }

    public PackageBuilder addHotel(Hotel hotel) {
        packageInstance.addHotel(hotel);
        return this;
    }

    public PackageBuilder setVilla(Villa villa) {
```

```

        packageInstance.setVilla(villa);
        return this;
    }

    public PackageBuilder addFlights(List<Flight> flights) {
        for (Flight flight : flights) {
            packageInstance.addFlight(flight);
        }
        return this;
    }

    public PackageBuilder addTrains(List<Train> trains) {
        for (Train train : trains) {
            packageInstance.addTrain(train);
        }
        return this;
    }

    public PackageBuilder addHotels(List<Hotel> hotels) {
        for (Hotel hotel : hotels) {
            packageInstance.addHotel(hotel);
        }
        return this;
    }

    public Package build() {
        Package builtPackage = packageInstance;
        packageInstance = null;
        return builtPackage;
    }
}

```

● SINGLETON PATTERN

The Singleton Pattern is employed to ensure that a single instance of the database connection is maintained across the application, preventing unnecessary resource consumption and ensuring consistency in data access. By encapsulating the creation and management of the database connection within a single class, it enhances code efficiency and promotes centralized control over database interactions.

The code for Singleton pattern can be found under the "database" folder and it's used everywhere, for eg, in controllers, managers etc.

```

public TrainController() {
    try {
        // Attempt to connect to MongoDB
        MongoClient connectedClient = MongoClient.create("mongodb://localhost:27017/");

        // Perform pre-flight checks and handle potential issues
        if (!preFlightChecks(connectedClient)) {
            throw new RuntimeException("Failed to connect to MongoDB during pre-flight checks.");
        }

        System.out.println("=> Connection successful: " +
            preFlightChecks(connectedClient));
    }
}

```

```

trainCollection =
connectedClient.getDatabase("Travel_Management_System").getCollection("trains");
trainBookingCollection =
connectedClient.getDatabase("Travel_Management_System").getCollection("trainBookings");
// ... rest of the initialization logic using connectedClient
} catch (MongoException e) {
    // Handle MongoException in case of connection issues
    throw new RuntimeException("Error connecting to MongoDB: " + e.getMessage());
}

```

- STRATEGY PATTERN

This pattern is used because it easily allows to add new booking types without modifying the existing code. You simply need to create a new concrete BookingStrategy implementation and update the BookingContext to use the new strategy.

To implement this pattern, an interface named BookingStrategy is created that consists of the method book which is a common method for all booking types. Then concrete implementation classes are created for each booking type, such as TrainBookingStrategy, HotelBookingStrategy, etc.

There is a BookingContext class which will take a BookingStrategy object and delegate the booking operations to the appropriate concrete BookingStrategy implementation. In the respective booking views of the Train, Hotel, etc., after the booking details are entered, an instance of the BookingContext is created and is used to set the appropriate BookingStrategy.

The code for this can be found in the strategies folder.

```

package com.example.application.strategies;

import com.mongodb.MongoException;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import org.bson.Document;

import java.util.Map;

public class FlightBookingStrategy implements BookingStrategy {
    private MongoCollection<Document> flightBookingCollection;
    public FlightBookingStrategy() {
        try {
            // Attempt to connect to MongoDB
            MongoClient connectedClient =
MongoClients.create("mongodb://localhost:27017/");

            flightBookingCollection =
connectedClient.getDatabase("Travel_Management_System").getCollection("flightBookings");
            // ... rest of the initialization logic using connectedClient
        } catch (MongoException e) {
            // Handle MongoException in case of connection issues
            throw new RuntimeException("Error connecting to MongoDB: " + e.getMessage());
        }
    }
}

```

