**INTERNSHIP**

**TRAVEL MANAGEMENT SYSTEM**

*A report submitted in partial fulfillment of the requirements for the Award of Degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**By**

**MAGANTI BHAVYA SRI**

**Regd.no: 21B91A04D0**

**Under Supervision of Mr. Rafikh (Trainer name)**

**Henotic Technology Pvt Ltd, Hyderabad**

**(Duration: 5th July 2023 to 5th September 2023)**



ELECTRONICS AND COMMUNICATION ENGINEERING

S.R.K.R. ENGINEERING COLLEGE

(Autonomous)

SRKR MARG, CHINNA AMIRAM, BHIMAVARAM-534204, A.P

(Recognized by A.I.C.T.E New Delhi) (Accredited by NBA & NAAC)

(Affiliated to JNTU, KAKINADA)

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE

(Autonomous)

A logo with several images

Description automatically generated with medium confidence ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the **"Summer Internship Report**” titled **“Travel Management System”** is the bonafide work done by Miss. **Maganti Bhavya Sri** bearing Regd.no: 21B91A04D0 at the end of second year second semester at **Henotic IT Solutions PVT Ltd , Hyderabad** from 5th July 2023 to 5th September 2023in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering.

**Department Internship Coordinator Dean -T & P Cell Head of the Department**

**Project Report**

**On**

# TRAVEL MANAGEMENT SYSTEM

# 

**S. R. K. R ENGINEERING COLLEGE (A) (Affiliated to JNTU, Kakinada)**

**Bhimavaram- (534 204).**

**(02-07-2023 TO 28-07-2023)**



**Guided By**

**MR. RAFIKH**

**Submitted By**

**M. BHAVYA SRI (21B91A04D0)-ECE**

**K. LAKSHMI PADMAJA (21B91A04A7) – ECE**

**P. MANASA (21B91A04I1)-ECE**

**ACKNOWLEDGEMENT**

I would like to extend my heartfelt gratitude to **Mr. Rafikh** working in Henotic IT Solutions Pvt Ltd in Hyderabad, for affording me the valuable opportunity to embark on an enriching internship journey within the organization. His belief in me and the chance he provided will forever be appreciated.

I am also deeply indebted to the incredible team at Henotic IT Solutions Pvt Ltd Hyderabad, who worked alongside me during this internship. Their unwavering patience, openness, and collaborative spirit not only made the workplace enjoyable but also fostered an environment conducive to growth and learning. Their support was instrumental in making this experience truly remarkable.

I wish to express my profound gratitude to Principal **Prof. M. Jagapathi Raju Garu** for the facilities and resources provided, which were instrumental in the successful completion of my internship.

I would like to acknowledge and thank my Head of the Department **Prof. DR. N. Uday Kumar**, for his invaluable constructive guidance throughout my internship, which played a pivotal role in enhancing my skills and knowledge.

I am also indebted to **Sri. K N V S Varma**, the internship coordinator of the Department of IT, for their unwavering support and valuable advice that not only helped me secure this internship but also guided me in successfully completing it.

I am extremely greatful to my department staff members and friends who helped me in successful completion of this internship.

M.BHAVYA SRI

(21B91A04D0)

**Internship Objectives**

* Internships are generally thought of to be reserved for college students looking to gain experience in a particular field.
* However, a wide array of people can benefit from Training Internships in order to receive real world experience and develop their skills.
* An objective for this position should emphasize the skills you already possess in the area and your interest in learning more.
* Internships are utilized in a number of different career fields, including architecture, engineering, healthcare, economics, advertising and many more.
* Some internship is used to allow individuals to perform scientific research while others are specifically designed to allow people to gain first-hand experience working.
* Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs.
* When you are applying for a Training Internship, make sure to highlight any special skills or talents that can make you stand apart from the rest of the applicants so that you have an improved chance of landing the positions.

**WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES**

|  |  |  |  |
| --- | --- | --- | --- |
| **1st WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| 05/07/23 | Wednesday | Introduction to java |
| 06/07/23 | Thursday | Array and Loops concepts |
| 07/07/23 | Friday | Eclipse (IDE) installation |
| 08/07/23 | Saturday | OOPs concepts |
| 09/07/23 | Sunday | Modifiers and Interfaces |

|  |  |  |  |
| --- | --- | --- | --- |
| **2nd WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| 10/07/23 | Monday | Abstraction and Debugging |
| 11/07/23 | Tuesday | Special keywords in java |
| 12/07/23 | Wednesday | Basics of DS |
| 13/07/23 | Thursday | Stack and Linked Lists |
| 14/07/23 | Friday | Double linked List |
| 15/07/23 | Saturday | Queue and circular queues |

