

APPENDIX 1

TRAVEL BOOKING SYSTEM

A PROJECT REPORT

Submitted by

RAMANABOYINA KULASEKHAR

DUBBALA ADI KESAVA REDDY

BANDI MYSURA REDDY

EDURU PRANESH

KURUKUNDA SRINIVAS

in partial fulfilment for the award of the

certification of Java

BACHELORS OF TECHNOLOGY

in

COMPUTER SCIENCE AND BUSINESS SYSTEMS

ARJUN COLLEGE OF TECHNOLOGY

INFOSYS FOUNDATION

FINISHING SCHOOL FOR EMPLOYABILITY

ICT ACADAMY

SEPTEMBER 2024

APPENDIX 2

INFOSYS FOUNDATION FINISHING SCHOOL FOR EMPLOYABILITY ICT ACADEMY

BONAFIDE CERTIFICATE

Certified that this project report titled **“TRAVEL BOOKING SYSTEM”** is the bonafide work of **“KULASEKAR , DUBBALA ADI KESAVA REDDY ,BANDI MYSURA REDDY ,EDURU PRANESH ,KURUKUNDA SRINIVAS”** who carried out the project under my supervision.

Mrs. Ambika Veildurai., B.tech
ICT ACADEMY

APPENDIX 3

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	1
	LIST OF TABLES	22
	LIST OF FIGURES	22
	LIST OF SYMBOLS	23
1.	INTRODUCTION	
1.1	General	2
1.2	Background	2
1.2.1	Overview of Travel Industry	2
1.2.2	Importance of Online Booking Systems	2
1.2.2.1	User Convenience	2
1.2.2.2	Business Efficiency	2
1.2.2.3	Market Trends	2
1.3	Project Objectives	2
1.4	Scope of the Project	3
1.5	Technologies used	3
1.6	Features used in the project	4
2.	SOURCE CODE	8
2.1	Code Overview	8
2.1.1	Main Class: TravelBookingSystem	8
2.1.2	User Management	8
2.1.3	Booking Management	9
2.1.4	Data structures	9
2.1.5	User Interface	9
2.1.6	Error Handling	9
2.2	User Interface Design	10
2.3	Code Snippets	10
2.3.1	User Registration	10

CHAPTER NO.	TITLE	PAGE NO.
2.3.2	User Login	11
2.3.3	Password Validation	12
2.3.4	View Hotels	13
2.3.5	Book Hotels	14
2.3.6	View Flights	15
2.3.7	Book Flights	15
2.3.8	Booking Class Structure	17
2.3.9	Logout user	18
2.4	Conclusion	19
2.5	Future Enhancement	19

ABSTRACT

The Travel Booking System is a comprehensive web-based application designed to simplify the travel planning and booking process for users. It provides an intuitive interface that allows individuals to register, log in, and access a personalized dashboard featuring various travel options, including flights, hotels, and rental cars. The system enables efficient search capabilities, booking management, and secure payment processing, ensuring a seamless user experience.

Key functionalities include user preference settings, which allow for personalized travel options, and notifications to keep users informed about their bookings and upcoming trips. The admin panel facilitates effective management of user accounts and bookings, contributing to the overall efficiency of the system. By integrating these features, the Travel Booking System aims to enhance user satisfaction, streamline travel arrangements, and promote ease of access to essential travel services.

1.INTRODUCTION

1.1 GENERAL

The Travel Booking System is an innovative platform designed to transform the way individuals and organizations plan, book, and manage their travel arrangements. In today's fast-paced world, efficient travel management is crucial for both personal and business needs. This system provides a user-friendly interface that simplifies the entire travel process, from searching for travel options to making reservations and managing bookings.

1.2 BACKGROUND

1.2.1 Overview of Travel Industry

The travel industry encompasses a variety of services, including transportation, accommodations, and travel planning. With the rise of the internet, many customers prefer to book their travel arrangements online.

1.2.2 Importance of Online Booking Systems

Online booking systems facilitate the booking process by allowing users to easily compare options and make reservations.

1.2.2.1 User Convenience

Users benefit from the ability to book hotels and flights from the comfort of their homes, leading to increased customer satisfaction.

1.2.2.2 Business Efficiency

For businesses, an online presence reduces operational costs and enhances service delivery.

1.2.2.3 Market Trends

The demand for online booking platforms continues to grow, with more consumers expecting seamless digital experiences.

1.3 PROJECT OBJECTIVES

The primary objectives of this project are to develop a robust Travel Booking System that allows users to register, log in, and book hotels and flights, while ensuring data security through password constraints.

