Group Number: 10

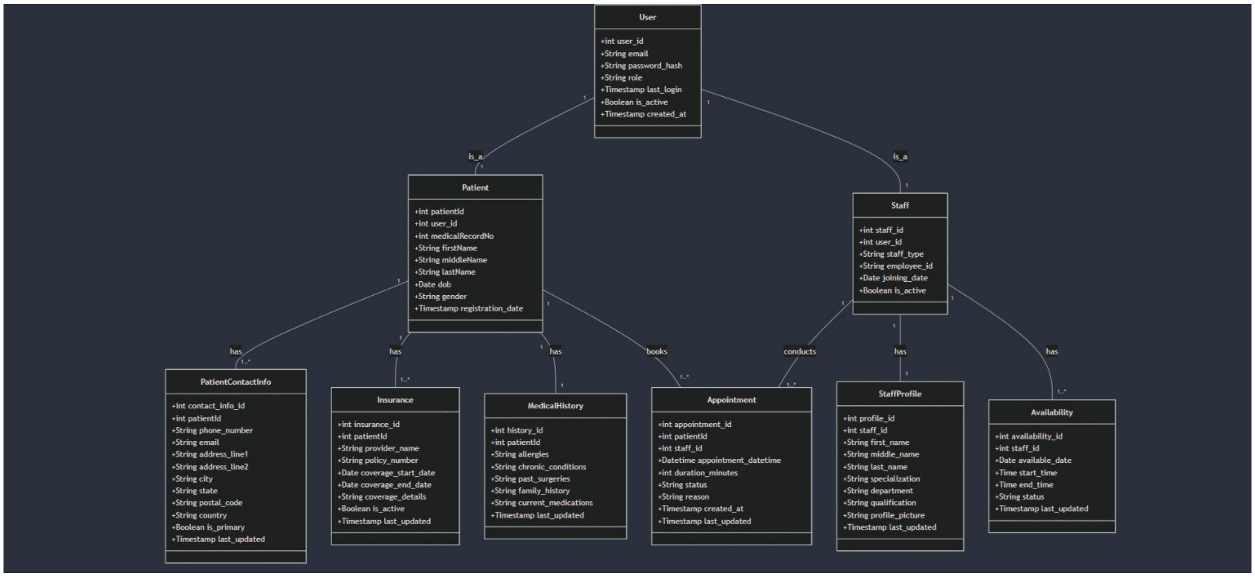
Project Topic: WellCare – Hospital Management System

Problem Statement:

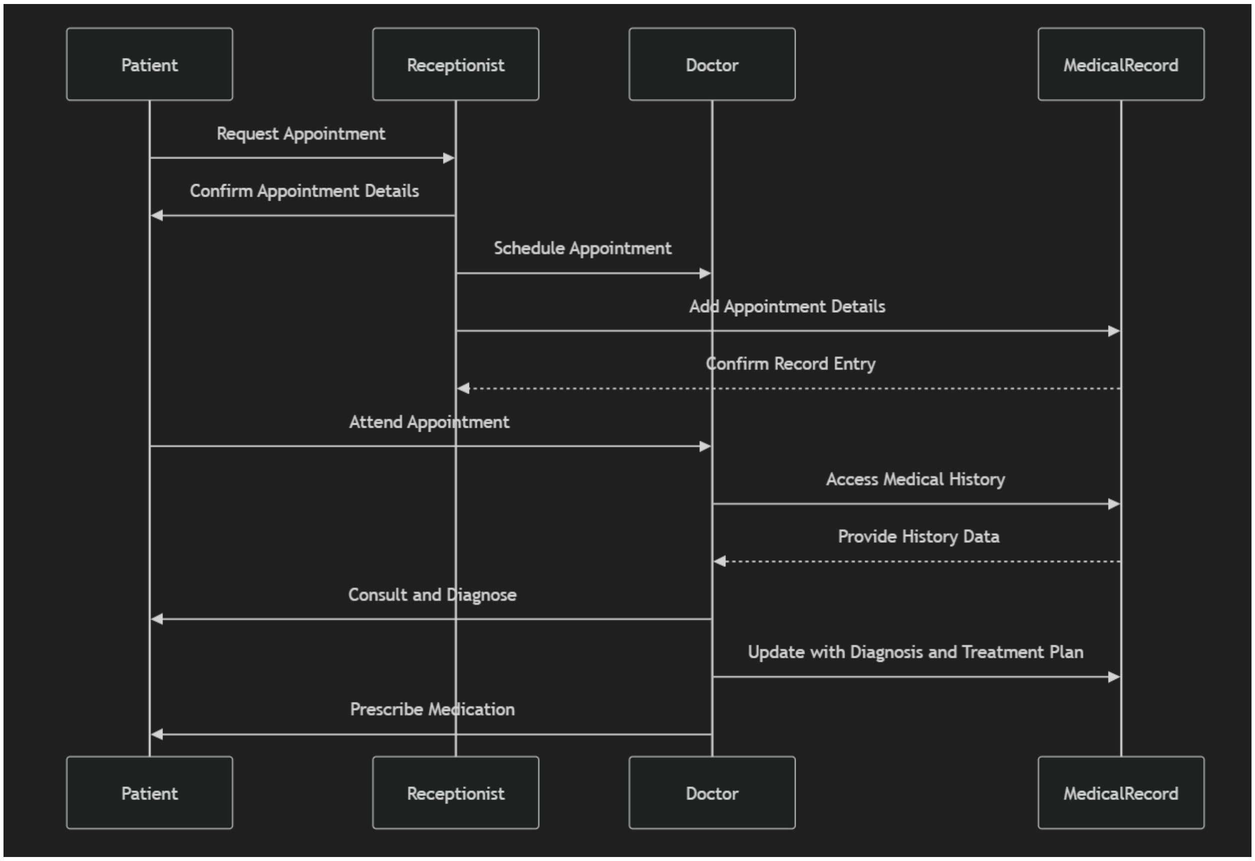
Hospitals today face increasing challenges in managing their operations efficiently due to growing patient demands, fragmented workflows, and reliance on manual processes. Issues such as inefficient appointment scheduling, lack of centralized medical records, and inadequate staff and resource management contribute to delays, errors, and diminished patient experiences. One critical improvement is displaying billing information directly on the Patients Tab, enabling patients to stay informed about their expenses, reducing confusion, and fostering transparency. This not only enhances communication but also builds trust between patients and healthcare providers. Moreover, hospitals often struggle to engage patients with timely notifications for appointments, test results, and follow-ups, further impacting care quality. The WellCare - Hospital Management System addresses these challenges by offering a comprehensive platform to streamline scheduling, centralize patient data, and improve patient engagement. By integrating modern technologies, it aims to enhance operational efficiency, optimize resource utilization, and ultimately elevate the quality of patient care.

UML and ER Diagrams

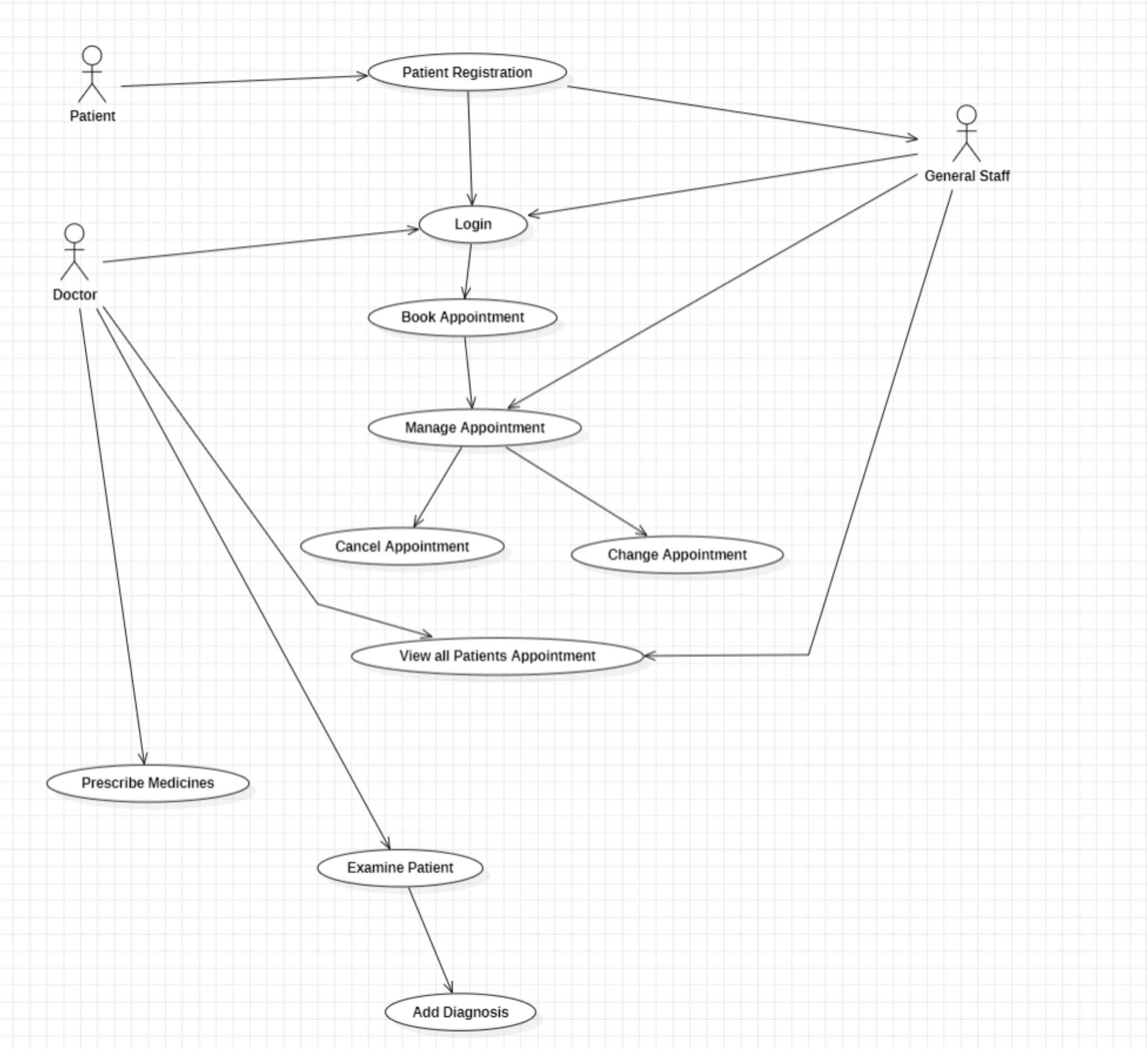
1) Class Diagram:



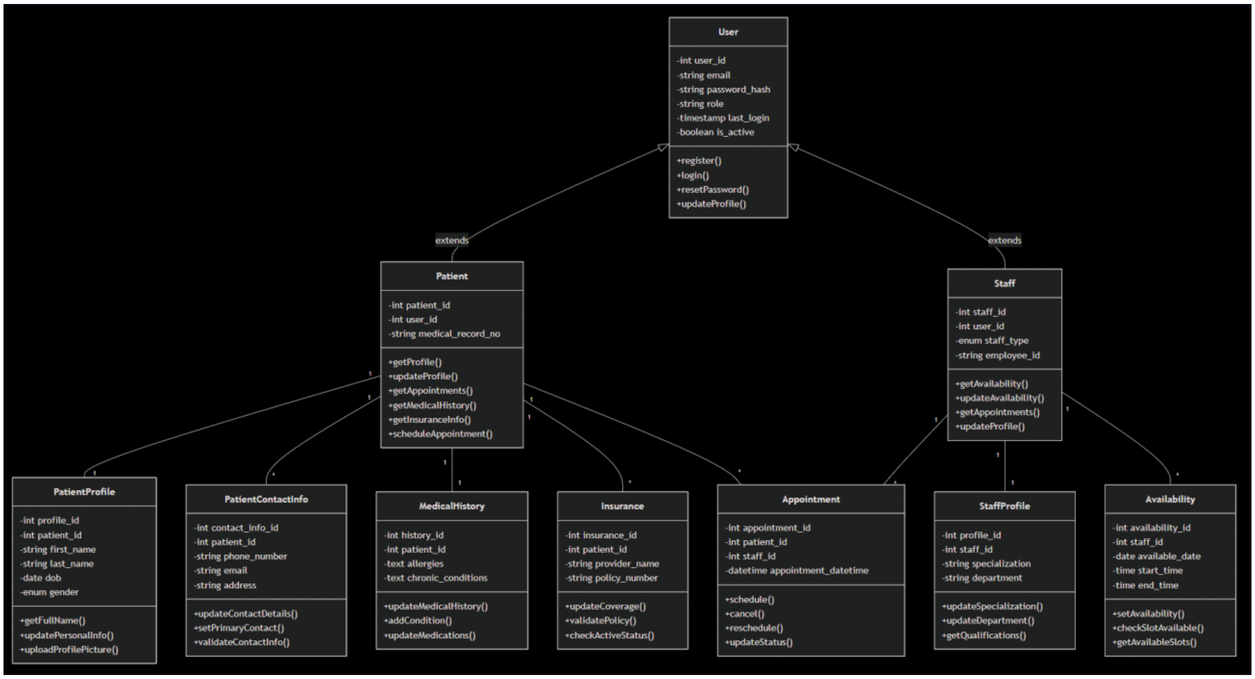
2) Sequence Diagram:



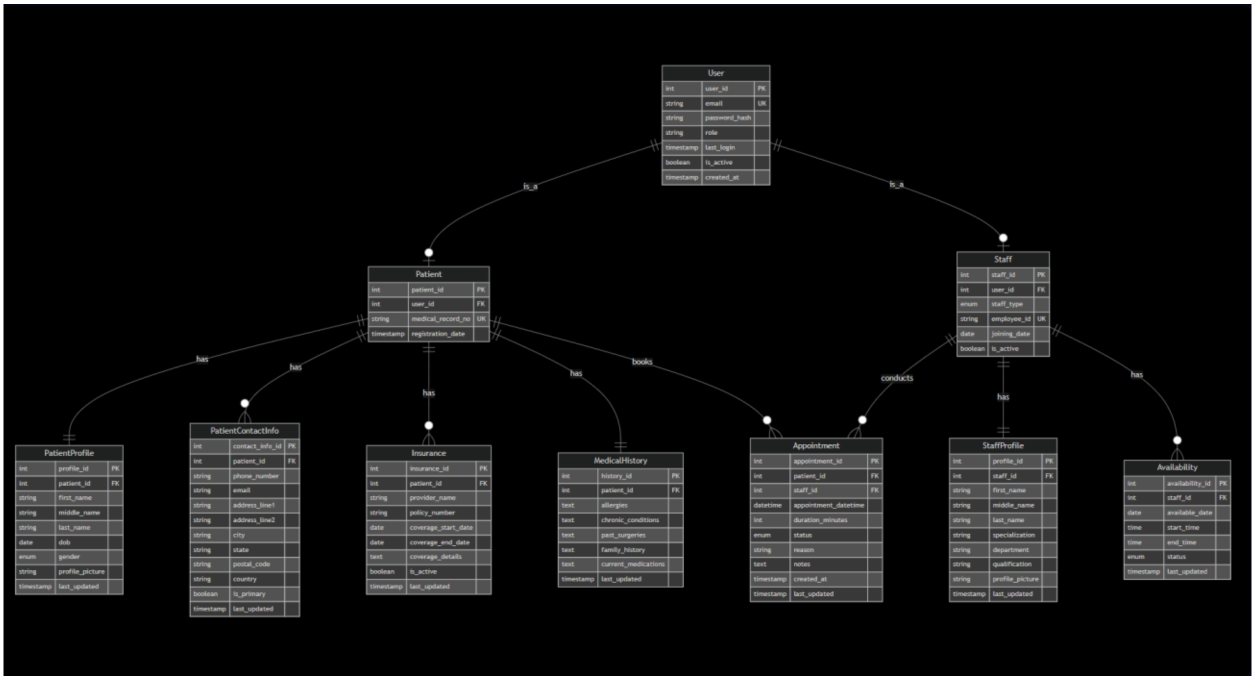
3) Use Case Diagram:



4) Object Diagram:



5) ER Diagram:



User		
int	user_id	PK
string	email	UK
string	password_hash	
string	role	
timestamp	last_login	
boolean	is_active	
timestamp	created_at	

Patient		
int	patient_id	PK
int	user_id	FK
string	medical_record_no	UK
timestamp	registration_date	

Staff		
int	staff_id	PK
int	user_id	FK
enum	staff_type	
string	employee_id	UK
date	joining_date	
boolean	is_active	

PatientProfile		
int	profile_id	PK
int	patient_id	FK
string	first_name	
string	middle_name	
string	last_name	
date	dob	
enum	gender	
string	profile_picture	
timestamp	last_updated	

PatientContactInfo		
int	contact_info_id	PK
int	patient_id	FK
string	phone_number	
string	email	
string	address_line1	
string	address_line2	
string	city	
string	state	
string	postal_code	
string	country	
boolean	is_primary	
timestamp	last_updated	

Insurance		
int	insurance_id	PK
int	patient_id	FK
string	provider_name	
string	policy_number	
date	coverage_start_date	
date	coverage_end_date	
text	coverage_details	
boolean	is_active	
timestamp	last_updated	

MedicalHistory		
int	history_id	PK
int	patient_id	FK
text	allergies	
text	chronic_conditions	
text	past_surgeries	
text	family_history	
text	current_medications	
timestamp	last_updated	

Appointment		
int	appointment_id	PK
int	patient_id	FK
int	staff_id	FK
datetime	appointment_datetime	
int	duration_minutes	
enum	status	
string	reason	
text	notes	
timestamp	created_at	
timestamp	last_updated	

StaffProfile		
int	profile_id	PK
int	staff_id	FK
string	first_name	
string	middle_name	
string	last_name	
string	specialization	
string	department	
string	qualification	
string	profile_picture	
timestamp	last_updated	

Availability		
int	availability_id	PK
int	staff_id	FK
date	available_date	
time	start_time	
time	end_time	
enum	status	
timestamp	last_updated	

Object Oriented Concepts:

1) Class and Objects
2) Interfaces
3) Encapsulation
4) Abstraction
5) Inheritance
6) Polymorphism
7) Singleton
8) Factory Design Pattern

Tech Stack:

Frontend:

- Thymeleaf, HTML, CSS, JavaScript, Bootstrap

Backend:

- Java, Spring Boot, Spring MVC, Spring Data JPA
- APIs

Database:

- MySQL
- WampServer (for data schema visualization and management)

DevOps & Tools:

- Git (version control), GitHub (repository hosting)
- Maven (build automation)
- Bruno (API testing)

Security:

- Spring Security, JWT (for authentication and session handling)

IDE:

- IntelliJ IDEA (for Java development and project management)

Functionalities that will be implemented by end of Milestone 2

- Develop CRUD APIs for various database tables.
- Integrate the developed APIs with the frontend to ensure seamless communication between the frontend and backend.
- **Division of Work Among Team Members**

1. **Nidhi Mehta -** Implement JWT for user authentication in the login API and develop CRUD APIs for the Staff table.
2. **Katha Patel** - Create CRUD APIs for Patient Insurance Details and Medical History
3. **Yashika Lodh** - Create CRUD APIs for Patient Contact Information and Appointment
4. **Saniya Azmat** - Create CRUD APIs for Patient Basic Information and Prescription

Each team member is responsible for writing clean, well-documented code and testing their APIs to ensure functionality and reliability before integration with the frontend.

