UE21CS352B - Object-Oriented Analysis & Design using Java

Mini Project Report

" HOSPITAL MANAGEMENT SYSTEM"

Submitted by:

THEJASHWINI K	**PES1UG22CS843**
ULLASA G	**PES1UG22CS844**
AUM UPADHYAY	**PES1UG22CS845**
VANUSHREE DM	**PES1UG22CS846**

6th Semester K Section

Prof. Bargavi Mokashi

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

PROBLEM STATEMENT

The weather forecast management system is designed to provide users with live weather updates seamlessly. The application is based on the robust MVC (Model-View-Controller) architecture and is enriched with design patterns and principles to ensure a smooth user experience.

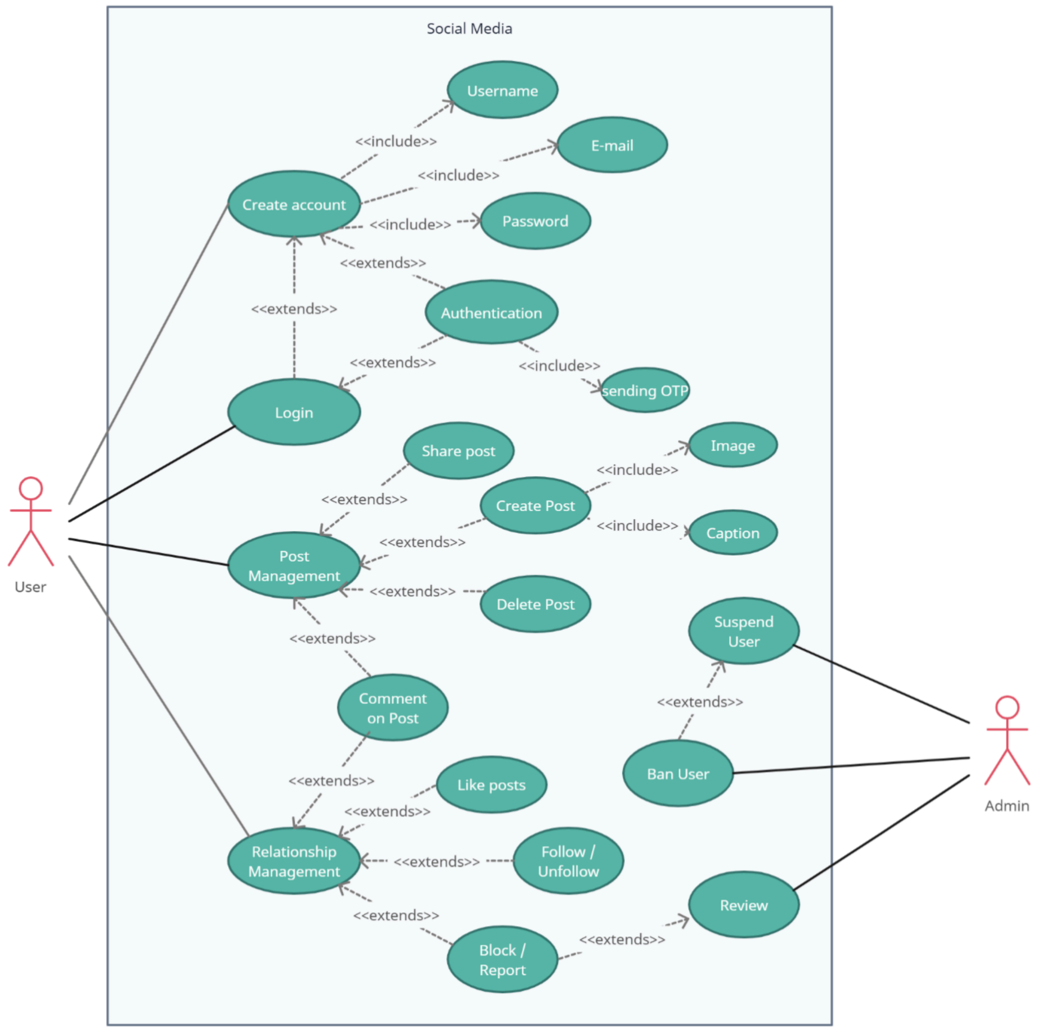
Each new user is required to register themselves and log in to begin using the application. Users then gain access to the main dashboard to view profile details and request real-time forecasts for any location worldwide. Authentication and data storage are seamlessly managed through database integration. Live weather data, including temperature, humidity, wind speed, precipitation and UV index is retrieved from the WeatherAPI.

Ultimately, the weather forecast web application serves as an indispensable resource for individuals and helps in planning daily activities, travel arrangements and outdoor activities. Its intuitive interface and comprehensive data presentation make it a reliable tool to stay updated on current weather conditions worldwide.

