# SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

**(An Autonomous Institution)**

**(Approved by AICTE and Affiliated to Anna University,**

**Chennai) ACCREDITED BY NAAC WITH "A" GRADE**

## BONAFIDE CERTIFICATE

Certified that this mini project report titled "**FLIGHT MANAGEMENT SYSTEM"** is the Bonafede work of **"NIRANJANA GURUMALLESH M (727721EUCS092)**, **MOHAMMED IFRAN S(727721EUCS076), NITHISH KUMAR P(727721EUCS095)"** who carried out the mini project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **Ms. S. BIRUNTHA** | **Dr. K. SASI KALA RANI** |
| **SUPERVISOR,** | **HEAD OF THE** |
| **ASSISTANT PROFESSOR** | **DEPARTMENT** |

**Department of Computer Science and Engineering**

**Sri Krishna College of Engineering and Technology Kuniamuthur,**

**Coimbatore.**

**This Mini Project report is submitted for Autonomous Mini Project Viva-Voce examination held on ……...**

| | |
|---|---|
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

## ACKNOWLEDGEMENT

We express our sincere thanks to the management and **Dr. J. JANET M.E., Ph.D.,** Principal, Sri Krishna College of Engineering and Technology, Coimbatore for providing us the facilities to carry out this mini project work.

We are thankful to **Dr. K. SASI KALA RANI M.E., Ph.D**., Professor and Head, Department of Computer Science and Engineering, for her continuous evaluation and comments given during the course of the mini project work.

We express our deep sense of gratitude to our supervisor and **Ms. S. BIRUNTHA (Ph.D.),** Assistant Professor, Department of Computer science and Engineering for her valuable advice, guidance and support during the course of our mini project work.

We would also like to thank our mini project coordinators **Ms. S. BIRUNTHA (Ph.D.).,** Assistant Professor, Department of Computer Science and Engineering for helping us in completing our mini project work

## ABSTRACT

The Flight Management System (FMS) represents a technological milestone with the potential to redefine modern aviation practices. This ambitious project is driven by the goal of creating a versatile and efficient system that can meet the diverse needs of the aviation industry. It encompasses an array of critical features, including precise navigation, route optimization, real-time weather integration, and automated flight control.

At its core, the primary objective of this project is to elevate aircraft navigation accuracy and operational efficiency. By automating complex navigation tasks, the FMS enhances flight safety, ensuring precise altitude and speed control while seamlessly integrating real-time weather data. The system's capacity to optimize flight routes and reduce fuel consumption leads to substantial cost savings, even though initial investments and pilot training may pose challenges.

In conclusion, the Flight Management System project aspires to revolutionize aviation practices by offering a comprehensive solution that addresses key aspects of navigation, safety, and efficiency. Its potential to minimize environmental impact, improve punctuality, and enhance passenger experiences positions it as a strategic imperative for airlines in navigating the dynamic landscape of contemporary aviation. Embracing the FMS represents a transformative step towards a safer, more efficient, and sustainable future in the aviation industry.

**TABLE OF CONTENTS**

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| ABBREVIATION | FULL FORM |
|---|---|
| DB | DATABASE |
| JDBC | JAVA DATABASE CONNECTIVITY |
| CSR | CUSTOMER SERVICE REPRESENTATIVE |
| RO | REPAIR ORDER |
| CRM | CUSTOMER RELATIONSHIP MANAGEMENT |

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW OF FLIGHT MANAGEMENT SYSTEM

The Flight Management System (FMS) represents a pivotal advancement in aviation technology, underpinning the modern air travel experience. It is a multifaceted and hardware solution designed to optimize aircraft navigation, flight planning, and control sys, thereby ensuring the efficiency, safety, and precision of each flight. One of the most crucial aspects of the FMS is its role in flight planning and route optimization. Leveraging extensa databases of aviation data, including airways, waypoints, and real-time weather information system calculates the most optimal flight path for each journey. This not only reduces flight time but also minimizes fuel consumption, contributing to cost savings for airlines and a reduced environmental footprint. The FMS is a cornerstone of navigation accuracy. By continually analysing the aircraft's position and comparing it to planned route, the system provides real-time guidance to pilots. It calculates corrections, altitude adjustments, and variations as needed, ensuring that the aircraft remains precisely on course. Furthermore, the integration of the FMS with autothrottle and autopilot systems automates critical aspects of control. This reduces pilot workload and enhances flight stability, especially during long-haul journeys. In addition to real-time navigation, the FMS continuously monitors aircraft performance. It tracks vital parameters like engine performance, fuel consumption, and weight distribution, making adjustments to maintain optimal efficiency and safety throughout the flight. The significance of the Flight Management System is evident in its wide-ranging benefits. It enhances flight safety by reducing the potential for human errors in navigation and control, improves operational efficiency by optimizing routes and fuel consumption, and contributes to more enjoyable travel experience for passengers through smoother flights and -time arrivals

## 1.2 EXPLANATION OF CAR WORKSHOP MANAGEMENT SYSTEM

A Flight Management System (FMS) represents an advanced and integral software solution at the forefront of contemporary aviation. This system serves as the digital nucleus of aircraft, unifying a spectrum of critical functions that are paramount for ensuring safe, efficient, and precise air travel.

The aviation landscape is complex, involving intricate flight planning, dynamic navigation, real-time weather monitoring, and the need for precise control during flight. The FMS excels in simplifying and optimizing these multifaceted operations, making it indispensable for modern aviation.

Key components integral to an FMS typically encompass route planning, navigation, autothrottle and autopilot integration, performance monitoring, and real-time data processing. In addition to these fundamental functions, modern FMS systems are often equipped with advanced data analytics and predictive maintenance capabilities. These features empower pilots, airline operators, and maintenance crews to make informed decisions, enhance aircraft performance, and ensure passenger safety and satisfaction.

At its core, the FMS is responsible for creating efficient flight paths, optimizing fuel consumption, and reducing environmental impact

# CHAPTER-2

# LITERATURE STUDY

## 2.1 FUNCTIONS AND STRENGTH OF THE SYSTEM

The Flight Management System (FMS) has revolutionized aviation operations, marking a stark departure from traditional manual flight processes. In the manual paradigm, flight planning involves labour-intensive calculations, paper charts, and maps, which are both time-consuming and susceptible to human error. Navigational tasks are equally manual, relying on visual references and instruments. Route adjustments during flight necessitate radio communication with air traffic control and manual calculations, potentially leading to inefficiencies. Fuel management is error-prone, with pilots manually monitoring consumption and making adjustments. Communication relies on voice exchanges, which can be prone to misinterpretation and delays.

In contrast, the FMS automates and optimizes virtually every aspect of flight operations. It calculates optimal flight routes based on real-time data, including weather conditions and air traffic, reducing planning time and enhancing accuracy. Precise navigation guidance is achieved through continuous updates of the aircraft's position, altitude, and heading, utilizing GPS and sensors for unparalleled reliability. Real-time route adjustments are calculated and executed seamlessly, optimizing fuel efficiency and avoiding adverse conditions. Fuel management becomes automated, calculating consumption and remaining fuel, optimizing throttle settings for efficiency. Communication with air traffic control is digitized, reducing the potential for miscommunication and delays. The FMS, therefore, stands as a transformative technology, elevating efficiency, safety, and precision in modern aviation by significantly reducing manual reliance and mitigating the potential

for human errors

## 2.3 STUDY ABOUT FLIGHT MANAGEMENT SYSTEM

A comprehensive study on Flight Management Systems (FMS) in modern aviation reveals a robust and evolving body of research and development within this critical domain. Aviation, as a dynamic and safety-driven industry, heavily relies on technological advancements to enhance operational efficiency and passenger safety. The study's primary focus is to investigate the multifaceted aspects of FMS, ranging from its historical evolution to its current functions, challenges, and future trends.