Team Contributions

1. **Nidhi Mehta**
    1. Set up the development environment in alignment with the selected tech stack, ensuring all dependencies, tools, and frameworks were properly configured in each team member's device.
    2. Contributed significantly to designing and implementing the database schema, ensuring its compatibility with the project requirements.
    3. Developed the basic login API, which includes user authentication and successfully connected it with the frontend, enabling smooth user login functionality.
2. **Katha Patel**
    1. Designed the Entity-Relationship (ER) Diagram to define database structure and relationships between tables and created the Object Diagram to visually represent objects and their interactions within the system.
    2. Played a key role in building the database schema and ensuring its integration with the backend.
    3. Developed the API for inserting Patient Insurance Details.
    4. Took responsibility for managing the project workflow on GitHub by tracking tasks, managing issues, and ensuring team collaboration.
3. **Yashika Lodh**
    1. Designed the Sequence Diagram to map out the order of operations for critical workflows and the Use Case Diagram to detail system functionality from the user's perspective.
    2. Planned the Feature Flow/UI Flow for the project, ensuring the front-end user experience aligns with backend functionality.
    3. Developed the API for inserting Patient Contact Information.
4. **Saniya Azmat**
    1. Created the Class Diagram to define the structure of key system classes, their attributes, and methods, which helped guide API development.
    2. Conducted research on the requirements of a hospital management system.
    3. Developed the API for inserting Patient Basic Information.

Group Number: 10

Project Topic: WellCare – Hospital Management System

Tech Stack:

Frontend:

- Thymeleaf, HTML, CSS, JavaScript, Bootstrap

Backend:

- Java, Spring Boot, Spring MVC, Spring Data JPA
- APIs

Database:

- MySQL
- WampServer (for data schema visualization and management)

DevOps & Tools:

- Git (version control), GitHub (repository hosting)
- Maven (build automation)
- Bruno (API testing)

Security:

- Spring Security, JWT (for authentication and session handling)

IDE:

- IntelliJ IDEA (for Java development and project management)

Functionalities implemented for Milestone 2

Comprehensive CRUD Operations

CRUD (Create, Read, Update, Delete) functionality was implemented for several critical entities in the hospital management system:

- **Medical History**: Enables tracking and updating a patient's past and ongoing medical conditions.
- **Patient Insurance**: Facilitates storing and managing insurance details for billing and verification.
- **Patient Contact Information**: Manages patient communication details like phone numbers and addresses.
- **Appointments**: Allows scheduling, modifying, and canceling appointments between patients and staff.
- **Staff**: Supports staff management, including details about roles, departments, and availability.
- **Patient Basic Information**: Provides CRUD capabilities for managing general patient data such as name, gender, and date of birth.
- **Prescriptions**: Handles storing and retrieving prescription details issued to patients.

Backend-Frontend Integration

All backend APIs developed for the above entities were seamlessly integrated with the frontend. This ensures real-time functionality and enables a smooth user experience. Users can now interact with the system's key modules through intuitive UI components backed by dynamic server operations.

JWT-Based Authentication

A secure login mechanism was implemented using JSON Web Tokens (JWT), ensuring:

- **Authentication**: Verifies user identity with secure token-based login.

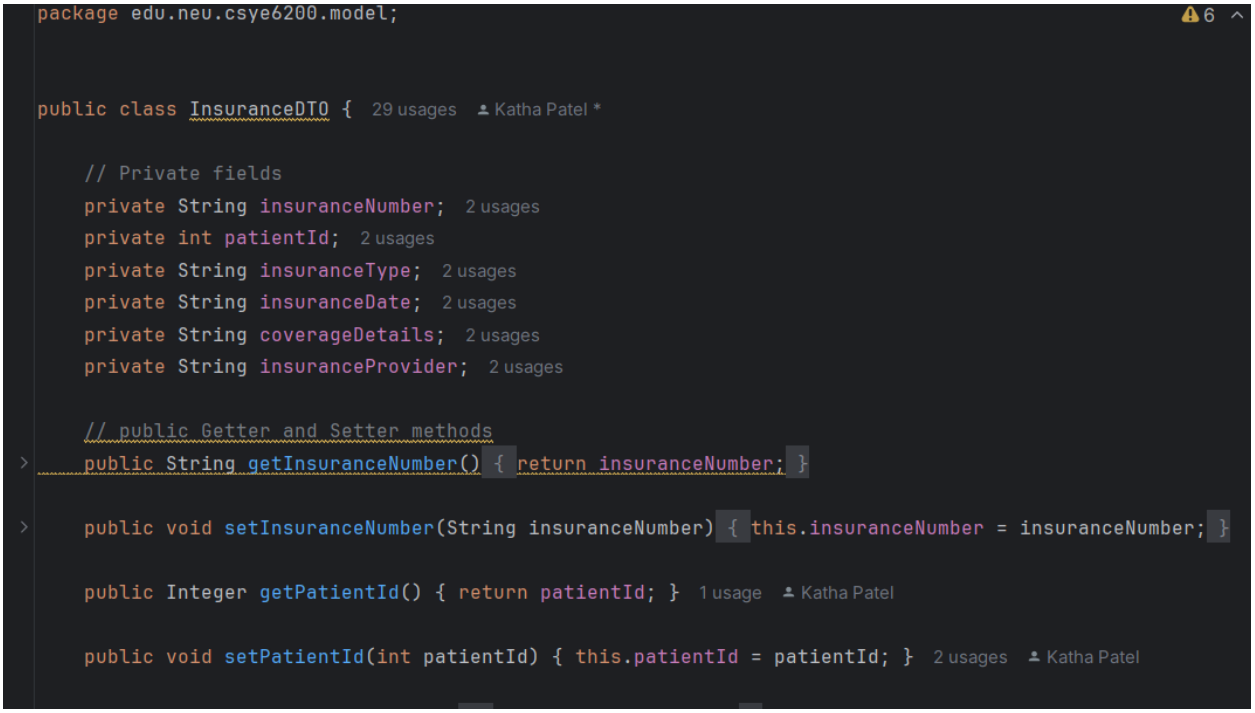
- **Role-Based Redirection**: Directs users to appropriate dashboards (e.g., staff or patient) based on their role, enhancing usability and workflow management.

These features collectively improve security, provide efficient management of hospital data, and offer a user-friendly interface.

Object Oriented Concepts Implemented

1. Encapsulation

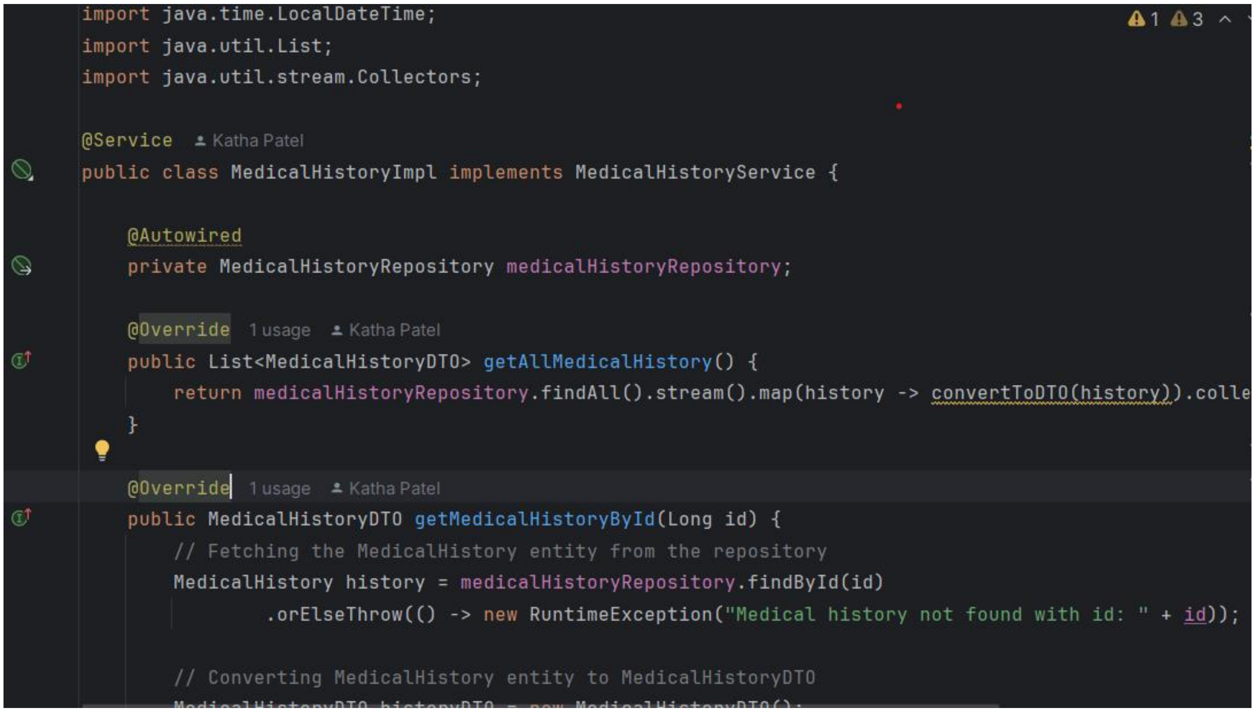
- Hiding the internal state of an object and only exposing necessary components through getters and setters
- Usage: Fields in InsuranceDTO are private and accessed via getters and setters, ensuring encapsulation.



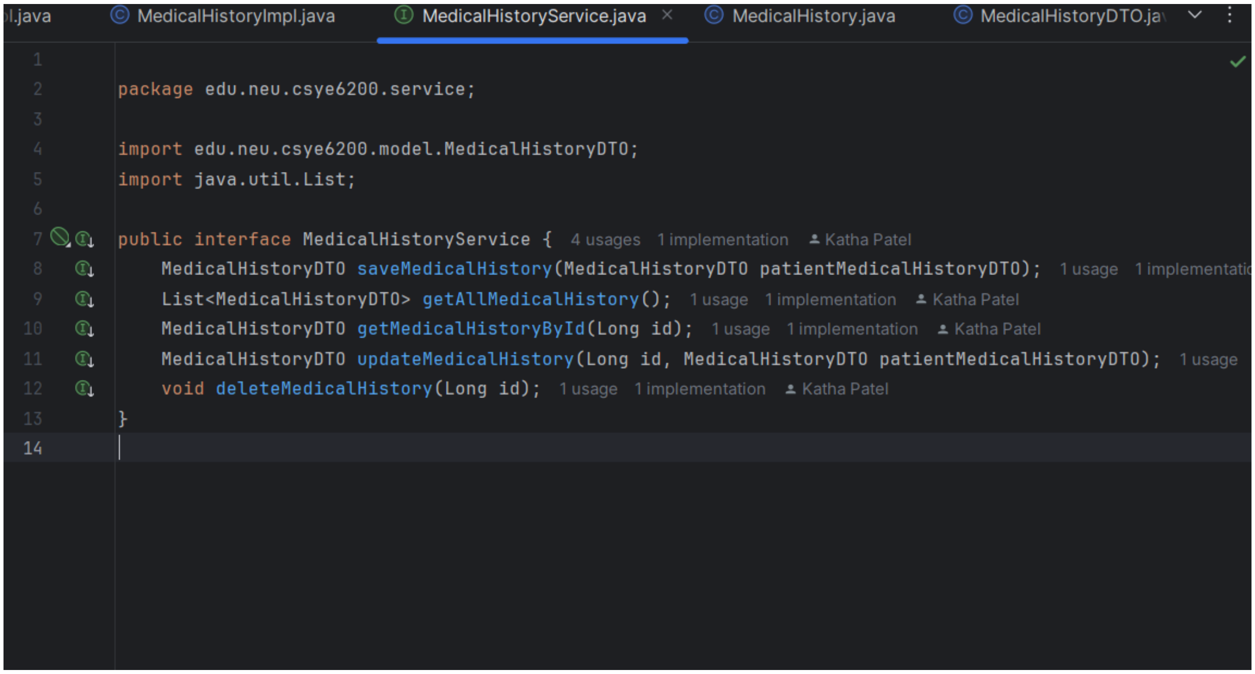
```java
package edu.neu.csye6200.model;


public class InsuranceDTO {  29 usages  Katha Patel *

    // Private fields
    private String insuranceNumber;  2 usages
    private int patientId;  2 usages
    private String insuranceType;  2 usages
    private String insuranceDate;  2 usages
    private String coverageDetails;  2 usages
    private String insuranceProvider;  2 usages

    // public Getter and Setter methods
    public String getInsuranceNumber() { return insuranceNumber; }

    public void setInsuranceNumber(String insuranceNumber) { this.insuranceNumber = insuranceNumber; }

    public Integer getPatientId() { return patientId; }  1 usage  Katha Patel

    public void setPatientId(int patientId) { this.patientId = patientId; }  2 usages  Katha Patel
```

2. Abstraction

- Hiding implementation details and exposing only essential features.
- Usage: The MedicalHistoryService interface hides the implementation of logic provided in MedicalHistoryImpl.



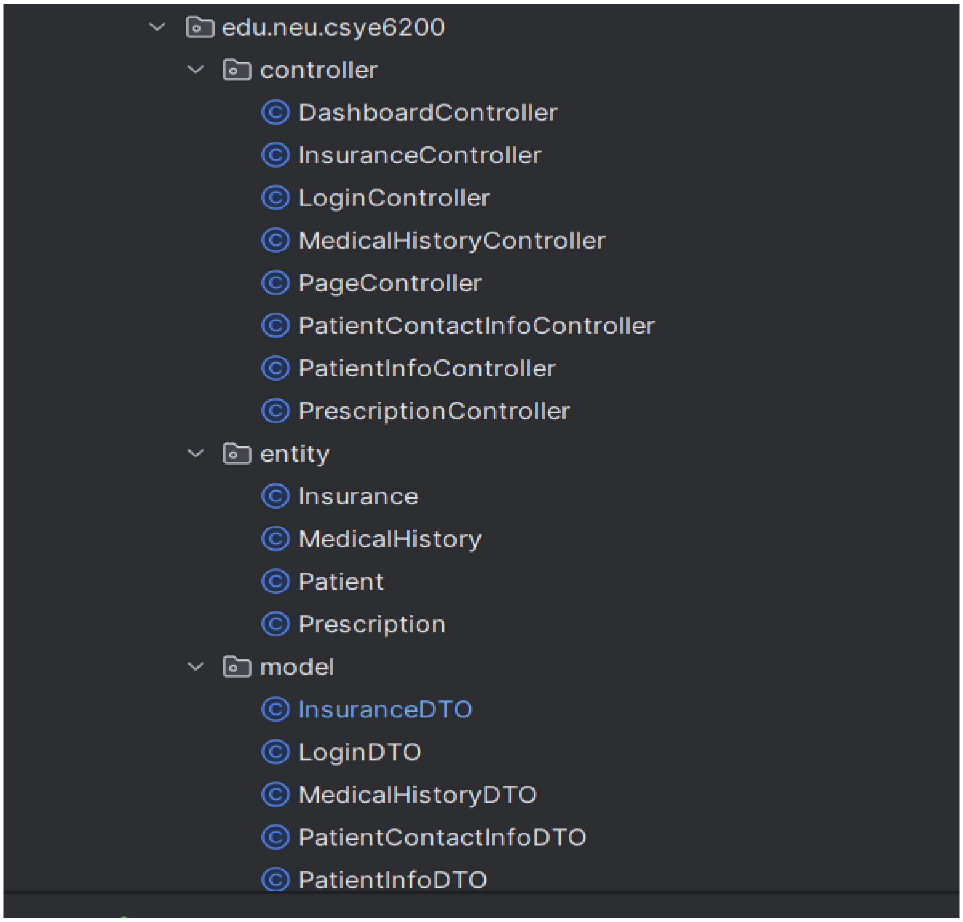
```
import java.time.LocalDateTime;
import java.util.List;
import java.util.stream.Collectors;

@Service  Katha Patel
public class MedicalHistoryImpl implements MedicalHistoryService {

    @Autowired
    private MedicalHistoryRepository medicalHistoryRepository;

    @Override  1 usage  Katha Patel
    public List<MedicalHistoryDTO> getAllMedicalHistory() {
        return medicalHistoryRepository.findAll().stream().map(history -> convertToDTO(history)).colle
    }

    @Override  1 usage  Katha Patel
    public MedicalHistoryDTO getMedicalHistoryById(Long id) {
        // Fetching the MedicalHistory entity from the repository
        MedicalHistory history = medicalHistoryRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Medical history not found with id: " + id));

        // Converting MedicalHistory entity to MedicalHistoryDTO
        MedicalHistoryDTO historyDTO = new MedicalHistoryDTO();
```



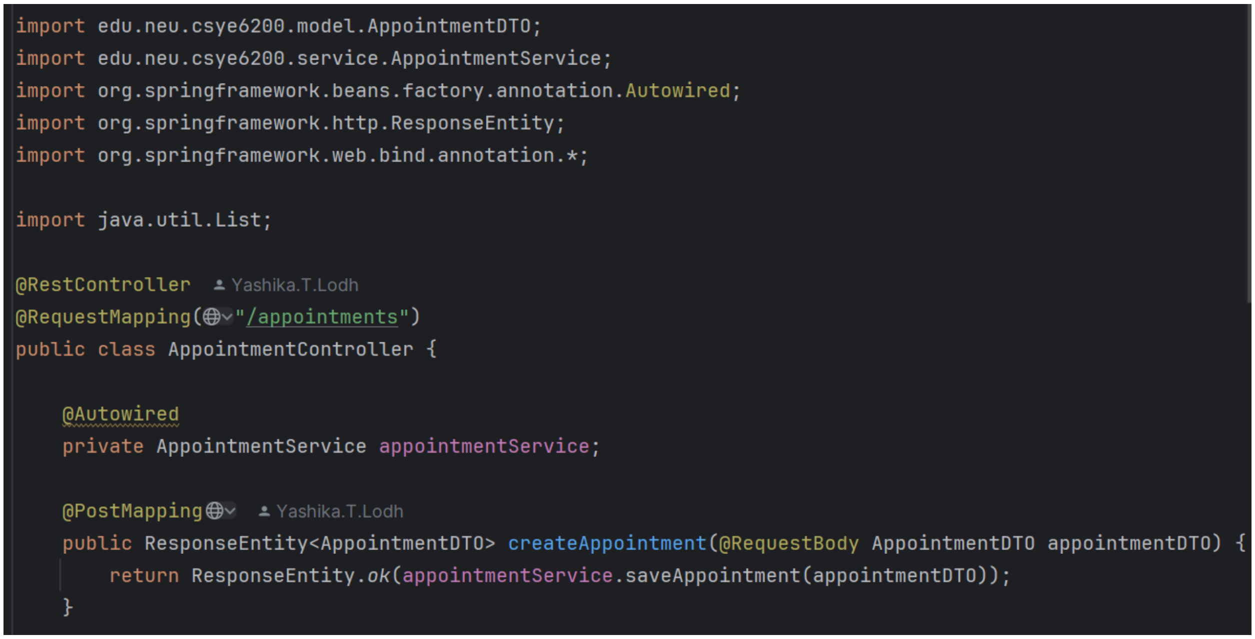
```

package edu.neu.csye6200.service;

import edu.neu.csye6200.model.MedicalHistoryDTO;
import java.util.List;

public interface MedicalHistoryService {  4 usages  1 implementation  Katha Patel
    MedicalHistoryDTO saveMedicalHistory(MedicalHistoryDTO patientMedicalHistoryDTO);  1 usage  1 implementatic
    List<MedicalHistoryDTO> getAllMedicalHistory();  1 usage  1 implementation  Katha Patel
    MedicalHistoryDTO getMedicalHistoryById(Long id);  1 usage  1 implementation  Katha Patel
    MedicalHistoryDTO updateMedicalHistory(Long id, MedicalHistoryDTO patientMedicalHistoryDTO);  1 usage
    void deleteMedicalHistory(Long id);  1 usage  1 implementation  Katha Patel
}

```

3. Class and Objects

- A class is a blueprint for creating objects. An object is an instance of a class.
- Classes such as LoginController, InsuranceController, MedicalHistoryContoller, PatientInfoController, PatientContactInfoDTO and LoginDTO define the structure and behavior of the application.