```

    }

    @Override
    public void book(Map<String, String> bookingDetails) {
        // Implement booking logic for flight
        Document flightBooking = new Document("username", bookingDetails.get("username"))
            .append("flight_number", bookingDetails.get("flight_number"))
            .append("airline", bookingDetails.get("airline"))
            .append("departure_date", bookingDetails.get("departure_date"))
            .append("arrival_date", bookingDetails.get("arrival_date"))
            .append("seats_booked", bookingDetails.get("seats_booked"))
            .append("total_price", bookingDetails.get("total_price"))
            .append("date_booked", bookingDetails.get("date_booked"))
            .append("paid", bookingDetails.get("paid"));

        flightBookingCollection.insertOne(flightBooking);
    }
}

```

● ADAPTER PATTERN

The Adapter pattern is employed to abstract the conversion of MongoDB documents of various types into Java's List format, providing a uniform interface for data retrieval and manipulation. By implementing adapter classes for each MongoDB document type, the pattern facilitates seamless integration with Java code, enhancing flexibility and maintainability.

It follows OCP too, since new types of services can be added just by extending the "abstractAdapter" Class. without the need to change any of the existing code

```

package com.example.application.Adapters.Booking;

import com.example.application.Adapters.Booking.abstractBookingAdapter;
import com.example.application.models.FlightBooking;
import org.bson.Document;

public class FlightBookingAdapter extends abstractBookingAdapter {
    private String flightNumber;
    private String flightName;
    private String departureDate;
    private String arrivalDate;
    private String seatsBooked;

    public String getFlightNumber() {
        return flightNumber;
    }

    public void setFlightNumber(String flightNumber) {
        this.flightNumber = flightNumber;
    }

    public String getFlightName() {
        return flightName;
    }
}

```

```

public void setFlightName(String flightName) {
    this.flightName = flightName;
}

public String getDepartureDate() {
    return departureDate;
}

public void setDepartureDate(String departureDate) {
    this.departureDate = departureDate;
}

public String getArrivalDate() {
    return arrivalDate;
}

public void setArrivalDate(String arrivalDate) {
    this.arrivalDate = arrivalDate;
}

public String getSeatsBooked() {
    return seatsBooked;
}

public void setSeatsBooked(String seatsBooked) {
    this.seatsBooked = seatsBooked;
}

