UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Event Management System

Submitted by:

MANJUNATH	**PES1UG21CS327**
MALLIKARJUN	**PES1UG21CS320**
MALLIKARJUN MELMALGI	**PES1UG21CS321**
NANDEESH HIREMATH	**PES1UG21CS363**

6th Semester ***F*** *Section*

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

1. Project Description:

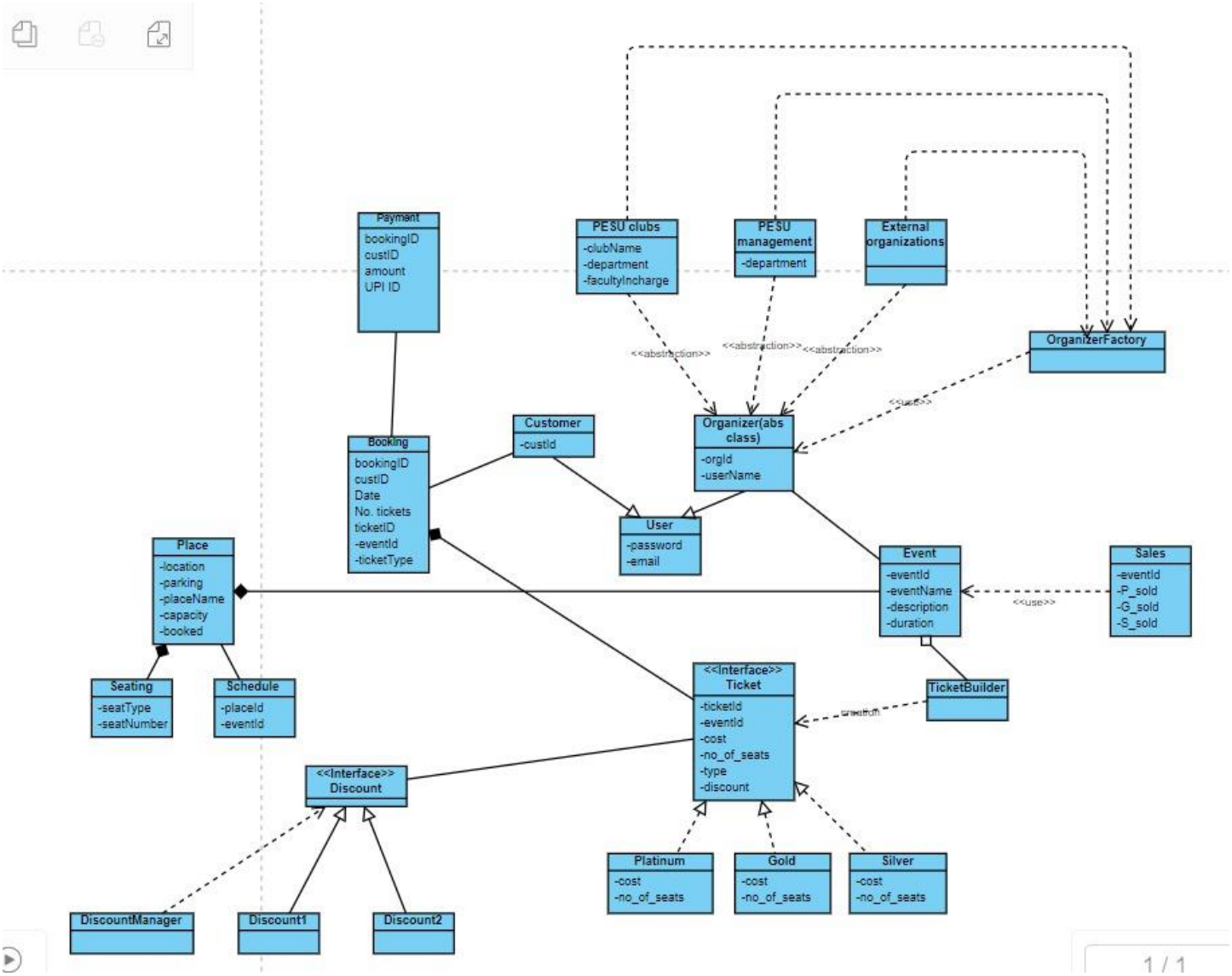
- Event Management System - a comprehensive application where users can book tickets to attend events and review them, while event organizers can schedule and manage events.
- There are two roles in this system: User and Organizer.
- If the user is a new user, they must first register and then be able to log in to book tickets. Once logged in, users can view the available events and book tickets for events that have been scheduled by the organizer. After booking the tickets and completing payment, users can also leave reviews and ratings for the events they attended.
- An organizer can view the current event schedule. They can add new events to the schedule, update event details, or remove existing events from the schedule.
- Both the organizer and user views are synchronized, ensuring that any updates made by the organizer are reflected in real-time for the users.
- Github repository link