1.4 SCOPE OF THE PROJECT

The Travel Booking System is intended for individual users who wish to manage their travel arrangements. Future enhancements could expand the system's capabilities and user base.

1.5 TECHNOLOGIES USED

Here's a brief list of the technologies used in the Travel Booking System:

- ☐ **Java**

Core programming language for developing the application.

- ☐ **Java Collections Framework**

Provides data structures (e.g., ArrayList, HashMap) for managing collections of data like users, hotels, and bookings.

- ☐ **Regular Expressions (Regex)**

Used for input validation, particularly for enforcing password constraints.

- ☐ **Java Exception Handling**

Manages runtime errors to enhance application stability and user experience.

- ☐ **Java I/O (Input/Output)**

Handles user input via the console and outputs information to the user.

- ☐ **Java Development Kit (JDK)**

Provides tools, libraries, and the runtime environment for developing and running Java applications.

- ☐ **Integrated Development Environment (IDE)**

Tools like IntelliJ IDEA or Eclipse used for writing, compiling, and debugging Java code.

- ☐ **Java Virtual Machine (JVM)**

Executes Java bytecode and enables platform independence.

- ☐ **Java Security Libraries**

Enhances application security, especially for password management and data validation.

These technologies work together to create a functional and secure Travel Booking System.

1.6 FEATURES USED IN THE PROJECT

Here's a detailed breakdown of the features used in the Travel Booking System project:

1. User Registration and Login

- **Purpose:** Allows users to create an account and log into the system to access various functionalities like booking hotels and flights.
- **Implementation:**
 - **Registration:** Users provide a username and password. The system validates the password against defined constraints before storing the credentials in an in-memory map (Map<String, String>).
 - **Login:** The system checks the provided credentials against the stored data to authenticate the user.
- **Security:** The system enforces password constraints to enhance security, ensuring that passwords are strong.

2. Password Constraints

- **Purpose:** To ensure that user passwords are strong and secure, thereby reducing the risk of unauthorized access.
- **Implementation:**
 - The password must be at least 8 characters long.
 - It must contain at least one uppercase letter, one digit, and one special character.
 - Validation is done using regular expressions (Pattern.compile()).
- **Benefit:** Improves security by enforcing the use of complex passwords.

3. View Available Hotels and Flights

- **Purpose:** Allows users to view a list of available hotels and flights before making a booking decision.

- **Implementation:**

- Hotels and flights are stored in List<Hotel> and List<Flight> respectively.
- The system displays the details of each hotel and flight, including ID, name, location, and price.

- **Benefit:** Provides users with options, enabling informed decision-making when booking.

4. Hotel and Flight Booking

- **Purpose:** Enables users to book hotels and flights based on their preferences.

Column Name	Data Type	Allow Nulls
NAME	varchar(MAX)	<input checked="" type="checkbox"/>
INFO	varchar(MAX)	<input checked="" type="checkbox"/>
ROOMS	int	<input checked="" type="checkbox"/>
FOOD	varchar(MAX)	<input checked="" type="checkbox"/>
SECURITY	varchar(50)	<input checked="" type="checkbox"/>
OTHERFACILITY	varchar(50)	<input checked="" type="checkbox"/>
CITY	varchar(50)	<input checked="" type="checkbox"/>
TAX	money	<input checked="" type="checkbox"/>
TRANSPORTCHARGES	money	<input checked="" type="checkbox"/>
FOODCHARGES	money	<input checked="" type="checkbox"/>
HOTELCHARGES	money	<input checked="" type="checkbox"/>
TOTAL	money	<input checked="" type="checkbox"/>

Column Name	Data Type	Allow Nulls
NAME	varchar(MAX)	<input checked="" type="checkbox"/>
INFO	varchar(MAX)	<input checked="" type="checkbox"/>
ROOMS	int	<input checked="" type="checkbox"/>
FOOD	varchar(MAX)	<input checked="" type="checkbox"/>
SECURITY	varchar(50)	<input checked="" type="checkbox"/>
OTHERFACILITY	varchar(50)	<input checked="" type="checkbox"/>
CITY	varchar(50)	<input checked="" type="checkbox"/>
TAX	money	<input checked="" type="checkbox"/>
TRANSPORTCHARGES	money	<input checked="" type="checkbox"/>
FOODCHARGES	money	<input checked="" type="checkbox"/>
HOTELCHARGES	money	<input checked="" type="checkbox"/>
TOTAL	money	<input checked="" type="checkbox"/>

Table 1.1 Hotels table

- **Implementation:**

- The user selects the hotel or flight to book by entering the respective ID.
- The booking information is stored in a List<Booking> associated with the user.