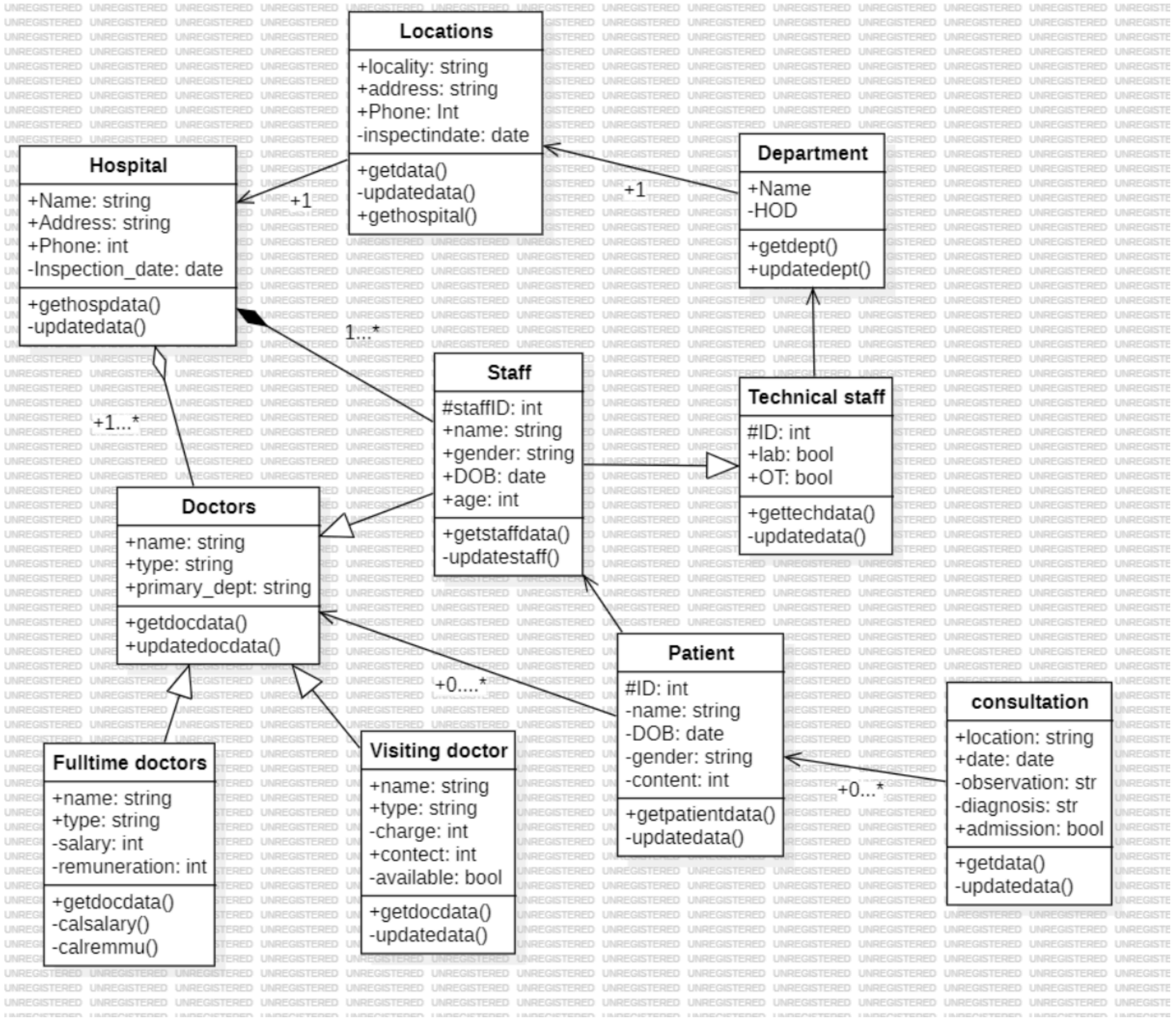
The front end of the system is designed using HTML, CSS and Javascript. MySQL is used at the backend to store user data.

