A REPORT

ON

AUDI TICKET BOOKING SYSTEM

CS F213 OBJECT ORIENTED PROGRAMMING

Dhruv Agarwal	**2019B2A30892P**
Anushka Patil	**2020B3A70767P**
Aryan Bansal	**2021A7PS2776P**
Rishabh Sahni	**2021A7PS1630P**

Submitted To

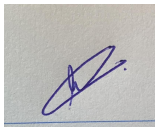
Prof. Amit Dua

TABLE OF CONTENTS

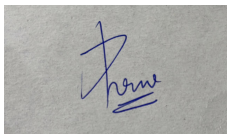
PLAGIARISM STATEMENT

I know that plagiarism means taking and using the ideas, writings, works or inventions of another as if they were one's own. I know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement (which includes the proper use of quotation marks). I know that plagiarism covers this sort of use of material found in textual sources and from the Internet. I acknowledge and understand that plagiarism is wrong. I understand that my research must be accurately referenced. I have followed the rules and conventions concerning referencing, citation and the use of quotations as set out in the departmental guide. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

NAME: ANUSHKA PATIL

SIGNATURE: 

NAME: DHRUV AGARWAL

SIGNATURE: 

WORK DIVISION

The entirety of the project was done by Dhruv Agarwal (2019B2A30892P) and Anushka Patil(2020B3A70767P)

- Feature implementation : Dhruv and Anushka
- Project Report : Dhruv and Anushka
- Object-Oriented Programming Principles : Dhruv and Anushka
- Use-case : Dhruv and Anushka
- Sequence : Dhruv and Anushka

- UML diagrams : Dhruv and Anushka
- GUI : Dhruv and Anushka

INTRODUCTION

This chapter gives an overview of the system's aim, targets, background, and operating environment.

I. PROJECT AIMS AND OBJECTIVES

The aims and objectives are as follows:

- Online booking system for booking Audi tickets during Oasis
- Provision for the user to check event details including timings
- Provision for user to check seating availability
- Provision for admin to create new events/change details of existing events
- Provision for admin to track revenue for events and check seating availability

II. BACKGROUND OF PROJECT

The main objective of Audi ticketing system is to provide a convenient way to the users to book the tickets for the events. User and admin can perform various functions according to the designated constraints. Classes like Seat and Event are included in this system which would keep track of the events and seating in the Audi and also provide a detailed description about the same. With this computerised system there will be no inefficiency or manual labour involved in ticket booking which generally happens when a non-computerised system is used.

PROCESSOR	INTEL CORE PROCESSOR OR BETTER PERFORMANCE
OPERATING SYSTEM	WINDOWS 10 ,WINDOWS 7, UBUNTU
IDE	ECLIPSE, INTELLIJ
LANGUAGE USED	JAVA

HELPER DOCUMENT

Execution Instructions:

The code was tested on Pop!_OS 21.04 which is an ubuntu-based Linux distro using Visual Studio Code. Instructions to execute the application are given below:

● Install JDK 17 using sudo apt install openjdk-17-jdk

● Download SDK from here.

● Unzip it to a desired location

● Add an environment variable pointing to the lib directory of the runtime in .bashrc

● export PATH_TO_FX=path/to/javafx-sdk-17/lib

● In VS Code settings.json file in the project directory, add the path to lib folder to referencedLibraries like this: { "java.project.sourcePaths": ["src"], "java.project.outputPath": "bin", "java.project.referencedLibraries": ["lib/**/*.jar", "/home/aviralomar/development/sdk/javafx-sdk-17.0.1/lib/*.jar"] }

● Install following: sudo apt install libswt-gtk-4-java

● To compile the code go into the bin folder of the project directory and run: javac --module-path $PATH_TO_FX --add-modules javafx.controls ../src/*.java -d .

● To execute the application, from the bin directory, run: java --module-path $PATH_TO_FX --add-modules javafx.controls Monopoly

SYSTEM ANALYSIS

The following section contains analysis of the process of making an Audi Ticket Booking System, including the software requirement specification (SRS). The SRS part includes both functional and non-functional requirements to give a full description and overview of system requirements before the development process begins.

SOFTWARE REQUIREMENT SPECIFICATION

PRODUCT DESCRIPTION

Audi Ticket Booking System is a computerised system that helps the user to keep track of the day-to-day events at the Audi and book tickets in an electronic format. It helps the administrator to make backend changes to the existing events and add events for a more efficient and streamlined process.