|  |  |  |  |
| --- | --- | --- | --- |
| **3rd WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| 17/07/23 | Monday | Tree Data Structures |
| 18/07/23 | Tuesday | Tree Traversals |
| 19/07/23 | Wednesday | Sorting Algorthims |
| 20/07/23 | Thursday | Array List |
| 21/07/23 | Friday | Heap Data Structure |
| 22/07/23 | Saturday | Hash Table and Hash Maps |

|  |  |  |  |
| --- | --- | --- | --- |
| **4th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| 24/07/23 | Monday | Exploring Projects Ideas |
| 25/07/23 | Tuesday | Discussing With Trainer |
| 26/07/23 | Wednesday | Project Session |
| 27/07/23 | Thursday | Finalizing the ideas |
| 28/07/23 | Friday | Building Up the Requirements |
| 29/07/23 | Saturday | Coding |

|  |  |  |  |
| --- | --- | --- | --- |
| **5th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| 30/07/23 | Monday | Coding |
| 31/07/23 | Tuesday |
| 01/08/23 | Wednesday |
|  | 02/08/23 | Thursday | Testing |
|  | 03/08/23 | Friday | Enhancement |
|  | 04/08/23 | Saturday | Submission |

**TABLE OF CONTENTS**

**S.NO CONTENTS PAGE.NO**

1. ABSTRACT 9
2. INTRODUCTION 10-11
3. PROJECT DESCRIPTION 11-12
4. PROBLEM STATEMENT 12-13
5. SOFTWARE REQUIREMENT SPECIFICATIONS 13
6. HARDWARE REQUIREMENTS 14
7. SYSTEM ARCHITECTURE 14
8. IMPLEMENTATION 14-34
9. FUNCTIONALITY AND USAGE 34
10. OUTPUT 35-37
11. CONCLUSION 38

**1.ABSTRACT**

Developed in Java programming language, Travel Management System is a console-based project or application for travel agencies. The main objective of this project is to create a fast, effective and reliable online working platform to develop a communication system between customers and the agency. After the implementation of this project, one doesn’t need to visit the travel agency office to plan any tour.

The Travel Management System (TMS) is a comprehensive software solution designed to streamline and optimize the planning, booking, and management of travel-related activities. In an increasingly globalized and interconnected world, efficient travel management is crucial for businesses, organizations, and individuals alike. The TMS offers a user-friendly and integrated platform that simplifies the entire travel process, from trip planning and itinerary creation to expense reporting and analytics.

The proposed java project on travel management is an solution to the existing problems regarding cruise management in various travel agencies. It assists in promotion of tourism by providing information regarding different places, distances and the ways of getting there. This project is beneficial for both travel management agencies and the customers in many ways.

**2.INTRODUCTION**

**2.1 Background**

Data Structures and Algorithms (DSA) serve as the cornerstone of computer science and software development. They are the underlying framework that enables efficient data storage, retrieval, and manipulation, and they play a pivotal role in the functionality of a wide range of applications and systems. Without a solid grasp of DSA, it is challenging to craft robust and high-performing software solutions.

**2.2 The Significance of DSA**

In the world of software engineering, DSA is the bedrock upon which innovation and efficiency are built. Whether it's searching for a specific element in a massive dataset, sorting a list of items, or efficiently traversing complex data structures, DSA provides the tools and techniques to solve these problems. By mastering DSA, we equip ourselves with the capability to design and implement elegant, scalable, and optimized software.

**2.3 The Power of Java**

In this project, we combine the power of Data Structures and Algorithms with the versatility of the Java programming language. Java is renowned for its platform independence, robustness, and extensive library support. It is a language that not only empowers software developers but also provides a fertile ground for implementing and experimenting with DSA.

**2.4 Project Overview**

In this project, we embark on a journey into the realm of Data Structures and Algorithms through, harnessing the robust capabilities of the Java programming language. Our goal is to explore, understand, and apply various DSA concepts to real-world scenarios, highlighting their practical significance within the context of Java.

**2.5 Objectives**

Our project revolves around a set of fundamental objectives:

**2.5.1 Implementation and Analysis of Data Structures**

We aim to implement and thoroughly analyze common data structures, including arrays, linked lists, stacks, queues, trees, and graphs. By working with these structures within the Java environment, we seek to gain an in-depth understanding of their characteristics, advantages, and trade-offs.

**2.5.2 Development and Evaluation of Algorithms**

Another central aspect of our project is the development and evaluation of algorithms. We will design and implement algorithms for various critical operations, such as sorting, searching, and traversal, leveraging the power of Java to create efficient solutions.

**2.5.3 Real-World Applications in Java**

We are committed to bridging theory and practice within the Java ecosystem. To this end, we will apply the knowledge and skills acquired from DSA to solve a specific problem that showcases the real-world relevance and impact of these principles in Java-based software development. This application will serve as a testament to the practical significance of DSA in the context of Java.