Moreover, this comprehensive examination highlights the pivotal role of FMS in mitigating some of the most pressing challenges faced in the aviation industry. FMS systems reduce the probability of human errors, significantly enhance navigation accuracy, and introduce automation into critical flight operations. Safety records, statistics, and real world examples underscore the positive impact of FMS on aviation safety.

Efficiency and cost savings are another noteworthy focus in the literature. The study
emphasizes how FMS optimizes flight routes, minimizes fuel consumption, and reduces operational costs for airlines. Cost-benefit analyses and empirical data reveal the substantial economic benefits associated with the adoption of FMS technology.

Furthermore, the study casts a forward-looking lens on the future of Flight Management Systems. Emerging technologies such as artificial intelligence and data analytics are identified as potential game-changers in shaping the capabilities and scope of FMS. Additionally, the increasing emphasis on sustainability and environmental concerns within the aviation sector aligns with the FMS's role in promoting fuel efficiency and reducing carbon emissions.

# CHAPTER - 3

# REQUIREMENTS

## 3.1 FUNCTIONAL REQUIREMENTS

Functional requirements for a Flight Management System (FMS) are essential specifications that define the system's capabilities and functionalities, ensuring safe and efficient aircraft operations. These requirements serve as the foundation for designing and developing an FMS that meets the rigorous demands of modern aviation.

First the FMS must excel in flight planning and route optimization. It should enable flight crews to input departure and destination airports, calculate optimal flight routes considering dynamic factors such as weather conditions, air traffic, and fuel efficiency, and continuously adapt these routes in real-time.

In addition to route planning, the FMS must provide precise navigation guidance to the flight crew, including recommendations for heading, altitude, and speed. Integration with the aircraft's autopilot and autothrottle systems is crucial for automating flight control when necessary, enhancing flight stability and precision. Importantly, the system should include fail-safe mechanisms to revert to manual control in case of system failures or emergency situations. Efficient performance monitoring and optimization are paramount.

The system should continuously monitor engine efficiency, fuel consumption, and aircraft weight, making real-time adjustments to optimize engine thrust and fuel consumption for maximum efficiency, contributing to cost savings and environmental sustainability. Furthermore, the FMS must seamlessly integrate real-time weather data, utilizing it to make informed decisions about routing, altitude adjustments, and weather avoidance. Providing weather-related advisories and warnings to the flight crew is crucial for safe flight operations.

Flexibility in route adjustments is essential. The system should allow flight crews to input route modifications, such as changes to waypoints or altitude, based on air traffic control instructions or unforeseen circumstances. It should calculate and implement these adjustments efficiently to ensure smooth flight progression. Effective communication with Air Traffic Control (ATC) is vital. The FMS should facilitate data exchange with ATC,

enabling the transmission of flight plans, sending position reports, and receiving instructions from ATC. The FMS should support auto-sequencing of waypoints and procedures, streamlining operational processes. A user-friendly interface is key to efficient operation. The FMS should provide an intuitive and easily navigable interface, allowing flight crews to input data, make adjustments, and access critical information with ease. Safety and reliability are paramount. The system should incorporate robust fail-safe mechanisms and redundancy to ensure system reliability in all scenarios.

## 3.2 NON-FUNCTIONAL REQUIREMENTS

The Flight Management System's non-functional requirements are essential for ensuring the system's reliability, performance, and usability within the beauty and wellness industry. These non-functional requirements include, the system must efficiently handle a large volume of concurrent users and data, responding promptly to user requests, and ensuring a smooth user experience. The system should easily adapt to accommodate the growth of salon and spa operations, increasing customer demands, and expanding service offerings.

Compatibility across various devices and browsers enhances accessibility for both customers and employees. Regulatory compliance is vital to ensure adherence to automotive industry standards and relevant legal requirements. Integration capabilities enable seamless connectivity with external systems, such as accounting software and parts suppliers, streamlining workflows. Performance metrics and monitoring tools should be in place to assess system health and response times, while cost-effectiveness should be considered within budget constraints.

Ensuring accessibility for users with disabilities through compliance with accessibility standards is essential. Finally, maintainability is key, allowing for easy updates and adaptations as the business evolves, ensuring the system remains a valuable asset to the car workshop's operations. In summary, these non-functional requirements collectively contribute to a robust, secure, and efficient car workshop management system, enhancing the overall effectiveness and reliability of automotive repair and service operations.

### 3.3 HARDWARE REQUIREMENTS

- Processor Type : Intel Core i5
- Speed : 3.40 GHz
- RAM : 4 GB
- Hard Disk : 500 GB
- Keyboard : 101/102 Standard Keys
- Mouse : Optical Mouse

### 3.4 SOFTWARE REQUIREMENTS

- Operating System : Windows 10 / Windows 11
- Coding Language : Java
- Database : MySQL

# MODULE-4

# SYSTEM ANALYSIS

## 4.1 PROBLEM STATEMENT

Design and implement a flight management software system that enables an airline to efficiently manage its flight schedules, seat availability, and booking confirmations. The system should provide airline staff with the tools to input and update flight schedules, monitor seat availability in real-time, and confirm bookings for passengers. It should also allow passengers to search for available flights, select seats, and make reservations. The system should maintain accurate and up-to-date flight information, handle concurrent user interactions, and ensure data security and integrity.

## 4.2 EXISTING APPROACH

The current approach to managing flight operations in the airline industry relies predominantly on traditional methods and exhibits limited digital integration. Flight schedules are typically managed using legacy software and manual processes, potentially resulting in scheduling inefficiencies and challenges in adapting to changing operational demands. Seat availability is tracked through reservation systems, but real-time updates may be lacking, leading to discrepancies in displayed seat availability for passengers.

Booking confirmations are issued through these systems, yet passengers may encounter delays or uncertainties in receiving their confirmations. Reservation systems, while essential, may have limitations in terms of real-time updates and flexibility, affecting the ability to efficiently

Payment processing is securely managed, but discrepancies in payment gateways and processes across airlines can result in inconsistent booking experiences. Moreover, data

management encompasses passenger information, flight schedules, and bookings, with potential challenges in data integration and synchronization between systems.

Moreover, reservation systems should evolve to offer flexibility and adaptability in a dynamic travel landscapes. Prioritizing transparent and personalized customer communication, streamlining payment processing, and improving data integration are imperative for airlines to remain competitive and meet the evolving demands of today's travellers.

In summary, the current approach highlights the need for modernization to meet the expectations of today's travellers and enhance operational efficiency in the airline industry.

## 4.3 DISADVANTAGE OF EXISTING APPROACH

The existing approach to flight management in the airline industry is marked by several significant disadvantages that hinder its efficiency and effectiveness. Firstly, reliance on manual processes and outdated software for flight scheduling, seat management, and booking confirmations poses a substantial risk of errors, delays, and miscommunication

Customer relationship management (CRM) represents another significant drawback. The absence of a robust CRM system hampers airlines' capacity to maintain a loyal customer base, personalize services, gather valuable customer feedback, and offer tailored promotions. Ultimately, this deficiency impacts customer retention and profitability.

In summary, the existing approach to flight management in the airline industry is characterized by inefficiencies, data management challenges, customer relationship issues, and sustainability deficiencies.

# MODULE-5

# SYSTEM DESIGN

## 5.1 INPUT DESIGN

Effective input design is essential for a user-friendly and efficient Flight Management System (FMS) used by airline staff to manage flight schedules, seat availability, and booking confirmations. It should prioritize user needs, offer an intuitive interface with role-based access, incorporate input validation for data accuracy, provide real-time updates, and support features like date pickers, dropdowns, and confirmation prompts. Accessibility, responsiveness across devices, and multi-language support are critical considerations, along with usability testing to ensure alignment with user workflows. Such design principles enhance system usability, reduce errors, and streamline operations for airline personnel, ultimately improving the overall functionality and user satisfaction of the FMS.