PROBLEM STATEMENT

Problems in a non-computerised system include the following:

- Damage and loss of ticketing records: When a computerised system is not implemented, records kept can be lost or damaged due to human error.
- Difficulty to sort and access record databases: Sorting and searching for user records is difficult to do manually.
- Difficulty to change event details dynamically
- Difficulty to show seating availability dynamically
- Errors in tracking revenue

SYSTEM OBJECTIVES

a. Control and performance improvements
b. Incorporation of dynamic changes
c. Ease of ticket booking
d. Ease of backend tracking

SYSTEM REQUIREMENTS

I. NON-FUNCTIONAL REQUIREMENTS

a. EFFICIENCY REQUIREMENT: Easier access to ticket booking and revenue tracking is implemented.

b. RELIABILITY REQUIREMENT: The system should accurately performs user registration, admin registration, event registration, ticket booking, and modify event details.

c. USABILITY REQUIREMENT: The system is designed for a user friendly environment so that the user and admin can perform specified functionality efficiently.

d. IMPLEMENTATION REQUIREMENTS: Implementation involves using the programming language Java.

II. FUNCTIONAL REQUIREMENTS

a. User

1. User Login: Users have to create their account which involves entering their first name, last name, password and email ID. After name and email are stored, the user can login through the registration window by entering email ID and password to book tickets.
2. Ticket Booking: Users can view event details, including timings and description, before booking tickets. After choosing the desired event, the user can view available seating in a matrix format to choose desired seats.

b. Admin

1. Admin Login: Admin will have to create their account which involves entering their first name, last name, password and email ID. After name and email are stored, the admin can login through the registration window by entering email ID and password to book tickets.
2. Revenue Tracking: The admin can track revenue for specific events.
3. Add/change event details: The admin can create new events. The admin can also change existing event timings, ticket cost, and description.

SOFTWARE AND HARDWARE REQUIREMENTS

I. SOFTWARE REQUIREMENTS

- Operating system: Windows 10 is used as the operating system due to efficiency and user-friendliness
- Programming language: Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Java offers features like
    1. Compiled and Interpreted.
    2. Platform-Independent and Portable.
    3. Object-Oriented.
    4. Robust and Secure.
    5. Distributed
    6. Architecture-Neutral
    7. Familiar and Simple.
    8. Multithreaded and Interactive.
    9. High Performance.
    10. Dynamic and Extensible

II. HARDWARE REQUIREMENTS

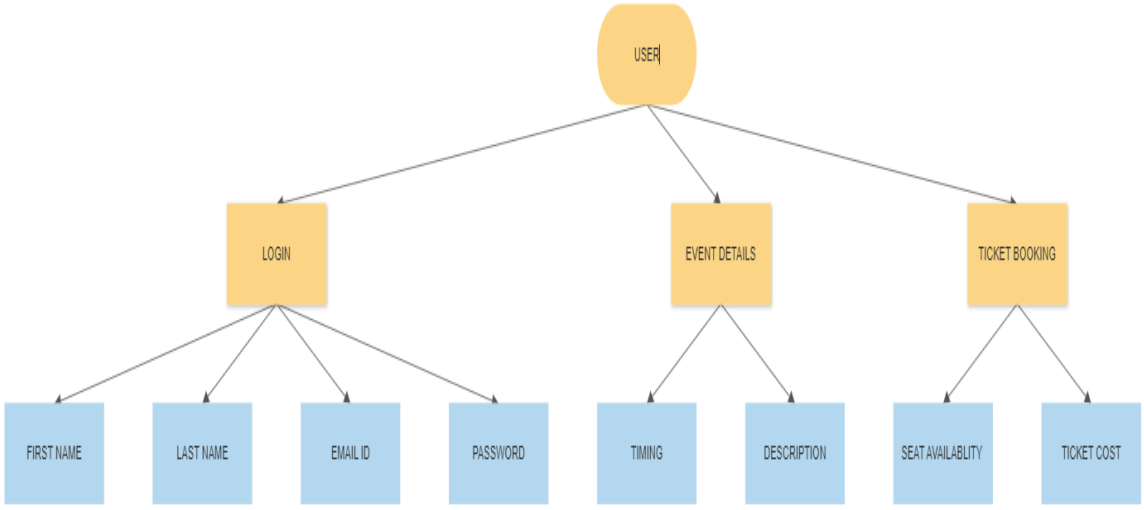
- Intel Core Processor
- 1 GB Ram

This application employs a comprehensive strategy to reduce manual labour and schedule resources and time in a logical manner. The original information is entered to the database and the ticket is confirmed after meeting all requirements.