**2.6 Structure of the Report**

To provide a comprehensive understanding of our project's journey, this report is structured as follows:

* We commence with introduction about the significance of Java and small concepts of DSA, project goals and description followed by system architecture.
* Subsequently, we delve into methodology which explains the algorithms and data structures that are chosen for the project with provided pseudocode, elucidating the algorithms and data structures we have selected and explaining their implementations within the Java programming language.
* The report then showcases the outcomes of rigorous testing and presents an in-depth discussion of the implications of our findings. In conclusion, the key insights are summarized and the implications of your work and its potential impact are discussed.

Through this project, we aspire to make a meaningful contribution to the collective knowledge of DSA and underscore their practical significance in the realm of Java-based software development. We firmly believe that the knowledge and experience gained in this endeavor will not only enrich our academic pursuits but also have tangible applications in creating efficient, optimized, and innovative software solutions.

**3. PROJECT DESCRIPTION**

**3.1 Project Goals**

The system is designed to interact with users through a command-line interface, allowing them to perform the actions.

The Travel Management System is essentially a program that can be used for managing and booking flights for a group of users. It maintains a list of users, flights, and bookings, and provides user-friendly options to interact with the system. The project's ultimate goal is to provide a convenient way for users to manage their travel plans and for administrators to oversee the system's operation.

**3.2 Team Members**

The project was developed by a team consisting of K. Lakshmi Padmaja, M. Bhavya Sri, and P.Manasa.

**4.PROBLEM STATEMENT**

In today's fast-paced world, efficient travel management is crucial for individuals and organizations alike. Booking flights, keeping track of reservations, and managing user accounts can be complex tasks. To address these challenges, we aim to develop a comprehensive Travel Management System (TMS) that simplifies the process of booking flights, managing bookings, and maintaining user information. The Travel Management System should offer the following solutions:.

**User Management:** Create a user-friendly and secure platform for user registration, authentication, and profile management, with support for different user roles.

**Flight Management:** Enable administrators to add and manage flight details, and provide users with a convenient way to view available flights.

**Booking Management:** Offer users the ability to book, cancel, and view their flight reservations. Calculate and display the total charges for each booking.

**User Details Management:** Allow users and administrators to update user information, including usernames, passwords, group size, and associated flights.

**Exit Functionality:** Provide a smooth exit option for users when they are finished using the system.

**SOLUTION:**

* Automated System: Introduce a travel booking Application.
* Efficient Inventory Management: Implement tools for real-time inventory tracking and management.
* Accurate booking: Utilize the system to add user, checking the flights , make a booking , updating info, cancel booking and having a complete view on bookings.
* Enhanced Customer Experience: Provide a user-friendly interface for customers, enabling seamless browsing the available flights and effortless booking.
* Java Programming: Employ Java programming language for robust backend functionality.
* Object-Oriented Principles: Implement object-oriented principles for modular and maintainable codebase.
* Advanced Data Structures: Utilize advanced data structures for optimized data storage and retrieval.
* Improved Accuracy: Eliminate manual errors, leading to accurate account calculations.

**5**.**SOFTWARE REQUIREMENTS SPECIFICATIONS**

**System configurations :**

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behavior, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

**Software Requirements**:

1. **Operating System** : Windows 11
2. **Coding Language :** JAVA.
3. **IDE Used :** NetBeans
4. **TECHNOLOGIES:-**
5. **JAVA LANGUAGE:**
6. Java is a versatile, high-level programming language widely used in a variety of applications, Java is an object-oriented, class-based programming language renowned for its platform independence, reliability, and extensive libraries. Developed by Sun Microsystems (now owned by Oracle), Java has earned its popularity due to its "Write Once, Run Anywhere".
7. **NETBEANS(IDE):**
8. An Integrated Development Environment (IDE) in Java is a software application
9. **1.Code Editing:** Java IDEs offer advanced code editing features such as syntax highlighting, auto-completion, and code formatting to enhance productivity and code quality
10. **2. Project Management:** IDEs help developers organize their projects by providing project templates, source code version control integration, and build tools for managing dependencies.
11. **3. Debugging:** Debugging tools in an IDE allow developers to identify and fix issues in their Java code. Developers can set breakpoints, step through code, inspect variables, and view stack traces.
12. **4. Compiler and Build Tools:** Java IDEs often come with integrated compilers and build tools like Ant, Maven, or Gradle, streamlining the process of building and running Java application

**6.HARDWARE REQUIREMENTS**

1. Processor-Intel Core i3

2. RAM-8GB

**7. SYSTEM ARCHITECTURE**

**7.1 Software Components**

**TravelManagementSystem:** This class serves as the main entry point for the application. It manages users, flights, bookings, and provides methods for booking management.

**User**: Represents a user of the system, storing username, password, and booking-related information.

**Flight**: Represents flight information, including flight number, origin, destination, available seats, and price.

**Booking:** Represents a booking made by a user, including a unique booking ID, user, flight, and the number of people booked.