MODELS

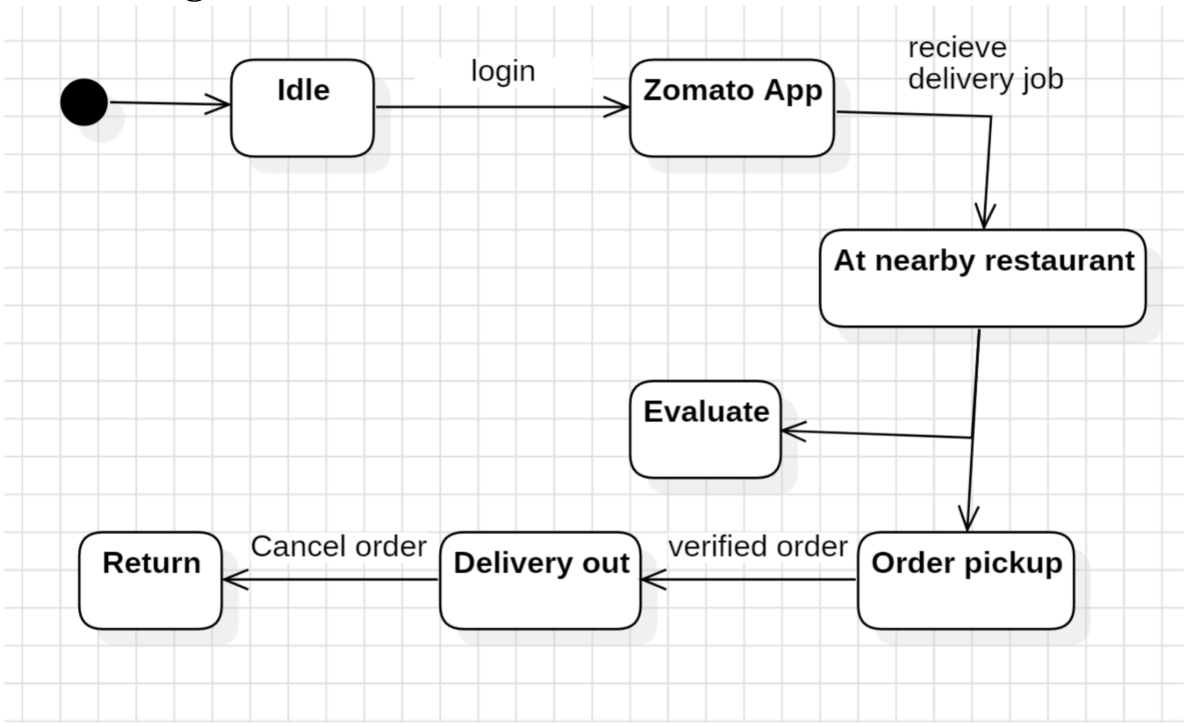
Use Case Diagram



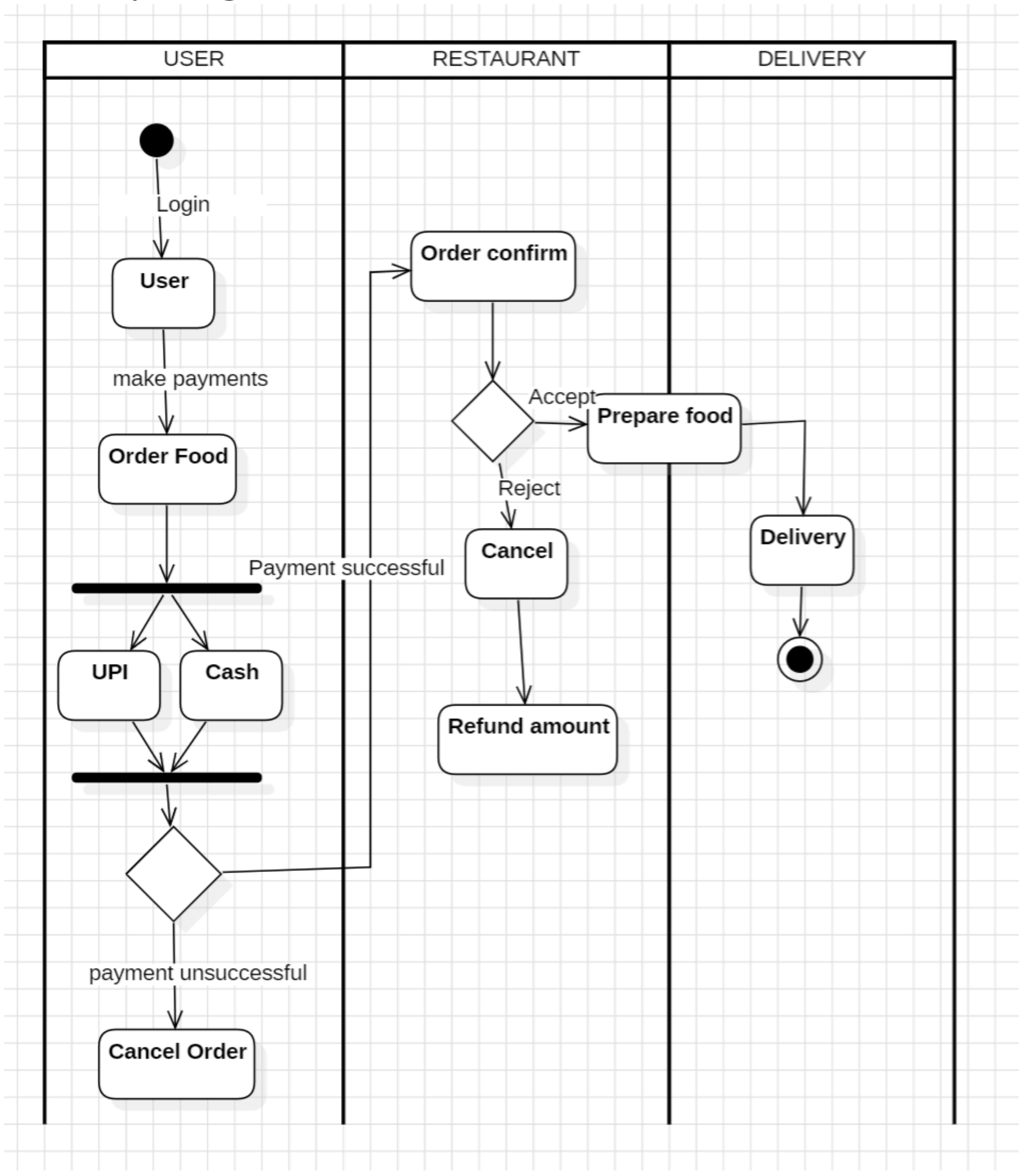
Class Diagram



State Diagram



Activity Diagram



ARCHITECTURE PATTERN

Model-View-Controller (MVC) is a widely used architectural pattern that divides an application into three interconnected components: Model, View, and Controller.
The model represents the data and business logic, the view displays the data to the user, and the controller processes user input and updates the model and view accordingly.

Use of each component for our application:

1. **Model**:

- User Model: This model will handle user authentication and authorization. It will include fields like username, password, role (patient, doctor, pharmacist, etc.), and permissions.
- Patient Model: Includes patient information such as name, contact details, medical history, etc.
- Doctor Model: Contains information about doctors such as name, specialization, availability, etc.
- Pharmacy Model: Stores information about medicines, including name, quantity, price, etc.
- Appointment Model: Tracks appointments between patients and doctors.
- Prescription Model: Represents prescriptions issued by doctors to patients.
- Ambulance Model: Manages ambulance details like availability, location, etc.

2. **View:**

- User Interface: The UI will consist of web pages or mobile app screens for different functionalities like patient registration, doctor appointment scheduling, pharmacy browsing, etc.
- Patient Dashboard: Allows patients to view their medical history, schedule appointments, consult with doctors, order medicines, etc.
- Doctor Dashboard: Provides doctors with functionalities like viewing patient appointments, writing prescriptions, updating their availability, etc.
- Pharmacy Interface: Enables pharmacists to manage their inventory, view orders, update medicine details, etc.
- Admin Dashboard: Administrators will have access to manage users,

doctors, pharmacists, ambulances, etc.

3. **Controller:**

- Authentication Controller: Handles user login, registration, and logout.
- Patient Controller: Manages patient-related operations like scheduling appointments, viewing medical history, etc.