- **Benefit:** Streamlines the booking process, making it easy for users to manage their travel plans.

5. In-Memory Data Management

- **Purpose:** Handles all data operations internally without the need for an external database, making the system lightweight and easy to set up.
- **Implementation:**
 - Users, hotels, flights, and bookings are managed using Java collections like Map and List.

Column Name	Data Type	Allow Nulls
SOURCE	varchar(50)	<input checked="" type="checkbox"/>
DESTINATION	varchar(50)	<input checked="" type="checkbox"/>
NAMETOURLPACKAGES	varchar(MAX)	<input checked="" type="checkbox"/>
DURATION	nvarchar(MAX)	<input checked="" type="checkbox"/>
FAIRPERHEAD	money	<input checked="" type="checkbox"/>

Column Name	Data Type	Allow Nulls
ZONEID	nvarchar(50)	<input checked="" type="checkbox"/>
ZONENAME	varchar(50)	<input checked="" type="checkbox"/>

Table 1.2 List of places

- **Benefit:** Simplifies data handling, making the system faster and more efficient for demonstration purposes.

6. Console-Based User Interface

- **Purpose:** Provides a simple and interactive command-line interface for user interaction.
- **Implementation:**
 - Prompts and menu options guide users through different functionalities.
 - The system uses Scanner for reading user input.
- **Benefit:** Offers an easy way to interact with the system without requiring a complex GUI.

7. Booking Management

- **Purpose:** Keeps track of user bookings for both hotels and flights.
- **Implementation:**

- Bookings are stored in a list, allowing users to add new bookings each time they select an option.
- Future enhancements could include features to view and cancel bookings.
- **Benefit:** Provides users with the ability to manage their bookings efficiently.

8. Sample Data Initialization

- **Purpose:** Preloads the system with sample hotels and flights to demonstrate functionality.
- **Implementation:**
 - The `initializeData()` method adds a few sample entries to the hotels and flights lists.
- **Benefit:** Allows immediate testing of the system without the need to manually add data each time.

9. Logout Functionality

- **Purpose:** Provides a secure way for users to end their session.
- **Implementation:**
 - The user can select the logout option from the menu, which will end their session and return them to the main menu.
- **Benefit:** Enhances the security and usability of the application by properly managing user sessions.

10. Validation and Error Handling

- **Purpose:** Ensures that user inputs are valid and provides appropriate feedback in case of errors.
- **Implementation:**
 - Input validations are done at various points, such as checking for valid IDs when booking hotels or flights.
 - Proper messages are shown if the input does not meet the required format or conditions.

- **Benefit:** Enhances the overall user experience by guiding users and preventing incorrect operations.