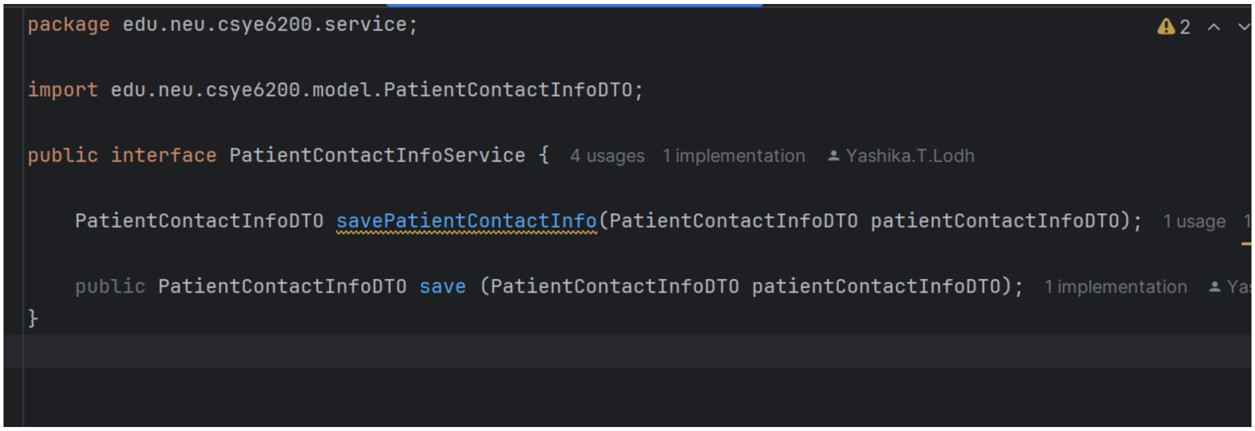
```
edu.neu.csye6200
    controller
        DashboardController
        InsuranceController
        LoginController
        MedicalHistoryController
        PageController
        PatientContactInfoController
        PatientInfoController
        PrescriptionController
    entity
        Insurance
        MedicalHistory
        Patient
        Prescription
    model
        InsuranceDTO
        LoginDTO
        MedicalHistoryDTO
        PatientContactInfoDTO
        PatientInfoDTO
```



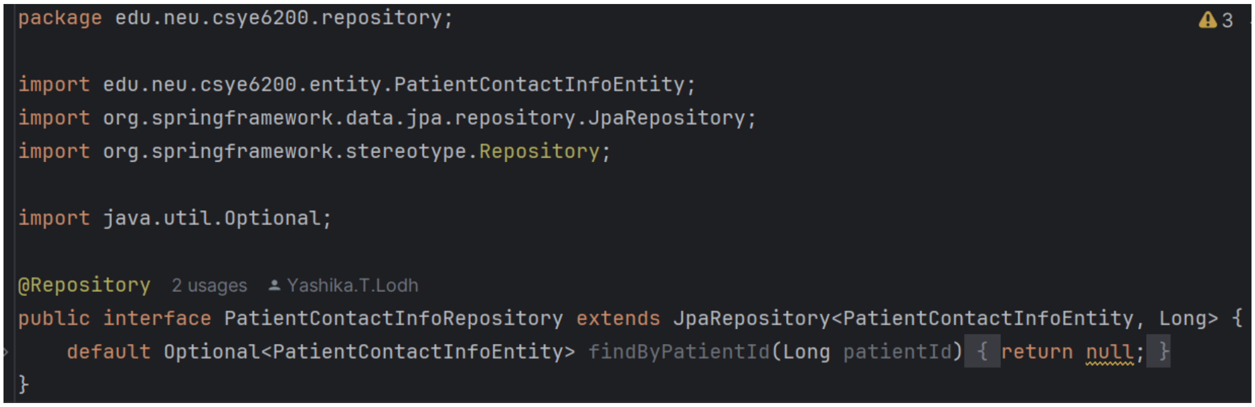
```java
import edu.neu.csye6200.model.AppointmentDTO;
import edu.neu.csye6200.service.AppointmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController  Yashika.T.Lodh
@RequestMapping("/appointments")
public class AppointmentController {

    @Autowired
    private AppointmentService appointmentService;

    @PostMapping  Yashika.T.Lodh
    public ResponseEntity<AppointmentDTO> createAppointment(@RequestBody AppointmentDTO appointmentDTO) {
        return ResponseEntity.ok(appointmentService.saveAppointment(appointmentDTO));
    }
```

4. Interfaces

- An interface defines a contract that implementing classes must adhere to, without providing implementation details.
- PatientContactInfoService and PatientContactInfoRespository are interfaces. They abstract business logic and database operations, respectively.



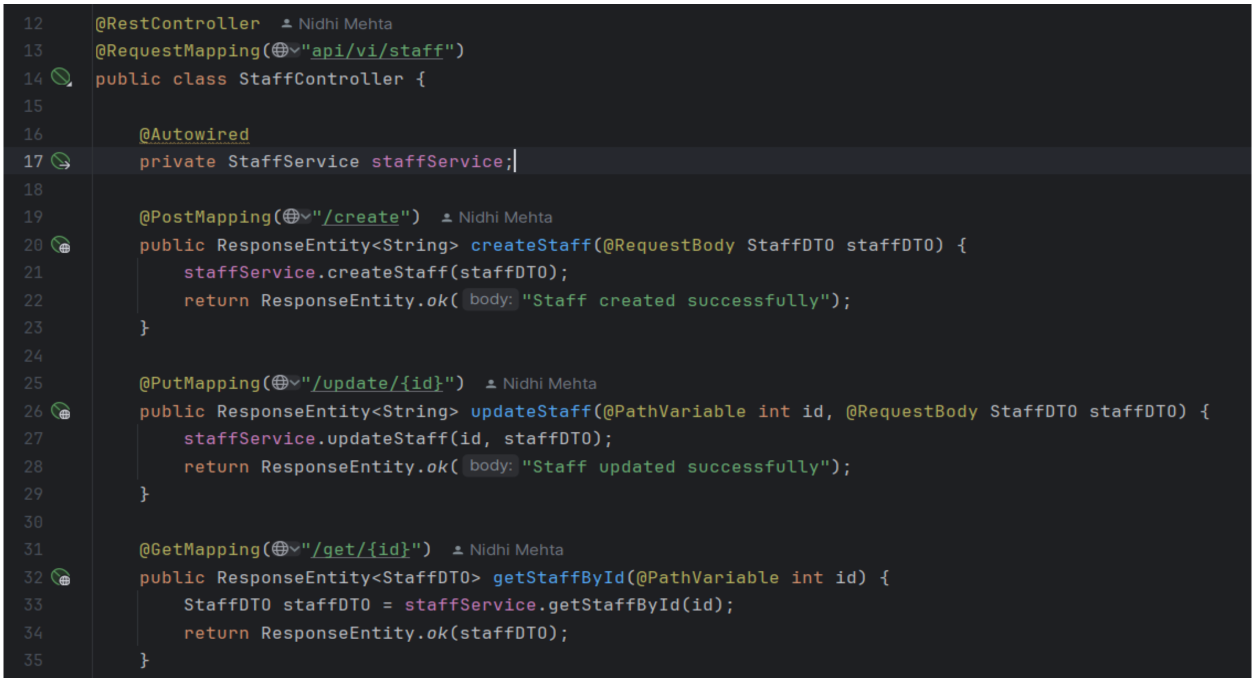
```
package edu.neu.csye6200.service;                                                                          ⚠2

import edu.neu.csye6200.model.PatientContactInfoDTO;

public interface PatientContactInfoService {  4 usages  1 implementation  Yashika.T.Lodh

    PatientContactInfoDTO savePatientContactInfo(PatientContactInfoDTO patientContactInfoDTO);  1 usage  1

    public PatientContactInfoDTO save (PatientContactInfoDTO patientContactInfoDTO);  1 implementation  Ya
}
```



```
package edu.neu.csye6200.repository;                                                                       ⚠3

import edu.neu.csye6200.entity.PatientContactInfoEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository  2 usages  Yashika.T.Lodh
public interface PatientContactInfoRepository extends JpaRepository<PatientContactInfoEntity, Long> {
    default Optional<PatientContactInfoEntity> findByPatientId(Long patientId) { return null; }
}
```

5. Inheritance

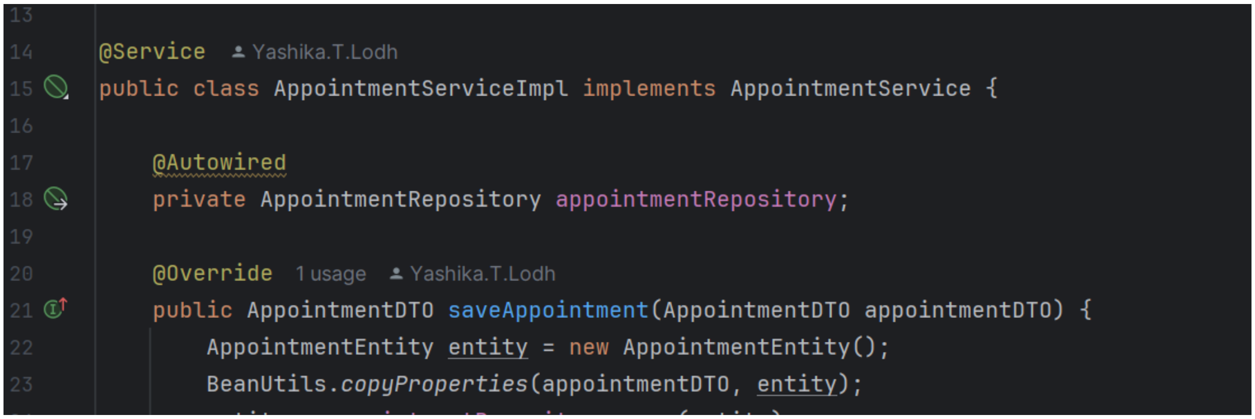
- A class derives from another class to reuse code or extend functionality.
- Usage: Used indirectly via Spring annotations (e.g., `@RestController` and `@Repository`) where classes like StaffController and StaffRepository inherit behavior from Spring Framework base classes.



```
@RestController  Nidhi Mehta
@RequestMapping("api/vi/staff")
public class StaffController {

    @Autowired
    private StaffService staffService;

    @PostMapping("/create")  Nidhi Mehta
    public ResponseEntity<String> createStaff(@RequestBody StaffDTO staffDTO) {
        staffService.createStaff(staffDTO);
        return ResponseEntity.ok( body: "Staff created successfully");
    }

    @PutMapping("/update/{id}")  Nidhi Mehta
    public ResponseEntity<String> updateStaff(@PathVariable int id, @RequestBody StaffDTO staffDTO) {
        staffService.updateStaff(id, staffDTO);
        return ResponseEntity.ok( body: "Staff updated successfully");
    }

    @GetMapping("/get/{id}")  Nidhi Mehta
    public ResponseEntity<StaffDTO> getStaffById(@PathVariable int id) {
        StaffDTO staffDTO = staffService.getStaffById(id);
        return ResponseEntity.ok(staffDTO);
    }
```

6. Polymorphism

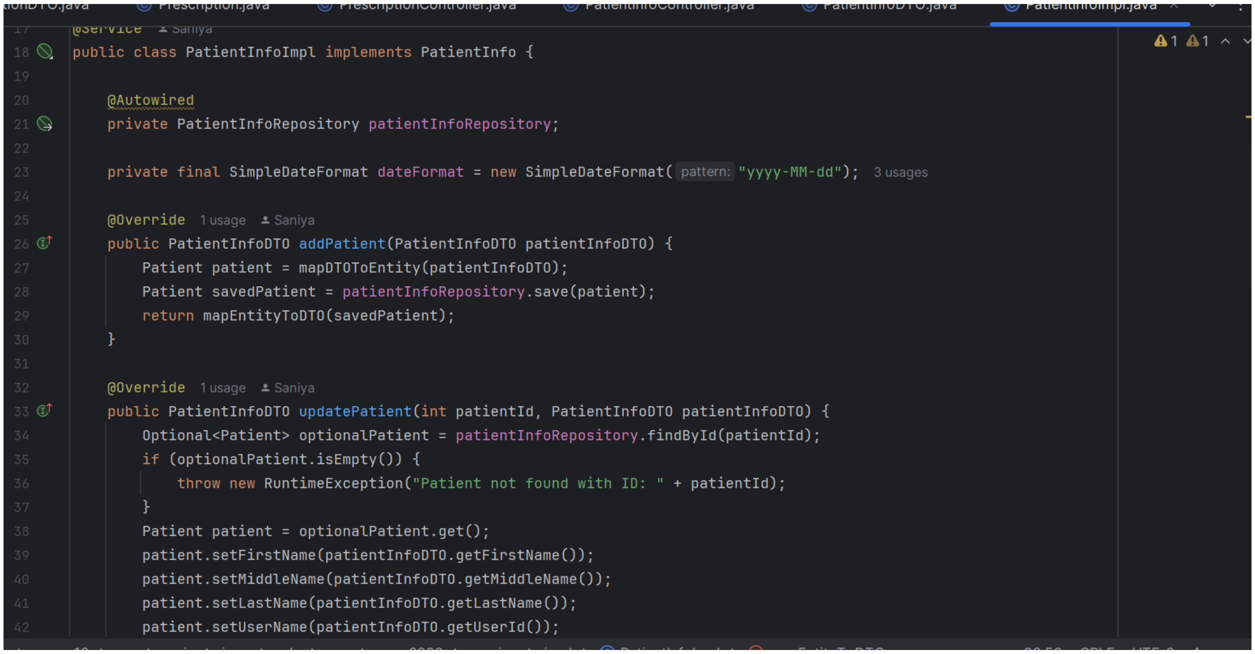
- Polymorphism allows one interface to be used for different data types, meaning a method or function can work with objects of different classes or types.



```
@Service  Yashika.T.Lodh
public class AppointmentServiceImpl implements AppointmentService {

    @Autowired
    private AppointmentRepository appointmentRepository;

    @Override  1 usage  Yashika.T.Lodh
    public AppointmentDTO saveAppointment(AppointmentDTO appointmentDTO) {
        AppointmentEntity entity = new AppointmentEntity();
        BeanUtils.copyProperties(appointmentDTO, entity);
```

7. Singleton

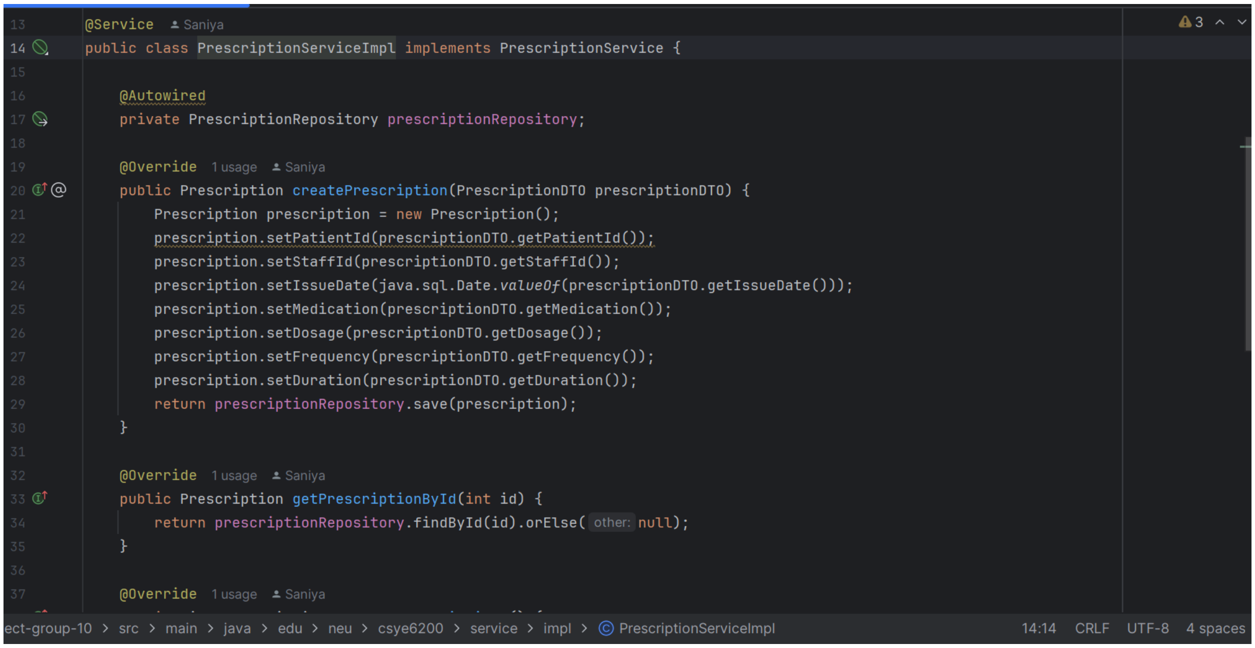
- A class ensures only one instance is created and provides global access to it.
- Usage: Spring automatically ensures that beans like PatientInfoImpl are singletons.



```java
@Service  Saniya
public class PatientInfoImpl implements PatientInfo {

    @Autowired
    private PatientInfoRepository patientInfoRepository;

    private final SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd");  3 usages

    @Override  1 usage  Saniya
    public PatientInfoDTO addPatient(PatientInfoDTO patientInfoDTO) {
        Patient patient = mapDTOToEntity(patientInfoDTO);
        Patient savedPatient = patientInfoRepository.save(patient);
        return mapEntityToDTO(savedPatient);
    }

    @Override  1 usage  Saniya
    public PatientInfoDTO updatePatient(int patientId, PatientInfoDTO patientInfoDTO) {
        Optional<Patient> optionalPatient = patientInfoRepository.findById(patientId);
        if (optionalPatient.isEmpty()) {
            throw new RuntimeException("Patient not found with ID: " + patientId);
        }
        Patient patient = optionalPatient.get();
        patient.setFirstName(patientInfoDTO.getFirstName());
        patient.setMiddleName(patientInfoDTO.getMiddleName());
        patient.setLastName(patientInfoDTO.getLastName());
        patient.setUserName(patientInfoDTO.getUserId());
```

8. Factory Design Pattern

- Provides a way to create objects without specifying their exact class.
- Spring uses the Factory pattern internally to create and inject beans like PrescriptionServiceImpl.