- Doctor Controller: Handles doctor-related functionalities such as managing appointments, issuing prescriptions, etc.
- Pharmacy Controller: Controls operations related to pharmacy management like managing inventory, processing orders, etc.
- Ambulance Controller: Manages ambulance-related operations such as tracking availability, dispatching ambulances, etc.
- Admin Controller: Manages administrative tasks like user management, system configuration, etc.

DESIGN PRINCIPLES

1. Open/Closed Principle (OCP): states that a class should be open to extension but closed to change, meaning it should allow new functionality to be added without changing existing code. Our system design allows for easy expansion and modification without affecting existing functionality.

For example, new features such as additional weather attributes or users can be added to the system without changing the core components

2. Single Responsibility Principle (SRP): states that a class should have only one reason to change. In other words, a class should have only one responsibility or job within the software system. This principle emphasizes the importance of keeping classes focused, understandable, and maintainable. By adhering to the SRP, developers create classes that are more cohesive, easier to understand, and less likely to require modification when changes are made to the system.

DESIGN PATTERNS

1. Factory Design Pattern:

The Factory design pattern is a creational pattern that provides an interface for creating objects without specifying their concrete classes. It encapsulates the object creation process within a dedicated factory method, abstracting the client code from the complexities of object instantiation. This promotes code scalability, extensibility, and maintainability by decoupling the client code from the specifics of object creation.

2. Application in Hospital Management System:

In the Hospital Management System, the Factory design pattern is employed to facilitate the creation of diverse objects, such as users, doctors, and services, based on specific requirements and input parameters. This promotes modularity and flexibility within the system, allowing for seamless addition or modification of object creation logic without impacting the client code.
.