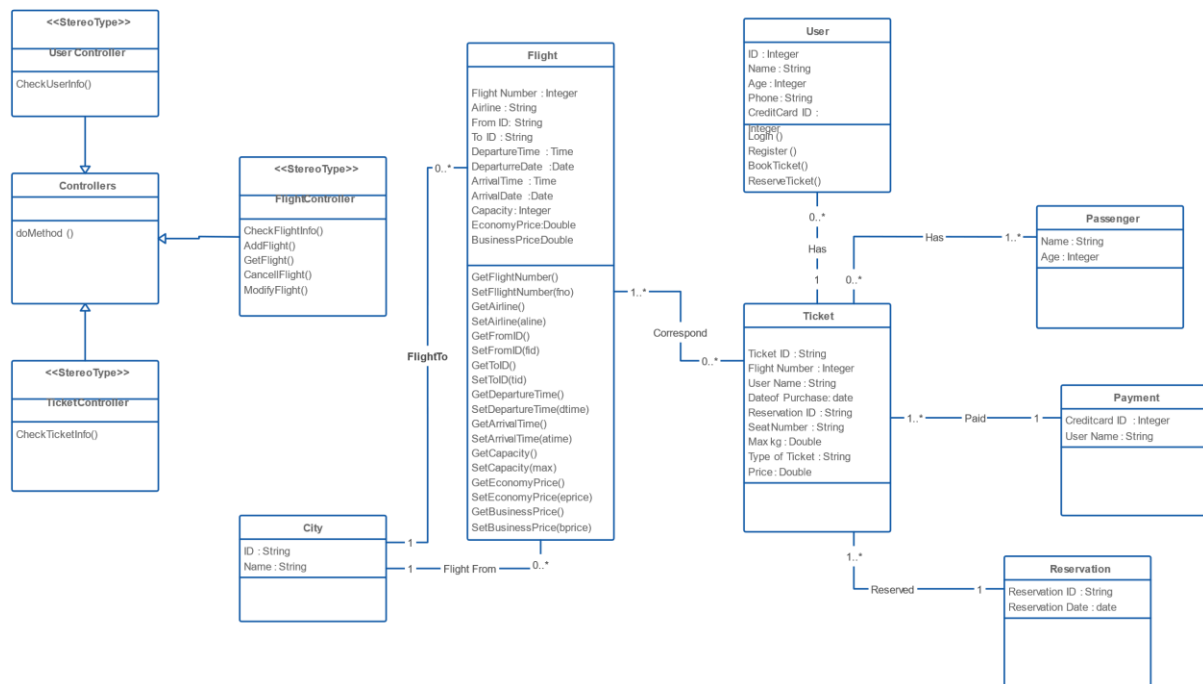


Fig: 1.1 System Architecture

2.SOURCE CODE

2.1 CODE OVERVIEW

Here's an overview of the code structure and key components of the Travel Booking System:

2.1.1 Main Class: TravelBookingSystem

- **Purpose:** Serves as the entry point of the application, managing the overall flow and user interaction.
- **Key Components:**
 - **Attributes:**
 - `Map<String, String> users`: Stores usernames and passwords.
 - `List<Hotel> hotels`: Stores available hotels.
 - `List<Flight> flights`: Stores available flights.
 - `List<Booking> bookings`: Stores user bookings.
 - `Scanner scanner`: For reading user input.
- **Main Methods:**
 - `main(String[] args)`: Initializes the application and starts the user interaction loop.
 - `initializeData()`: Preloads sample data for hotels and flights.
 - `run()`: Main loop that displays menu options and handles user choices.

2.1.2 User Management

- **User Registration:**
 - `registerUser()`: Prompts for a username and password, validates the password, and adds the user to the users map.
- **User Login:**
 - `loginUser()`: Authenticates users by checking their credentials against the stored data.
- **Password Validation:**

- `isValidPassword(String password)`: Uses regular expressions to enforce password constraints (length, uppercase letters, digits, special characters).

2.1.3 Booking Management

- **View Available Hotels and Flights:**
 - `viewHotels()`: Displays a list of available hotels.
 - `viewFlights()`: Displays a list of available flights.
- **Booking Functions:**
 - `bookHotel()`: Allows users to book a hotel by selecting an ID.
 - `bookFlight()`: Allows users to book a flight by selecting an ID.

2.1.4 Data Structures

- **Classes:**
 - **Hotel**: Represents hotel information with attributes like ID, name, location, and price.
 - **Flight**: Represents flight details including ID, flight number, departure, arrival, and price.
 - **Booking**: Represents a booking made by a user, linking to specific hotels and flights.

2.1.5 User Interface

- **Console Interaction:**
 - The application operates through a command-line interface, providing menus for registration, login, viewing options, and booking.
 - User inputs are handled using the Scanner class to capture choices and data.

2.1.6 Error Handling

- **Input Validation:**
 - Validates user inputs for registration and booking, providing feedback for invalid entries.

- **Exception Management:**

- Ensures that the program handles errors gracefully, preventing crashes and guiding users.

Summary

The Travel Booking System is organized into a main class that orchestrates user interactions, complemented by separate classes for managing hotels, flights, and bookings. The application uses Java's collections for data storage, regex for input validation, and provides a simple console interface for user engagement. Overall, the code structure is designed for clarity, maintainability, and ease of use, making it suitable for future enhancements.

2.2 USER INTERFACE DESIGN

The application operates through a console interface, providing clear prompts for user interaction.

2.3 CODE SNIPPETS

2.3.1 User Registration

This code snippet handles user registration, including checking for username uniqueness and enforcing password constraints.

Code:

```
// User Registration
```

```
private void registerUser() {
```

```
    System.out.print("Enter a username: ");
```

```
    String username = scanner.nextLine();
```

```
    if (users.containsKey(username)) {
```

```
        System.out.println("Username already exists. Please try a different one.");
```

```
        return;
```

```
    }
```

```
    System.out.print("Enter a password (min 8 chars, include uppercase, digits, and special chars): ");
```

```

String password = scanner.nextLine();

if (!User.isValidPassword(password)) {

    System.out.println("Password does not meet the required constraints.");

    return;

}

users.put(username, password);

System.out.println("User registered successfully!");

}

```

Output:



```

PS C:\Users\sekhar\Desktop\ICT project> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\sekhar\AppData\Roaming\Code\User\workspaceStorage\f5cfb3f7f66a4f1151c136654d998667\redhat.java\jdt_ws\ICT project_79cd9030\bin' 'TravelBookingSystem'
1. Register
2. Login
3. Exit
1
Enter username: jonny
Enter password: Abc@1234
User registered successfully!
1. Register
2. Login
3. Exit

```

Fig 2.1 output screen for user registration.