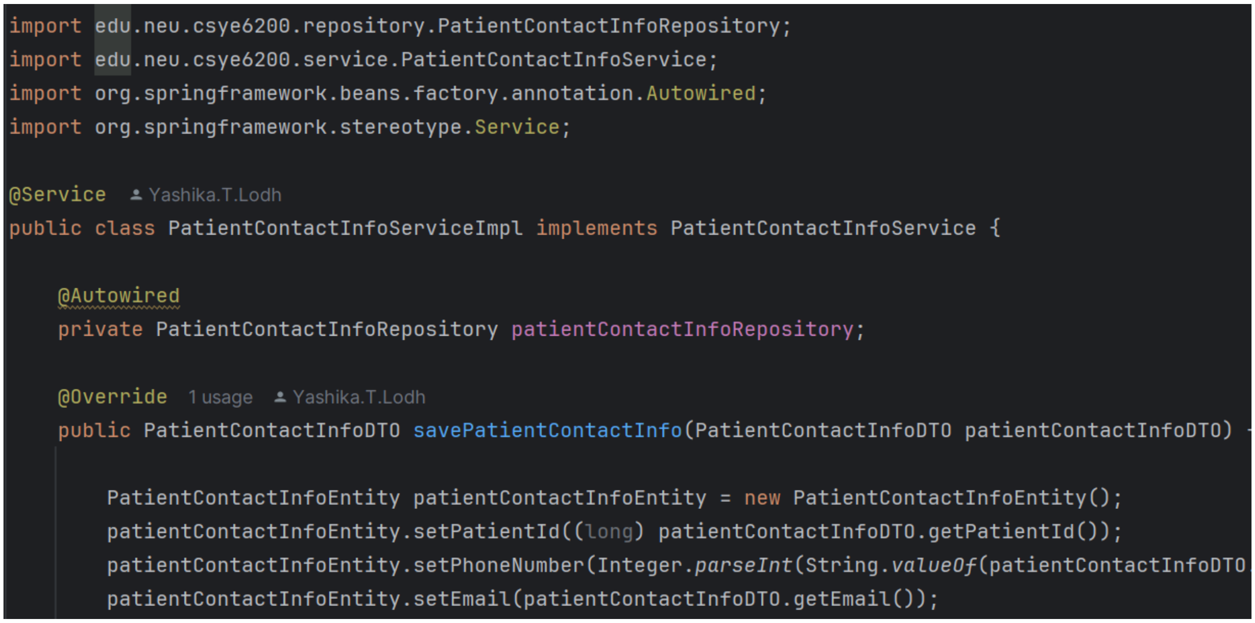


```
@Service  Saniya
public class PrescriptionServiceImpl implements PrescriptionService {

    @Autowired
    private PrescriptionRepository prescriptionRepository;

    @Override  1 usage  Saniya
    public Prescription createPrescription(PrescriptionDTO prescriptionDTO) {
        Prescription prescription = new Prescription();
        prescription.setPatientId(prescriptionDTO.getPatientId());
        prescription.setStaffId(prescriptionDTO.getStaffId());
        prescription.setIssueDate(java.sql.Date.valueOf(prescriptionDTO.getIssueDate()));
        prescription.setMedication(prescriptionDTO.getMedication());
        prescription.setDosage(prescriptionDTO.getDosage());
        prescription.setFrequency(prescriptionDTO.getFrequency());
        prescription.setDuration(prescriptionDTO.getDuration());
        return prescriptionRepository.save(prescription);
    }

    @Override  1 usage  Saniya
    public Prescription getPrescriptionById(int id) {
        return prescriptionRepository.findById(id).orElse( other: null);
    }

    @Override  1 usage  Saniya
```

⚠3
ect-group-10 › src › main › java › edu › neu › csye6200 › service › impl › PrescriptionServiceImpl
14:14 CRLF UTF-8 4 spaces

9. Dependency Injection

- An object receives its dependencies from an external source rather than creating them itself.
- Usage: @Autowired annotation in PatientContactInfoController and PatientContactInfoImpl demonstrates dependency injection.



```
import edu.neu.csye6200.repository.PatientContactInfoRepository;
import edu.neu.csye6200.service.PatientContactInfoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service  Yashika.T.Lodh
public class PatientContactInfoServiceImpl implements PatientContactInfoService {

    @Autowired
    private PatientContactInfoRepository patientContactInfoRepository;

    @Override  1 usage  Yashika.T.Lodh
    public PatientContactInfoDTO savePatientContactInfo(PatientContactInfoDTO patientContactInfoDTO) 

        PatientContactInfoEntity patientContactInfoEntity = new PatientContactInfoEntity();
        patientContactInfoEntity.setPatientId((long) patientContactInfoDTO.getPatientId());
        patientContactInfoEntity.setPhoneNumber(Integer.parseInt(String.valueOf(patientContactInfoDTO
        patientContactInfoEntity.setEmail(patientContactInfoDTO.getEmail());
```

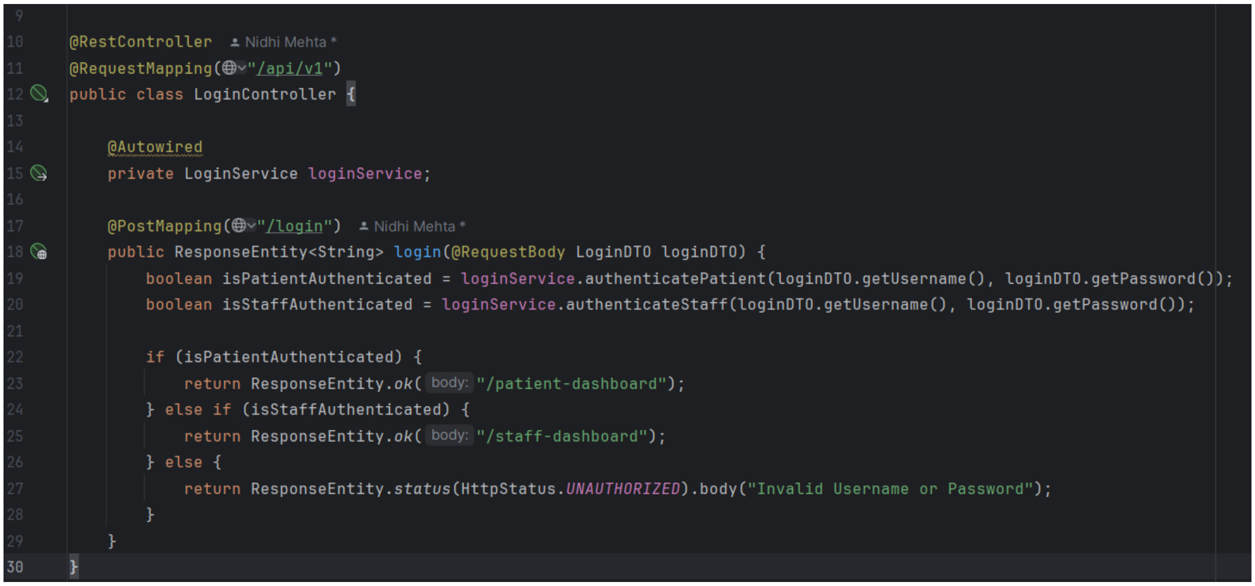
10. Separation of Concerns

- Dividing responsibilities among different classes or layers to improve modularity.
- Usage

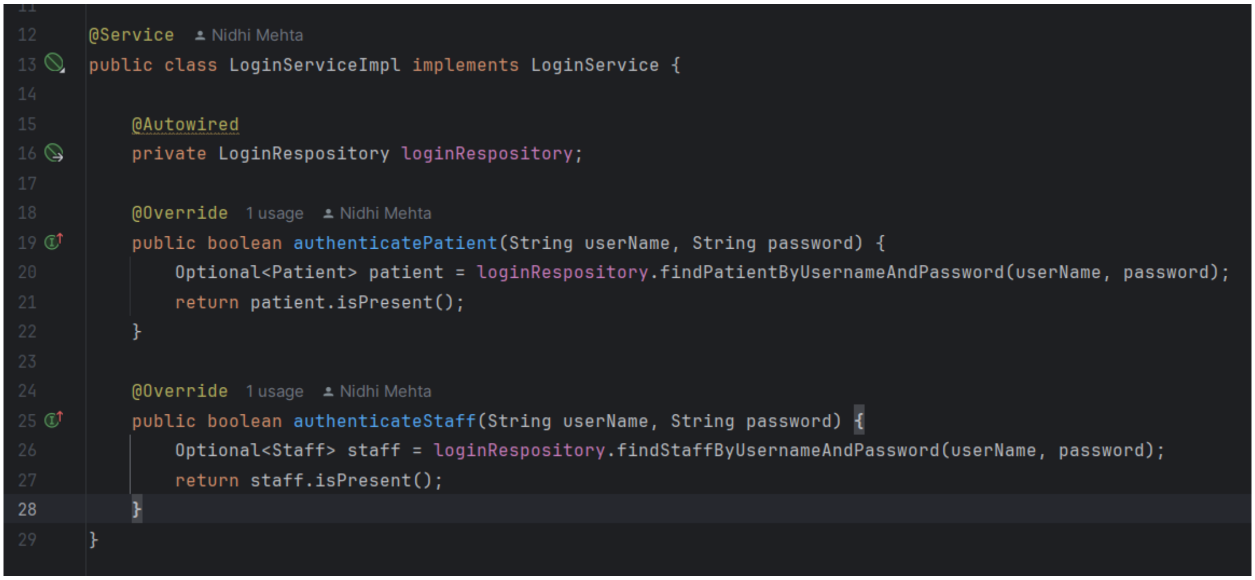
  - Controller Layer: Handles HTTP requests (LoginController).

  - Service Layer: Contains business logic (LoginServiceImpl).

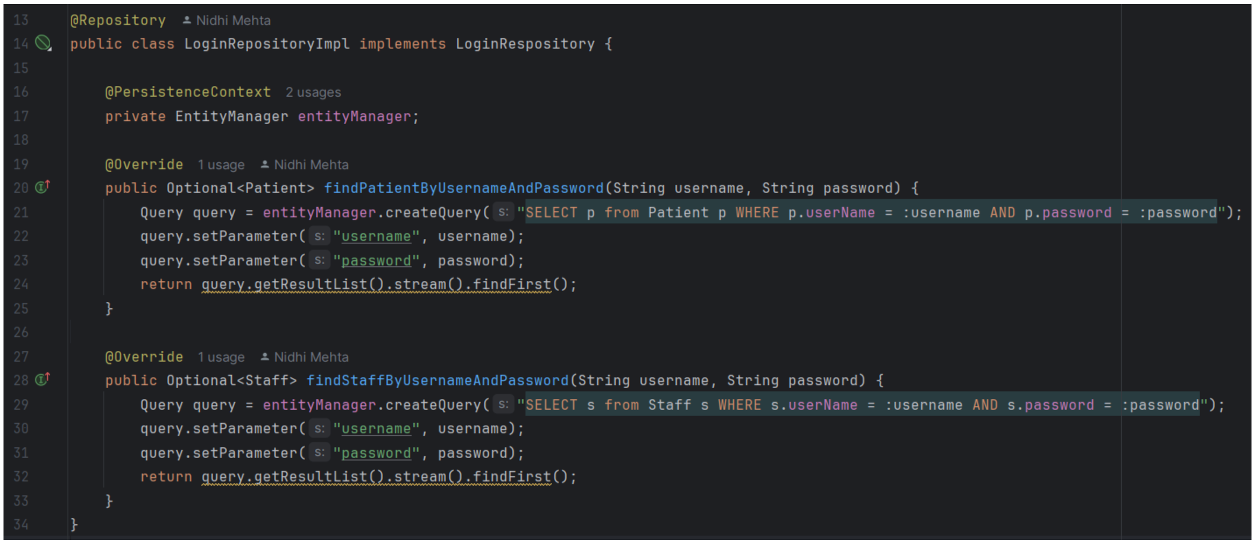
  - Repository Layer: Handles data access (LoginRepositoryImpl).



```java
@RestController
@RequestMapping("/api/v1")
public class LoginController {

    @Autowired
    private LoginService loginService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody LoginDTO loginDTO) {
        boolean isPatientAuthenticated = loginService.authenticatePatient(loginDTO.getUsername(), loginDTO.getPassword());
        boolean isStaffAuthenticated = loginService.authenticateStaff(loginDTO.getUsername(), loginDTO.getPassword());

        if (isPatientAuthenticated) {
            return ResponseEntity.ok("/patient-dashboard");
        } else if (isStaffAuthenticated) {
            return ResponseEntity.ok("/staff-dashboard");
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid Username or Password");
        }
    }
}
```



```java
@Service
public class LoginServiceImpl implements LoginService {

    @Autowired
    private LoginRespository loginRespository;

    @Override
    public boolean authenticatePatient(String userName, String password) {
        Optional<Patient> patient = loginRespository.findPatientByUsernameAndPassword(userName, password);
        return patient.isPresent();
    }

    @Override
    public boolean authenticateStaff(String userName, String password) {
        Optional<Staff> staff = loginRespository.findStaffByUsernameAndPassword(userName, password);
        return staff.isPresent();
    }
}
```



```
@Repository  Nidhi Mehta
public class LoginRepositoryImpl implements LoginRespository {

    @PersistenceContext  2 usages
    private EntityManager entityManager;

    @Override  1 usage  Nidhi Mehta
    public Optional<Patient> findPatientByUsernameAndPassword(String username, String password) {
        Query query = entityManager.createQuery( s: "SELECT p from Patient p WHERE p.userName = :username AND p.password = :password");
        query.setParameter( s: "username", username);
        query.setParameter( s: "password", password);
        return query.getResultList().stream().findFirst();
    }

    @Override  1 usage  Nidhi Mehta
    public Optional<Staff> findStaffByUsernameAndPassword(String username, String password) {
        Query query = entityManager.createQuery( s: "SELECT s from Staff s WHERE s.userName = :username AND s.password = :password");
        query.setParameter( s: "username", username);
        query.setParameter( s: "password", password);
        return query.getResultList().stream().findFirst();
    }
}
```

Functionalities that will be implemented by end of Milestone 3

1. Connecting APIs to Frontend

- Link all existing APIs with their respective front-end components.
- Ensure that data flows seamlessly between the frontend UI and backend logic, enabling dynamic updates and real-time user interaction.
- Make necessary updates to APIs, such as refining response formats or handling additional edge cases, to ensure compatibility with frontend requirements.

2. Setting the Flow of Frontend

- **Staff Dashboard**:
    - Define a clear navigation flow for staff members, such as managing appointments, viewing patient details, and accessing specific staff-related data.
    - Streamline user interactions and improve usability with intuitive design and responsive layouts.
- **Patient Dashboard**:
    - Organize front-end workflows for patients, including viewing personal medical history, managing insurance details, booking appointments, and accessing prescriptions.
    - Ensure smooth navigation with a consistent and user-friendly design.

3. Performing Necessary Testing

- **Integration Testing**: Verify that all APIs work as expected with their connected front-end components.
- **Functional Testing**: Test the correctness of both staff and patient workflows to ensure all features behave as intended.

By the end of this milestone, the system should have fully integrated APIs and a well-defined, tested frontend flow for both staff and patients, offering a cohesive and functional user experience.

Team Contributions

1. **Katha Patel**
   a. Developed CRUD APIs for **medical history**.
   b. Built CRUD APIs for **patient insurance information**.
   c. Integrated **medical history** and **patient insurance APIs** with the frontend.

2. **Yashika Lodh**
   a. Implemented CRUD APIs for **patient contact information**.
   b. Created CRUD APIs for **appointment details**.
   c. Integrated **patient contact information APIs** with the frontend.

3. **Nidhi Mehta**
   a. Developed CRUD APIs for **staff management**.
   b. Enhanced the login API by adding **JWT authentication**.
   c. Implemented **redirection logic** between staff and patient dashboards post-login.

4. **Saniya Azmat**
   a. Built CRUD APIs for **patient basic information**.
   b. Developed CRUD APIs for **prescription details**.
   c. Integrated **patient basic information APIs** with the frontend.

Group Number: 10

Project Topic: WellCare – Hospital Management System

Final Tech Stack:

Frontend:

- Thymeleaf, HTML, CSS, JavaScript, Bootstrap

Backend:

- Java, Spring Boot, Spring MVC, Spring Data JPA
- APIs

Database:

- MySQL
- WampServer (for data schema visualization and management)

DevOps & Tools:

- Git (version control), GitHub (repository hosting)
- Maven (build automation)
- Bruno (API testing)

Security:

- Spring Security, JWT (for authentication and session handling)

IDE:

- IntelliJ IDEA (for Java development and project management)

Functionalities Implemented:

Dashboard UI:
Tables displaying data such as appointments.

Search or filtering functionality.
Navigation features (sidebar, headers, etc.).

1. **doctor-profile.html**:
   Displays information about doctors.
   Likely includes fields for personal and professional details.

2. **insurance.html**:
   Manages or displays insurance-related information.
   May include forms or tables for insurance policies and claims.

3. **login.html**:
   Implements the login functionality for users.
   Includes input fields for username/email and password.

4. **staff-all-appointments.html**:
   Displays a list of all appointments for staff.

5. **Patient-profile.html:**
   Displays and allows editing of patient details, such as contact info, medical records, and preferences.
   Could integrate with insurance and appointment systems for seamless profile updates.

6. **Doctor-dashboard.html:**
   A dashboard for doctors to manage their schedules, patient interactions, and performance.
   Could include an overview of upcoming appointments, recent patient updates.

Dynamic Data Handling:
Fetching or displaying data dynamically (e.g., appointments, patients).
pagination, and other data management.

Interactive Features:
Buttons with actions (e.g., "Details" button to view more about an appointment).

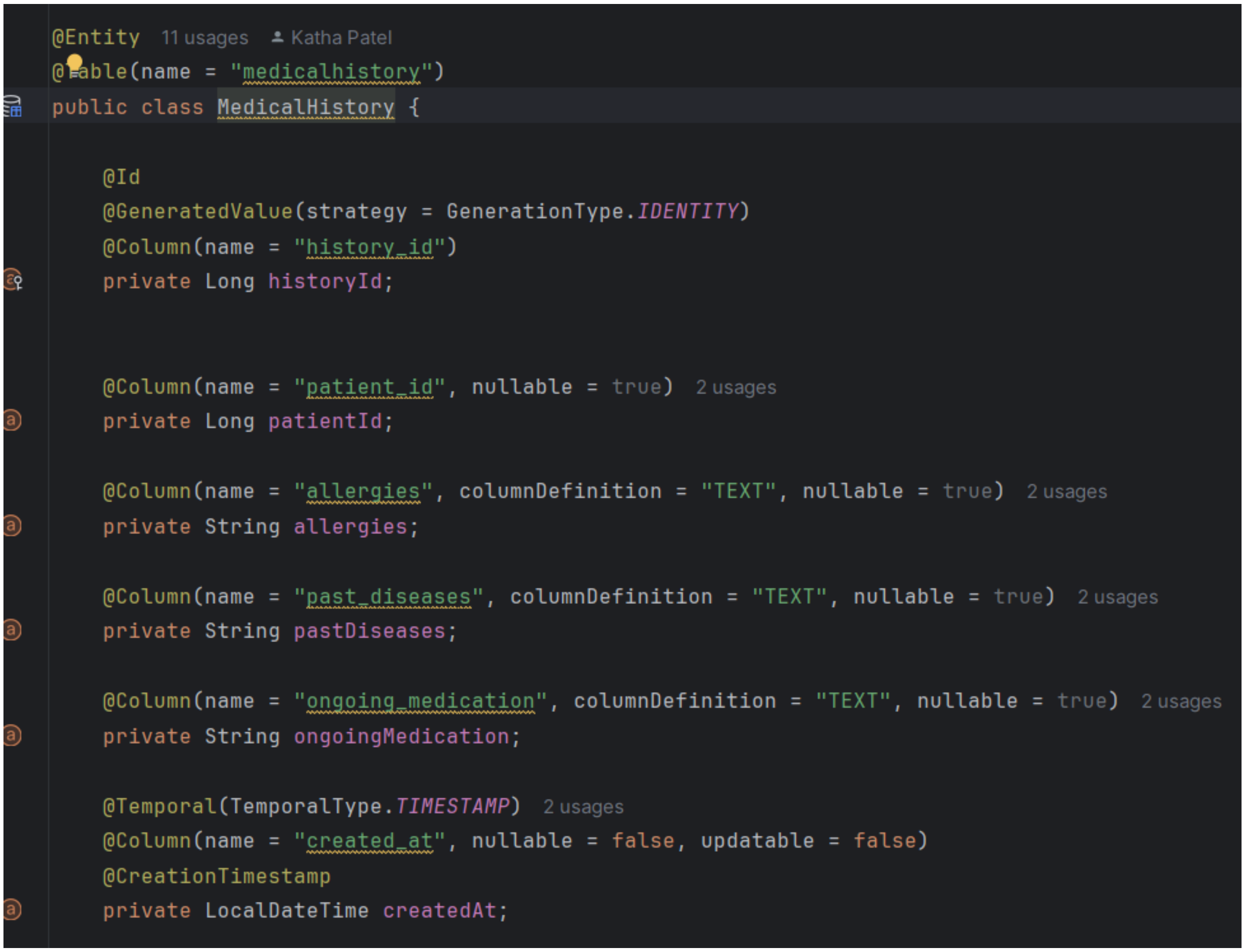
Authentication & Roles:
User login and authentication.
Role-based access (e.g., doctor, patient).

CRUD Operations:
Creating, reading, updating, and deleting entries in tables (e.g., appointments, patients).