Additionally, an effective input design for the Flight Management System (FMS) should consider the dynamic nature of airline operations, offering flexibility to adapt to changing circumstances and last-minute changes efficiently. It should prioritize the clear presentation of information, enable seamless communication, and facilitate streamlined data entry. Progressive enhancement, with features like keyboard shortcuts for power users, can further enhance user efficiency. By adhering to these principles, the FMS can deliver a comprehensive and user centric input experience, ultimately enhancing the productivity and satisfaction of airline staff.

Incorporating advanced input features, such as intelligent autocomplete and predictive text, can further improve the input design by reducing the effort required for data entry and minimizing errors. Additionally, designing a responsive interface that seamlessly transitions

between desktop and mobile devices ensures that airline staff can access and interact with the FMS efficiently, whether they are in an office or on the go. Furthermore, robust reporting and analytics tools can be integrated into the input design to empower users with actionable insights, helping them make data-driven decisions and optimize flight management processes. Usability testing, involving actual airline staff, should be an ongoing practice to continually refine the input interface and adapt it to evolving user needs and industry trends. These enhancements collectively contribute to a more sophisticated and user centric Flight Management System, improving operational efficiency and customer satisfaction within the airline industry.

### 5.2 OUTPUT DESIGN

The output is designed in such a way that it is attractive, convenient and informative. Forms are designed in java with various features, which make the console output more pleasing. As the outputs are the most important sources of information to the users, better design should improve the system's relationships with us and also will help in decision-making. Form design elaborates the way output is presented and the layout available for capturing information. In this project, the output is designed in the form reports. The system is designed to generate various user-friendly reports to help the business process. Following are the different reports supported.

### Module Description for Flight Management System:

The Flight Management System is a versatile software solution meticulously designed to optimize the operations of beauty and wellness establishments. This system comprises several interrelated modules, each dedicated to specific functions: User Management Module: This module oversees user registration, authentication, and user roles to ensure secure access for employees and customers.

Appointment and Scheduling Module: Customers can effortlessly book appointments, while staff can efficiently manage these bookings, including real-time availability checks and automated appointment reminders.

Customer and Service Information Module: It stores comprehensive customer profiles, including personal details, appointment history, and service preferences, facilitating personalized and tailored services.

Product Inventory Module: This module tracks beauty and wellness product inventory levels, automating restocking alerts to prevent shortages and disruptions.

Work Order Management Module: This module generates work orders for scheduled services, tracks their progress, and supports the reassignment of tasks, ensuring efficient service delivery.

Billing and Invoicing Module: It generates invoices for completed services, records payment transactions, and manages billing cycles for a seamless financial workflow. Reporting and Analytics Module: This module empowers management with valuable insights through various reports, aiding data-driven decisions regarding operations, revenue, and customer behaviour.