**MakeBooking:**  Manages the booking process, allowing users to select available flights and make bookings.

**CancelBooking:** Allows users to cancel existing bookings.

**UpdateUserDetails**: Allows users to update their details, including username, password, number of members booked, and the flight they are booked on.

**ViewAvailableFlights:** Provides functionality to view all available flights.

**ViewAllBookings:** Allows users and administrators to view all bookings in the system.

**ViewBookingsForUser**: Permits users and administrators to view bookings for a specific user.

**ViewUsers:** Allows administrators to view all registered users.

**Exit:** Provides an exit mechanism for the application.

**8. IMPLEMENTATION**

**8.1 Main Class (travelmanagementsystem.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : travelmanagementsystem

TRAVEL MANAGEMENT SYSTEM (MAIN CLASS):

package travel;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Scanner;

public class TravelManagementSystem {

Map<String, User> users;

private List<Flight> flights; // Changed to Flight

private List<Booking> bookings;

private int bookingIdCounter;

private String currentUsername;

public TravelManagementSystem() {

users = new HashMap<>();

flights = new ArrayList<>(); // Changed to Flight

bookings = new ArrayList<>();

bookingIdCounter = 1; // Initialize the booking ID counter

currentUsername = null;

}

// User management methods...

public void addUser(User user) {

users.put(user.getUsername(), user);

}

public User getUserByUsername(String username) {

return users.get(username);

}

// Flight management methods...

public void addFlight(Flight flight) { // Changed to Flight

flights.add(flight);

}

public List<Flight> getAllFlights() { // Changed to Flight

return flights;

}

// Booking methods...

public void makeBooking(User user, Flight flight, int numOfPeople) { // Changed to Flight

if (flight.getAvailableSeats() >= numOfPeople) {

Booking booking = new Booking(bookingIdCounter++, user, flight, numOfPeople); // Changed to Flight

bookings.add(booking);

flight.setAvailableSeats(flight.getAvailableSeats() - numOfPeople);

System.out.println("Booking successful! Your Booking ID is: " + booking.getBookingId());

} else {

System.out.println("Not enough seats available for booking.");

}

}

public void cancelBooking(int bookingId) {

Booking bookingToRemove = null;

for (Booking booking : bookings) {

if (booking.getBookingId() == bookingId) {

bookingToRemove = booking;

booking.getFlight().setAvailableSeats(booking.getFlight().getAvailableSeats() + booking.getNumOfPeople());

break;

}

}

if (bookingToRemove != null) {

bookings.remove(bookingToRemove);

System.out.println("Booking with ID " + bookingId + " has been canceled.");

} else {

System.out.println("Booking not found with ID " + bookingId + ". No booking canceled.");

}

}

public List<Booking> getBookingsByUsername(String username) {

List<Booking> userBookings = new ArrayList<>();

for (Booking booking : bookings) {

if (booking.getUser().getUsername().equals(username)) {

userBookings.add(booking);

}

}

return userBookings;

}

public List<Booking> getAllBookings() {

return bookings;

}

// Helper method to check if admin is logged in

public boolean isAdminLoggedIn() {

User currentUser = getUserByUsername(currentUsername);

return currentUser != null && currentUser.isAdmin();

}

// Helper method to get Flights by Flight Number

public Flight getFlightByNumber(String flightNumber) { // Changed to Flight

for (Flight flight : flights) {

if (flight.getFlightNumber().equalsIgnoreCase(flightNumber)) {

return flight;

}

}

return null;

}

public void updateFlightDetails(String flightNumber, String newOrigin, String newDestination, int newAvailableSeats, double newPrice) {

Flight flightToUpdate = getFlightByNumber(flightNumber);

if (flightToUpdate != null) {

flightToUpdate.setOrigin(newOrigin);

flightToUpdate.setDestination(newDestination);

flightToUpdate.setAvailableSeats(newAvailableSeats);

flightToUpdate.setPrice(newPrice);

System.out.println("Flight details updated successfully.");

} else {

System.out.println("Flight not found. Flight details not updated.");

}

}

public static void main(String[] args) {

TravelManagementSystem tms = new TravelManagementSystem();

Scanner scanner = new Scanner(System.in);

// Create and add users

User user1 = new User("user1", "password1");

User user2 = new User("user2", "password2");

//User admin = new User("admin", "adminpass");

tms.addUser(user1);

tms.addUser(user2);

//tms.addUser(admin);

Flight flight1 = new Flight("F101", "New York", "Los Angeles", 100, 250.0);

Flight flight2 = new Flight("F102", "London", "Paris", 80, 150.0);

Flight flight3 = new Flight("F103", "San Francisco", "Chicago", 120, 200.0);

Flight flight4 = new Flight("F104", "Los Angeles", "New York", 95, 220.0);