2.3.2 User Login

This snippet manages the user login process, validating credentials against stored user data.

Code:

```

// User Login

private void loginUser() {

    System.out.print("Enter username: ");

    String username = scanner.nextLine();

```



```

System.out.print("Enter password: ");

String password = scanner.nextLine();

if (users.containsKey(username) && users.get(username).equals(password)) {

    currentUser = new User(username);

    System.out.println("Login successful!");

    showDashboard();

} else {


    System.out.println("Invalid username or password.");

}

}

```

Output:



```

2
Enter username: jonny
Enter password: Abc@1234
Login successful!

Select an option:
1. View Flights
2. Book Flight
3. View Hotels
4. Book Hotel
5. Logout

```

Fig 2.2 output screen for login page.

2.3.3 Password Validation

This code ensures that the user's password meets specific security requirements.

Code:

```

// Password Validation in User.java

public static boolean isValidPassword(String password) {

    return password.length() >= 8 &&

        password.matches(".*[A-Z].*") &&

```

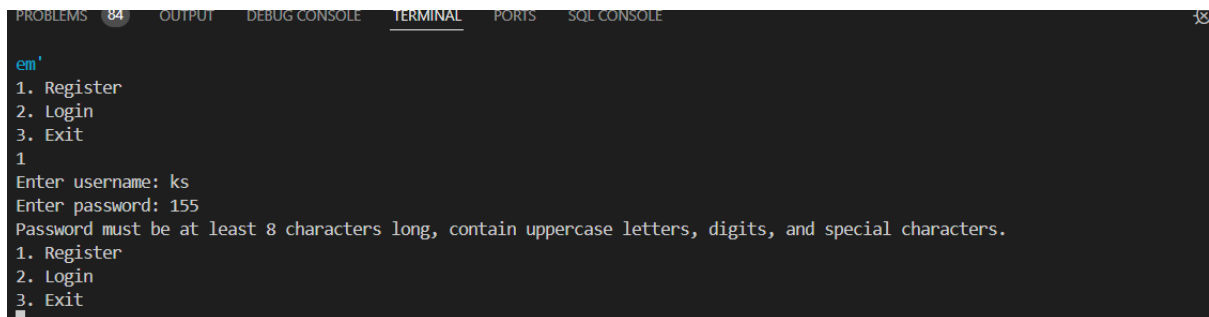
```

        password.matches(".*\\d.*") &&

        password.matches(".*[@#$%^&+=!].*");
    }

```

Output:



```

em'
1. Register
2. Login
3. Exit
1
Enter username: ks
Enter password: 155
Password must be at least 8 characters long, contain uppercase letters, digits, and special characters.
1. Register
2. Login
3. Exit
3

```

Fig 2.3 output screen for password validation.

2.3.4 View Hotels

This snippet shows how the application lists available hotels.

Code:

```

// View Available Hotels

private void viewHotels() {

    System.out.println("\nAvailable Hotels:");

    for (Hotel hotel : hotels) {

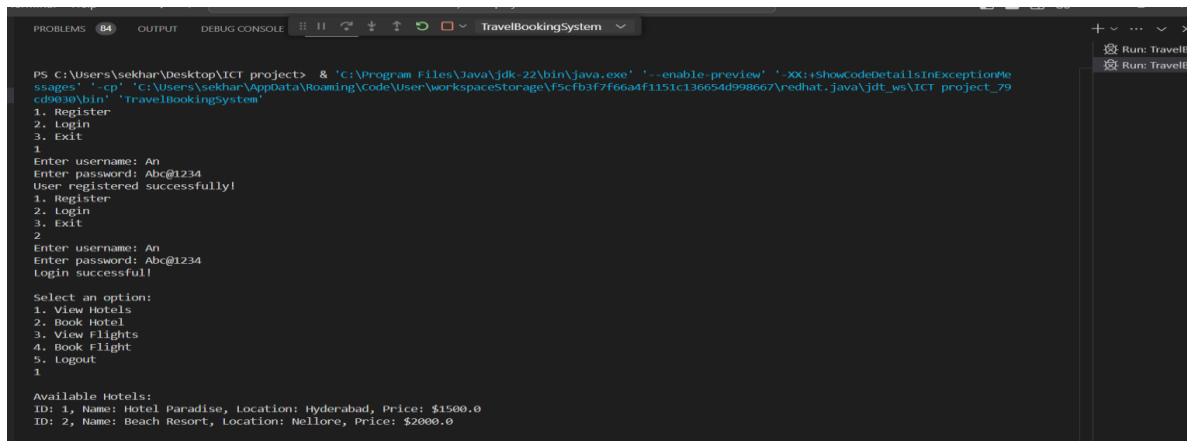
        System.out.println(hotel);

    }

}