public FlightBooking toBooking(Document doc) {
    return new FlightBooking(
        doc.getObjectId("_id").toString(),
        doc.getString("username"),
        doc.getString("flight_number"),
        doc.getString("airline"),
        doc.getString("departure_date"),
        doc.getString("arrival_date"),
        doc.getString("seats_booked"),
        doc.getString("total_price"),
        doc.getString("date_booked"),
        doc.getString("paid")
    );
}
}

```

SCREENSHOTS

Signup

← ↻ 🔍 localhost:8080/sign-up

Please enter your details

Name

Address

Email

Username

Password

Confirm password

[Register](#)

Login

My App

→ Login

Login View

Log in

Username •

Password •

[Log in](#)

[Forgot password](#)

[Register as new User?](#)

Book Flight

My App

→ Book Flights

→ Book Hotels

→ Book Villa

→ Book Trains

→ Book Packages

→ View Bookings

→ Review Booking

≡ Book Flight

All Flights

Flight Number	Airline	Departure Airport	Arrival Airport	No of Seats in Flight	Departure Date	Arrival Date	Price	Book
AI101	Air India	DEL	BOM	248	Fri Sep 13 11:30:00 IST 2024	Thu Sep 12 08:30:00 IST 2024	4500.0	Book
6E202	IndiGo	CCU	DEL	186	Wed Aug 14 17:45:00 IST 2024	Wed Aug 14 15:30:00 IST 2024	8000.0	Book
SG503	SpiceJet	BOM	MAA	180	Sun Dec 15 20:30:00 IST 2024	Sun Dec 15 18:00:00 IST 2024	7530.0	Book
6E204	IndiGo	BLR	VNS	173	Sun Apr 21 17:45:00 IST 2024	Sun Apr 21 15:30:00 IST 2024	8000.0	Book
6E205	IndiGo	VNS	BLR	173	Thu Apr 25 17:45:00 IST 2024	Thu Apr 25 15:30:00 IST 2024	8000.0	Book
AI202	Air India	BLR	UDR	186	Wed May 15 17:45:00 IST 2024	Wed May 15 15:30:00 IST 2024	8000.0	Book
AI203	Air India	UDR	BLR	186	Sat May 18 17:45:00 IST 2024	Fri May 24 15:30:00 IST 2024	8000.0	Book

Book hotel

My App

→ Book Flights

→ Book Hotels

→ Book Villa

→ Book Trains

→ Book Packages

→ View Bookings

→ Review Booking

≡ Book Hotel

All Hotels

Hotel Name	Location	No of Deluxe Rooms	Deluxe Room Price/Night	No of Standard Rooms	Standard Room Price/Night	Book
Taj Mahal Palace	Mumbai	99	5000.0	198	3000.0	Book
The Oberoi Udaivilas	Udaipur	79	6000.0	149	3500.0	Book
ITC Grand Chola	Chennai	119	5500.0	249	3200.0	Book
Sarovar Protico	Varanasi	60	4500.0	100	2500.0	Book

Book train

My App

→ Book Flights

→ Book Hotels

→ Book Villa

→ Book Trains

→ Book Packages

→ View Bookings

→ Review Booking

≡ Book Train

All Trains

Train Number	Train Name	Departure Airport	Arrival Airport	No of Seats in Train	Departure Date	Arrival Date	Price	Book
12345	Rajdhani Express	Mumbai Central	New Delhi	89	Wed May 15 17:30:00 IST 2024	Wed May 15 13:30:00 IST 2024	1500.0	Book
67890	Shatabdi Express	Chennai Central	Bangalore City	58	Tue Jul 16 18:30:00 IST 2024	Tue Jul 16 14:30:00 IST 2024	900.0	Book
54321	Duronto Express	Howrah Junction	Sealdah	107	Fri Nov 29 19:30:00 IST 2024	Sun Nov 17 15:30:00 IST 2024	700.0	Book