**5.3 FLOW DIAGRAM**

## USE CASE DIAGRAM:

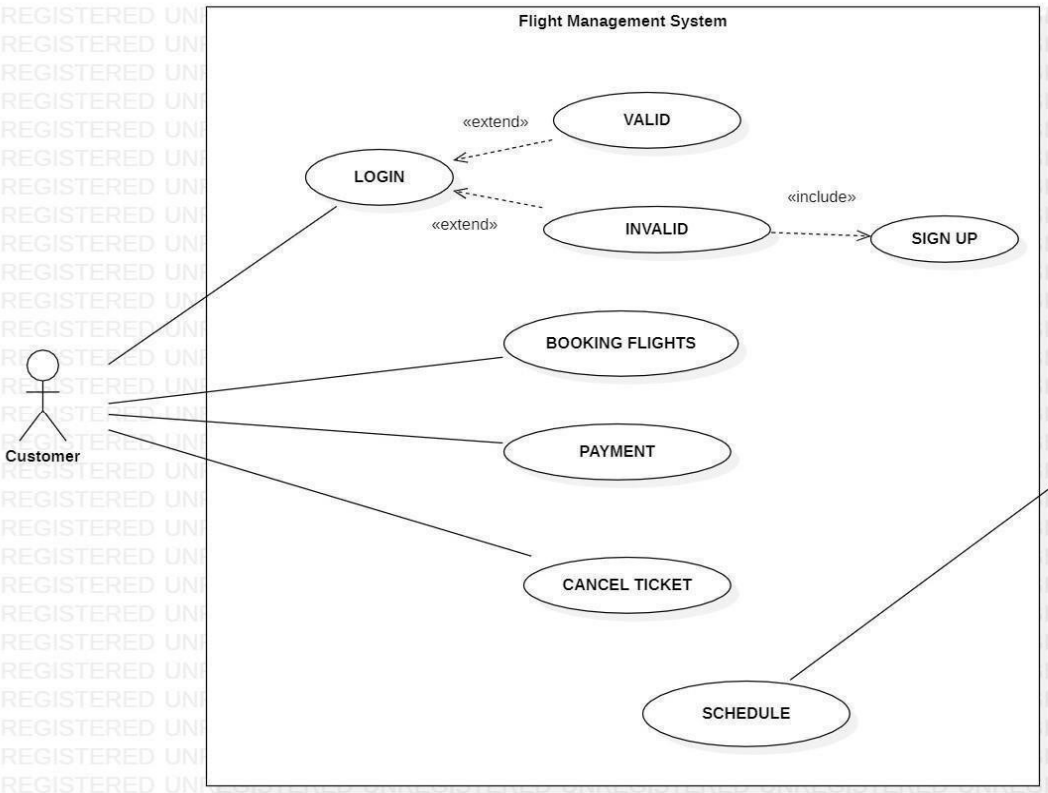


*Figure 5.3.1 – Use case diagram*

**STATE DIAGRAM:**

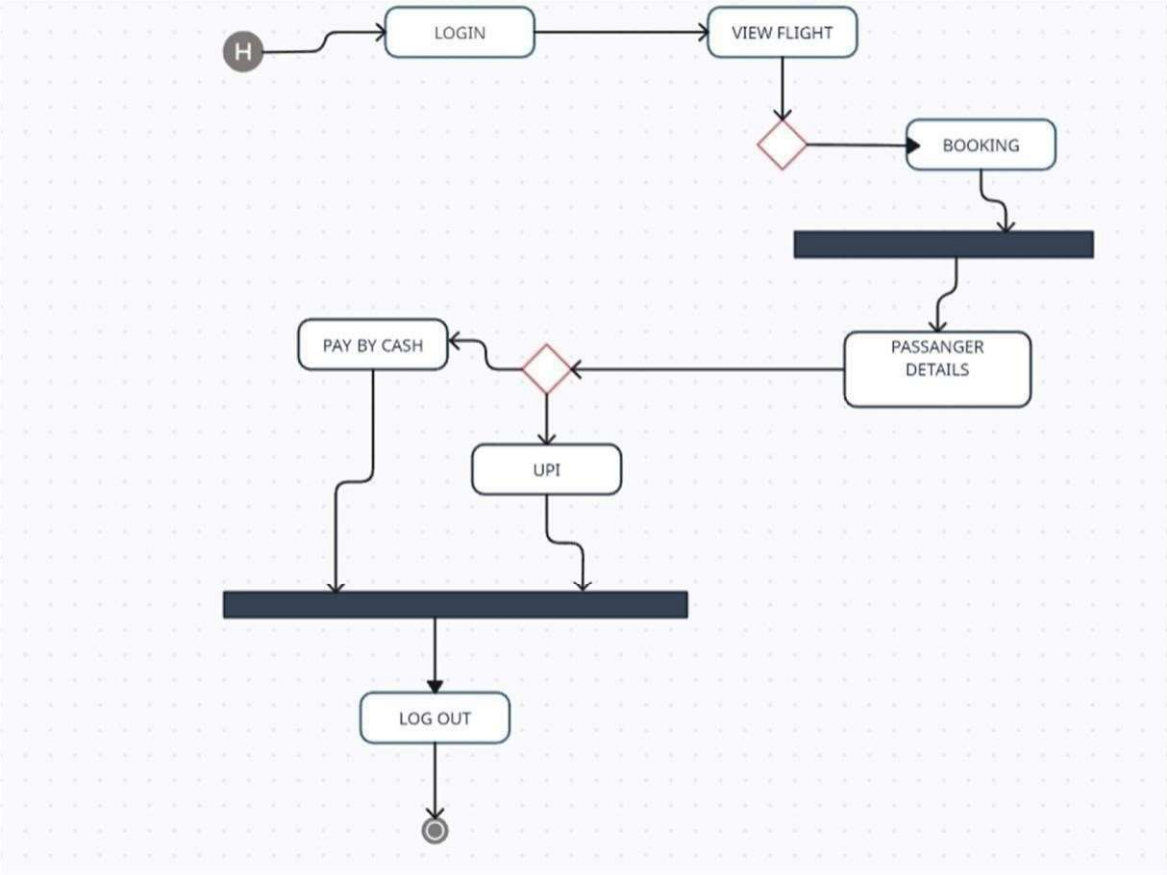


*Figure5.3.2 – State diagram*

**ER DIAGRAM:**

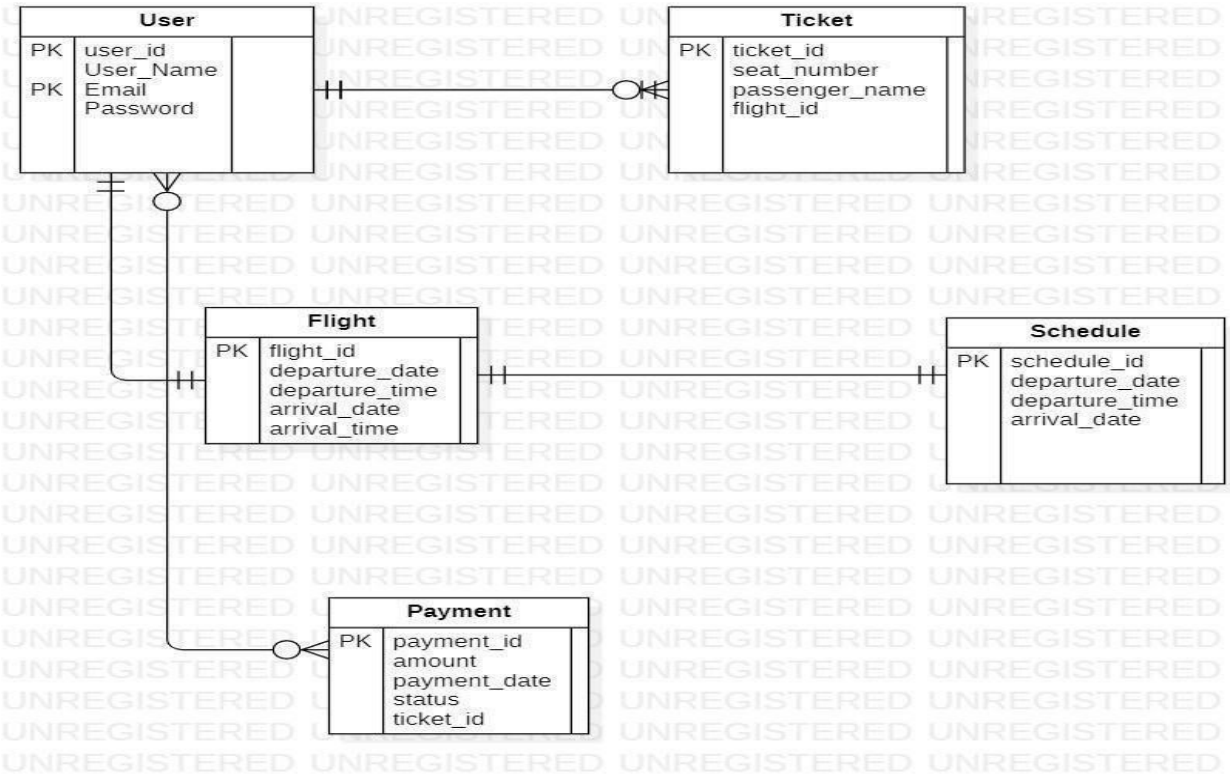


*Figure 5.3.3 – ER diagram*

## CLASS DIAGRAM

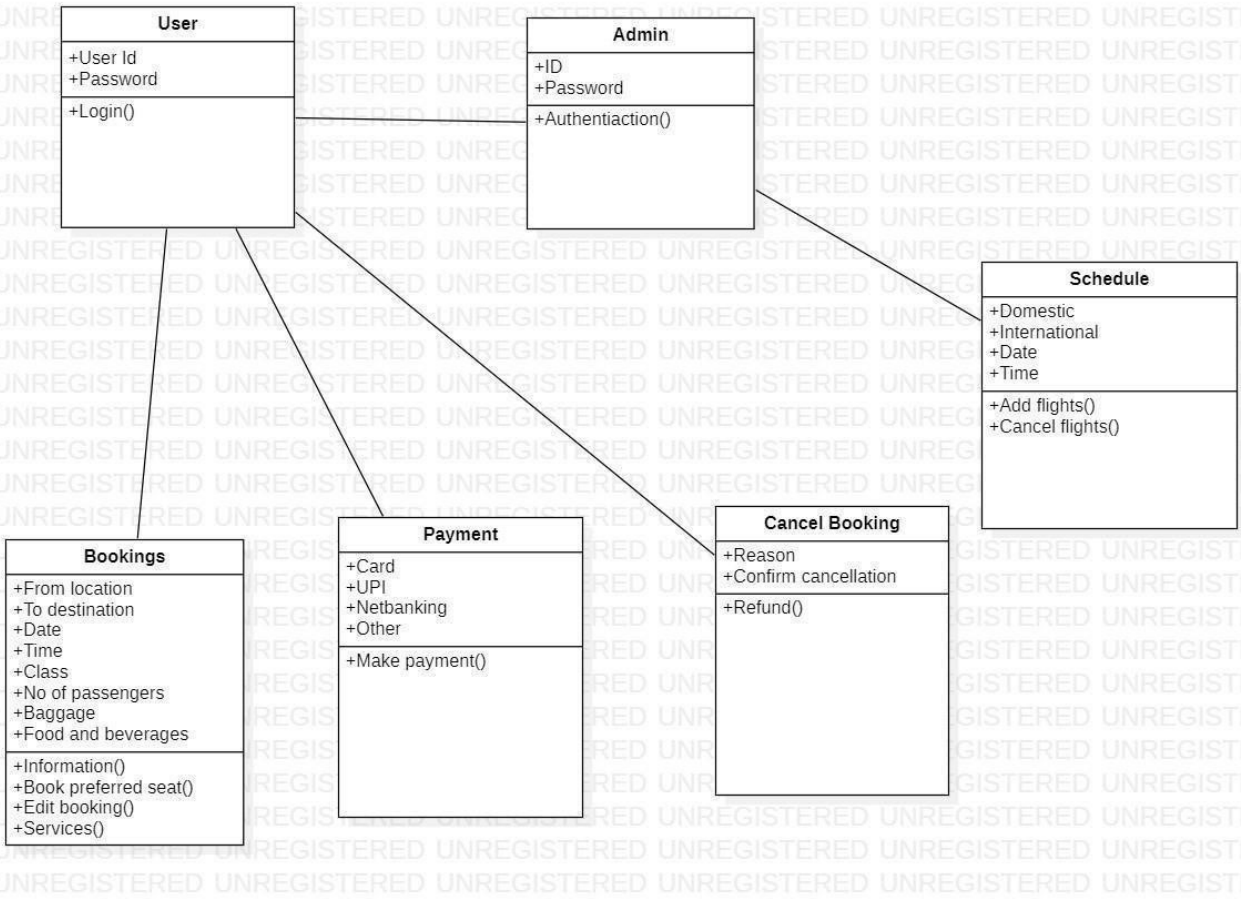


*Figure 5.3.4 – Class diagram*

**ENHANCED E R DIAGRAM**

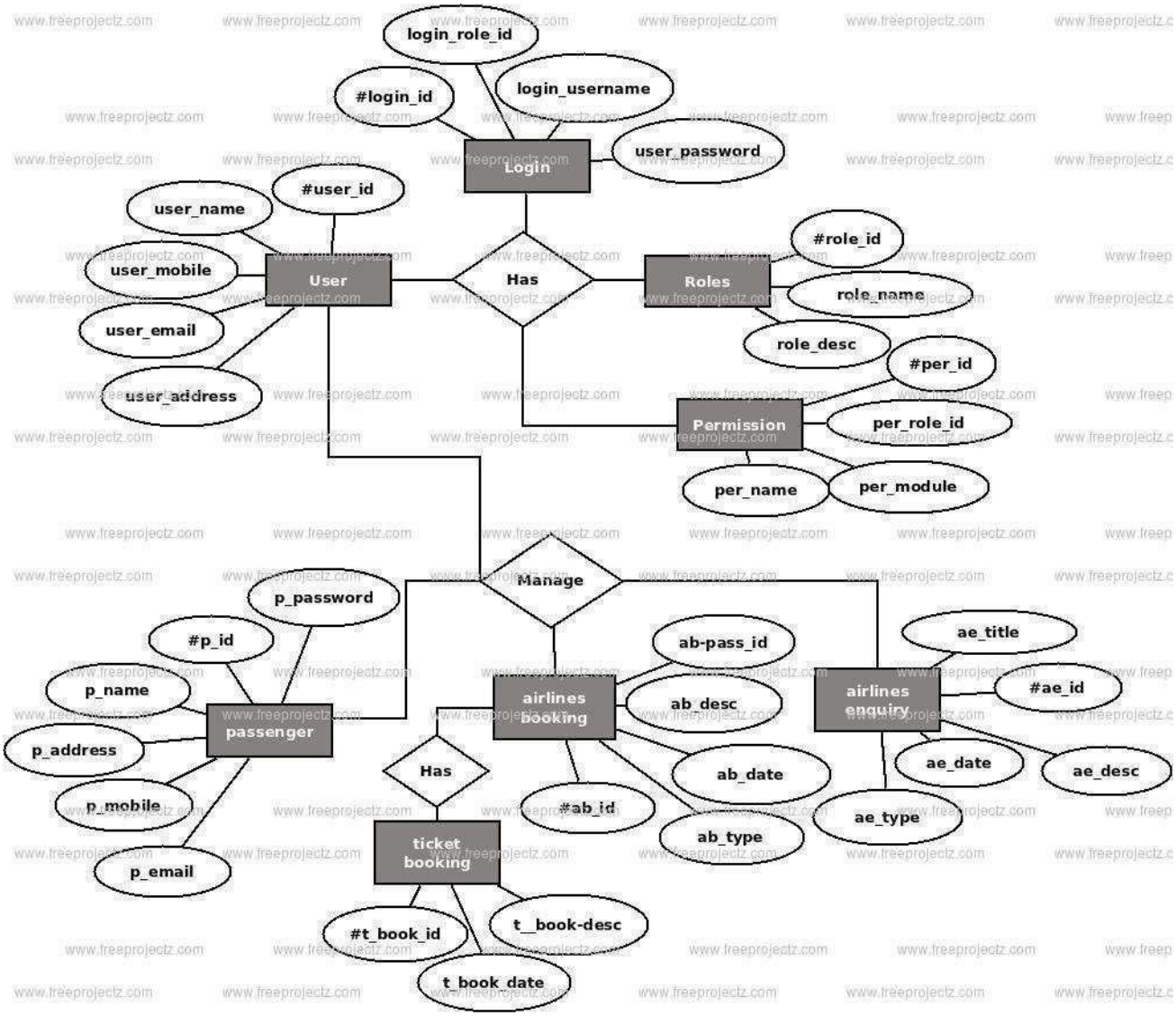


*Figure 5.3.5 – E ER diagram*

## SEQUENCE DIAGRAM

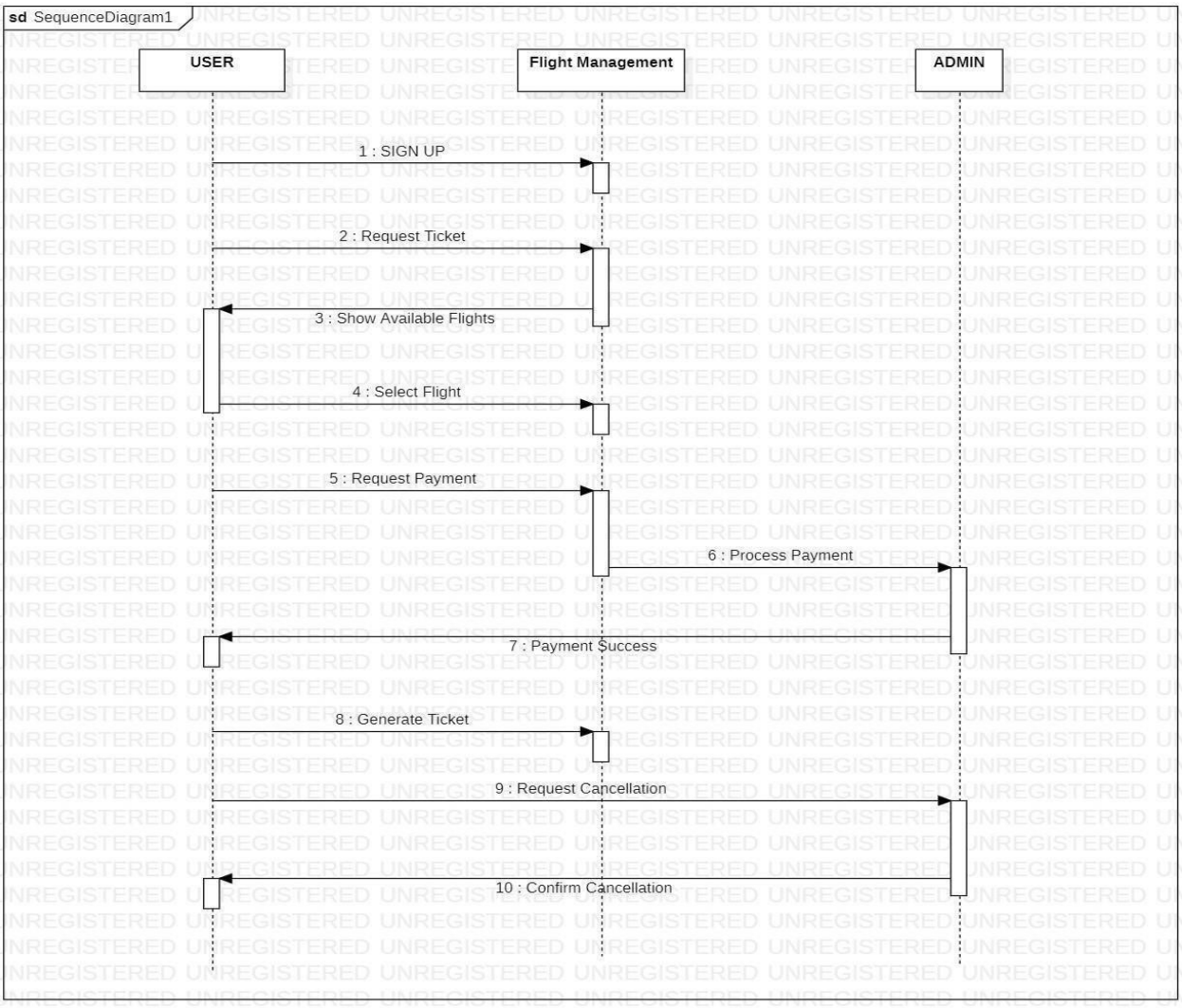


*Figure 5.3.6 – Sequence diagram*

## MODULE 6

## IMPLEMENTATION

### 6.1 CODING

Flight.java:

```
package Code;
import java. time. LocalDateTime; import
java.time.format.DateTimeFormatter; class
Flight {
    private String flightNumber;
    private String destination;
    private LocalDateTime departureTime;
    public String getFlightNumber() {
    return flightNumber;
    }    public void
setFlightNumber(String flightNumber) {
    this.flightNumber = flightNumber;
    }    public
String getDestination() {
    return destination;
    }
```

```
            public void setDestination(String destination) {
    this.destination = destination;
                    }
  public LocalDateTime getDepartureTime() {     return departureTime;
                     }                    public void
setDepartureTime(LocalDateTime departureTime) {
    this.departureTime = departureTime;
                    }
  public int getCapacity() {     return capacity;
                    }
  public void setCapacity(int capacity) {     this.capacity = capacity;
                    }
  public int getBookedSeats() {     return bookedSeats;
                     }                    public void
setBookedSeats(int bookedSeats) {
    this.bookedSeats = bookedSeats;
                    }
    private int capacity;
                     private int
bookedSeats;
                          public Flight(String flightNumber, String destination, LocalDateTime
                    departureTime, int capacity) {
```

```
                this.flightNumber = flightNumber;
        this.destination = destination;
        this.departureTime = departureTime;
        this.capacity = capacity;
        this.bookedSeats = 0;
                }

                // Getters and setters
        public boolean checkAvailability() {
        return bookedSeats < capacity;
                }
public void bookSeat() {  if
(checkAvailability()) {
bookedSeats++;
                System.out.println("Seat booked successfully!");
                } else {
                System.out.println("Sorry, no seats available.");
                }
        }
        @Override
        public String toString() {
        DateTimeFormatterformatter =
        DateTimeFormatter.ofPattern("yyyyMMdd HH:mm");  return "Flight " +
        flightNumber + " to " + destination + " at " + departureTime.format(formatter)+ "
```

```
            (" + bookedSeats + "/" + capacity + " seats
            booked)";
            }
    }
```

**FLIGHT MANAGEMENT SYSTEM:**

```
  package Code;              import java. SQL.*;    import  java.
time.LocalDateTime;      import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException; import
java.util.ArrayList; import java.util.List; import java.util.Scanner;
public class FlightManagementSystem extends
AbstractClass{//inheritance     private List<Flight> Flight;     private
Scanner   scanner;//encapsulation          private   Connection   connection;          public
FlightManagementSystem() {        Flight = new ArrayList<>();        scanner
= new Scanner(System.in);        establishConnection(); //
Connect to the MySQL database
    }
    private void establishConnection() {
        String url = "jdbc:mysql://localhost:3306/flight_db?useSSL=false";
        String user = "root";
        String password = "tiger"; try
{
            Class.forName("com.mysql.cj.jdbc.Driver");  //  Updated  driver  class  name