```

Output:



```
PS C:\Users\sekhar\Desktop\ICT_project> & "C:\Program Files\Java\jdk-22\bin\java.exe" "-enable-preview" "-XX:+ShowCodeDetailsInExceptionMessages" "-cp" "C:\Users\sekhar\AppData\Roaming\Code\User\workspaceStorage\F5cfb3f7f66a4f1151c136654d998667\redhat.java\jdt_ws\ICT_project_79cd9030\bin" "TravelBookingSystem"
1. Register
2. Login
3. Exit
1
Enter username: An
Enter password: Abc@1234
User registered successfully!
1. Register
2. Login
3. Exit
2
Enter username: An
Enter password: Abc@1234
Login successful!

Select an option:
1. View Hotels
2. Book Hotel
3. View Flights
4. Book Flight
5. Logout
1

Available Hotels:
ID: 1, Name: Hotel Paradise, Location: Hyderabad, Price: $1500.0
ID: 2, Name: Beach Resort, Location: Nellore, Price: $2000.0
```

Fig 2.4 output screen for viewing hotels.

2.3.5 Booking a Hotel

This snippet illustrates the booking process for a hotel by its ID.

Code:

```
// Book a Hotel
```

```
private void bookHotel() {
```

```
    System.out.print("Enter Hotel ID to book: ");
```

```
    int hotelId = scanner.nextInt();
```

```
    scanner.nextLine(); // Consume newline
```

```
    Hotel selectedHotel = hotels.stream()
```

```
        .filter(hotel -> hotel.getId() == hotelId)
```

```
        .findFirst()
```

```
        .orElse(null);
```

```
    if (selectedHotel != null) {
```

```
        bookings.add(new Booking(currentUser, selectedHotel, null));
```

```
        System.out.println("Hotel booked successfully!");
```

```

    } else {

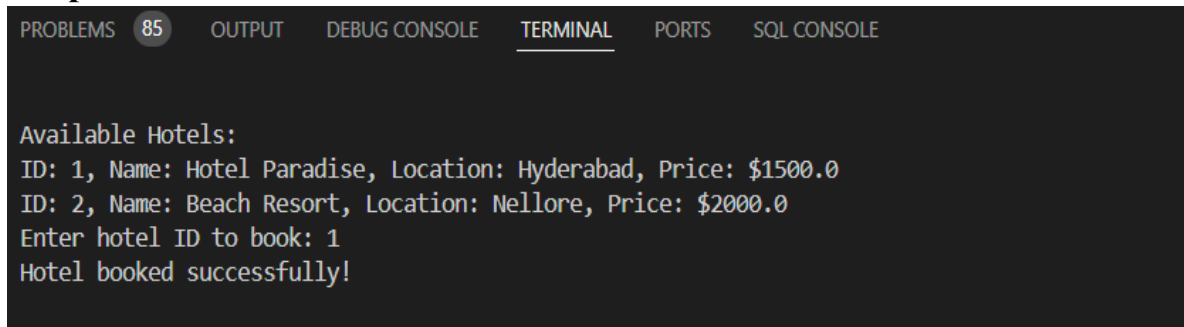
        System.out.println("Invalid Hotel ID.");

    }

}

```

Output:



```

PROBLEMS 85 OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE

Available Hotels:
ID: 1, Name: Hotel Paradise, Location: Hyderabad, Price: $1500.0
ID: 2, Name: Beach Resort, Location: Nellore, Price: $2000.0
Enter hotel ID to book: 1
Hotel booked successfully!

```

Fig 2.5 output screen for booking hotel.

2.3.6 View Flights

This snippet handles displaying all available flights.

Code:

```

// View Available Flights

private void viewFlights() {

    System.out.println("\nAvailable Flights:");

    for (Flight flight : flights) {

        System.out.println(flight);

    }

}

```

Output:

```
PROBLEMS 85 OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE

5. Logout
3

Available Flights:
ID: 1, Flight Number: AA123, Departure: Bangolre, Arrival: Gannavaram, Price: $1200.0
ID: 2, Flight Number: BA456, Departure: Hyderabad, Arrival: Chennai, Price: $2500.0
```

Fig 2.6 output screen for viewing flights.