Object Oriented Concepts:

1. Encapsulation

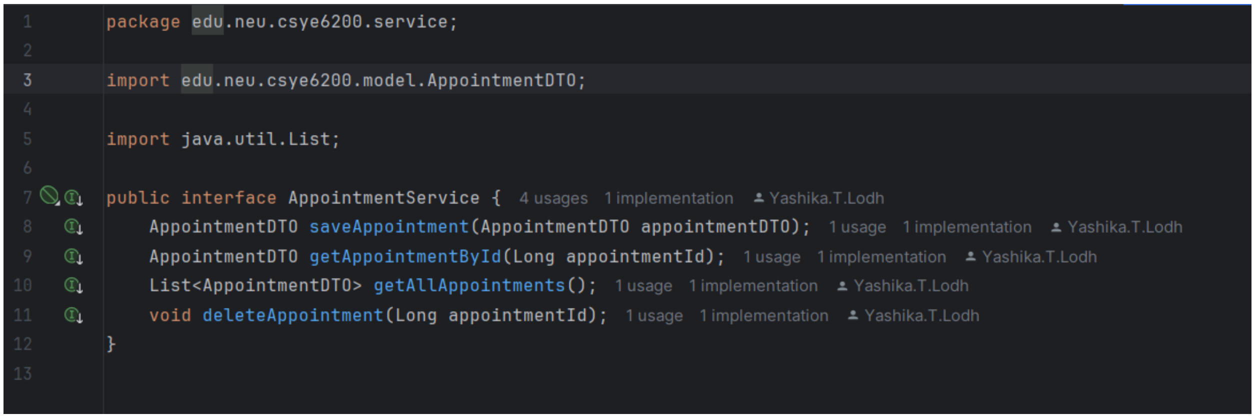
- Hiding the internal state of an object and only exposing necessary components through getters and setters
- Usage: Fields in MedicalHistoryDTO are private and accessed via getters and setters, ensuring encapsulation.



```
@Entity  11 usages  Katha Patel
@Table(name = "medicalhistory")
public class MedicalHistory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "history_id")
    private Long historyId;


    @Column(name = "patient_id", nullable = true)  2 usages
    private Long patientId;

    @Column(name = "allergies", columnDefinition = "TEXT", nullable = true)  2 usages
    private String allergies;

    @Column(name = "past_diseases", columnDefinition = "TEXT", nullable = true)  2 usages
    private String pastDiseases;

    @Column(name = "ongoing_medication", columnDefinition = "TEXT", nullable = true)  2 usages
    private String ongoingMedication;

    @Temporal(TemporalType.TIMESTAMP)  2 usages
    @Column(name = "created_at", nullable = false, updatable = false)
    @CreationTimestamp
    private LocalDateTime createdAt;
```

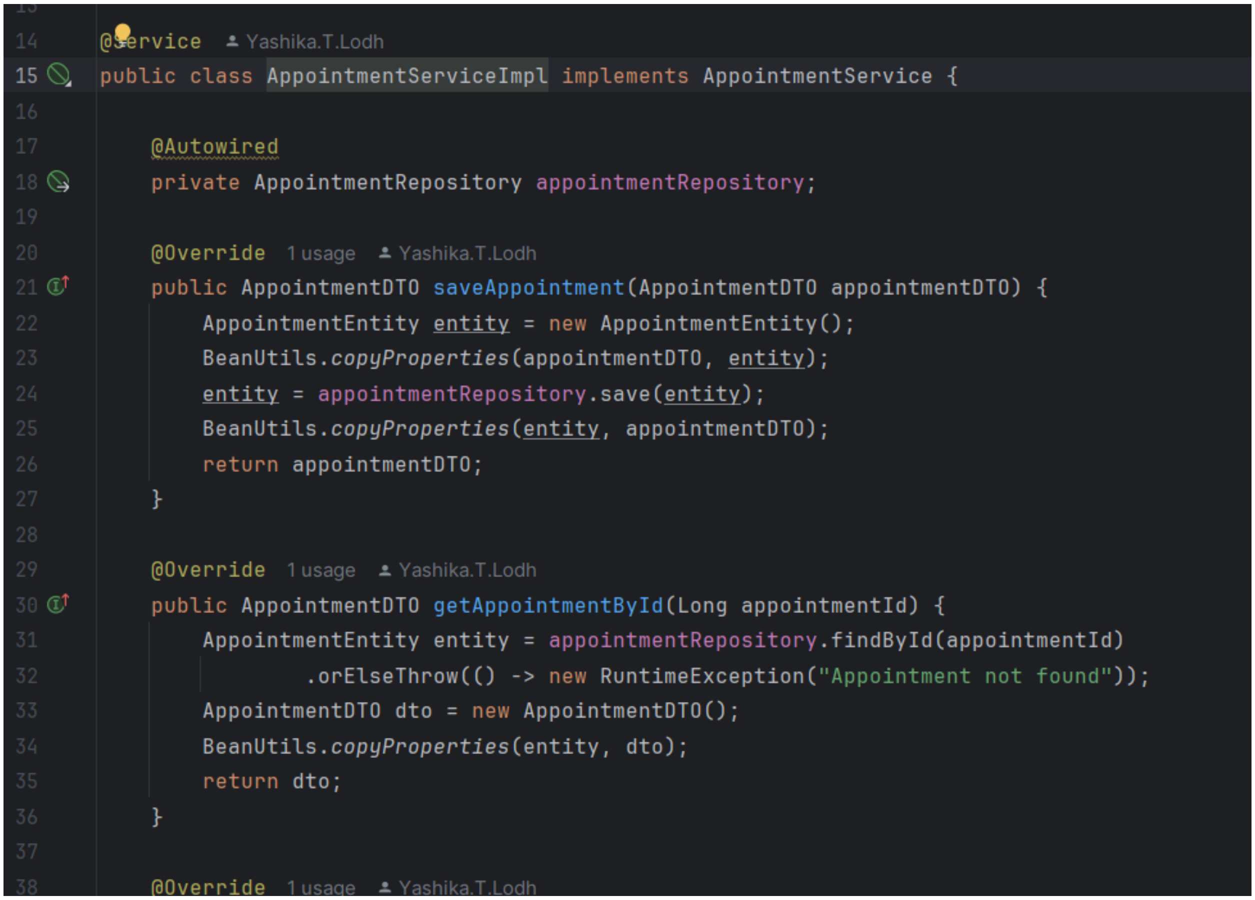
2. Abstraction

- Hiding implementation details and exposing only essential features.
- Usage: The AppointmentService interface hides the implementation of logic provided in AppointmentImpl.



```java
package edu.neu.csye6200.service;

import edu.neu.csye6200.model.AppointmentDTO;

import java.util.List;

public interface AppointmentService {
    AppointmentDTO saveAppointment(AppointmentDTO appointmentDTO);
    AppointmentDTO getAppointmentById(Long appointmentId);
    List<AppointmentDTO> getAllAppointments();
    void deleteAppointment(Long appointmentId);
}
```

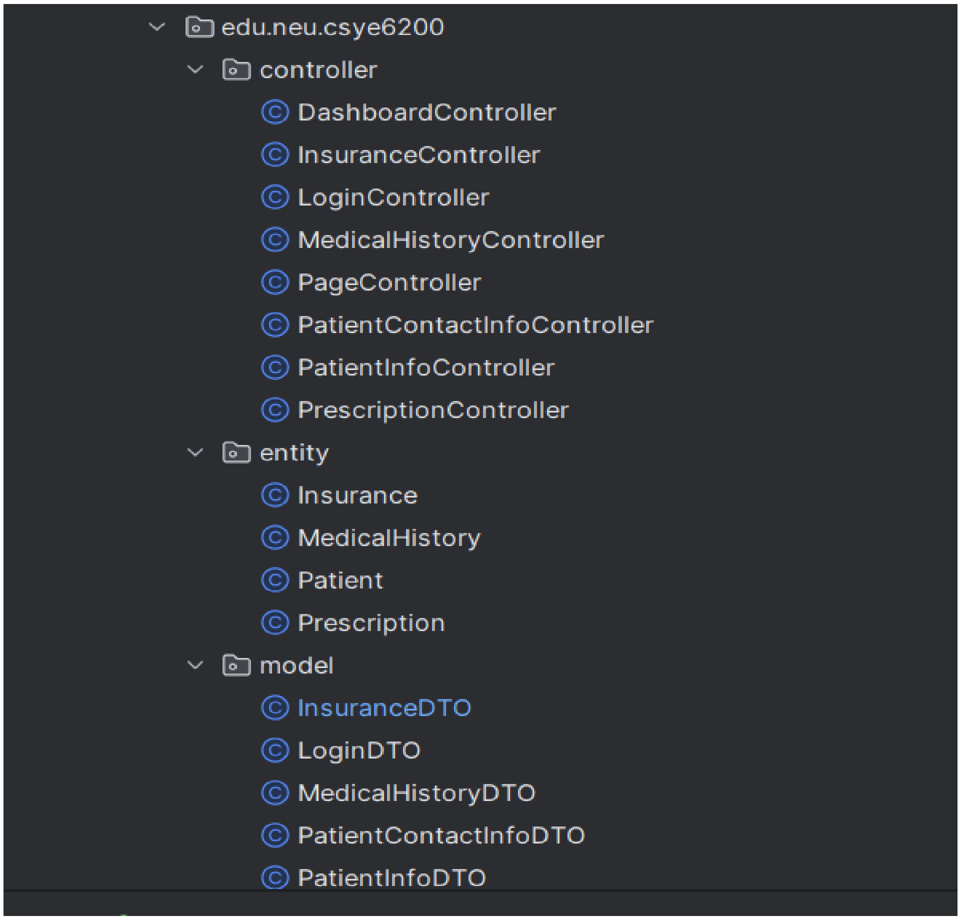
4 usages 1 implementation Yashika.T.Lodh
1 usage 1 implementation Yashika.T.Lodh
1 usage 1 implementation Yashika.T.Lodh
1 usage 1 implementation Yashika.T.Lodh
1 usage 1 implementation Yashika.T.Lodh



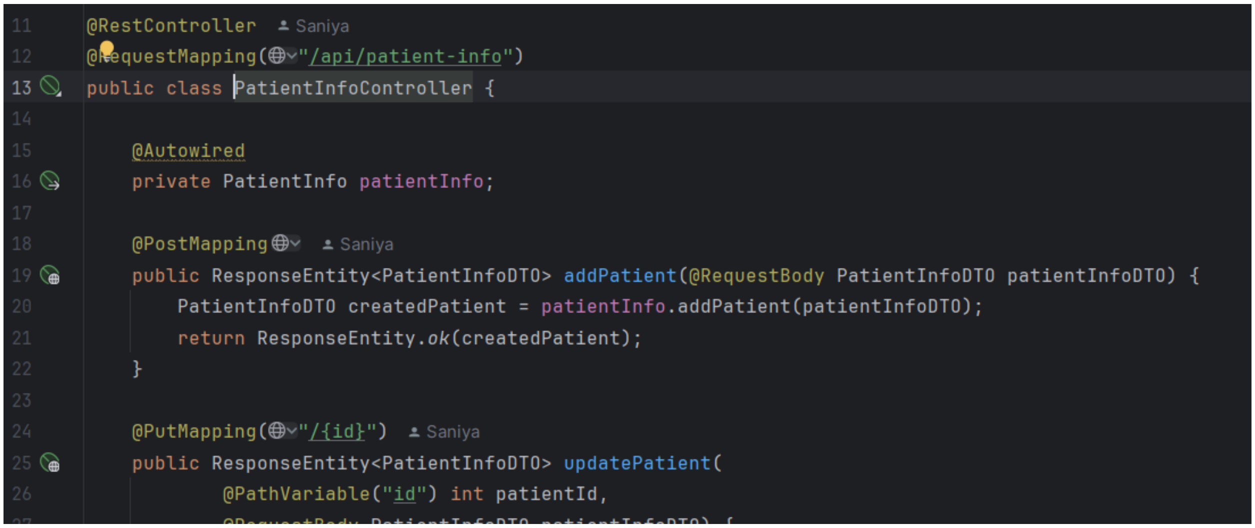
```java
@Service
public class AppointmentServiceImpl implements AppointmentService {

    @Autowired
    private AppointmentRepository appointmentRepository;

    @Override
    public AppointmentDTO saveAppointment(AppointmentDTO appointmentDTO) {
        AppointmentEntity entity = new AppointmentEntity();
        BeanUtils.copyProperties(appointmentDTO, entity);
        entity = appointmentRepository.save(entity);
        BeanUtils.copyProperties(entity, appointmentDTO);
        return appointmentDTO;
    }

    @Override
    public AppointmentDTO getAppointmentById(Long appointmentId) {
        AppointmentEntity entity = appointmentRepository.findById(appointmentId)
                .orElseThrow(() -> new RuntimeException("Appointment not found"));
        AppointmentDTO dto = new AppointmentDTO();
        BeanUtils.copyProperties(entity, dto);
        return dto;
    }

    @Override
```

3. Class and Objects

- A class is a blueprint for creating objects. An object is an instance of a class.
- Classes such as LoginController, InsuranceController, MedicalHistoryContoller, PatientInfoController, PatientContactInfoDTO and LoginDTO define the structure and behavior of the application.



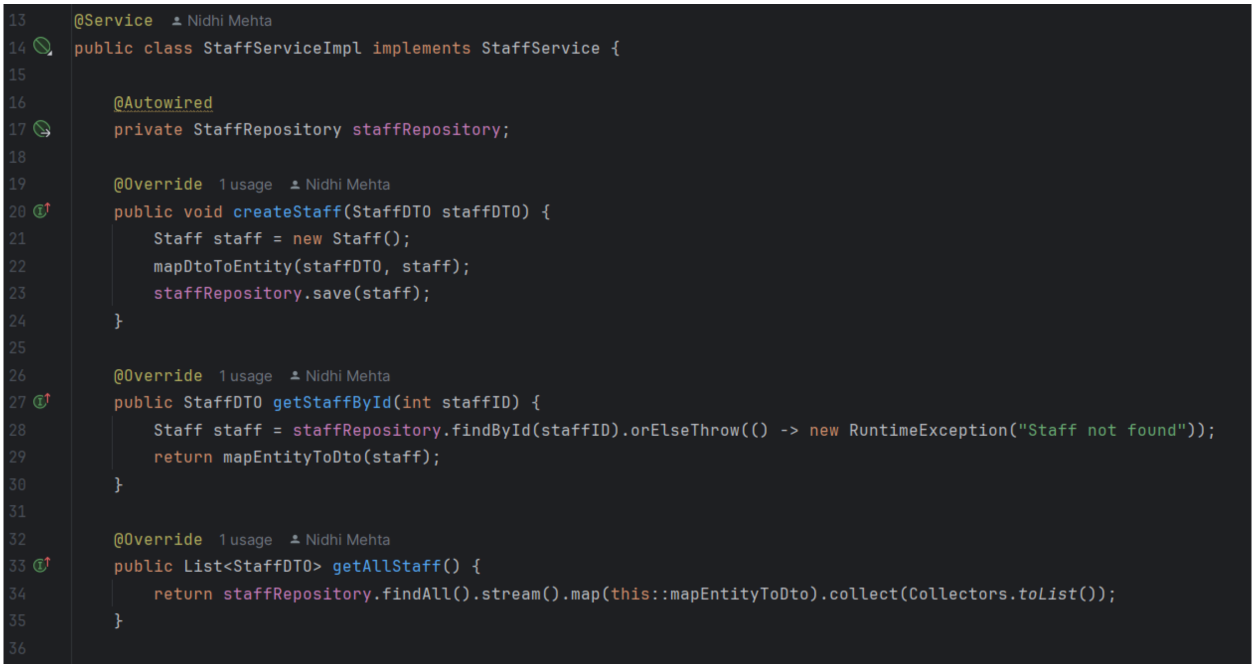
```
edu.neu.csye6200
    controller
        DashboardController
        InsuranceController
        LoginController
        MedicalHistoryController
        PageController
        PatientContactInfoController
        PatientInfoController
        PrescriptionController
    entity
        Insurance
        MedicalHistory
        Patient
        Prescription
    model
        InsuranceDTO
        LoginDTO
        MedicalHistoryDTO
        PatientContactInfoDTO
        PatientInfoDTO
```



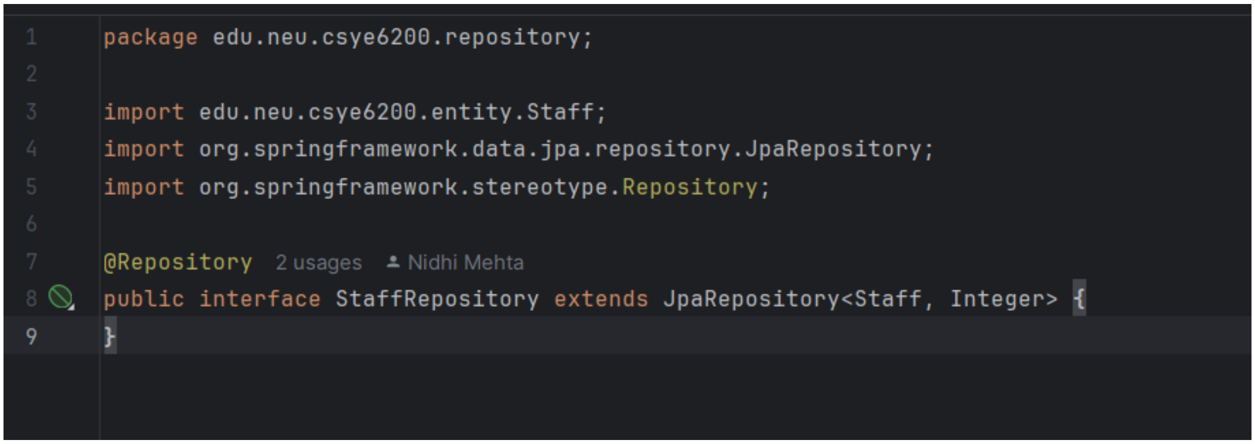
```java
@RestController  Saniya
@RequestMapping("/api/patient-info")
public class PatientInfoController {

    @Autowired
    private PatientInfo patientInfo;

    @PostMapping  Saniya
    public ResponseEntity<PatientInfoDTO> addPatient(@RequestBody PatientInfoDTO patientInfoDTO) {
        PatientInfoDTO createdPatient = patientInfo.addPatient(patientInfoDTO);
        return ResponseEntity.ok(createdPatient);
    }

    @PutMapping("/{id}")  Saniya
    public ResponseEntity<PatientInfoDTO> updatePatient(
            @PathVariable("id") int patientId,
            @RequestBody PatientInfoDTO patientInfoDTO) {
```

4. Interfaces

- An interface defines a contract that implementing classes must adhere to, without providing implementation details.
- StaffService and StaffRespository are interfaces. They abstract business logic and database operations, respectively.