GITHUB

Code repository link:
https://github.com/Ullasgsagar563/OOAD.git

Individual contributions of the team members:

<u>Ullas G (PES1UG22CS844)</u>

Code Implementation:

- Lead the development of backend modules for the Hospital Management System, including controllers, services, DAOs, and models.
- Implement database connectivity and data access logic to interact with the database.
- Integrate frontend and backend components to ensure smooth interaction and data flow.
- Conduct testing and debugging to ensure functionality and reliability of the code.
- Document the code thoroughly, including comments and explanations for better understanding and maintainability.

<u>Aum Upadhyay (PES1UG22CS845):</u>

Frontend Design and Overall Modules Implementation:

- Lead the design and implementation of the user interface for the Hospital Management System.
- Develop frontend components using appropriate technologies (e.g., Java Swing, JavaFX, HTML/CSS/JavaScript).
- Design navigation flow and user interaction mechanisms for the frontend.
- Integrate frontend components with backend modules for seamless functionality.

<u>Vanushree DM (PES1UG22CS846):</u>

Code Implementation:

- Lead the development of backend modules for the Hospital Management System, including controllers, services, DAOs, and models.
- Implement database connectivity and data access logic to interact with the database.
- Integrate frontend and backend components to ensure smooth interaction and data flow.
- Conduct testing and debugging to ensure functionality and reliability of the code.

- Document the code thoroughly, including comments and explanations for better understanding and maintainability.

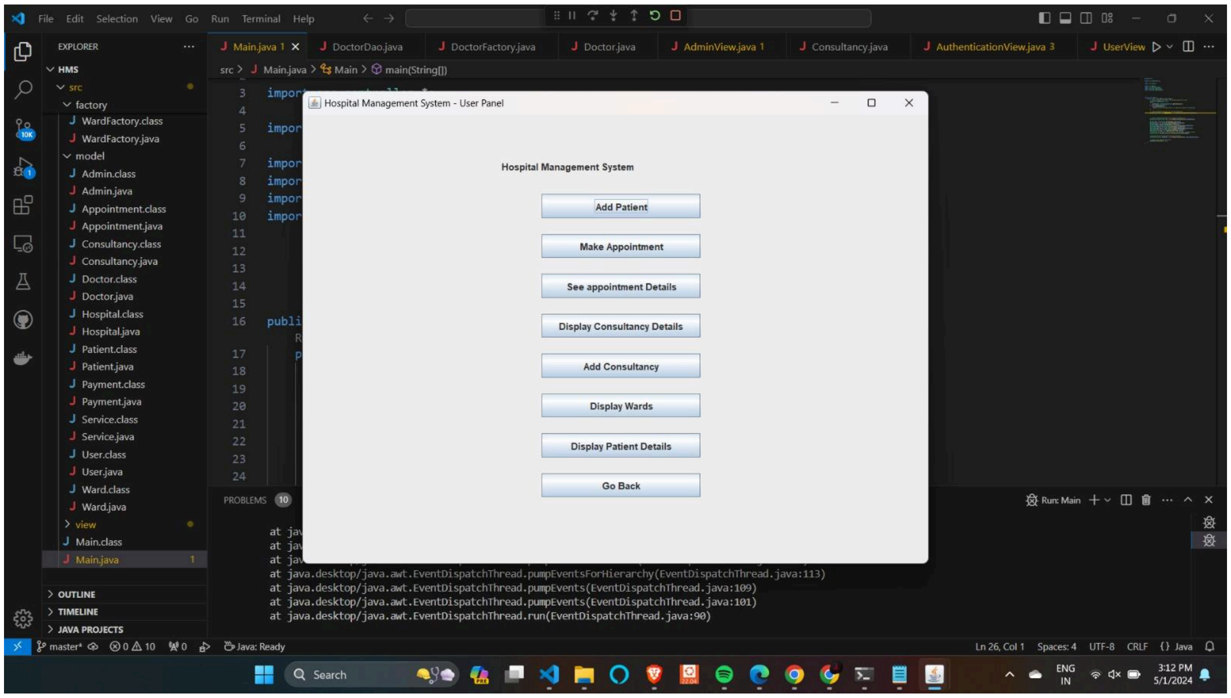
Thejaswini K (PES1UG22CS843):

Creating Documentation to Apply Factory Pattern:

- Research the Factory design pattern and its relevance in software development.
- Analyze project requirements to identify suitable areas for applying the Factory pattern.
- Document the process of applying the Factory pattern in the Hospital Management System.
- Provide examples and code snippets demonstrating the implementation of the Factory pattern in relevant modules.