Flight flight5 = new Flight("F105", "Paris", "London", 75, 160.0);

Flight flight6 = new Flight("F106", "Chicago", "San Francisco", 110, 190.0);

Flight flight7 = new Flight("F107", "Tokyo", "Seoul", 150, 300.0);

Flight flight8 = new Flight("F108", "Seoul", "Beijing", 140, 280.0);

Flight flight9 = new Flight("F109", "Beijing", "Shanghai", 90, 180.0);

Flight flight10 = new Flight("F110", "Shanghai", "Tokyo", 120, 240.0);

tms.addFlight(flight1);

tms.addFlight(flight2);

tms.addFlight(flight3);

tms.addFlight(flight4);

tms.addFlight(flight5);

tms.addFlight(flight6);

tms.addFlight(flight7);

tms.addFlight(flight8);

tms.addFlight(flight9);

tms.addFlight(flight10);

// Interaction with users

boolean exit = false;

while (!exit) {

System.out.println("1. Add user");

System.out.println("2. View available flights");

System.out.println("3. Make a booking");

System.out.println("4. View all bookings");

System.out.println("5. View bookings for a specific user");

System.out.println("6. Update Username");

System.out.println("7. Cancel Booking");

System.out.println("8. View Users");

System.out.println("9. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

// Option 1: Add a new user

addUser(tms, scanner);

break;

case 2:

// Option 2: View available Flights

ViewAvailableFlights.show(tms.getAllFlights());

break;

case 3:

// Option 3: Make a booking

MakeBooking.make(tms, scanner);

break;

case 4:

// Option 4: View all bookings

ViewAllBookings.show(tms.getAllBookings());

break;

case 5:

// Option 5: View bookings for a specific user

ViewBookingsForUser.show(tms, scanner);

break;

case 6:

// Option 6: Update User Details

UpdateUserDetails userDetailsUpdater = new UpdateUserDetails(tms);

System.out.print("Enter the username to update details: ");

String usernameToUpdate = scanner.next();

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

String newUsername = scanner.next();

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

String newPassword = scanner.next();

System.out.print("Enter the new number of members booked: ");

int newNumOfMembers = scanner.nextInt();

System.out.print("Enter the new flight number: ");

String newFlightNumber = scanner.next();

userDetailsUpdater.updateUserDetails(usernameToUpdate, newUsername, newPassword, newNumOfMembers, newFlightNumber);

break;

case 7:

// Option 7: Cancel Booking

CancelBooking.cancel(tms, scanner);

break;

case 8:

// Option 8: View Users

ViewUsers.show(tms);

break;

case 9:

// Option 9: Exit

Exit.exit();

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

// Method to add a new user

private static void addUser(TravelManagementSystem tms, Scanner scanner) {

System.out.print("Enter a username for the new user: ");

String newUsername = scanner.next();

System.out.print("Enter a password for the new user: ");

String newPassword = scanner.next();

// System.out.print("Is the new user an admin? (true/false): ");

// boolean newIsAdmin = scanner.nextBoolean();

// Create and add the new user

User newUser = new User(newUsername, newPassword);

tms.addUser(newUser);

System.out.println("New user added successfully.");

}

public double calculateTotalCharges(String flightNumber, int numOfPeople) {

Flight flight = getFlightByNumber(flightNumber);

if (flight != null) {

double pricePerPerson = flight.getPrice();

double totalCharges = pricePerPerson \* numOfPeople;

return totalCharges;

} else {

System.out.println("Flight not found. Cannot calculate total charges.");

return -1.0; // Return a negative value to indicate an error

}

}

}

**8.2 Booking Class (booking.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : booking

package travel;

public class Booking {

private int bookingId;

private User user;

private Flight flight;

private int numOfPeople;

public Booking(int bookingId, User user, Flight flight, int numOfPeople) {

this.bookingId = bookingId;

this.user = user;

this.flight = flight;

this.numOfPeople = numOfPeople;

}

// Getters and setters

public int getBookingId() {

return bookingId;

}

public void setBookingId(int bookingId) {

this.bookingId = bookingId;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Flight getFlight() {

return flight;

}

public void setFlight(Flight flight) {

this.flight = flight;

}

public int getNumOfPeople() {

return numOfPeople;

}

public void setNumOfPeople(int numOfPeople) {

this.numOfPeople = numOfPeople;

}

}

**8.3 Cancel booking class(cancelbooking.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : cancelbooking

package travel;

//package mainproject;

import java.util.Scanner;

public class CancelBooking {

public static void cancel(TravelManagementSystem tms, Scanner scanner) {

System.out.print("Enter the booking ID to cancel: ");

int bookingId = scanner.nextInt();

tms.cancelBooking(bookingId);

}

}

**8.4 Exit class(exit.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : exit

package travel;

public class Exit {

public static void exit() {

System.out.println("Exiting the Travel Management System. Goodbye!");