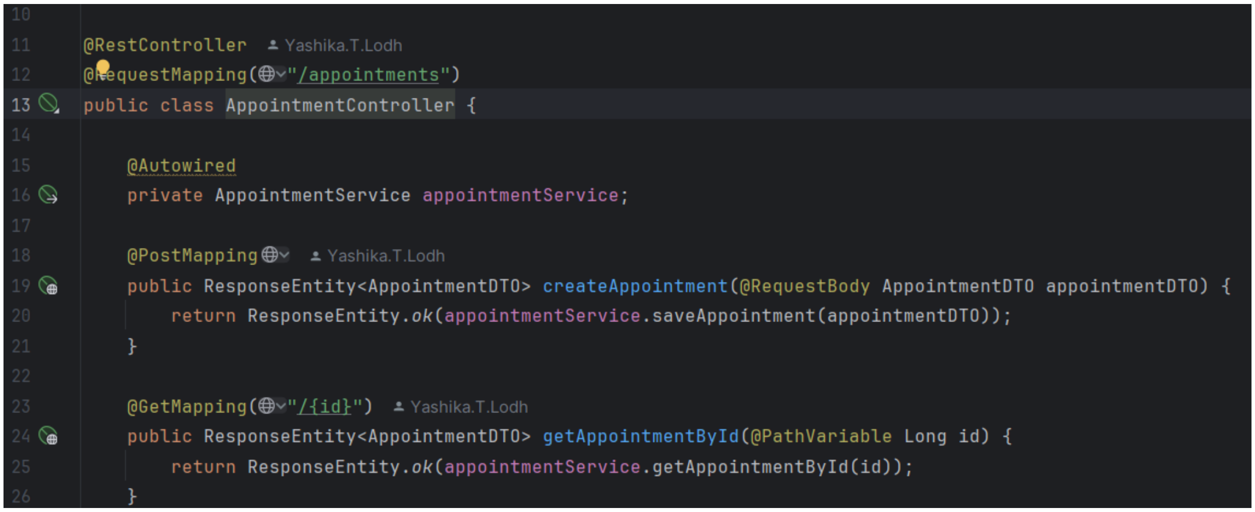
```java
@Service  Nidhi Mehta
public class StaffServiceImpl implements StaffService {

    @Autowired
    private StaffRepository staffRepository;

    @Override  1 usage  Nidhi Mehta
    public void createStaff(StaffDTO staffDTO) {
        Staff staff = new Staff();
        mapDtoToEntity(staffDTO, staff);
        staffRepository.save(staff);
    }

    @Override  1 usage  Nidhi Mehta
    public StaffDTO getStaffById(int staffID) {
        Staff staff = staffRepository.findById(staffID).orElseThrow(() -> new RuntimeException("Staff not found"));
        return mapEntityToDto(staff);
    }

    @Override  1 usage  Nidhi Mehta
    public List<StaffDTO> getAllStaff() {
        return staffRepository.findAll().stream().map(this::mapEntityToDto).collect(Collectors.toList());
    }

```



```java
package edu.neu.csye6200.repository;

import edu.neu.csye6200.entity.Staff;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository  2 usages  Nidhi Mehta
public interface StaffRepository extends JpaRepository<Staff, Integer> {
}
```

5. Inheritance

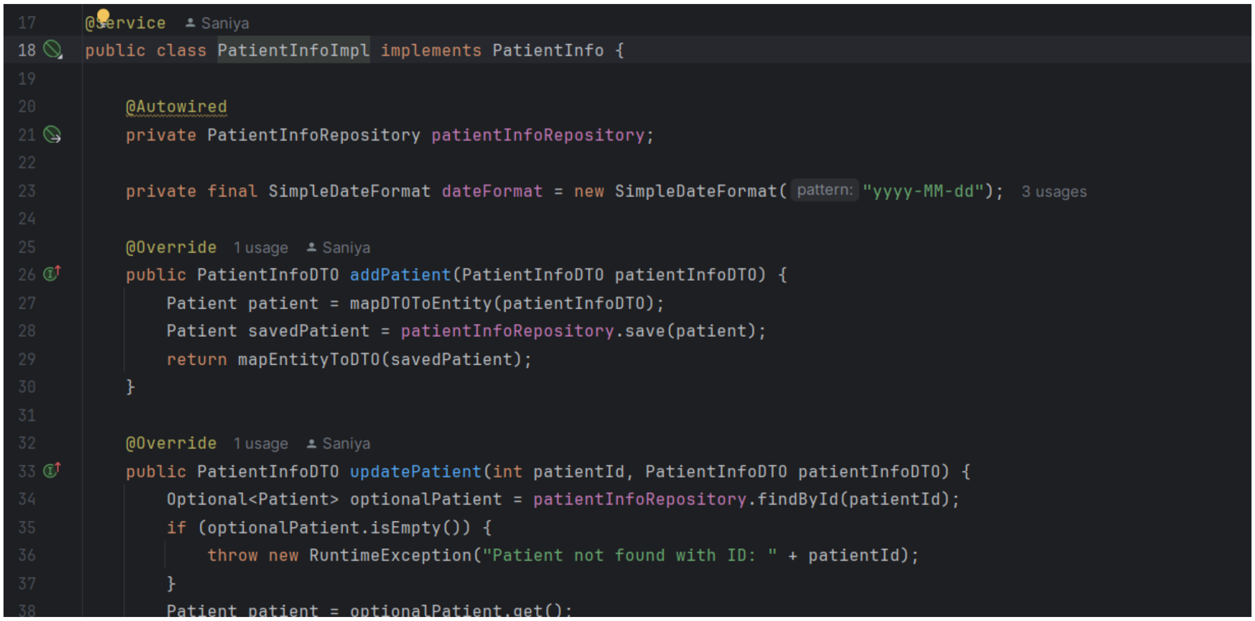
- A class derives from another class to reuse code or extend functionality.
- Usage: Used indirectly via Spring annotations (e.g., `@RestController` and `@Repository`) where classes like AppointmentController and AppointmentRepository inherit behavior from Spring Framework base classes.



```java

@RestController
@RequestMapping("/appointments")
public class AppointmentController {

    @Autowired
    private AppointmentService appointmentService;

    @PostMapping
    public ResponseEntity<AppointmentDTO> createAppointment(@RequestBody AppointmentDTO appointmentDTO) {
        return ResponseEntity.ok(appointmentService.saveAppointment(appointmentDTO));
    }

    @GetMapping("/{id}")
    public ResponseEntity<AppointmentDTO> getAppointmentById(@PathVariable Long id) {
        return ResponseEntity.ok(appointmentService.getAppointmentById(id));
    }
```

6. Polymorphism

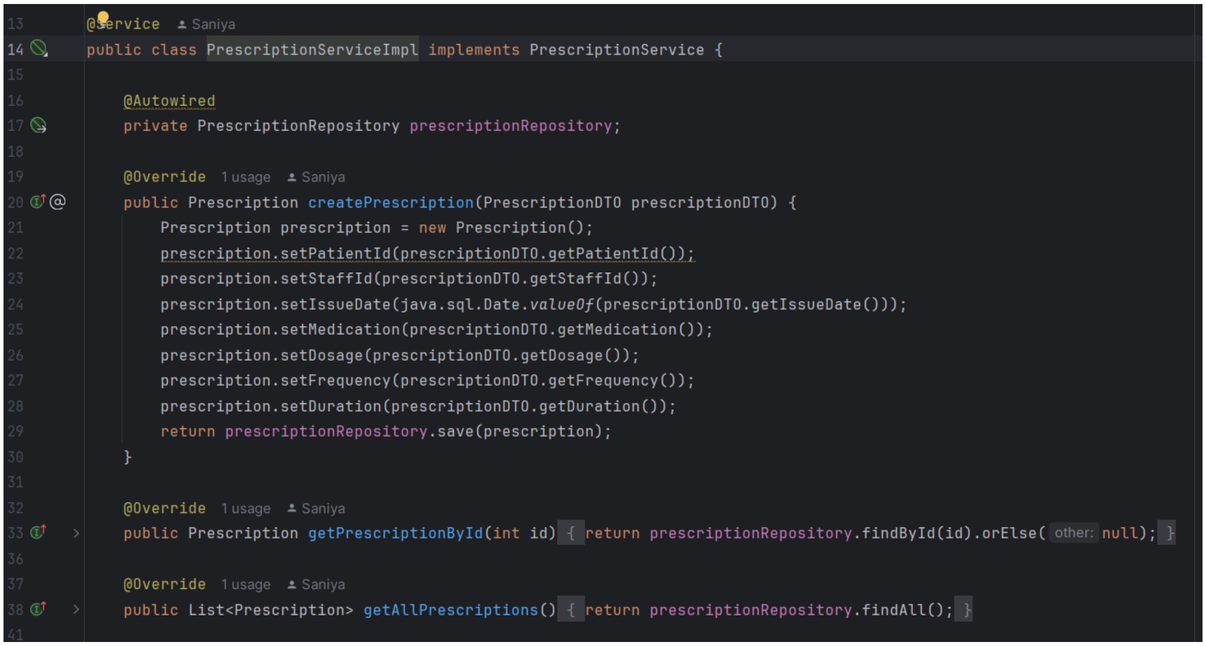
- Polymorphism allows one interface to be used for different data types, meaning a method or function can work with objects of different classes or types.



```java
@Service
public class PatientInfoImpl implements PatientInfo {

    @Autowired
    private PatientInfoRepository patientInfoRepository;

    private final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

    @Override
    public PatientInfoDTO addPatient(PatientInfoDTO patientInfoDTO) {
        Patient patient = mapDTOToEntity(patientInfoDTO);
        Patient savedPatient = patientInfoRepository.save(patient);
        return mapEntityToDTO(savedPatient);
    }

    @Override
    public PatientInfoDTO updatePatient(int patientId, PatientInfoDTO patientInfoDTO) {
        Optional<Patient> optionalPatient = patientInfoRepository.findById(patientId);
        if (optionalPatient.isEmpty()) {
            throw new RuntimeException("Patient not found with ID: " + patientId);
        }
        Patient patient = optionalPatient.get();
```

7. Singleton

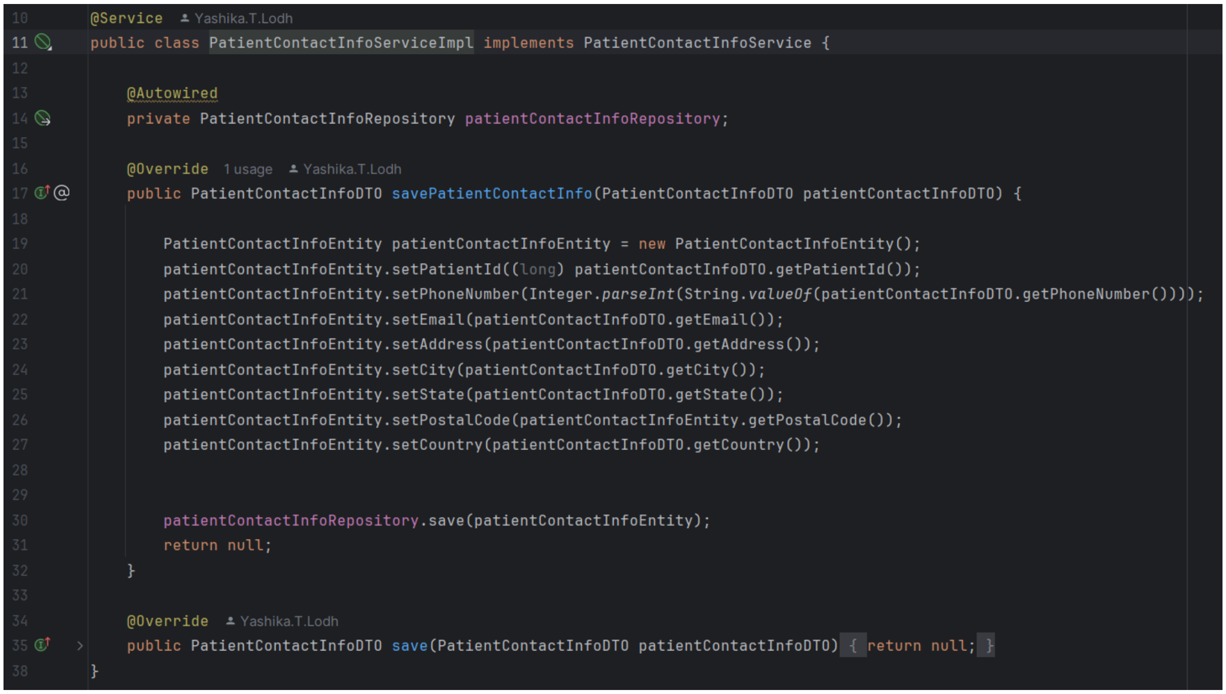
- A class ensures only one instance is created and provides global access to it.
- Usage: Spring automatically ensures that beans like PrescriptionServiceServiceImpl are singletons.



```
@Service  Saniya
public class PrescriptionServiceImpl implements PrescriptionService {

    @Autowired
    private PrescriptionRepository prescriptionRepository;

    @Override  1 usage  Saniya
    public Prescription createPrescription(PrescriptionDTO prescriptionDTO) {
        Prescription prescription = new Prescription();
        prescription.setPatientId(prescriptionDTO.getPatientId());
        prescription.setStaffId(prescriptionDTO.getStaffId());
        prescription.setIssueDate(java.sql.Date.valueOf(prescriptionDTO.getIssueDate()));
        prescription.setMedication(prescriptionDTO.getMedication());
        prescription.setDosage(prescriptionDTO.getDosage());
        prescription.setFrequency(prescriptionDTO.getFrequency());
        prescription.setDuration(prescriptionDTO.getDuration());
        return prescriptionRepository.save(prescription);
    }

    @Override  1 usage  Saniya
    public Prescription getPrescriptionById(int id) { return prescriptionRepository.findById(id).orElse( other: null); }

    @Override  1 usage  Saniya
    public List<Prescription> getAllPrescriptions() { return prescriptionRepository.findAll(); }
```

8. Factory Design Pattern

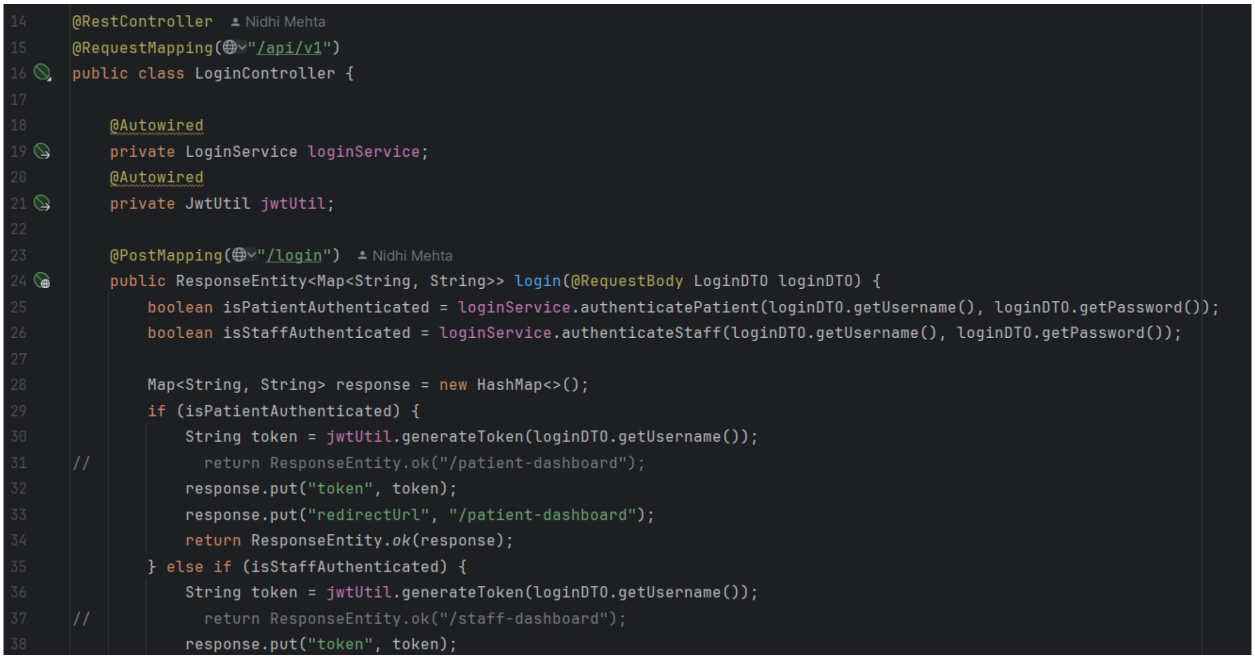
- Provides a way to create objects without specifying their exact class.
- Spring uses the Factory pattern internally to create and inject beans like PatientContactInfoServiceImpl.



```java
@Service  Yashika.T.Lodh
public class PatientContactInfoServiceImpl implements PatientContactInfoService {

    @Autowired
    private PatientContactInfoRepository patientContactInfoRepository;

    @Override  1 usage  Yashika.T.Lodh
    public PatientContactInfoDTO savePatientContactInfo(PatientContactInfoDTO patientContactInfoDTO) {

        PatientContactInfoEntity patientContactInfoEntity = new PatientContactInfoEntity();
        patientContactInfoEntity.setPatientId((long) patientContactInfoDTO.getPatientId());
        patientContactInfoEntity.setPhoneNumber(Integer.parseInt(String.valueOf(patientContactInfoDTO.getPhoneNumber())));
        patientContactInfoEntity.setEmail(patientContactInfoDTO.getEmail());
        patientContactInfoEntity.setAddress(patientContactInfoDTO.getAddress());
        patientContactInfoEntity.setCity(patientContactInfoDTO.getCity());
        patientContactInfoEntity.setState(patientContactInfoDTO.getState());
        patientContactInfoEntity.setPostalCode(patientContactInfoEntity.getPostalCode());
        patientContactInfoEntity.setCountry(patientContactInfoDTO.getCountry());


        patientContactInfoRepository.save(patientContactInfoEntity);
        return null;
    }

    @Override  Yashika.T.Lodh
    public PatientContactInfoDTO save(PatientContactInfoDTO patientContactInfoDTO) { return null; }
}
```

9. Dependency Injection

- An object receives its dependencies from an external source rather than creating them itself.
- Usage: @Autowired annotation in LoginController and LoginServiceImpl demonstrates dependency injection.