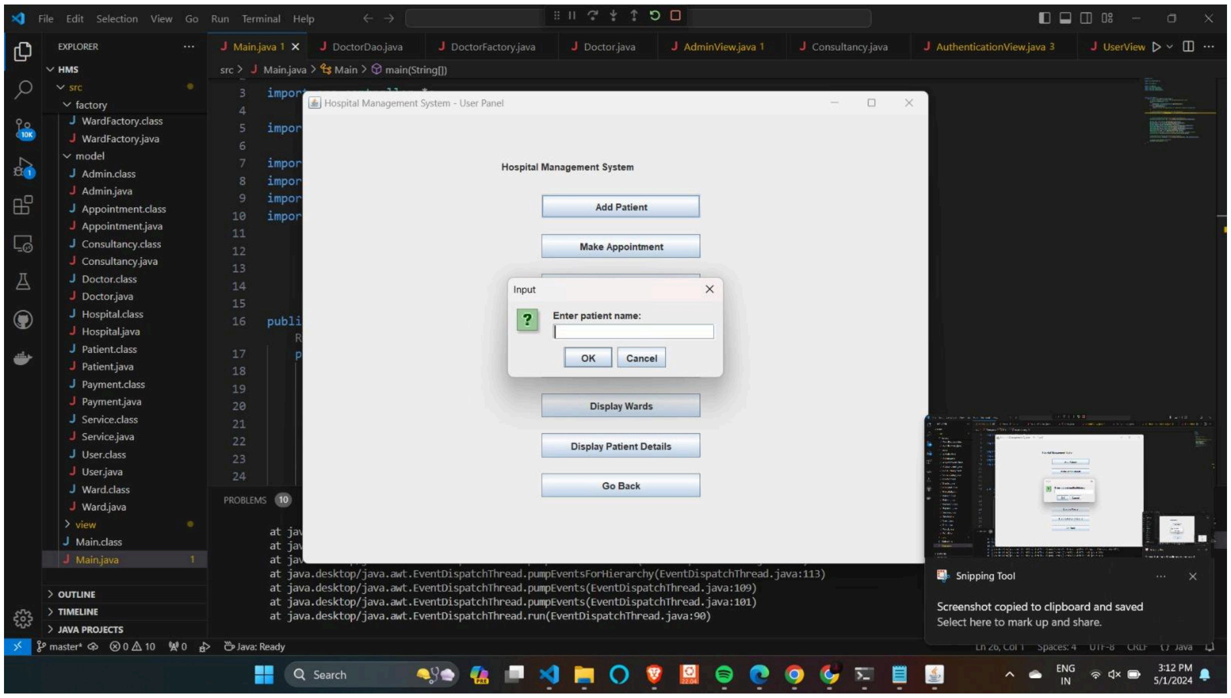
OUTPUT SCREENSHOTS

LANDING PAGE



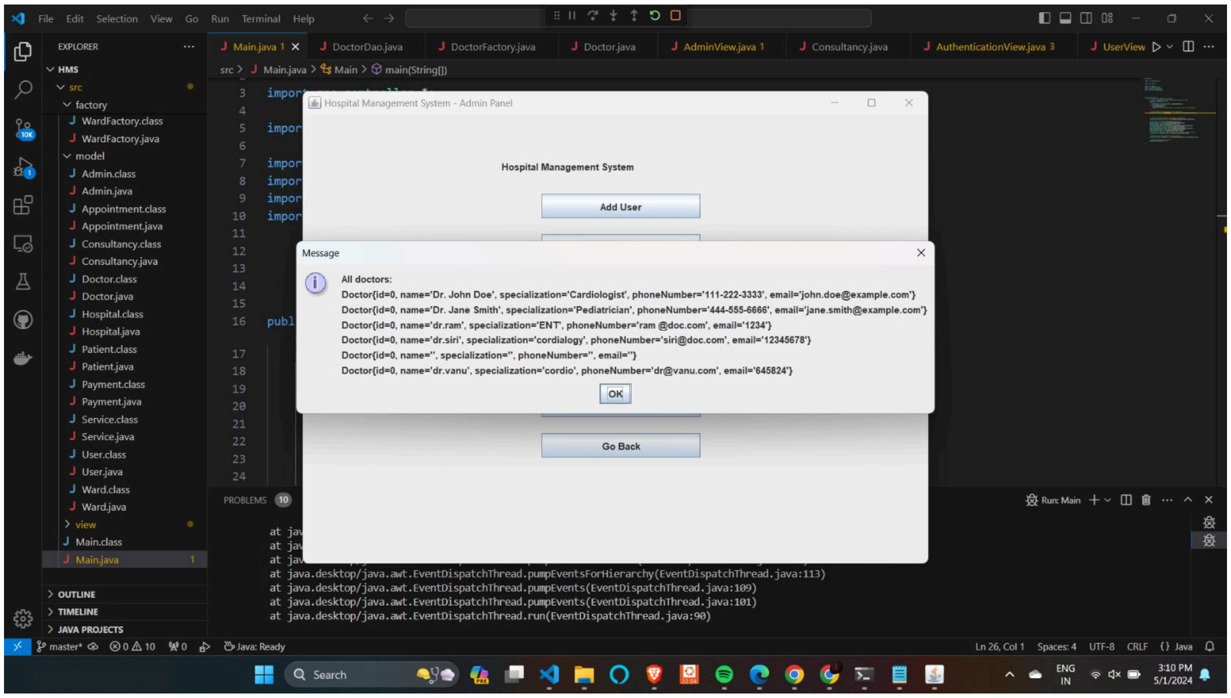
```
at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)
```