connection        =        DriverManager.getConnection(url,        user,        password);
```

```
System.out.println("Connected to the database successfully!");                    } catch
(ClassNotFoundException | SQLException e) {
            e.printStackTrace();
            System.out.println("Failed to connect to the database.");
        }
    }
    public void addFlight() {
        try {
            System.out.println("Enter flight number: ");
            String flightNumber = scanner.next();
            System.out.println("Enter destination: ");
            String destination = scanner.next();
            System.out.println("Enter departure date and time (yyyy-MM-dd HH:mm): ");
scanner.nextLine(); // Consume the newline character from the previous input
            String departureDateTimeStr = scanner.nextLine();
// Check if the input is empty or null            if
(departureDateTimeStr.isEmpty()) {
                System.out.println("Departure date and time cannot be empty.");              return;
            }
                LocalDateTime departureTime = LocalDateTime.parse(departureDateTimeStr,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"));
            System.out.println("Enter capacity: ");          int capacity = scanner.nextInt();
            // Convert LocalDateTime to Timestamp
            Timestamp departureTimestamp = Timestamp.valueOf(departureTime);
            // Insert the new flight into the database
```

```
        String insertQuery = "INSERT INTO flights (flight_number, destination,
departure_time, capacity) VALUES (?, ?, ?, ?)";
        PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);
preparedStatement.setString(1, flightNumber);          preparedStatement.setString(2, destination);
preparedStatement.setTimestamp(3, departureTimestamp);          preparedStatement.setInt(4,
capacity);          preparedStatement.executeUpdate();          preparedStatement.close();
System.out.println("Flight added successfully!");          } catch (DateTimeParseException e) {
e.printStackTrace();
        System.out.println("Invalid date and time format. Please use the format 'yyyy-MMdd
HH:mm'.");
      } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Failed to add the flight to the database.");
      }
   }
   public void viewFlights() {
      try {
        String selectQuery = "SELECT * FROM flights";
        Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(selectQuery);          if
(!resultSet.isBeforeFirst()) {
          System.out.println("No flights available.");
        } else {
          System.out.println("=== Available Flights
===");          while (resultSet.next()) {
resultSet.getInt("id");
```

```
                String flightNumber = resultSet.getString("flight_number");

                String destination = resultSet.getString("destination");

                LocalDateTime departureTime =
resultSet.getTimestamp("departure_time").toLocalDateTime();
int capacity = resultSet.getInt("capacity");                int bookedSeats =
resultSet.getInt("booked_seats");
System.out.println("Flight " + flightNumber + " to " + destination
+
                        " at " + departureTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm")) +
                        " (" + bookedSeats + "/" + capacity + " seats booked)");
            }
            System.out.println("==========================");
        }           resultSet.close();          statement.close();
} catch
(SQLException e) {
        e.printStackTrace();
        System.out.println("Failed to retrieve flights from the database.");
    }
  }
  // Method to book a seat on a flight
@Override//polymorphism    public void bookSeat()
{       viewFlights();
System.out.println("Enter the flight number: ");

String flightNumber = scanner.next();       try {
        // Check if the flight exists in the database
```

```