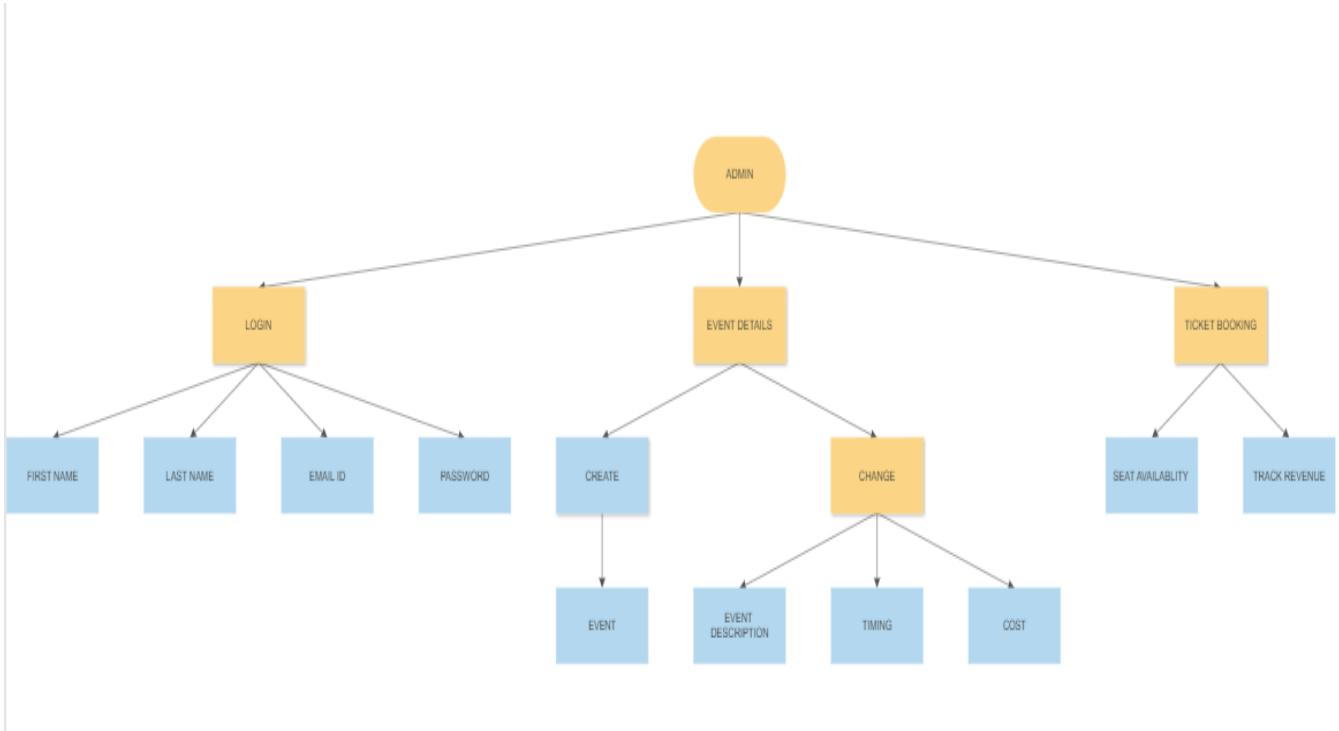
SYSTEM DESIGN

I. UML USE CASE DIAGRAM

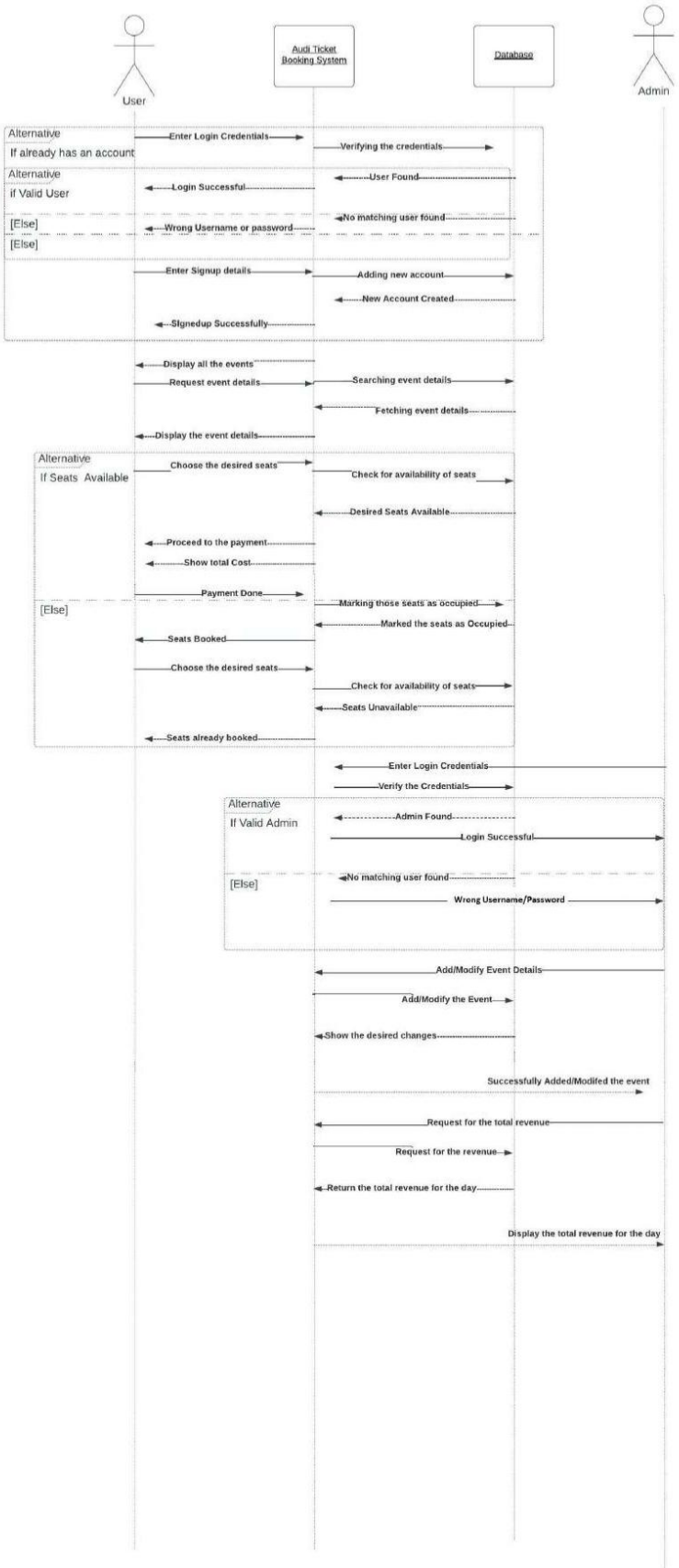
a. User



b. Admin



II. UML SEQUENCE DIAGRAM



SYSTEM IMPLEMENTATION

CLASS DESCRIPTION

1. Main: This is the main class which extends the Application class. It contains the start method and implements the main method.
2. Admin: The Admin class extends the User class and implements the Runnable interface. It contains the run() method which shows the Admin events, and a constructor which takes in a first name, last name, email, password and Audi object as parameters. It also contains the addEvent() method which allows the Admin to add an event, the changeEventDetails() method which allows the Admin to change event details, the showAdminEvents() method which shows the Admin events and the showTotalSales() method which shows the total sales from all events.
3. Audi: This class is the main class that holds the information about the events which are being held in the Audi. It also stores the details about the customers and admins who are attending the events. It contains methods to add customers, get customers and admins and also details about the events.
4. AlertBox: The class AlertBox has one static method called display(), which takes three parameters: a String title, a String message, and a String buttonmessage. This method creates a new instance of the Stage class, sets the title, dimensions, and modality of the window, creates a Label and Button instance, and adds them to a VBox layout. It then sets this layout as the Scene for the window and displays the window.
5. CancelEvent: CancelEvent class is responsible for displaying the event details and offers the functionality to cancel tickets for an event. It contains methods like display(), setOnAction(), getText(), getValue(), etc.
6. ChangeEventDetails: ChangeEventDetails contains a method called display(). The display() method takes three arguments which are of type Audi, Admin and Event respectively. The display() method is used to take inputs from the user and update the event details.
7. Event class: This class stores information about the events which are being held in the Audi. It contains details such as the event name, time, description, event ID and the ticket price.
8. Student class: This class stores information about the customers who are attending the event. It contains details such as the user name, email, password and a reference to the Audi object.
9. LoginDetails: This is a helper class which is used to accept login details from the user.
10. SignupDetails: This is a helper class which is used to accept signup details from the user.