98765	Nainital Express	Varanasi Junction	Kathgodam	62	Mon Apr 22 20:30:00 IST 2024	Mon Apr 22 14:00:00 IST 2024	1200.0	Book
56789	Varanasi Express	Kathgodam	Varanasi Junction	72	Wed Apr 24 22:00:00 IST 2024	Wed Apr 24 14:30:00 IST 2024	1100.0	Book

Book Package

My App

→ Book Flights

→ Book Hotels

→ Book Villa

→ Book Trains

→ Book Packages

→ View Bookings

→ Review Booking

Book Package

Package Name	Total Price	Flights	Trains	Hotels	Villa	Book
Varanasi Package	25000.0	<div>Flight Number ↕ Airline ↕</div>	<div>Train Number ↕ Train Na</div>	<div>Name ↕ Location</div>	<div>Villa Name ↕ Address</div>	Book
		6E204 IndiGo	98765 Nainita	Sarovar Proti... Varana:	Tranquil Oasis Naini	
		6E205 IndiGo	56789 Varana	Sarovar Proti... Varana:		

View bookings

My App

→ Dashboard

→ Manage Users

→ Manage Services

→ Manage Bookings

All Flights Bookings

Booking ID	Username	Flight Number	Flight Name	Departure Date	Arrival Date	Seats Booked	Total Price	Date Booked	Edit	Delete
6624d380fb3cca2dc3c67c1b	user2	BA202	British Airways		2024-05-11	1	4500.0	2024-04-15	Edit	Delete
6624d380fb3cca2dc3c67c1c	user1	DL303	Delta		2024-06-16	3	13500.0	2024-04-20	Edit	Delete
6624d380fb3cca2dc3c67c1d	user2	UA404	United		2024-07-21	2	9000.0	2024-04-25	Edit	Delete
6624d380fb3cca2dc3c67c1e	user1	LH505	Lufthansa		2024-08-26	1	4500.0	2024-04-30	Edit	Delete
6629583050fe650475f9962b	user1	SG503	SpiceJet		2024-00-15	2	15060.0	2024-04-25	Edit	Delete
6629e957c1cbfc120026deb4	user1	SG503	SpiceJet		2024-00-15	2	15060.0	2024-04-25	Edit	Delete

Flight Number

BA202

Flight Name

British Airways

Departure Date

2024-05-10

Arrival Date

2024-05-11

Seats booked

1

Total Price

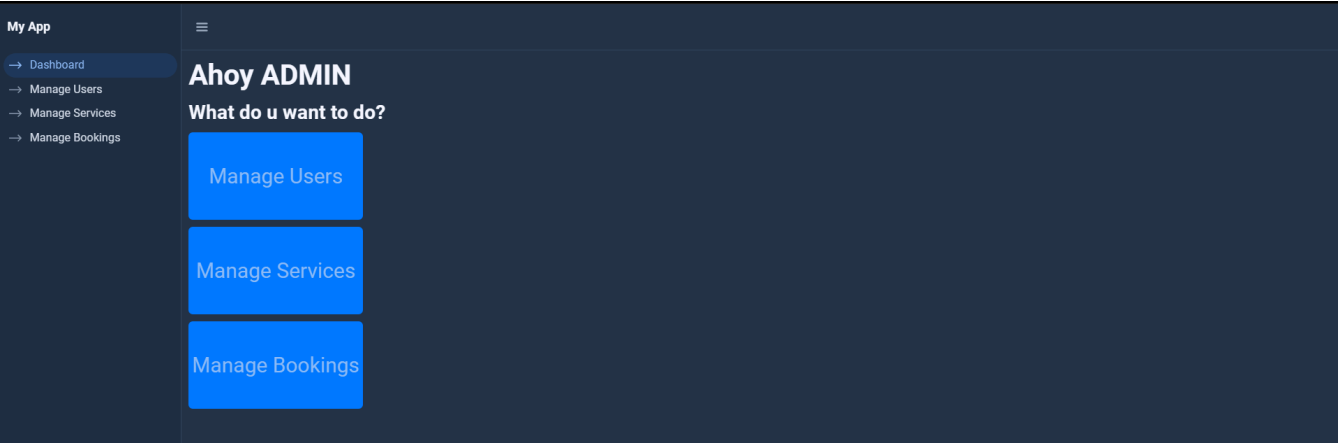
4500.0

save

All Train Bookings

Booking ID	Username	Train Number	Train Name	Departure Date	Arrival Date	Seats Booked	Total Price	Date Booked	Edit	Delete
6624ebe6fb3cca2dc3c67c22	user2	98765	Shatabdi Express	2024-05-15	2024-05-15	2	1500.0	2024-04-16	Edit	Delete
6624ebe6fb3cca2dc3c67c23	user1	67890	Rajdhani Express	2024-06-20	2024-06-21	3	4500.0	2024-04-22	Edit	Delete
6624ebe6fb3cca2dc3c67c24	user2	23456	Garib Rath Express	2024-07-25	2024-07-26	1	800.0	2024-04-28	Edit	Delete
66261ba6f7703ec7619b46	user1	54321	Duronto Express	2024-30-29	2024-30-17	10	700.0	2024-04-22	Edit	Delete
6630e01fe1abfd320036deb3	user1	54321	Duronto Express	2024-30-29	2024-30-17	4	3800.0	2024-04-25	Edit	Delete

Admin dashboard



Manage services by admin

<div>My App</div> <div><div>→ Dashboard</div><div>→ Manage Users</div><div>→ Manage Services</div><div>→ Manage Bookings</div></div>	Manage Flights							
	Airline	Arrival Airport	Arrival Date	Departure Airport	Departure Date	Flight Number	Number Of Seats In Flight	Price
	Air India	BOM	Fri Sep 13 11:30:00 IST ...	DEL	Thu Sep 12 08:30:00 IS...	AI101	248	4500.0
	IndiGo	DEL	Wed Aug 14 17:45:00 L...	CCU	Wed Aug 14 15:30:00 L...	6E202	186	8000.0