}

}

**8.5 Flight class(flight.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : flight

package travel;

//package mainproject;

public class Flight {

private String flightNumber;

private String origin;

private String destination;

private int availableSeats;

private double price;

public Flight(String flightNumber, String origin, String destination, int availableSeats, double price) {

this.flightNumber = flightNumber;

this.origin = origin;

this.destination = destination;

this.availableSeats = availableSeats;

this.price = price;

}

// Getters and setters

public String getFlightNumber() {

return flightNumber;

}

public void setFlightNumber(String flightNumber) {

this.flightNumber = flightNumber;

}

public String getOrigin() {

return origin;

}

public void setOrigin(String origin) {

this.origin = origin;

}

public String getDestination() {

return destination;

}

public void setDestination(String destination) {

this.destination = destination;

}

public int getAvailableSeats() {

return availableSeats;

}

public void setAvailableSeats(int availableSeats) {

this.availableSeats = availableSeats;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**8.6 Make booking class(makebooking.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : makebooking

package travel;

import java.util.List;

import java.util.Scanner;

public class MakeBooking {

public static void make(TravelManagementSystem tms, Scanner scanner) {

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

String username = scanner.next();

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

String password = scanner.next();

User user = tms.getUserByUsername(username);

if (user != null && user.getPassword().equals(password)) {

System.out.println("Available Flights for Booking:");

List<Flight> availableFlights = tms.getAllFlights();

for (Flight flight : availableFlights) {

System.out.println(flight.getFlightNumber() + " - " + flight.getOrigin() + " to " + flight.getDestination()

+ " (Price: $" + flight.getPrice() + ", Available Seats: " + flight.getAvailableSeats() + ")");

}

System.out.print("Enter the flight number for booking: ");

String flightNumber = scanner.next();

System.out.print("Enter the number of people for booking: ");

int numOfPeople = scanner.nextInt();

Flight flight = tms.getFlightByNumber(flightNumber);

if (flight != null) {

tms.makeBooking(user, flight, numOfPeople);

// Calculate and display the total charges

double totalCharges = tms.calculateTotalCharges(flightNumber, numOfPeople);

if (totalCharges >= 0) {

System.out.println("Total Charges: $" + totalCharges);

} else {

System.out.println("Unable to calculate total charges.");

}

} else {

System.out.println("Invalid flight number. Booking failed.");

}

} else {

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

}

}

}

**8.7 Update user details class(updateuserdetails.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : updateuserdetails

package travel;

public class UpdateUserDetails {

private final TravelManagementSystem tms;

public UpdateUserDetails(TravelManagementSystem tms) {

this.tms = tms;

}

public void updateUserDetails(String usernameToUpdate, String newUsername, String newPassword, int newNumOfMembers, String newFlightNumber) {

User userToUpdate = tms.getUserByUsername(usernameToUpdate);

if (userToUpdate != null) {

// Update username and password

userToUpdate.setUsername(newUsername);

userToUpdate.setPassword(newPassword);

// Update the number of members booked

userToUpdate.setNumOfMembersBooked(newNumOfMembers);

// Update the flight

Flight newFlight = tms.getFlightByNumber(newFlightNumber);

if (newFlight != null) {

userToUpdate.setFlight(newFlight);

System.out.println("User details updated successfully.");

} else {

System.out.println("Flight not found. User details not updated.");

}

} else {

System.out.println("User not found. User details not updated.");

}

}

}

**8.8 User class(user.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : user

package travel;

public class User {

private String username;

private String password;

private int numOfMembersBooked;

private Flight flight;

public User(String username, String password) {

this.username = username;

this.password = password;

this.numOfMembersBooked = 0;

this.flight = null;

}

// Getter methods for username and password

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

// Setter methods for username and password

public void setUsername(String newUsername) {

this.username = newUsername;

}

public void setPassword(String newPassword) {

this.password = newPassword;

}

// Getter and setter methods for numOfMembersBooked

public int getNumOfMembersBooked() {

return numOfMembersBooked;

}

public void setNumOfMembersBooked(int numOfMembersBooked) {

this.numOfMembersBooked = numOfMembersBooked;

}

// Getter and setter methods for the flight

public Flight getFlight() {

return flight;

}

public void setFlight(Flight flight) {

this.flight = flight;

}

// Other methods and properties as needed

public boolean isAdmin() {

return false; // Implement your admin logic here

}

}

**8.9 View all bookings(viewallbookings.java)**

CODE:

PACKAGE NAME : travel

FILE NAME :viewallbookings

package travel;

import java.util.Scanner;

import java.util.List;

public class ViewAllBookings {

public static void show(List<Booking> bookings) {

if (bookings.isEmpty()) {

System.out.println("No bookings available.");

} else {

System.out.println("List of all bookings:");

for (Booking booking : bookings) {

System.out.println(booking);