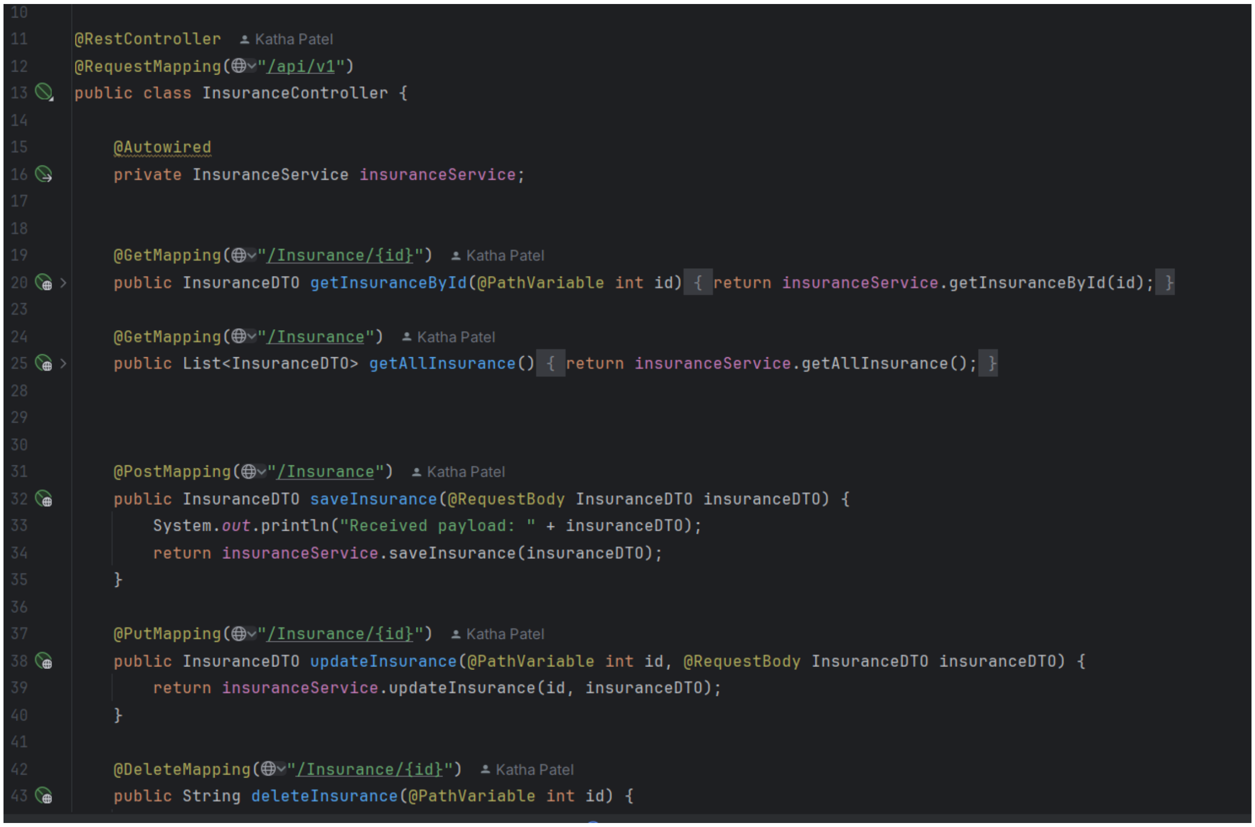


```java
@RestController  Nidhi Mehta
@RequestMapping("/api/v1")
public class LoginController {

    @Autowired
    private LoginService loginService;
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/login")  Nidhi Mehta
    public ResponseEntity<Map<String, String>> login(@RequestBody LoginDTO loginDTO) {
        boolean isPatientAuthenticated = loginService.authenticatePatient(loginDTO.getUsername(), loginDTO.getPassword());
        boolean isStaffAuthenticated = loginService.authenticateStaff(loginDTO.getUsername(), loginDTO.getPassword());

        Map<String, String> response = new HashMap<>();
        if (isPatientAuthenticated) {
            String token = jwtUtil.generateToken(loginDTO.getUsername());
//              return ResponseEntity.ok("/patient-dashboard");
            response.put("token", token);
            response.put("redirectUrl", "/patient-dashboard");
            return ResponseEntity.ok(response);
        } else if (isStaffAuthenticated) {
            String token = jwtUtil.generateToken(loginDTO.getUsername());
//              return ResponseEntity.ok("/staff-dashboard");
            response.put("token", token);
```

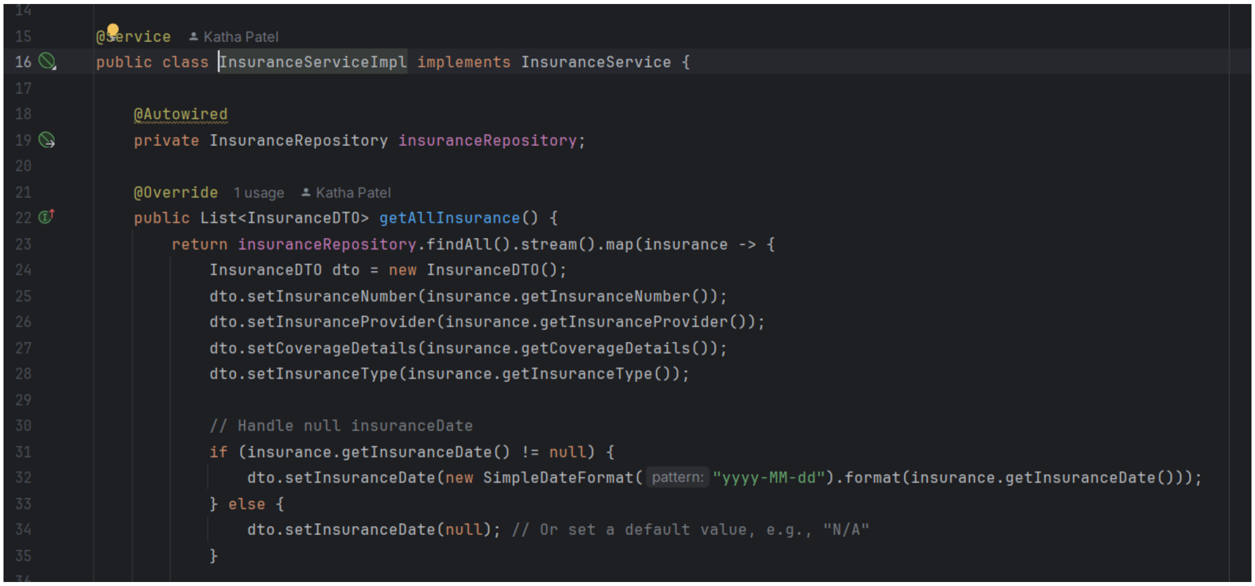
10. Separation of Concerns

- Dividing responsibilities among different classes or layers to improve modularity.
- Usage

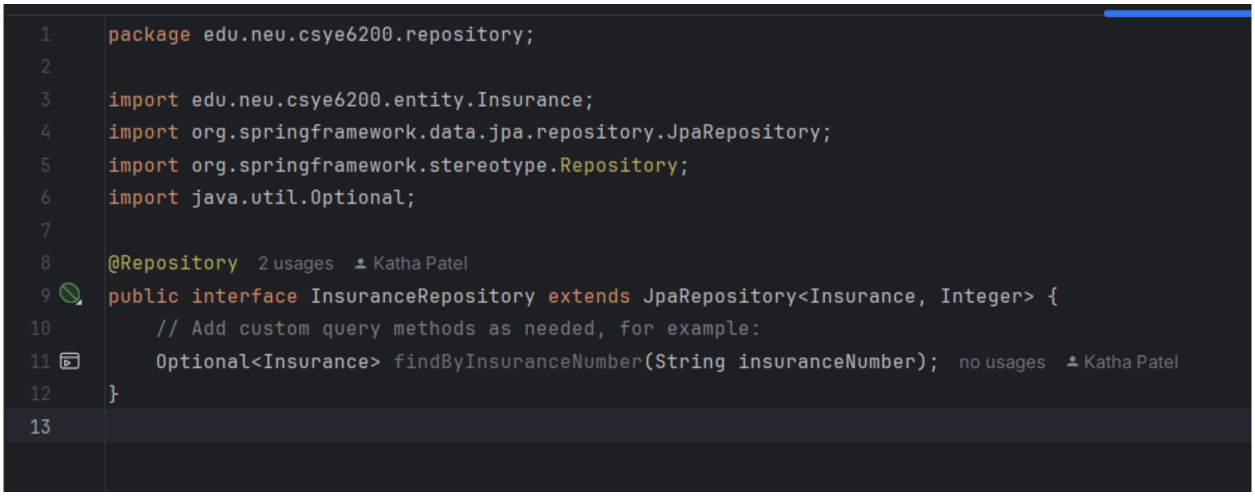
- Controller Layer: Handles HTTP requests (InsuranceController).
- Service Layer: Contains business logic (InsuranceServiceImpl).
- Repository Layer: Handles data access (InsuranceRepositoryImpl).



```java
@RestController  Katha Patel
@RequestMapping("/api/v1")
public class InsuranceController {

    @Autowired
    private InsuranceService insuranceService;


    @GetMapping("/Insurance/{id}")  Katha Patel
    public InsuranceDTO getInsuranceById(@PathVariable int id) { return insuranceService.getInsuranceById(id); }

    @GetMapping("/Insurance")  Katha Patel
    public List<InsuranceDTO> getAllInsurance() { return insuranceService.getAllInsurance(); }



    @PostMapping("/Insurance")  Katha Patel
    public InsuranceDTO saveInsurance(@RequestBody InsuranceDTO insuranceDTO) {
        System.out.println("Received payload: " + insuranceDTO);
        return insuranceService.saveInsurance(insuranceDTO);
    }

    @PutMapping("/Insurance/{id}")  Katha Patel
    public InsuranceDTO updateInsurance(@PathVariable int id, @RequestBody InsuranceDTO insuranceDTO) {
        return insuranceService.updateInsurance(id, insuranceDTO);
    }

    @DeleteMapping("/Insurance/{id}")  Katha Patel
    public String deleteInsurance(@PathVariable int id) {
```



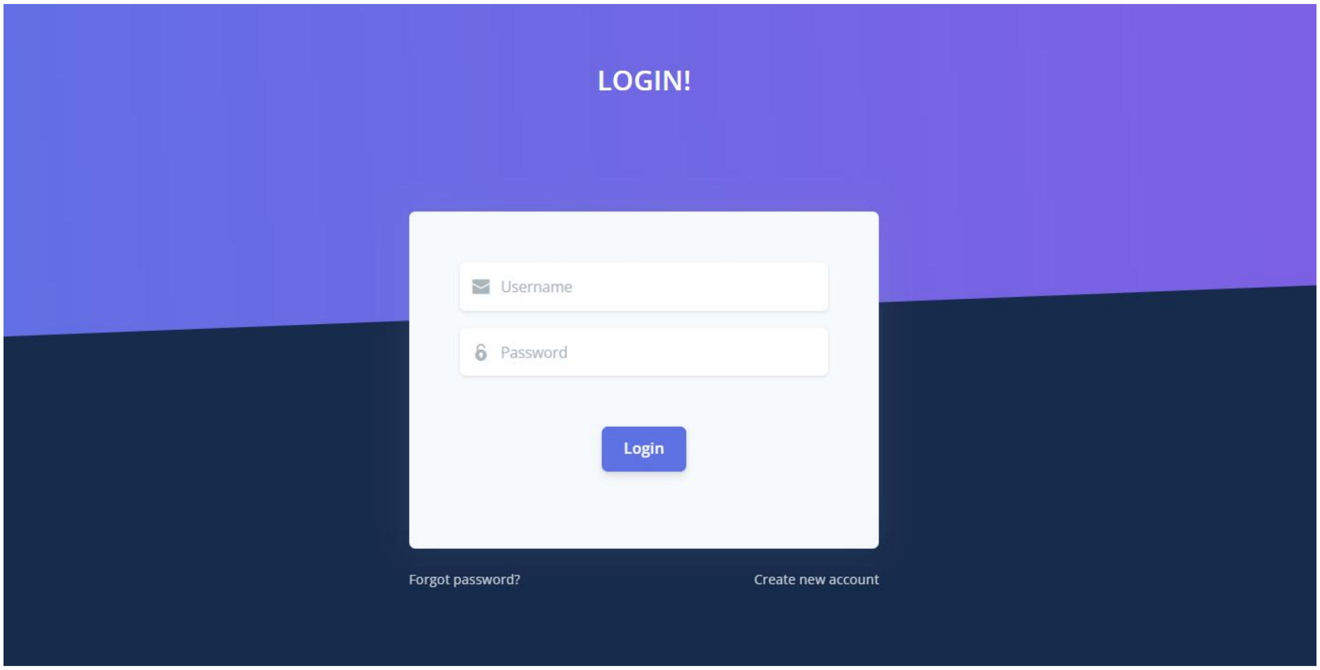
```java
@Service  Katha Patel
public class InsuranceServiceImpl implements InsuranceService {

    @Autowired
    private InsuranceRepository insuranceRepository;

    @Override  1 usage  Katha Patel
    public List<InsuranceDTO> getAllInsurance() {
        return insuranceRepository.findAll().stream().map(insurance -> {
            InsuranceDTO dto = new InsuranceDTO();
            dto.setInsuranceNumber(insurance.getInsuranceNumber());
            dto.setInsuranceProvider(insurance.getInsuranceProvider());
            dto.setCoverageDetails(insurance.getCoverageDetails());
            dto.setInsuranceType(insurance.getInsuranceType());

            // Handle null insuranceDate
            if (insurance.getInsuranceDate() != null) {
                dto.setInsuranceDate(new SimpleDateFormat( pattern: "yyyy-MM-dd").format(insurance.getInsuranceDate()));
            } else {
                dto.setInsuranceDate(null); // Or set a default value, e.g., "N/A"
            }
```



```java
package edu.neu.csye6200.repository;

import edu.neu.csye6200.entity.Insurance;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository  2 usages  Katha Patel
public interface InsuranceRepository extends JpaRepository<Insurance, Integer> {
    // Add custom query methods as needed, for example:
    Optional<Insurance> findByInsuranceNumber(String insuranceNumber);  no usages  Katha Patel
}

```

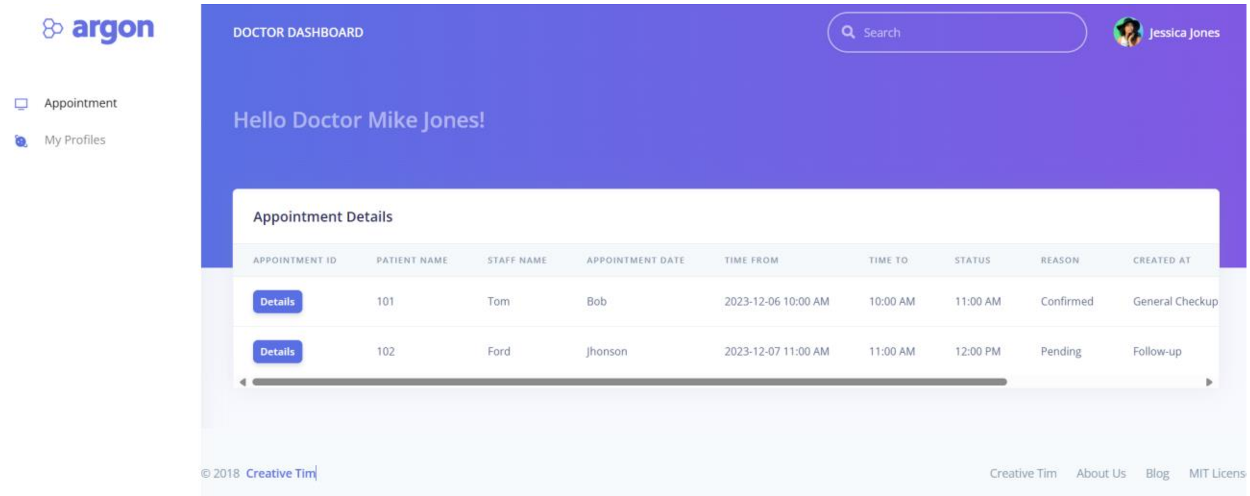
Screenshots of UI:

1. **Login:**



2. **Doctors Dashboard:**

Main Page that will be seen by doctor after logging in.

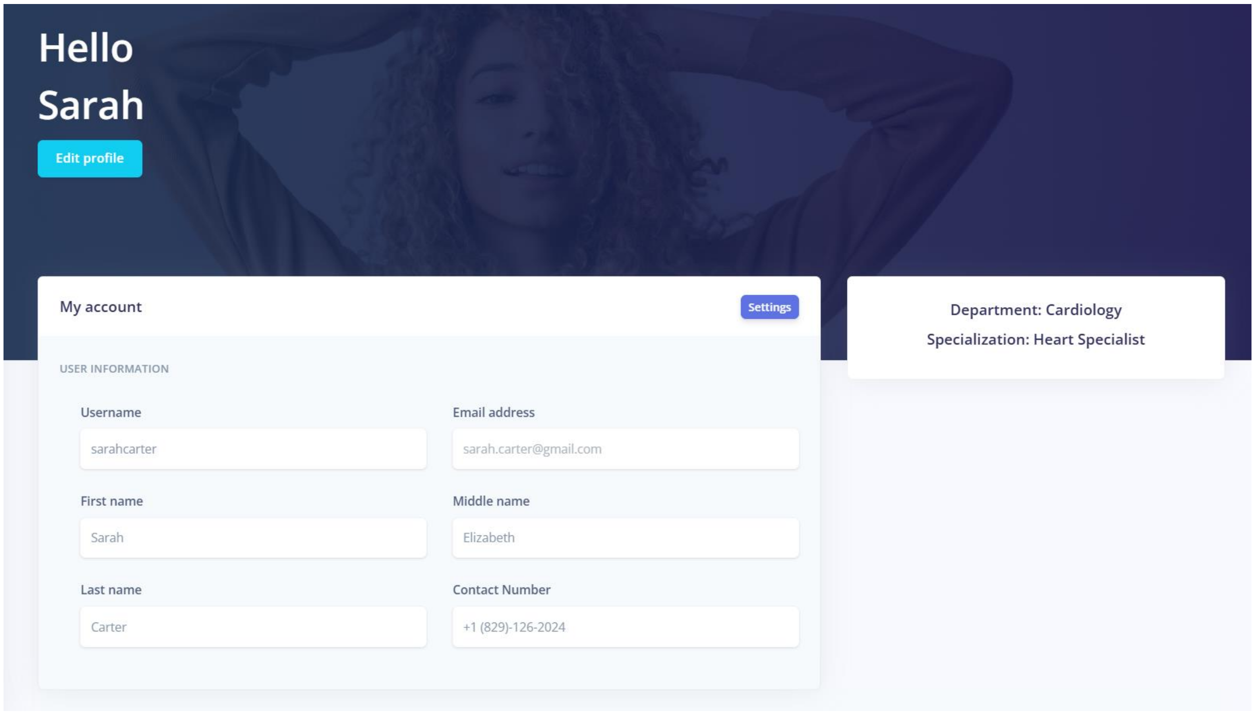


Hello Doctor Mike Jones!

Appointment Details

APPOINTMENT ID	PATIENT NAME	STAFF NAME	APPOINTMENT DATE	TIME FROM	TIME TO	STATUS	REASON	CREATED AT
Details	101	Tom	Bob	2023-12-06 10:00 AM	10:00 AM	11:00 AM	Confirmed	General Checkup
Details	102	Ford	Jhonson	2023-12-07 11:00 AM	11:00 AM	12:00 PM	Pending	Follow-up