2. Analysis and Design Models:

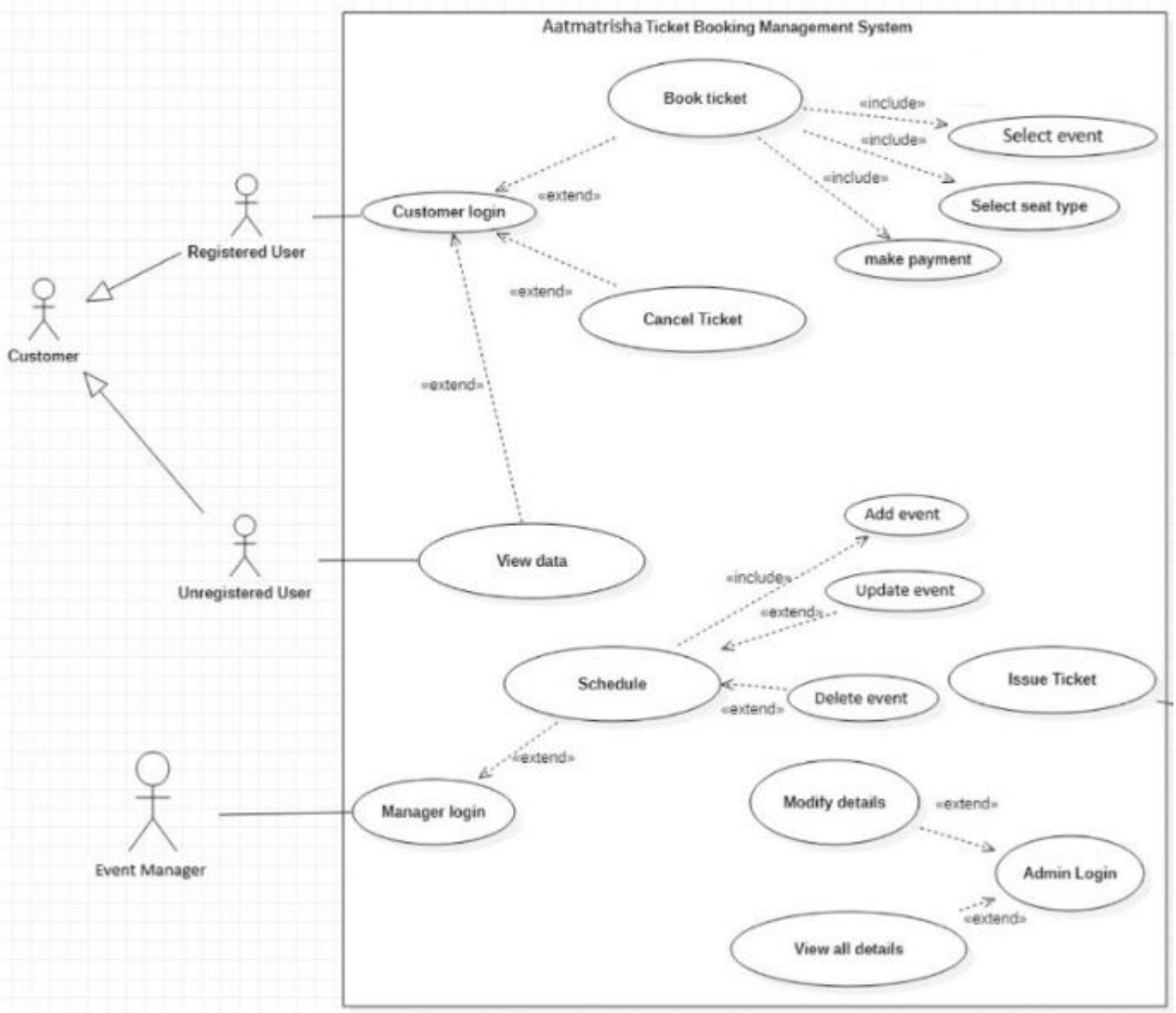
- Our architecture pattern is MVC.