}

}

}

public static void show(TravelManagementSystem tms, Scanner scanner) {

System.out.print("Enter the username to view bookings: ");

String username = scanner.next();

List<Booking> userBookings = tms.getBookingsByUsername(username);

if (userBookings.isEmpty()) {

System.out.println("No bookings found for user: " + username);

} else {

System.out.println("Bookings for user " + username + ":");

for (Booking booking : userBookings) {

System.out.println(booking);

}

}

}

}

**8.10 View available flights(viewavailableflights.java)**

CODE:

PACKAGE NAME : travel

FILE NAME :viewavailableflights

package travel;

import java.util.List;

public class ViewAvailableFlights {

public static void show(List<Flight> flights) {

System.out.println("Available Flights:");

for (Flight flight : flights) {

System.out.println(flight.getFlightNumber() + " - " + flight.getOrigin() + " to " + flight.getDestination()

+ " (Price: $" + flight.getPrice() + ", Available Seats: " + flight.getAvailableSeats() + ")");

}

}

}

**8.11 View bookings for user(viewbookingsforuser.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : **viewbookingsforuser**

package travel;

import java.util.List;

import java.util.Scanner;

public class ViewBookingsForUser {

public static void show(TravelManagementSystem tms, Scanner scanner) {

System.out.print("Enter the username to view bookings: ");

String viewUsername = scanner.next();

List<Booking> userBookings = tms.getBookingsByUsername(viewUsername);

if (!userBookings.isEmpty()) {

System.out.println("Bookings for User " + viewUsername + ":");

for (Booking booking : userBookings) {

System.out.println("Booking ID: " + booking.getBookingId());

System.out.println("Flight: " + booking.getFlight().getFlightNumber() + " - " +

booking.getFlight().getOrigin() + " to " + booking.getFlight().getDestination());

System.out.println("Num of People: " + booking.getNumOfPeople());

// Calculate and display the total charges for the booking

double totalCharges = tms.calculateTotalCharges(booking.getFlight().getFlightNumber(), booking.getNumOfPeople());

if (totalCharges >= 0) {

System.out.println("Total Charges: $" + totalCharges);

} else {

System.out.println("Unable to calculate total charges.");

}

System.out.println("--------------");

}

} else {

System.out.println("No bookings found for User " + viewUsername + ".");

}

}

}

**8.12 View users(viewusers.java)**

CODE:

PACKAGE NAME : travel

FILE NAME : **viewusers**

package travel;

public class ViewUsers {

public static void show(TravelManagementSystem tms) {

System.out.println("All Users:");

for (User user : tms.users.values()) {

System.out.println("Username: " + user.getUsername());

System.out.println("Admin: " + user.isAdmin());

System.out.println("--------------");

}

}

}

**Concepts used in the code:**

The code you provided for the Travel Management System (TMS) demonstrates the application of several Data Structures and Algorithms (DSA) concepts, along with Java programming principles. Here are some of the key DSA and Java concepts used in the code:

**Object-Oriented Programming (OOP):**

The code is organized into classes that represent real-world entities, such as User, Flight, and Booking.

Inheritance and encapsulation are used to model relationships between these classes, such as the User class having a reference to a Flight object.

Methods and attributes within classes facilitate modular and maintainable code.

**Collections Framework:**

The code uses Java's Collections Framework, specifically ArrayList and HashMap.

ArrayList is used to store lists of flights and bookings, allowing dynamic storage and retrieval of objects.

HashMap is used to store users, allowing efficient retrieval based on username.

**User Authentication and Authorization:**

The concept of user roles and authorization is implemented. An isAdmin method is used to determine if a user has administrator privileges.

This concept is critical for controlling access to certain features within the system.

**Searching and Retrieving Data:**

The code employs search and retrieval algorithms to find flights and users efficiently.

Methods like getFlightByNumber and getUserByUsername iterate through lists and maps to find objects.

**Data Validation and Error Handling:**

The code validates user inputs and provides error messages when invalid data is provided.

Error handling is crucial for maintaining the integrity of the system.

**Iterative Algorithms:**

Iterative loops, such as for loops, are used to traverse lists and perform tasks like canceling bookings and calculating total charges.

The code demonstrates how to iterate through data structures to find and manipulate data.

**Interactive User Interface:**

The code uses Java's input/output mechanisms to create an interactive console-based user interface.

It provides menus and prompts for user input, allowing users and administrators to interact with the system.

**Dynamic Memory Management:**

ArrayList dynamically manages memory for storing flight and booking objects.

The system can expand and shrink the data structures as needed.

**Code Modularity:**

The code is organized into classes and methods, enhancing modularity and reusability.

This is a fundamental OOP principle, which aids in code maintainability.

**Exception Handling:**

The code incorporates exception handling to deal with exceptional situations. For example, when a flight is not found, it returns a special value to indicate an error.