ADD A NEW PATIENT


Hospital Management System - User Panel
Hospital Management System
Add Patient
Make Appointment
Input
Enter patient name:
OK
Cancel
Display Wards
Display Patient Details
Go Back
Snipping Tool
Screenshot copied to clipboard and saved
Select here to mark up and share.

```
at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)
```

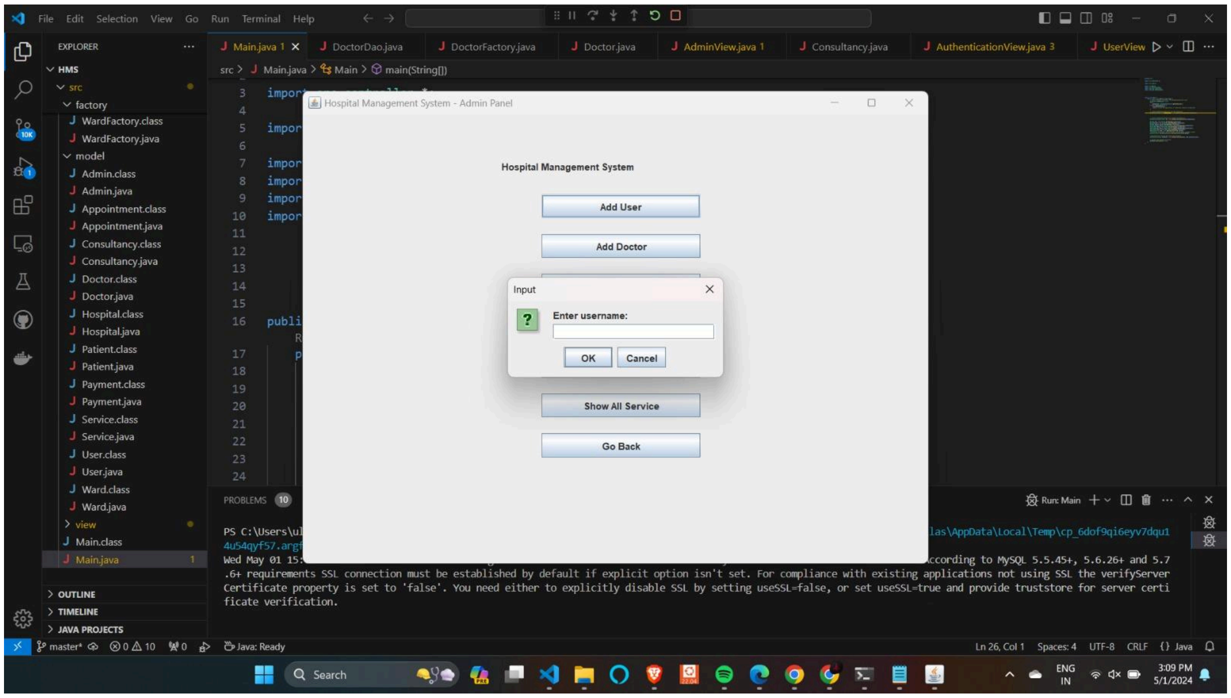
VIEW DOCTORS LIST


Hospital Management System - Admin Panel
Hospital Management System
Add User
Go Back
Message
OK

```
All doctors:
Doctor{id=0, name='Dr. John Doe', specialization='Cardiologist', phoneNumber='111-222-3333', email='john.doe@example.com'}
Doctor{id=0, name='Dr. Jane Smith', specialization='Pediatrician', phoneNumber='444-555-6666', email='jane.smith@example.com'}
Doctor{id=0, name='dr.ram', specialization='ENT', phoneNumber='ram @doc.com', email='1234'}
Doctor{id=0, name='dr.siri', specialization='cordialogy', phoneNumber='siri@doc.com', email='12345678'}
Doctor{id=0, name='', specialization='', phoneNumber='', email=''}
Doctor{id=0, name='dr.vanu', specialization='cordio', phoneNumber='dr@vanu.com', email='645824'}
```