        String selectQuery = "SELECT * FROM flights WHERE flight_number = ?";
 PreparedStatement preparedStatement = connection.prepareStatement(selectQuery);
preparedStatement.setString(1, flightNumber);
        ResultSet resultSet = preparedStatement.executeQuery();
 if
(resultSet.next()) {
 int capacity = resultSet.getInt("capacity");
int bookedSeats = resultSet.getInt("booked_seats");
if
(bookedSeats < capacity) {
                System.out.println("Enter passenger name: ");
                String name = scanner.next();
                System.out.println("Enter passport number: ");
                String passportNumber = scanner.next();
                // Insert the new passenger into the database
                    String     insertQuery    =    "INSERT  INTO  passengers
 (flight_id,           name, passport_number) "  "VALUES (?, ?, ?)";
PreparedStatementpassengerStatement=connection.prepareStatement(insertQuery);
passengerStatement.setInt(1, resultSet.getInt("id"));
passengerStatement.setString(2, name);
passengerStatement.setString(3, passportNumber);
passengerStatement.executeUpdate();
passengerStatement.close();
                // Update the booked seats count for the flight
```

```
String updateQuery = "UPDATE flights SET booked_seats = ? WHERE id = ?";
PreparedStatement  updateStatement  = connection.prepareStatement(updateQuery);
updateStatement.setInt(1, bookedSeats + 1);
updateStatement.setInt(2, resultSet.getInt("id")) updateStatement.executeUpdate();
updateStatement.close();
                System.out.println("Seat booked successfully for Passenger " + name +
                                " (Passport: " + passportNumber + ")");
} else {
                System.out.println("Sorry, no seats available on this flight.");
              }
          } else {
              System.out.println("Flight with number " + flightNumber + " not found.");
          }  resultSet.close();preparedStatement preparedStatement.close();
} catch
(SQLException e) {
          e.printStackTrace();
          System.out.println("Failed to book the seat on the flight.");
       }
   }
   // Method to close the JDBC connection when the application exits
private void closeConnection() {
       try {
          if (connection      !=       null    &&    !connection.isClosed())    {
connection.close();
              System.out.println("Connection to the database closed.");
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("Failed to close the database connection.");
        }
    }
    // Method to close resources and exit the
application     public void exit() {
closeConnection();         scanner.close();
        System.out.println("Exiting the application. Goodbye!");
        System.exit(0);
    }
    public static void main(String[] args) {
        FlightManagementSystem flightManagementSystem = new
FlightManagementSystem();         Scanner scanner = new Scanner(System.in);         int choice;
do {
            System.out.println("=== Flight Management System ===");
            System.out.println("1. Add Flight");
            System.out.println("2. View Available Flights");
            System.out.println("3. Book Seat");
            System.out.println("4. Exit");
            System.out.print("Enter  your   choice:   ");             choice = scanner.nextInt();
```

```
    switch      (choice)      {                                case  1:
flightManagementSystem.addFlight();                          break;
flightManagementSystem.viewFlights();                        break;
case 3:
flightManagementSystem.bookSeat();
}
 public    void   setPassportNumber(String  passportNumber)       {
    this.passportNumber = passportNumber;
 }

 public      Passenger(String name,  String  passportNumber)       {
 this.name = name;
this.passportNumber = passportNumber;
   }
   // Getters and setters
   @Override     public String toString() {          return "Passenger " + name  + "
(Passport: " + passportNumber + ")";
   }
}
```