- Decomposition of our project was done based on functionality.
- The below diagrams give an understanding as to how we
- designed our application and its use cases.

Class Diagram:

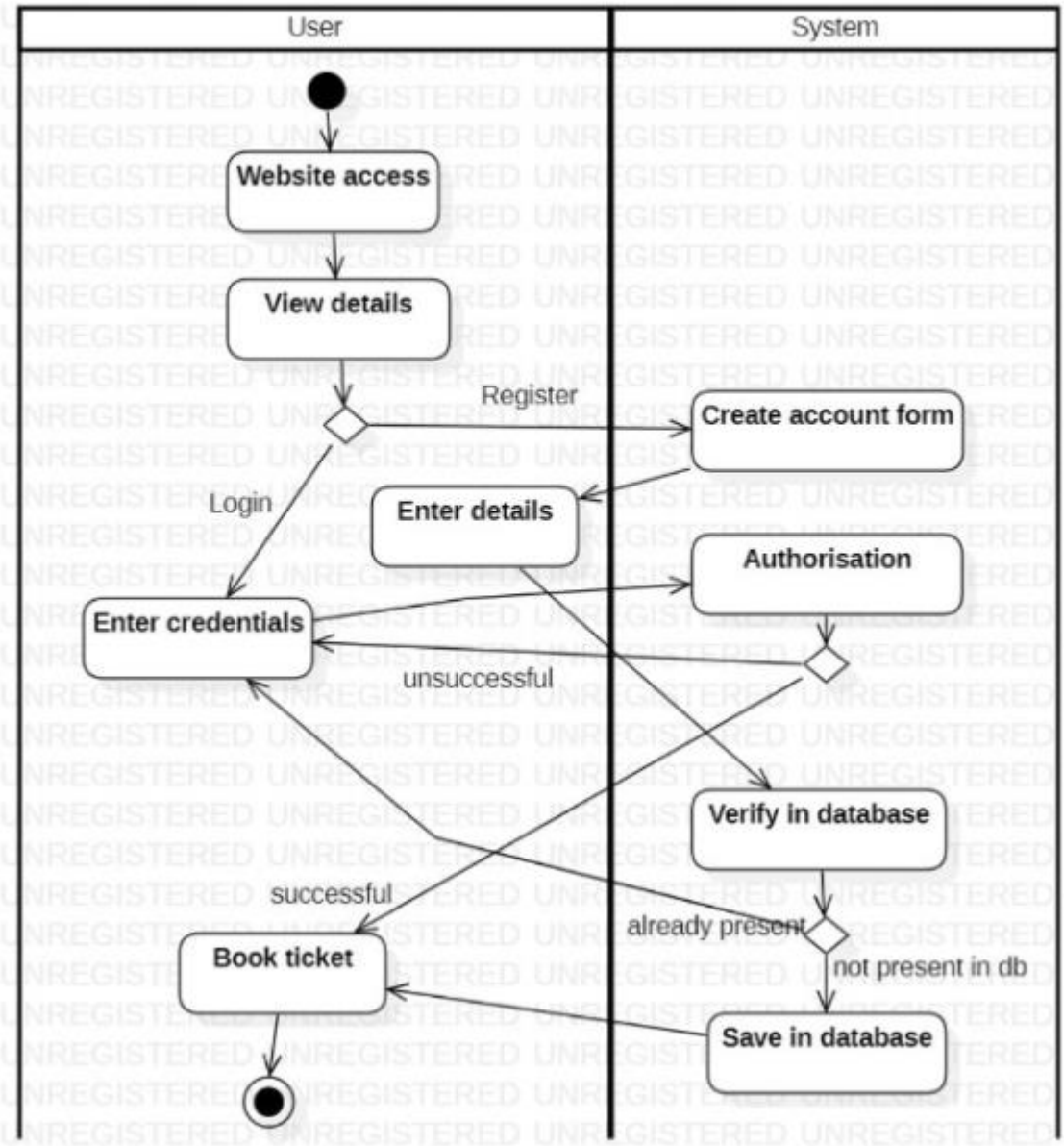


Use Cases Diagram:

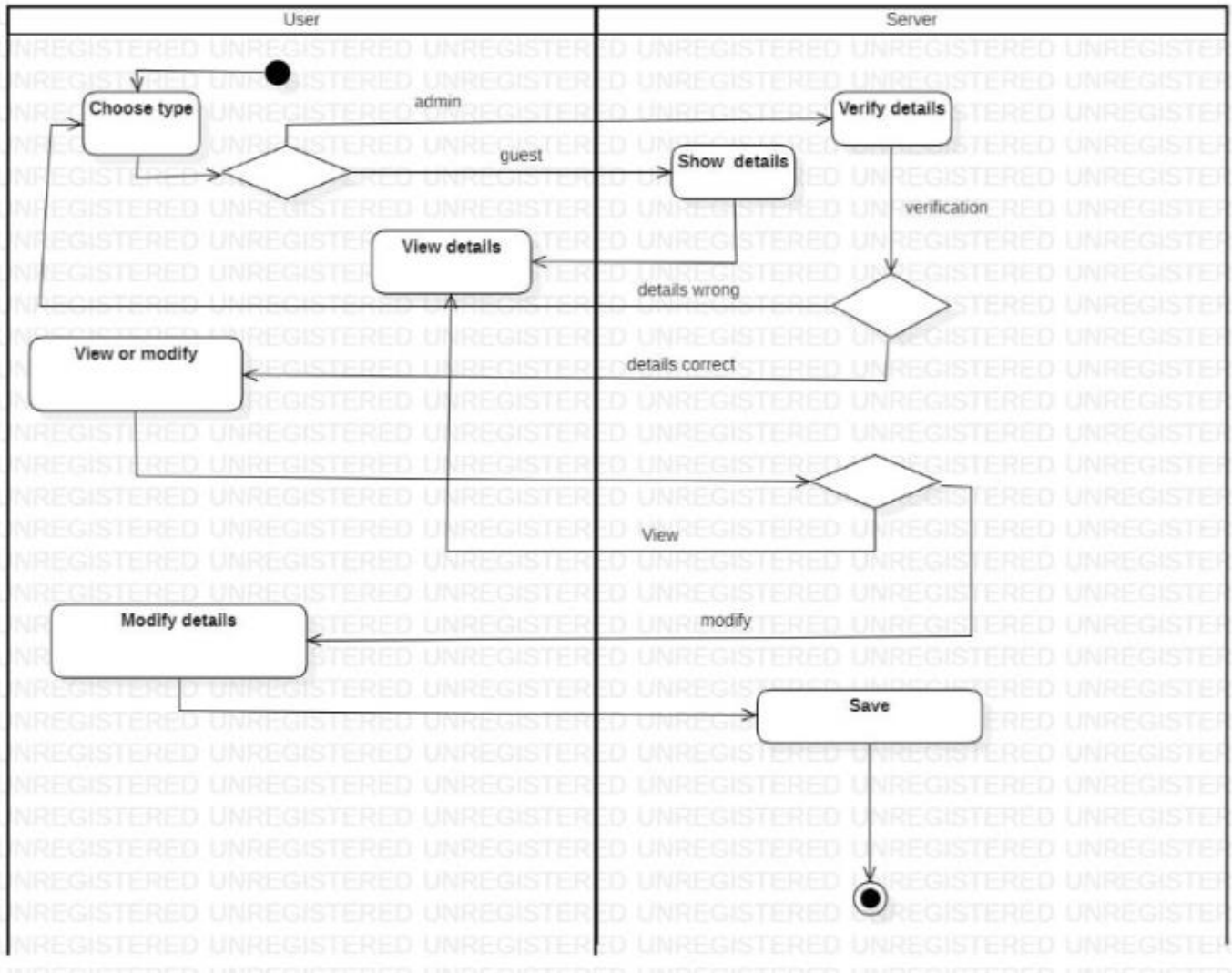


Activity Diagram:

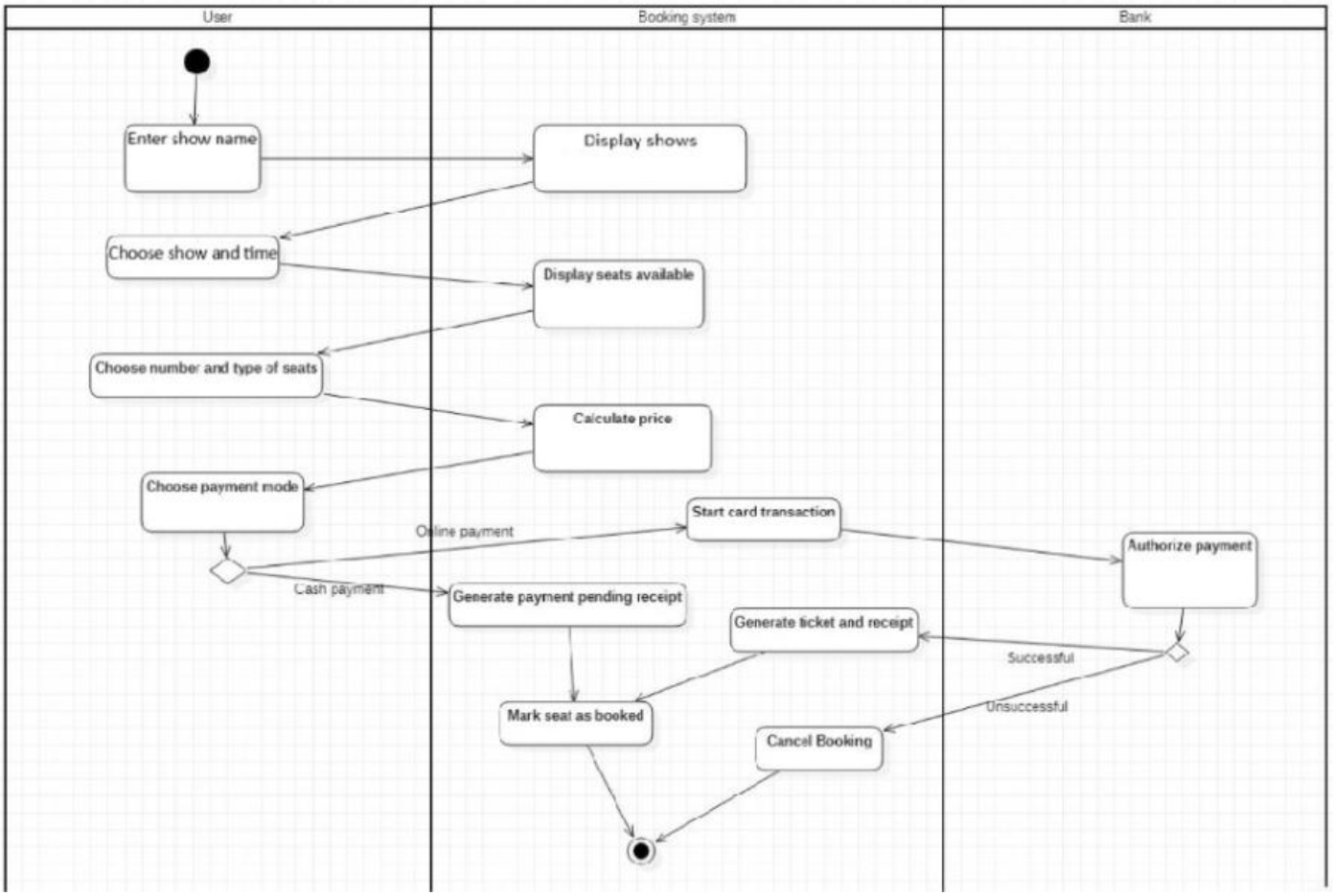
Login/Register Activity Diagram:



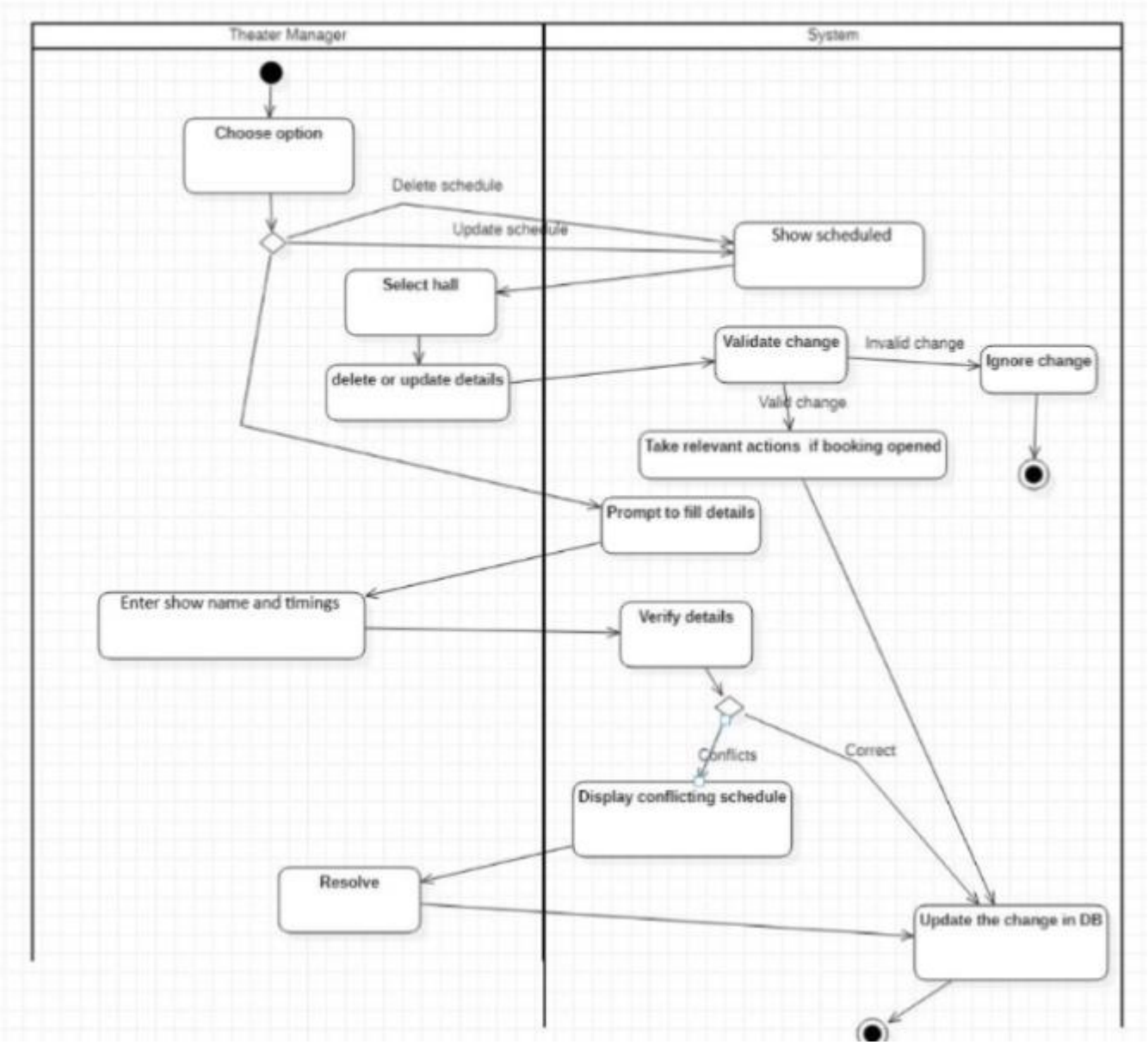
○ View/modify details Activity Diagram:



○ Book Ticket Activity Diagram:

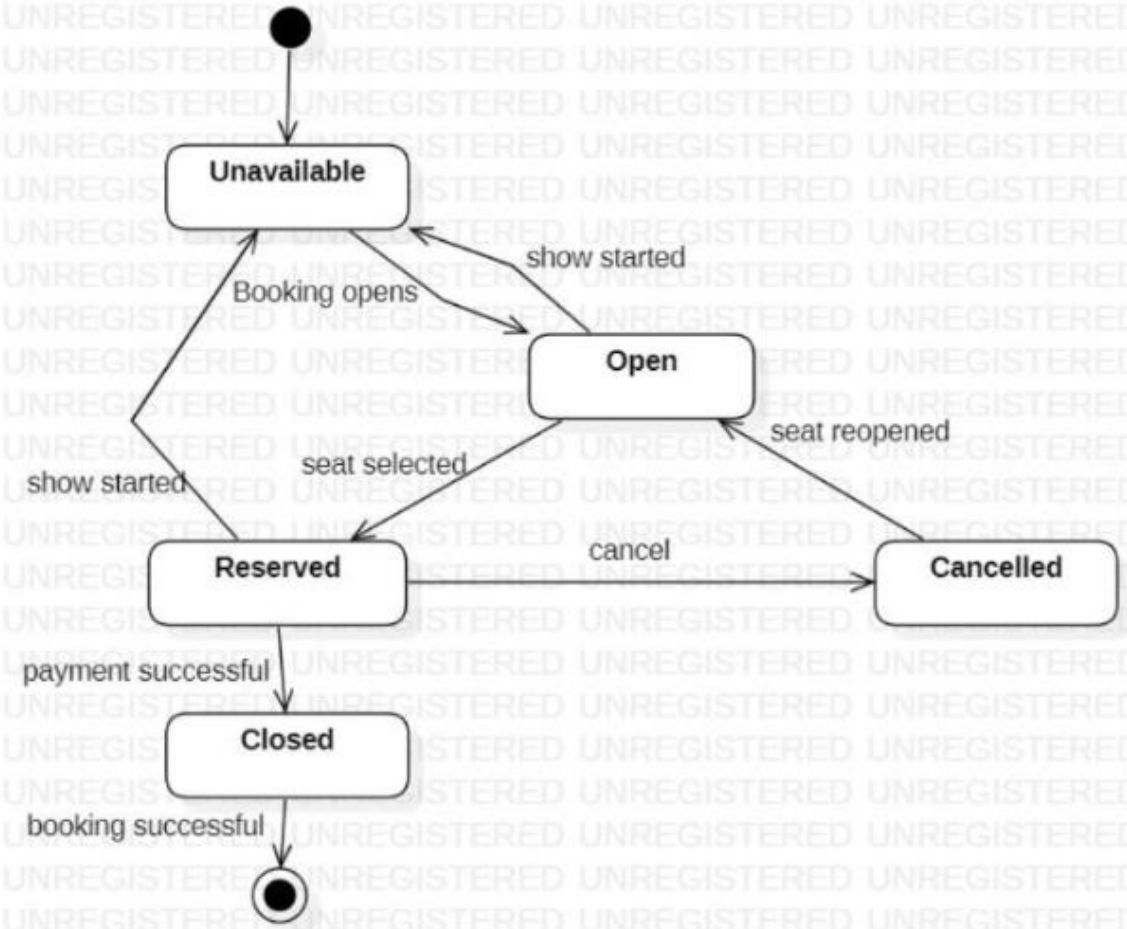


○ Schedule Activity Diagram:

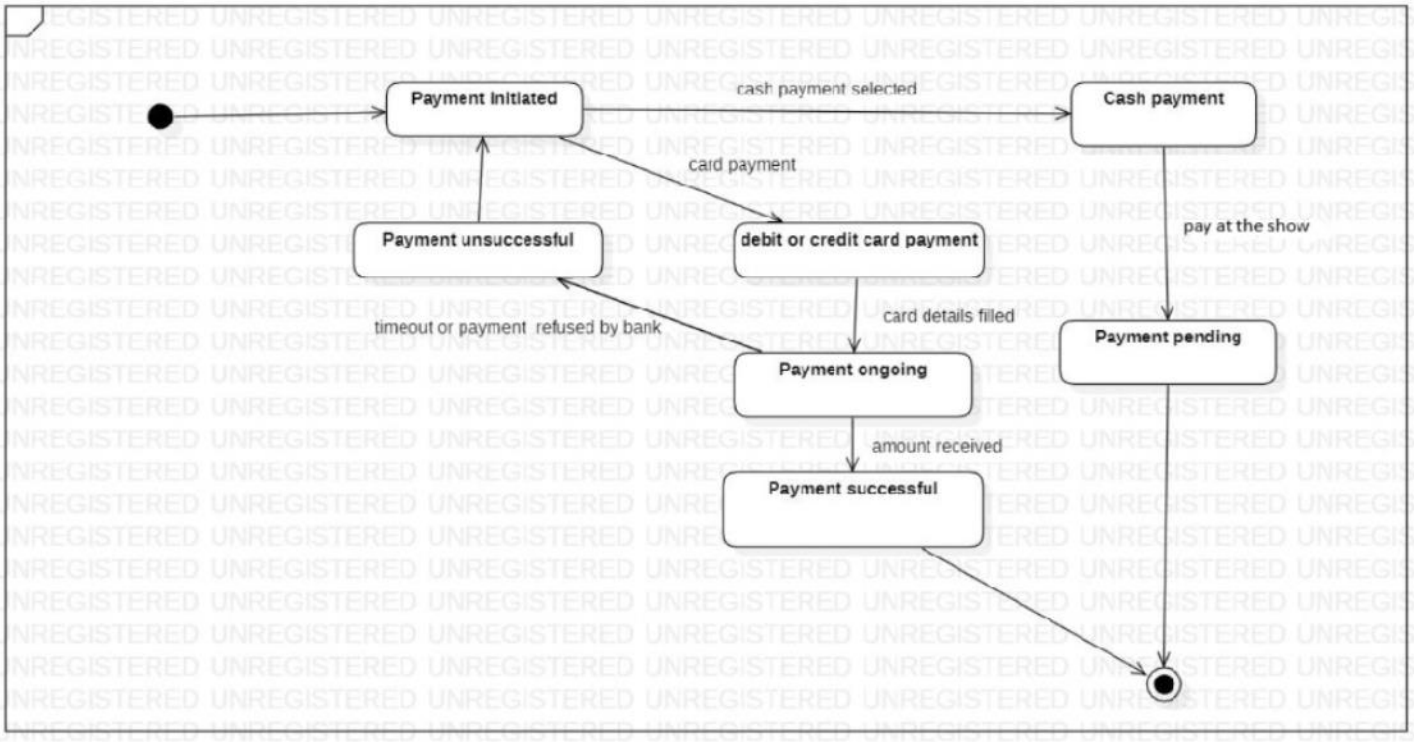


State Diagram:

○ Theatre seat State Diagram:



Payment State Diagram:



Architecture Pattern: Model-View-Controller (MVC):

- The Model-View-Controller (MVC) pattern is a popular architecture pattern used to separate concerns within an application. It divides an application into three interconnected components: Model, View, and Controller.
- Model: This component manages the data, logic, and rules of the system. In the Event Management System, the Model would handle data such as user information, event details, booking records, and reviews.
- View: The View is responsible for rendering the user interface and presenting information to the user. In this case, it would include the user and organizer interfaces for booking events, viewing schedules, and managing events.
- Controller: The Controller acts as an intermediary between the Model and the View, handling user interactions and updating the Model or View accordingly. In this system, Controllers manage operations like user registration, booking tickets, event scheduling, and user reviews.

Design Principles: SOLID

- The SOLID principles are a set of design principles that promote maintainable and scalable software design. Here is how they apply to the Event Management System:
- Single Responsibility Principle (SRP): Each class or module should have a single responsibility. For instance, the User class would handle user-specific data and operations, while the Event class would focus solely on event-related information.
- Open/Closed Principle (OCP): Software entities should be open for extension but closed for modification. The Event Management System could allow new event types or booking strategies to be added without modifying existing code.
- Liskov Substitution Principle (LSP): Derived classes must be substitutable for their base classes without altering the correctness of the program. This principle ensures that subclasses can replace parent classes without causing errors, allowing the system to be extended without introducing bugs.
- Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use. For example, the interfaces for user and organizer roles should be separate, ensuring that users do not have access to unnecessary organizer functions.

- Dependency Inversion Principle (DIP): High-level modules should not depend on low-level modules; both should depend on abstractions. In the Event Management System, interfaces would be used to decouple high-level functionalities from lower-level implementations.

Design Patterns: Factory and Builder

- Factory Pattern: The Factory pattern allows for the creation of objects without specifying the exact class to instantiate. This pattern can be used in the Event Management System to create different types of users (e.g., normal users, premium users) or different event types (e.g., concerts, conferences, workshops) based on input parameters.
- Builder Pattern: The Builder pattern is useful for constructing complex objects step-by-step. This pattern could be applied in the Event Management System for building complex event objects, allowing flexibility in setting attributes such as date, location, duration, and organizer details in a controlled manner. The Builder pattern would be particularly useful for creating event schedules, allowing organizers to customize events without violating the system's integrity.