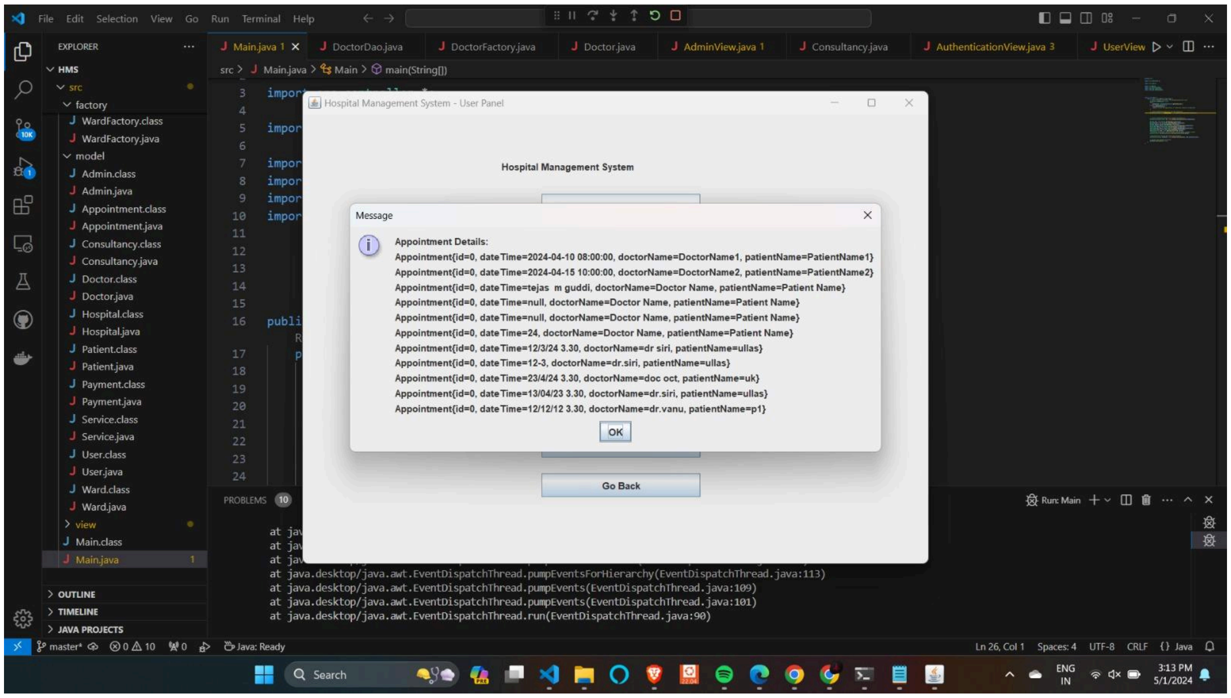
```
at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)
```

LOGIN PAGE



```
PS C:\Users\ul                                    las\AppData\Local\Temp\cp_6dof9qi6eyv7dqu1
4u54qyf57.argf
Wed May 01 15:                                    ccording to MySQL 5.5.45+, 5.6.26+ and 5.7
.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServer
Certificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certi
ficate verification.
```

APPOINTMENTS PAGE


Hospital Management System - User Panel
Hospital Management System
Message
OK
Go Back

```
Appointment Details:
Appointment{id=0, dateTime=2024-04-10 08:00:00, doctorName=DoctorName1, patientName=PatientName1}
Appointment{id=0, dateTime=2024-04-15 10:00:00, doctorName=DoctorName2, patientName=PatientName2}
Appointment{id=0, dateTime=tejas  m guddi, doctorName=Doctor Name, patientName=Patient Name}
Appointment{id=0, dateTime=null, doctorName=Doctor Name, patientName=Patient Name}
Appointment{id=0, dateTime=null, doctorName=Doctor Name, patientName=Patient Name}
Appointment{id=0, dateTime=24, doctorName=Doctor Name, patientName=Patient Name}
Appointment{id=0, dateTime=12/3/24 3.30, doctorName=dr siri, patientName=ullas}
Appointment{id=0, dateTime=12-3, doctorName=dr.siri, patientName=ullas}
Appointment{id=0, dateTime=23/4/24 3.30, doctorName=doc oct, patientName=uk}
Appointment{id=0, dateTime=13/04/23 3.30, doctorName=dr.siri, patientName=ullas}
Appointment{id=0, dateTime=12/12/12 3.30, doctorName=dr.vanu, patientName=p1}
```

```
at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)
```