This ensures that the application gracefully handles unexpected situations.

**Menu-Driven Program:**

The system's menu-driven interface is a common way to interact with users, where they select options by entering numbers. This pattern simplifies user interactions.

**Algorithms for Business Logic:**

The code implements logic for booking management, such as checking seat availability and calculating total charges.

These algorithms are at the core of the system's functionality.

**Exit Mechanism:**

The code properly manages the application's lifecycle, providing a graceful exit mechanism through the Exit class.

The Travel Management System code demonstrates the practical application of these DSA and Java concepts, providing a structured and functional solution for managing flight bookings and user information.

**9.FUNCTIONALITY AND USAGE**

**Features and Functionality**

The Travel Management System offers the following features:

**1.User Management:**

* Adding Users: Administrators can add users to the system, providing them with unique usernames and passwords.
* User Retrieval: The system allows for the retrieval of user details based on their username.
* User Updates: Users can update their details, including their username, password, and the number of members booked.

**2.Flight Management:**

* Adding Flights: Administrators can add flight information to the system, specifying the flight number, origin, destination, available seats, and price.
* Flight Retrieval: The system provides a list of all available flights.

**3.Booking Management:**

* Booking Creation: Users can make bookings for specific flights, provided there are enough available seats.
* Booking Cancellation: Users can cancel their bookings, which will increase the available seats for that flight.
* Total Charges Calculation: The system calculates and displays the total charges for a booking based on the flight's price and the number of people booked.
* Viewing Bookings: Both users and administrators can view bookings. Users can view their own bookings, while administrators can view all bookings in the system.

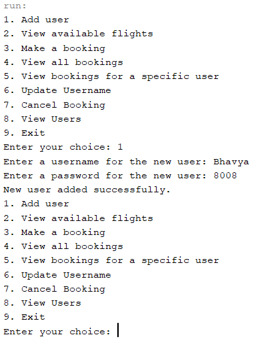
**4.User Interface:**

* The Travel Management System provides an interactive console-based interface for users and administrators to perform actions.
* Users are guided through the process of making bookings and updating their details with clear prompts and instructions.

**Usage**

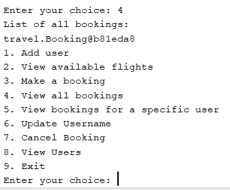
To run the Travel Management System, you can execute the TravelManagementSystem class. It will start an interactive menu-driven interface where you can perform various actions such as adding users, managing flights, making bookings, and more.

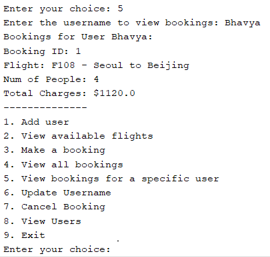
**10.OUTPUT FORMAT**

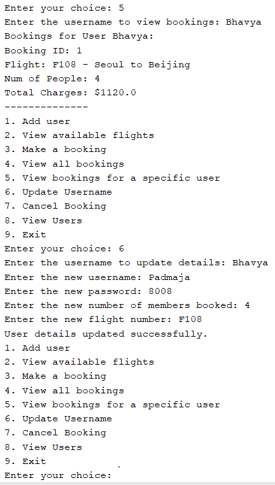


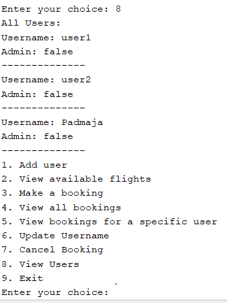


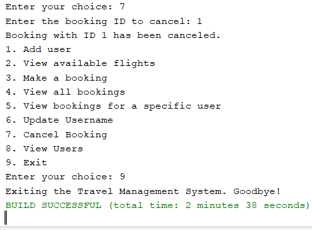












**11.CONCLUSION**

In conclusion, the Online Travel Management System project is a valuable development that caters to the increasing demand for secure and convenient online access to travel services. This project allows users add users, to check available flights, make a booking, can see a particular user details, all bookings, can update username, can cancel booking, view users and can exit.

Looking ahead, several opportunities for future improvements and enhancements are worth considering:

User Authentication and Security, Admin Features, Data Validation, User Profiles, Payment Integration, Email Notifications, Search and Filters, Flight Scheduling and Availability, Mobile Application, Reporting and Analytics, Localization, Feedback and Reviews, Integration with External APIs, Testing and Documentation, Scalability and Performance, User Support and Help Center.

Incorporating these future improvements will not only keep the Online Travel Management System competitive but also enhance the user experience and overall utility. It reflects the dynamic nature of online travelling systems, which need to adapt and evolve to meet the ever-changing demands of users in the digital era.

**REFERENCES:**

<https://www.javatpoint.com/java-tutorial>

<https://www.geeksforgeeks.org/data-structures/>