11. AddEvent: The class AddEvent which contains the display() method. The display() method has two parameters Audi and Admin and opens a window which allows the user to add events. It also has a button to go back to the previous window.
12. User: It has five private instance variables: first_name, last_name, email, password, and a (of type Audi). It also has five getter methods which return the corresponding instance variable.
13. Seat: The Seat class has a private boolean variable "occupied". The Seat class has methods getOccupied(), setOccupied(), setOwner(), and getOwner().

DESIGN PATTERNS AND OOP PRINCIPLES

OOP principles present in the given code:

1. Encapsulation - The code uses access modifiers to restrict the access of certain methods and variables to certain classes.
2. Abstraction - The code uses abstract classes such as ActionEvent, EventHandler, Application and Stage which are used as a base for creating other classes.
3. Inheritance - The code uses inheritance to create classes such as Main and LoginDetails which inherit from the Application class.
4. Polymorphism - The code uses polymorphism to handle different types of inputs from the user in the form of Strings, ints and doubles.

The Design Patterns present in the given code are:

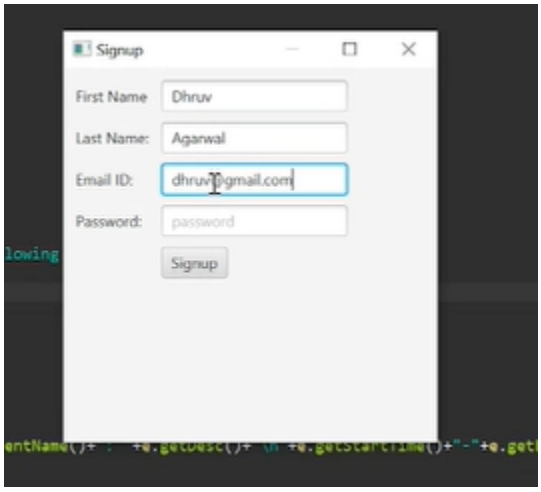
- The code in our project makes use of the **Factory Method Design Pattern**. This design pattern is used when an object needs to be created without specifying the exact class of object that will be created.
  Example: The CancelEvent class is used to create a window for cancelling tickets based on the given parameters (Audi, Student, Event).
- The code also uses the **Singleton Design Pattern**. Audi class is a public class which has only one Audi object which is a single instance and is used to store all the customers and admins.
- The code uses the **Command Design Pattern**. This pattern encapsulates a request as an object, allowing for different requests to be referenced through a common interface.
  Example: In the ShowAdminEvents class, the Command Design Pattern is seen in the edit and logout buttons, which are used to call different methods depending on the button clicked.

SOLID Design Principles used:

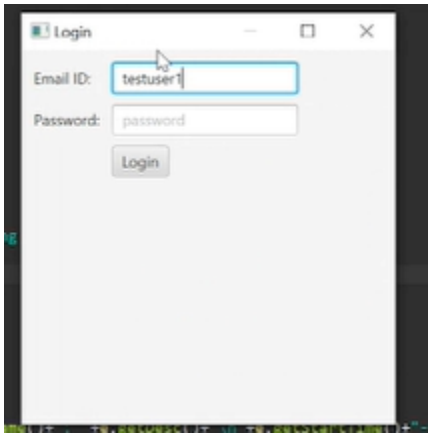
- Single Responsibility Principle: Each class is responsible for one specific part of the application, such as the Main class handling the application's startup and the LoginDetails and SignupDetails classes handling the login and sign up process.
- Open/Closed Principle: The application is designed to be open for extension by adding new features, but closed for modification since the existing code does not need to be modified for the new features.
- Liskov Substitution Principle: The application is designed with object-oriented programming in mind, making it possible to substitute classes with their subclasses.
- Interface Segregation Principle: The application is designed to have multiple small, independent interfaces instead of one large interface.
- Dependency Inversion Principle: The application is designed with high-level modules depending on abstractions, instead of low-level modules.