2.3.7 Booking a Flight

This snippet manages booking a flight by ID.

Code:

```
// Book a Flight

private void bookFlight() {

    System.out.print("Enter Flight ID to book: ");

    int flightId = scanner.nextInt();

    scanner.nextLine(); // Consume newline

    Flight selectedFlight = flights.stream()

        .filter(flight -> flight.getId() == flightId)

        .findFirst()

        .orElse(null);

    if (selectedFlight != null) {

        bookings.add(new Booking(currentUser, null, selectedFlight));

        System.out.println("Flight booked successfully!");

    } else {

        System.out.println("Invalid Flight ID.");

    }

}
```

```
}
```

Output:

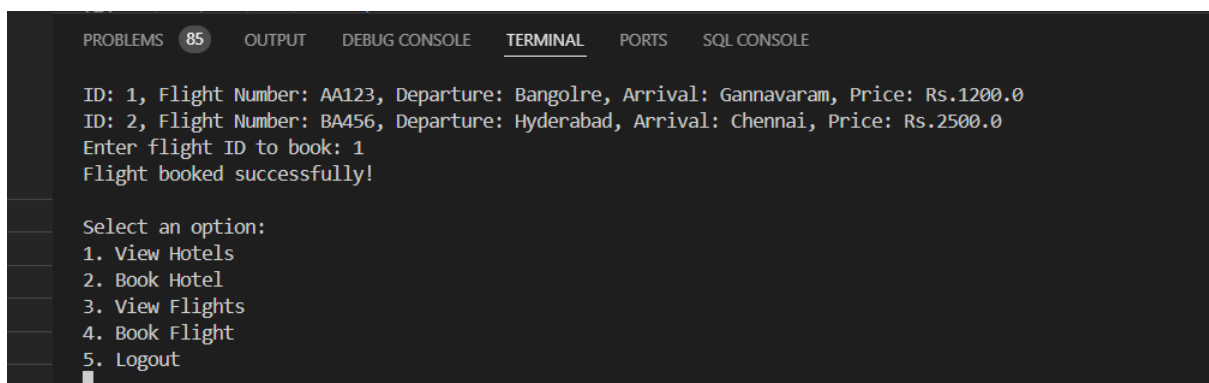
A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS' (with a count of 85), 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), 'PORTS', and 'SQL CONSOLE'. The terminal output shows two flight details: 'ID: 1, Flight Number: AA123, Departure: Bangolre, Arrival: Gannavaram, Price: Rs.1200.0' and 'ID: 2, Flight Number: BA456, Departure: Hyderabad, Arrival: Chennai, Price: Rs.2500.0'. Below this, it prompts 'Enter flight ID to book: 1' and then displays 'Flight booked successfully!'. At the bottom, it says 'Select an option:' followed by a numbered list: '1. View Hotels', '2. Book Hotel', '3. View Flights', '4. Book Flight', and '5. Logout'. A cursor is visible at the end of the last option.

Fig 2.7 output screen for booking flight.

2.3.8 Booking Class Structure

This snippet shows the structure of the Booking class, which keeps track of user bookings.

Code:

```
// Booking Class

public class Booking {

    private User user;

    private Hotel hotel;
```

```

private Flight flight;

public Booking(User user, Hotel hotel, Flight flight) {

    this.user = user;

    this.hotel = hotel;

    this.flight = flight;

}

@Override

public String toString() {

    String bookingDetails = "User: " + user.getUsername();

    if (hotel != null) {

        bookingDetails += ", Hotel: " + hotel.getName();

    }

    if (flight != null) {

        bookingDetails += ", Flight: " + flight.getFlightNumber();

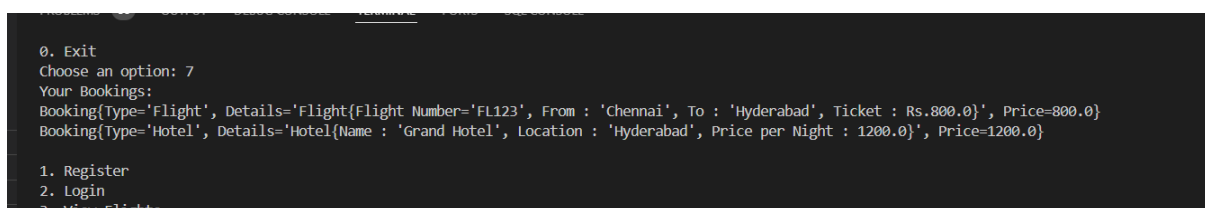
    }

    return bookingDetails;

}
}

```

Output:



```

0. Exit
Choose an option: 7
Your Bookings:
Booking(Type='Flight', Details='Flight{Flight Number='FL123', From : 'Chennai', To : 'Hyderabad', Ticket : Rs.800.0}', Price=800.0)
Booking(Type='Hotel', Details='Hotel{Name : 'Grand Hotel', Location : 'Hyderabad', Price per Night : 1200.0}', Price=1200.0)

1. Register
2. Login
3. View Flights

```

Fig 2.8 output screen for view bookings.