	SpiceJet	MAA	Sun Dec 15 20:30:00 IS...	BOM	Sun Dec 15 18:00:00 IS...	SG503	180	7530.0
	IndiGo	VNS	Sun Apr 21 17:45:00 IS...	BLR	Sun Apr 21 15:30:00 IS...	6E204	173	8000.0
	IndiGo	BLR	Thu Apr 25 17:45:00 IS...	VNS	Thu Apr 25 15:30:00 IS...	6E205	173	8000.0
	Air India	UDR	Wed May 15 17:45:00 L...	BLR	Wed May 15 15:30:00 L...	AI202	186	8000.0
	Air India	BLR	Sat May 18 17:45:00 IS...	UDR	Fri May 24 15:30:00 IST...	AI203	186	8000.0
<div>localhost:8080/manage-service</div>	Manage Trains							
	Arrival Date	Arrival Station	Departure Date	Departure Station	Number Of Seats In Train	Price	Train Name	Train Number
	Wed May 15 17:30:00 L...	New Delhi	Wed May 15 13:30:00 L...	Mumbai Central	89	1500.0	Rajdhani Express	12345
	Tue Jul 16 18:30:00 IST...	Bangalore City	Tue Jul 16 14:30:00 IST...	Chennai Central	58	900.0	Shatabdi Express	67890
	Fri Nov 29 19:30:00 IST...	Sealdah	Sun Nov 17 15:30:00 IS...	Howrah Junction	107	700.0	Duronto Express	54321
	Mon Apr 22 20:30:00 IS...	Kathgodam	Mon Apr 22 14:00:00 IS...	Varanasi Junction	62	1200.0	Nainital Express	98765
	Wed Apr 24 22:00:00 IS...	Varanasi Junction	Wed Apr 24 14:30:00 IS...	Kathgodam	72	1100.0	Varanasi Express	56789

Manage users by admin

My App

→ Dashboard

→ Manage Users

→ Manage Services

→ Manage Bookings

≡

Manage Users

Username	Role	Actions	Actions
user1	user	<div>Edit</div>	<div>Delete</div>
user2	user	<div>Edit</div>	<div>Delete</div>
jane_smith	manager	<div>Edit</div>	<div>Delete</div>
alice_wonder	manager	<div>Edit</div>	<div>Delete</div>
user3		<div>Edit</div>	<div>Delete</div>

Review your bookings

My App

→ Book Flights

→ Book Hotels

→ Book Villa

→ Book Trains

→ Book Packages

→ View Bookings

→ Review Booking

≡ Add review

Flights Bookings Done by You

Booking ID	Flight Number	Flight Name	Departure Date	Arrival Date	Seats Booked	Total Price	Date Booked	Give Feedback for this Booking
6624d380fb3cca2dc3c67c1c	DL303	Delta Airlines	2024-06-15	2024-06-16	3	13500.0	2024-04-20	<div>Review</div>
6624d380fb3cca2dc3c67c1e	LH505	Lufthansa	2024-08-25	2024-08-26	1	4500.0	2024-04-30	<div>Review</div>
6629583050fe650475f9962b	SG503	SpiceJet	2024-30-15	2024-00-15	2	15060.0	2024-04-25	Pay Before Reviewing
6629e957c1cbfc120026deb4	SG503					15060.0	2024-04-25	Pay Before Reviewing

How was the Stay/Travel Experience?

It was good

Any other Feedback?

Add Review

Train Bookings Done by You

Booking ID	Train Number	Train Name	Departure Date	Arrival Date	Seats Booked	Total Price	Date Booked	Give Feedback for this Booking
6624ebe6fb3cca2dc3c67c23	67890	Rajdhani Express	2024-06-20	2024-06-21	3	4500.0	2024-04-22	<div>Review</div>
66261ba6fbe7703ec7619b46	54321	Duronto Express	2024-30-29	2024-30-17	10	700.0	2024-04-22	Pay Before Reviewing
6629e91fc1cbfc120026deb2	54321	Duronto Express	2024-30-29	2024-30-17	4	2800.0	2024-04-25	Pay Before Reviewing