## 6.2 OUTPUT

Home page:

```
Connected to the database successfully!
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice:
```

Adding Flight:

```
Connected to the database successfully!
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 1
Enter flight number:
110
Enter destination:

chennai
Enter departure date and time (yyyy-MM-dd HH:mm):
2023-11-20 11:30
Enter capacity:
10
Flight added successfully!
```

Available Flights:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 2
=== Available Flights ===
Flight 848 to chennai at 2034-10-22 02:00 (2/200 seats booked)
Flight 110 to chennai at 2023-11-20 11:30 (0/10 seats booked)
==========================
```

Book Seat:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 3
=== Available Flights ===
Flight 848 to chennai at 2034-10-22 02:00 (2/200 seats booked)
Flight 110 to chennai at 2023-11-20 11:30 (0/10 seats booked)
==========================
Enter the flight number:
110
Enter passenger name:
nithish
Enter passport number:
12345
Seat booked successfully for Passenger nithish (Passport: 12345)
```

Exit:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 4
Connection to the database closed.
Exiting the application. Goodbye!
```

# CHAPTER – 7

# TESTING

## 7.1 INTRODUCTION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive. A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large- and small-scale systems.

## 7.2 STRATEGIC APPROACH TO SOFTWARE TESTING

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behaviours, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral in along streamlines that decrease the level of abstraction on each turn. A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code.

.

## 7.3 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing, we have is white box oriented and some modules the steps are conducted in parallel.

### 7.3.1 WHITE BOX TESTING

To follow the concept of white box testing we have tested each form. We have created independently to verify that Data flow is correct and all conditions are exercised to check their validity.

- All loops are executed on their boundaries. This type of testing ensures that all independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity paths.

### 7.3.2 CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested so that each path that may generate on a particular condition is traced to uncover any possible errors.

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was used in this type of testing.

### 7.3.3 DATA FLOW TESTING

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was used in this type of testing.

### 7.3.4 LOOP TESTING

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them. All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards. For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.
- Each unit has been separately tested by the development team itself and all the input have been validated.

## 7.4 TEST CASE

A test case, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.

In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released.

### 7.4.1 TEST CASE I

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 1
Enter flight number:
110
Enter destination:

chennai
Enter departure date and time (yyyy-MM-dd HH:mm):
2023-11-20 11:30
Enter capacity:
10




Flight added successfully!
```

EXPECTED:

=== Available Flights ===

Flight 110 to Chennai at 2023-11-20 11:30 (0/10 seats booked)

ACTUAL:

=== Available Flights ===

Flight 110 to Chennai at 2023-11-20 11:30 (0/10 seats booked)

### 7.4.2 TEST CASE II

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 3
=== Available Flights ===
Flight 848 to chennai at 2034-10-22 02:00 (2/200 seats booked)
Flight 110 to chennai at 2023-11-20 11:30 (0/10 seats booked)
========================
Enter the flight number:
110
Enter passenger name:
nithish
Enter passport number:
12345
```

```
Seat booked successfully for Passenger nithish (Passport: 12345)
```

EXPECTED:

=== Available Flights ===

Flight 110 to Chennai at 2023-11-20 11:30 (1/10 seats booked)

ACTUAL :

=== Available Flights ===

Flight 110 to Chennai at 2023-11-20 11:30 (1/10 seats booked)

# CHAPTER 8

# CONCLUSION AND FUTURE WORK

## 8.1 CONCLUSION

In conclusion, the implementation of a car workshop management system offers numerous benefits and improvements for automotive repair businesses. Such a system streamlines operations, enhances customer service, and ultimately contributes to increased efficiency and profitability. With features like appointment scheduling, inventory management, and customer relationship management

Additionally, these systems facilitate better communication and coordination among employees, leading to reduced errors and quicker turnaround times for repairs. By providing real-time data and analytics, they empower workshop owners and managers to make informed decisions, optimize resource allocation, and identify areas for improvement.

In summary, a well-implemented car workshop management system can transform the way automotive repair businesses operate. It can help them stay competitive in a rapidly changing industry, improve customer satisfaction, and drive growth and profitability. As technology continues to advance, car workshop management systems are likely to play an increasingly vital role in the success of automotive repair shops.

## 8.2 FUTURE WORK:

In the future, the car workshop management system can continue to evolve and expand its capabilities to further enhance efficiency and customer satisfaction. Integration with emerging technologies like artificial intelligence (AI) and Internet of things (IoT) could enable predictive maintenance, where the system anticipates potential vehicle issues and schedules maintenance before breakdowns occur.

## REFERENCES


[1] Web-Based Car Workshop Management System—A Review Mritha Ramalingam, Geetha Manoharan & R. Puviarasi Conference paper First Online: 15 June 2021

[2] Computerised Car Workshop System / Sivanesen Vaitilinggam Sivanesen, Vaitilinggam (2003) Computerised Car Workshop System / Sivanesen Vaitilinggam. Undergraduates thesis, University of Malaya.

[3] New Forms of Work Organisation in a French Car Company Workshop Author links open overlay panelJ.-P. Durand Department of Sociology, University of Rouen, rue Lavoisier, 76130 Mont Saint-Aignan, France.

[4] Multi-agent system “Smart Factory” for real-time workshop management in aircraft jet engines production JSC KUZNETSOV, 29 Zavodskoe shosse., Samara, Russian Federation, 423009

[5] Human Activity Recognition in car workshop Magdy Tawfik, Omar, October University For Modern Sciences and Arts.

## APPENDICES APPENDIX – I

## SOURCE CODE

Flight.java:

```
package Code;
import java. time. LocalDateTime;
import    java.time.format.DateTime
Formatter; class
Flight {
        private          String          flightNumber;
    private String destination;
    private LocalDateTime departureTime;
    public String getFlightNumber() {
        return flightNumber;
            }
    public void setFlightNumber(String flightNumber) {
this.flightNumber =  flightNumber;
            }                   public
String getDestination() {
    return destination;
        public void setDestination(String destination)
            this.destination = destination;
```

```
                }
  public LocalDateTime getDepartureTime() {
   return departureTime;
                }
          public void setDepartureTime(LocalDateTime departureTime) {
   this.departureTime = departureTime;
                }
  public int getCapacity() {
   return capacity;
                }
  public void setCapacity(int capacity) {
  this.capacity = capacity;
                }
  public int getBookedSeats() {
return bookedSeats;
                 }
   public void setBookedSeats(int bookedSeats) {
          this.bookedSeats = bookedSeats;
                 }
   private int capacity;
                  private int
bookedSeats;
```