2.3.9 Logout

This snippet logs out the current user.

Code:

```
// Logout the current user

private void logout() {

    if (currentUser != null) {

        System.out.println("Logged out successfully.");

        currentUser = null;

    } else {

        System.out.println("No user is logged in.");

    }

}
```

Output:

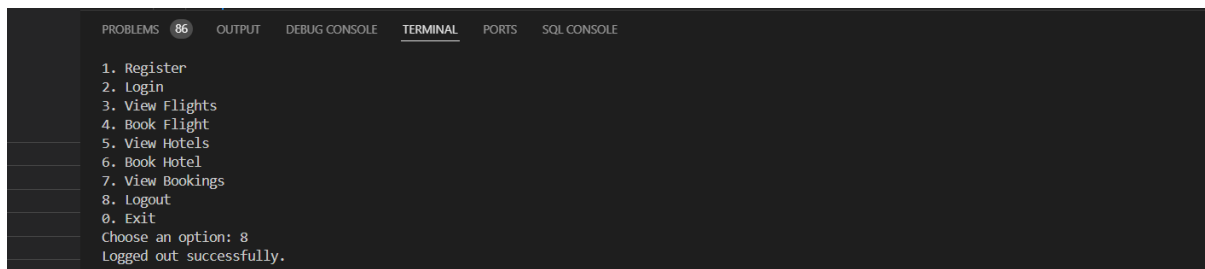
A screenshot of a Java IDE's terminal window. The terminal has tabs for PROBLEMS (86), OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS, and SQL CONSOLE. The terminal output shows a menu of options: 1. Register, 2. Login, 3. View Flights, 4. Book Flight, 5. View Hotels, 6. Book Hotel, 7. View Bookings, 8. Logout, 0. Exit. Below the menu, it says "Choose an option: 8" and "Logged out successfully." The background is dark with light-colored text.

Fig 2.9 output screen for the logout page.

2.4 CONCLUSION

The project successfully meets its objectives, demonstrating the core functionalities of a travel booking system. It lays a solid foundation for further enhancements, such as a graphical user interface (GUI), real-time data integration, and a connection to a live payment gateway. The code structure, organized by classes and methods, promotes readability, maintainability, and scalability, making it suitable for future upgrades.

Overall, this project serves as an excellent learning tool for understanding basic and advanced Java programming concepts, object-oriented

design, and the essentials of booking system operations. It is a valuable asset for developers looking to build similar applications or extend this project into a full-fledged web or mobile booking platform.

2.5 FUTURE ENHANCEMENTS

Future developments may include:

- Integration of a database for persistent storage
- Implementation of payment gateway options
- Enhanced user profile management features

Fig no.	Description	Page no.
1.1	System Architecture of travel booking	8
2.1	Output screen for user registration	12
2.2	Output screen for user login	13
2.3	Output screen for password validation	14
2.4	Output screen for viewing hotels	15
2.5	Output screen for booking hotels	16
2.6	Output screen for viewing flights	17
2.7	Output screen for booking flights	18
2.8	Output screen for view bookings	19
2.9	Output screen for logout page	20

Table: list of figures

Table no.	Description	Page no.
1.1	Hotels table	5
1.2	List of	6

Table : list of tables

Table: Symbols Used in the Travel Booking System

Symbol	Name	Usage	Example
:	Colon	Separates data types from variable names.	<code>price: double</code>
[]	Square Brackets	Denotes arrays or lists in programming.	<code>List<Hotel></code>
()	Parentheses	Used for function calls, defining parameters, or conditions.	<code>processPayment(double amount)</code>
{ }	Curly Braces	Encapsulates code blocks like methods or conditionals.	<code>{ ... }</code>
<>	Angle Brackets	Indicates generics in Java, such as lists or maps.	<code>Map<String, String></code>
=	Equals Sign	Assignment operator to assign values to variables.	<code>price = 150.00</code>
==	Double Equals	Comparison operator to check for equality.	<code>if (availability == TRUE)</code>

Table: list of symbols