VIDEO RECORDING

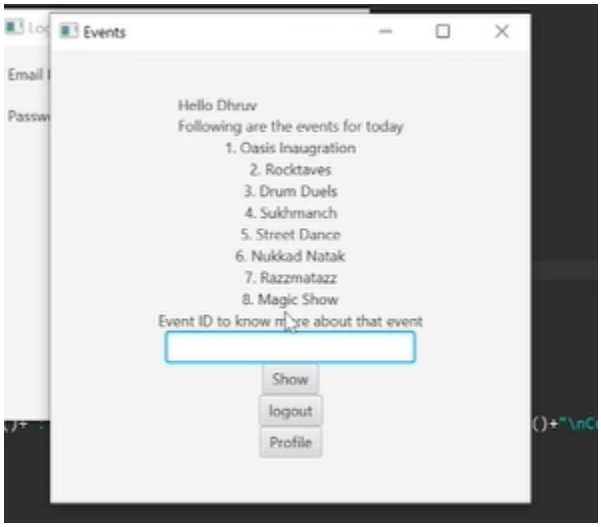
a. Registration Window



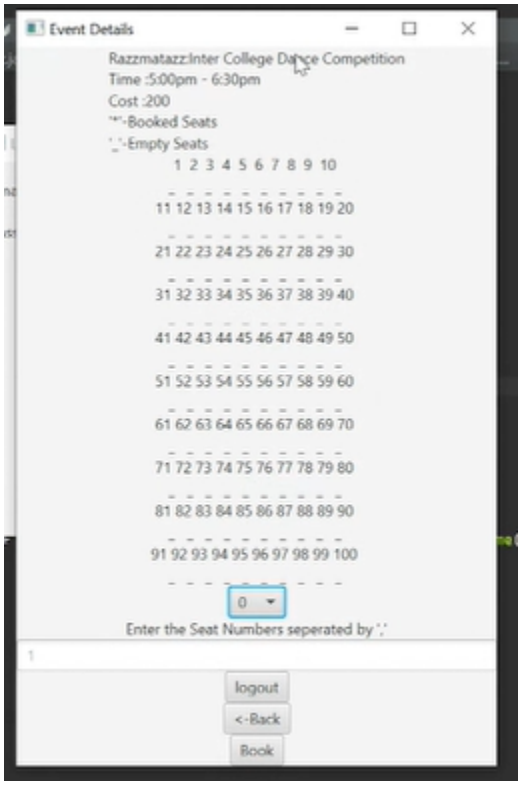
b. Login Window



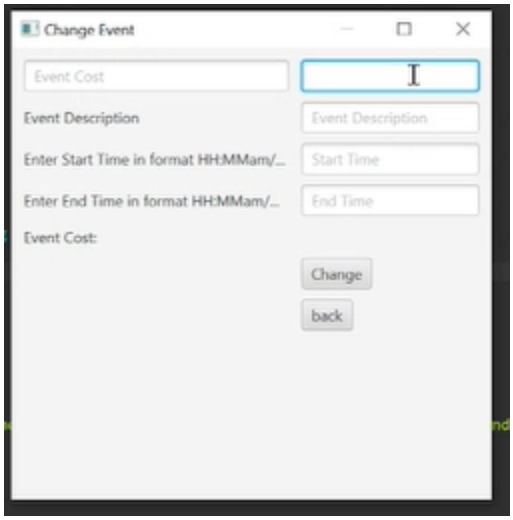
c. Event Board



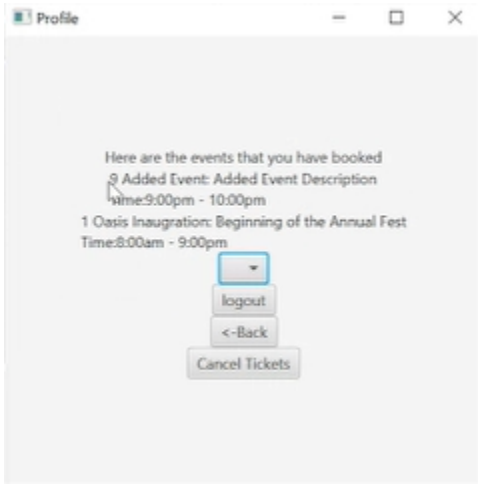
d. Ticket Booking



e. Event Editor



f. Ticket Cancellation



CONCLUSION

This project has a computerised version of a ticket booking system that will help both students and event organisers. The entire process is conducted on an online platform for ease of information availability and implementing dynamic changes so that the users and admin can successfully ensure that Audi event tickets are booked. The code also provides a login page for both admin and user and provides event details along with timings and ticket costs. Seating availability matrix is shown for both admin and user. The admin can make backend changes to the existing event database. This application employs a comprehensive strategy to reduce manual labour and schedule resources and time in a logical manner. The original information is entered to the database and the ticket is confirmed after meeting all requirements.