```
        public Flight(String flightNumber, String destination,
         LocalDateTime  departureTime, int capacity) {
                    this.flightNumber = flightNumber;
    this.destination = destination;
    this.departureTime = departureTime;
    this.capacity = capacity;
    this.bookedSeats = 0;
                    }                       public
  boolean checkAvailability() {             return
  bookedSeats < capacity;
                   }
                    public void bookSeat() {
    if (checkAvailability()) {
  bookedSeats++;
                    System.out.println("Seat booked successfully!");
                   } else {
                   System.out.println("Sorry, no seats available.");
                   }
              }
             @Override
              public String toString() {
DateTimeFormatterformatter  = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
return "Flight " + flightNumber + " to " + destination + " at " +
departureTime.format(formatter)+ " (" + bookedSeats + "/" + capacity + " seats  booked)";
             }
    }
```

**FLIGHT MANAGEMENT SYSTEM**:

```
FlightManagementSystem extends
AbstractClass{//inheritance     private List<Flight> Flight;     private
Scanner scanner;//encapsulation     private Connection connection;
    public FlightManagementSystem() {
Flight = new ArrayList<>();
        scanner = new Scanner(System.in);
Connect to the MySQL database
   }
   private void establishConnection() {
      String url = "jdbc:mysql://localhost:3306/flight_db?useSSL=false";
      String user = "root";
String password = "tiger"; try {
         Class.forName("com.mysql.cj.jdbc.Driver");    //    Updated    driver    class    name
connection = DriverManager.getConnection(url, user, password);
         System.out.println("Connected to the database successfully!");
        } catch (ClassNotFoundException | SQLException e) {
         e.printStackTrace();
         System.out.println("Failed to connect to the database.");
      }
   }
   public void addFlight() {
      try {
         System.out.println("Enter flight number: ");
```

```
            String flightNumber = scanner.next();
            System.out.println("Enter destination: ");
            String destination = scanner.next();
            System.out.println("Enter departure date and time (yyyy-MM-dd HH:mm): ");
scanner.nextLine(); // Consume the newline character from the previous input
            String departureDateTimeStr = scanner.nextLine();
// Check if the input is empty or null            if
(departureDateTimeStr.isEmpty()) {
                System.out.println("Departure date and time cannot be empty.");
              return;
            }
               LocalDateTime  departureTime  =  LocalDateTime.parse(departureDateTimeStr,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"));
            System.out.println("Enter capacity: ");            int capacity = scanner.nextInt();
            // Convert LocalDateTime to Timestamp
            Timestamp departureTimestamp = Timestamp.valueOf(departureTime);
            // Insert the new flight into the database
            String  insertQuery  =  "INSERT  INTO  flights  (flight_number,  destination,  departure_time,
capacity) VALUES (?, ?, ?, ?)";
preparedStatement.executeUpdate();
 preparedStatement.close();
            System.out.println("Flight  added  successfully!");
        }  catch (DateTimeParseException e) {
            e.printStackTrace();
            System.out.println("Invalid date and time format. Please
 use the format 'yyyy-MMdd
```

```
HH:mm'.");
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("Failed to add the flight to the database.");
        }
    }
    public void viewFlights() {
        try {
            String selectQuery = "SELECT * FROM flights";
            Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(selectQuery);            if
(!resultSet.isBeforeFirst()) {
                System.out.println("No flights available.");
            } else {
                System.out.println("=== Available Flights
===");                while (resultSet.next()) {
resultSet.getInt("id");
                    String flightNumber = resultSet.getString("flight_number");
                    String destination = resultSet.getString("destination");
                    LocalDateTime  =
resultSet.getTimestamp("departure_time").toLocalDateTime();
            int capacity = resultSet.getInt("capacity");
             int bookedSeats = resultSet.getInt("booked_seats");
                    System.out.println("Flight " + flightNumber + " to " + destination +
                            " at " + departureTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd
```

```
HH:mm")) +
                    " (" + bookedSeats + "/" + capacity + " seats booked)");
            }
            System.out.println("==========================");
        }
        resultSet.close();          statement.close();          } catch
(SQLException e) {
        e.printStackTrace();
        System.out.println("Failed to retrieve flights from the database.");
    }
  }
  // Method to book a seat on a flight
@Override//polymorphism    public void bookSeat()
{       viewFlights();
    System.out.println("Enter the flight number: ");
String flightNumber = scanner.next();       try {
        // Check if the flight exists in the database
        String selectQuery = "SELECT * FROM flights WHERE flight_number = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(selectQuery);
preparedStatement.setString(1, flightNumber);
        ResultSet resultSet = preparedStatement.executeQuery();
        if
(resultSet.next()) {            int capacity =
resultSet.getInt("capacity");           int
bookedSeats = resultSet.getInt("booked_seats");
if (bookedSeats < capacity) {
```

```
                System.out.println("Enter passenger name: ");
                String name = scanner.next();

                System.out.println("Enter passport number: ");
                String passportNumber = scanner.next();
                // Insert the new passenger into the database
               String    insertQuery    =    "INSERT    INTO  passengers
                String updateQuery = "UPDATE flights SET booked_seats = ? WHERE id = ?";
             PreparedStatement  updateStatement      = connection.prepareStatement
             (updateQuery);  updateStatement.setInt(1, bookedSeats + 1);
               updateStatement.setInt(2, resultSet.getInt("id"));
                 updateStatement.executeUpdate();
                    updateStatement.close();
                System.out.println("Seat booked successfully for Passenger " + name +
                                " (Passport: " + passportNumber + ")")
  // Method to close the JDBC connection when the application exits       private void
closeConnection() {
      try {
         if  (connection  !=        null  &&  !connection.isClosed())  {
connection.close();
            System.out.println("Connection to the database closed.");
         }
      } catch (SQLException e) {
         e.printStackTrace();
```

```
            System.out.println("Failed to close the database connection.");
        }
    }
    // Method to close resources and exit the
application     public void exit() {
closeConnection();         scanner.close();
        System.out.println("Exiting the application. Goodbye!");
        System.exit(0);
    }
    public static void main(String[] args) {
        FlightManagementSystem flightManagementSystem = new
FlightManagementSystem();         Scanner scanner = new Scanner(System.in);
         int choice;
do {
            System.out.println("=== Flight Management System ===");
            System.out.println("1. Add Flight");
            System.out.println("2. View Available Flights");
            System.out.println("3. Book Seat");
            System.out.println("4. Exit");

PASSANGER: package Code;
class Passenger {     private String name;     private
String passportNumber;     public
String getName() {
    return name;
```

```
    }
        public void setName(String name) {
this.name = name;
    }
    public String getPassportNumber() {  return passportNumber;
    }
    public void setPassportNumber(String passportNumber) {
       this.passportNumber = passportNumber;
    }
    public Passenger(String name, String passportNumber) { this.name
    = name;        this.passportNumber = passportNumber;
      // Getters and setter
    @Override    public String toString() {        return "Passenger " + name
    + " (Passport: " + passportNumber + ")";
      }
    }
```

## OUT PUT:

Home page:

```
Connected to the database successfully!
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice:
```

Adding Flight:

```
Connected to the database successfully!
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 1
Enter flight number:
110
Enter destination:

chennai
Enter departure date and time (yyyy-MM-dd HH:mm):
2023-11-20 11:30
Enter capacity:
10
Flight added successfully!
```

Available Flights:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 2
=== Available Flights ===
Flight 848 to chennai at 2034-10-22 02:00 (2/200 seats booked)
Flight 110 to chennai at 2023-11-20 11:30 (0/10 seats booked)
==========================
```

Book Seat:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 3
=== Available Flights ===
Flight 848 to chennai at 2034-10-22 02:00 (2/200 seats booked)
Flight 110 to chennai at 2023-11-20 11:30 (0/10 seats booked)
==========================
Enter the flight number:
110
Enter passenger name:
nithish
Enter passport number:
12345
Seat booked successfully for Passenger nithish (Passport: 12345)
```

Exit:

```
=== Flight Management System ===
1. Add Flight
2. View Available Flights
3. Book Seat
4. Exit
Enter your choice: 4
Connection to the database closed.
Exiting the application. Goodbye!
```