3. Doctors My Profile:



Hello Sarah

My account

USER INFORMATION

Username: sarahcarter

Email address: sarah.carter@gmail.com

First name: Sarah

Middle name: Elizabeth

Last name: Carter

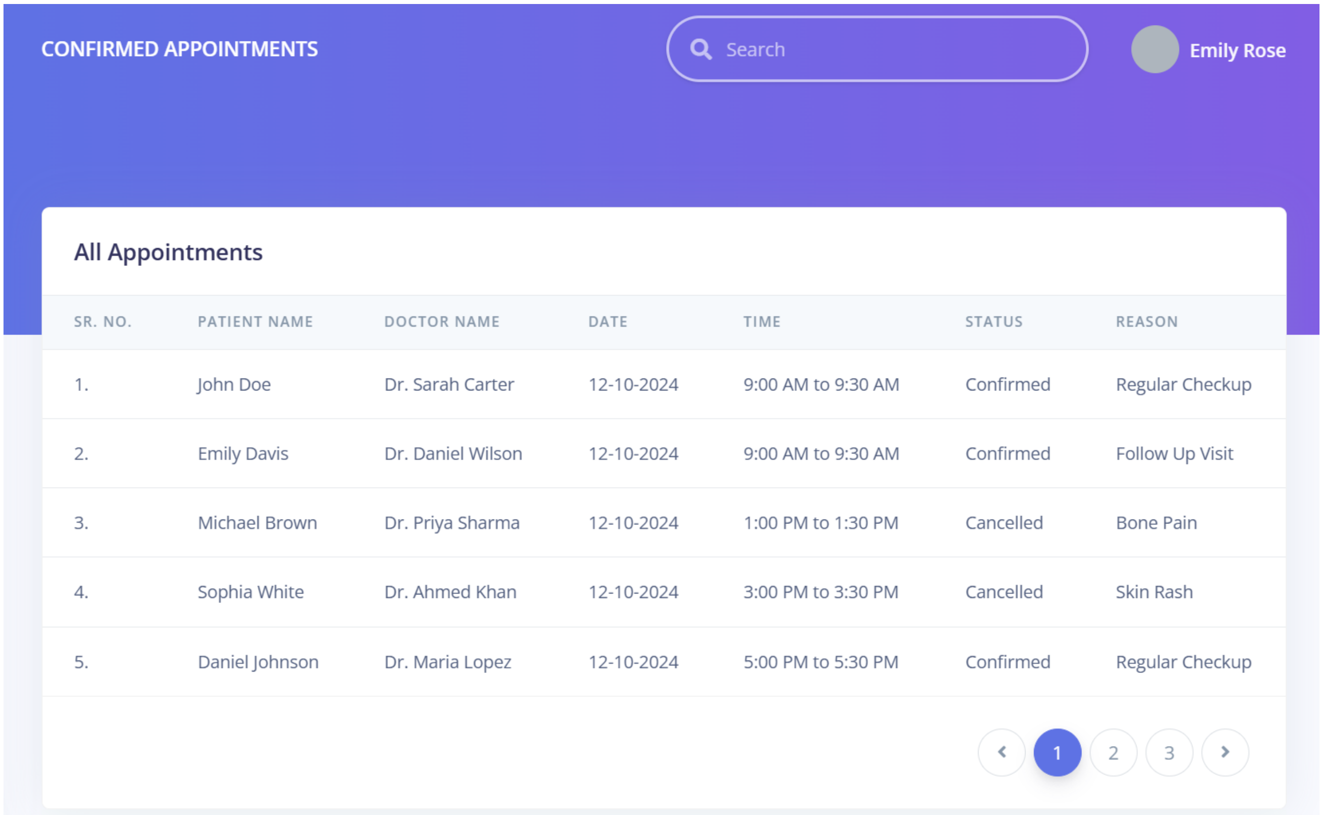
Contact Number: +1 (829)-126-2024

Department: Cardiology

Specialization: Heart Specialist

4. Staffs All Scheduled Appointments:

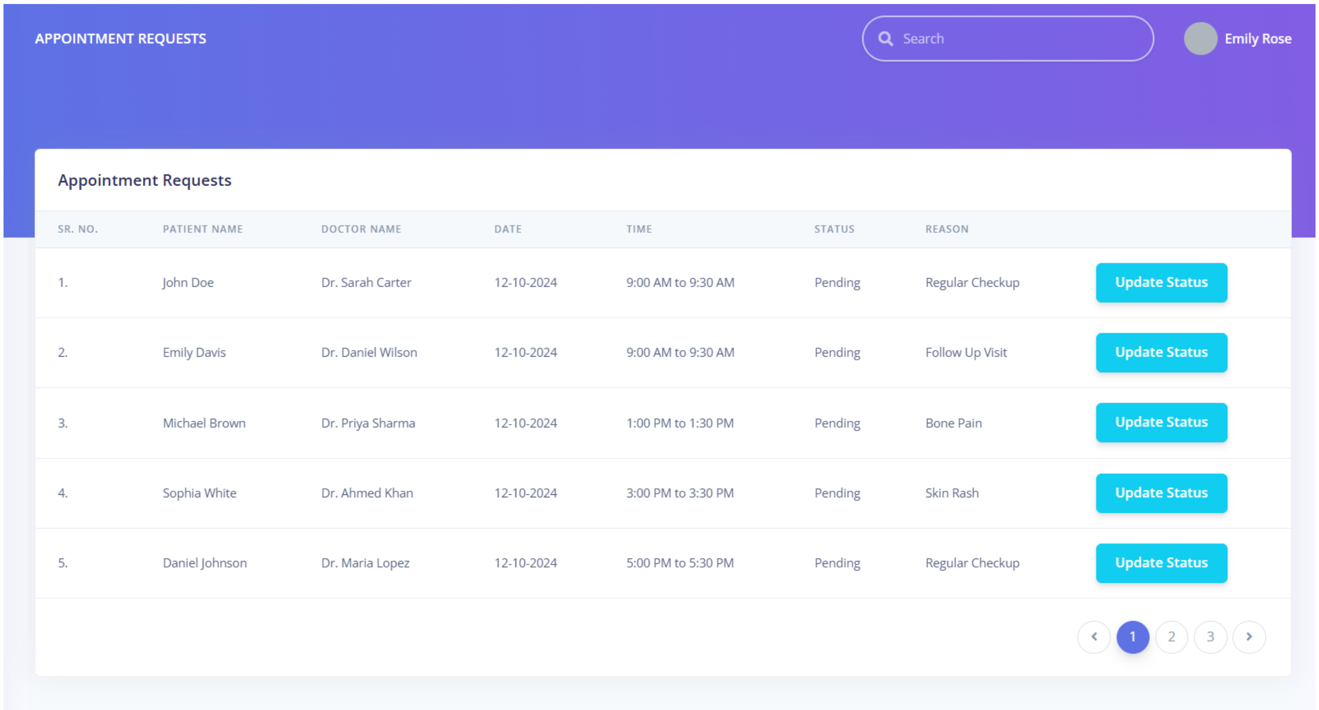
All confirmed appointments which will be seen on staff's dashboard



All Appointments

SR. NO.	PATIENT NAME	DOCTOR NAME	DATE	TIME	STATUS	REASON
1.	John Doe	Dr. Sarah Carter	12-10-2024	9:00 AM to 9:30 AM	Confirmed	Regular Checkup
2.	Emily Davis	Dr. Daniel Wilson	12-10-2024	9:00 AM to 9:30 AM	Confirmed	Follow Up Visit
3.	Michael Brown	Dr. Priya Sharma	12-10-2024	1:00 PM to 1:30 PM	Cancelled	Bone Pain
4.	Sophia White	Dr. Ahmed Khan	12-10-2024	3:00 PM to 3:30 PM	Cancelled	Skin Rash
5.	Daniel Johnson	Dr. Maria Lopez	12-10-2024	5:00 PM to 5:30 PM	Confirmed	Regular Checkup

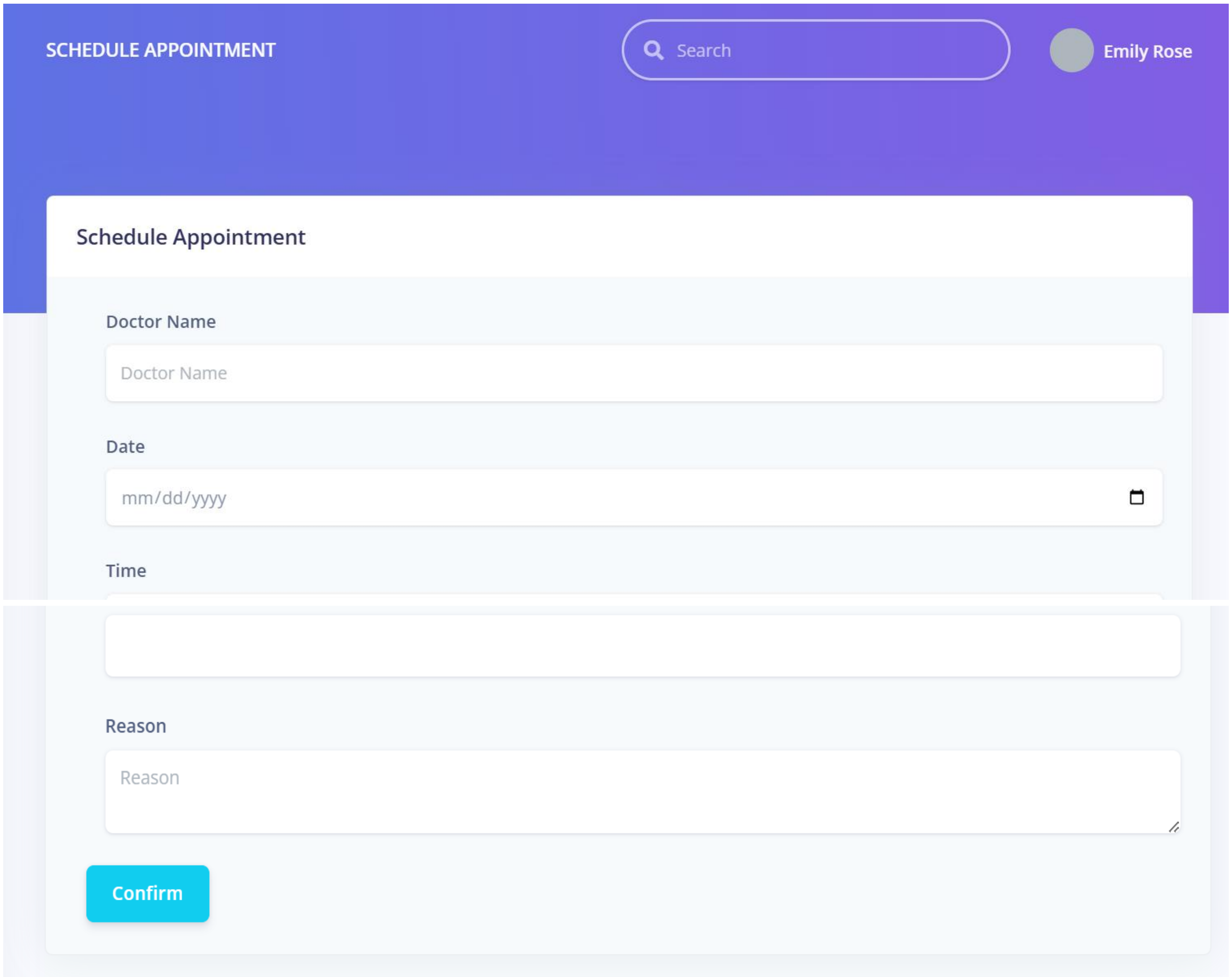
5. **Staffs All Appointment Request:**
   All the appointments which will be seen by staff to confirm



Appointment Requests

SR. NO.	PATIENT NAME	DOCTOR NAME	DATE	TIME	STATUS	REASON	
1.	John Doe	Dr. Sarah Carter	12-10-2024	9:00 AM to 9:30 AM	Pending	Regular Checkup	Update Status
2.	Emily Davis	Dr. Daniel Wilson	12-10-2024	9:00 AM to 9:30 AM	Pending	Follow Up Visit	Update Status
3.	Michael Brown	Dr. Priya Sharma	12-10-2024	1:00 PM to 1:30 PM	Pending	Bone Pain	Update Status
4.	Sophia White	Dr. Ahmed Khan	12-10-2024	3:00 PM to 3:30 PM	Pending	Skin Rash	Update Status
5.	Daniel Johnson	Dr. Maria Lopez	12-10-2024	5:00 PM to 5:30 PM	Pending	Regular Checkup	Update Status

6. **Patients Schedule Appointment:**



Schedule Appointment

Doctor Name

Doctor Name

Date

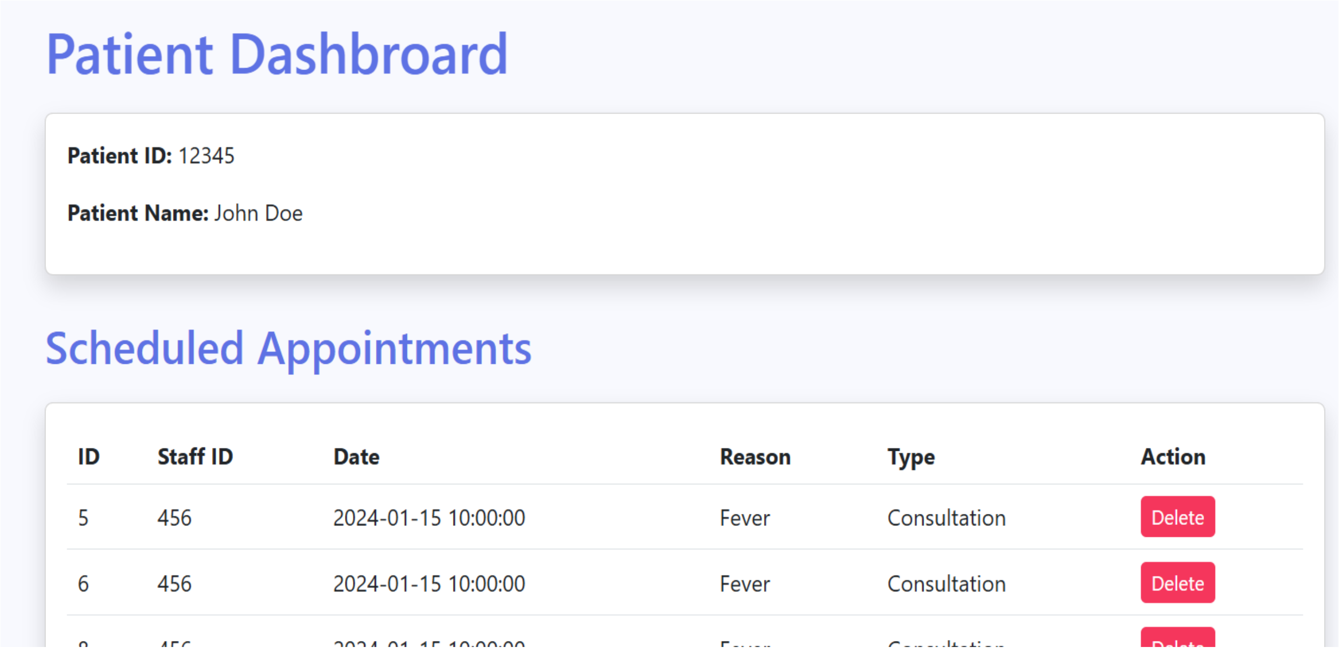
mm/dd/yyyy

Time

Reason

Reason

7. **Patients All Appointments:**



Patient Dashbroard

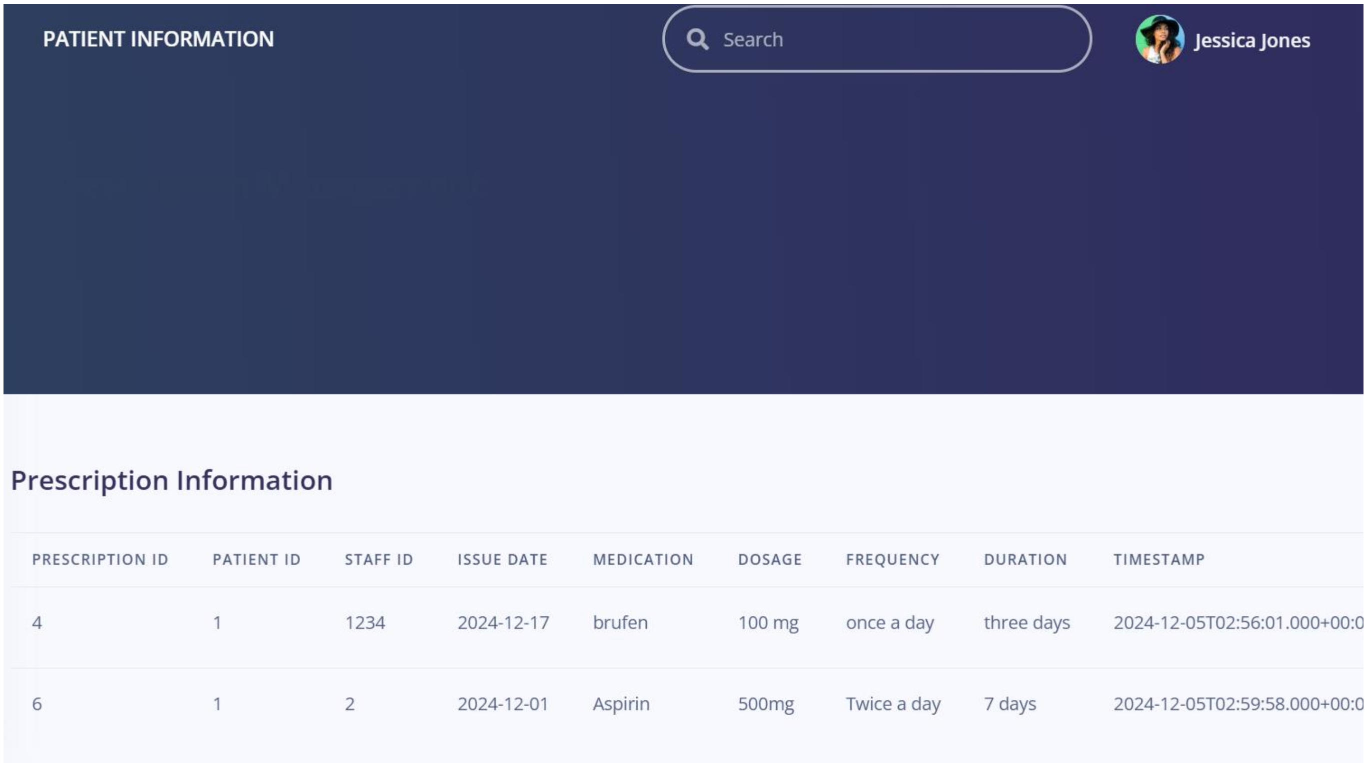
Patient ID: 12345

Patient Name: John Doe

Scheduled Appointments

ID	Staff ID	Date	Reason	Type	Action
5	456	2024-01-15 10:00:00	Fever	Consultation	Delete
6	456	2024-01-15 10:00:00	Fever	Consultation	Delete
8	456	2024-01-15 10:00:00	Fever	Consultation	Delete

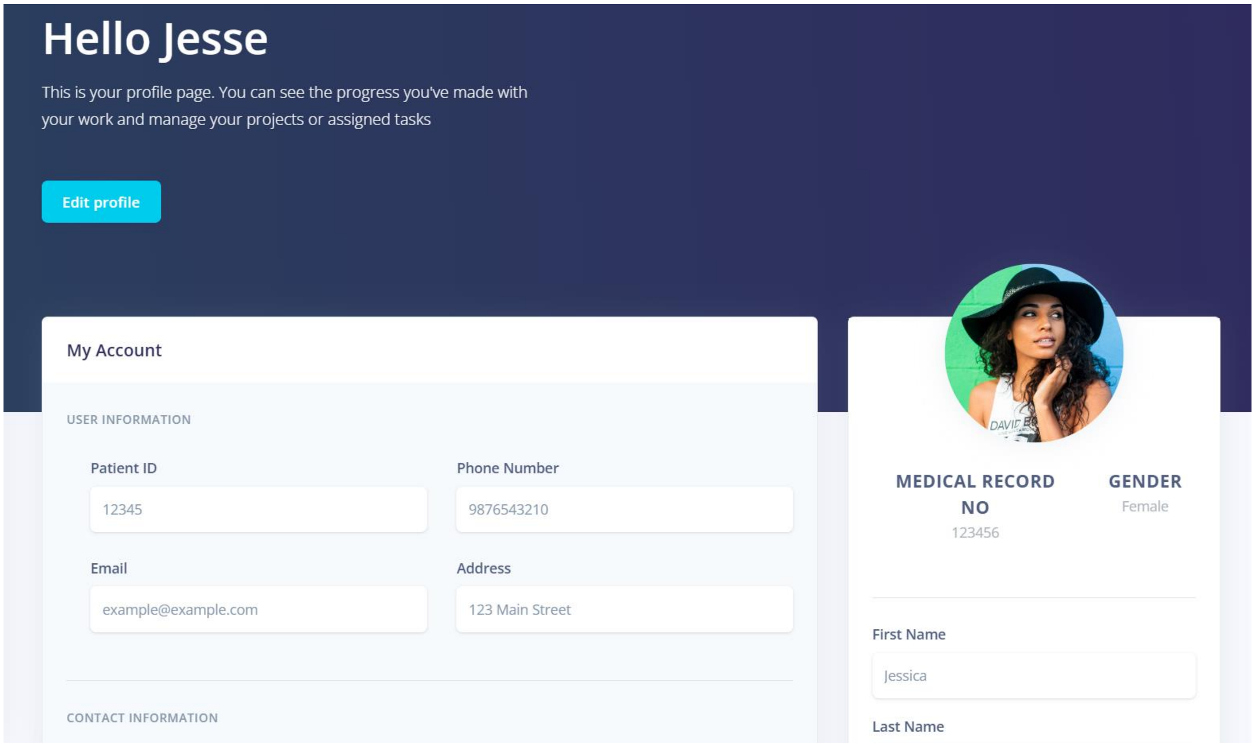
8. **Patients Medication and prescription:**



Prescription Information

PRESCRIPTION ID	PATIENT ID	STAFF ID	ISSUE DATE	MEDICATION	DOSAGE	FREQUENCY	DURATION	TIMESTAMP
4	1	1234	2024-12-17	brufen	100 mg	once a day	three days	2024-12-05T02:56:01.000+00:0
6	1	2	2024-12-01	Aspirin	500mg	Twice a day	7 days	2024-12-05T02:59:58.000+00:0

9. **Patient Profile:**



Hello Jesse

This is your profile page. You can see the progress you've made with your work and manage your projects or assigned tasks

My Account

USER INFORMATION

Patient ID

12345

Phone Number

9876543210

Email

example@example.com

Address

123 Main Street

CONTACT INFORMATION

MEDICAL RECORD NO

123456

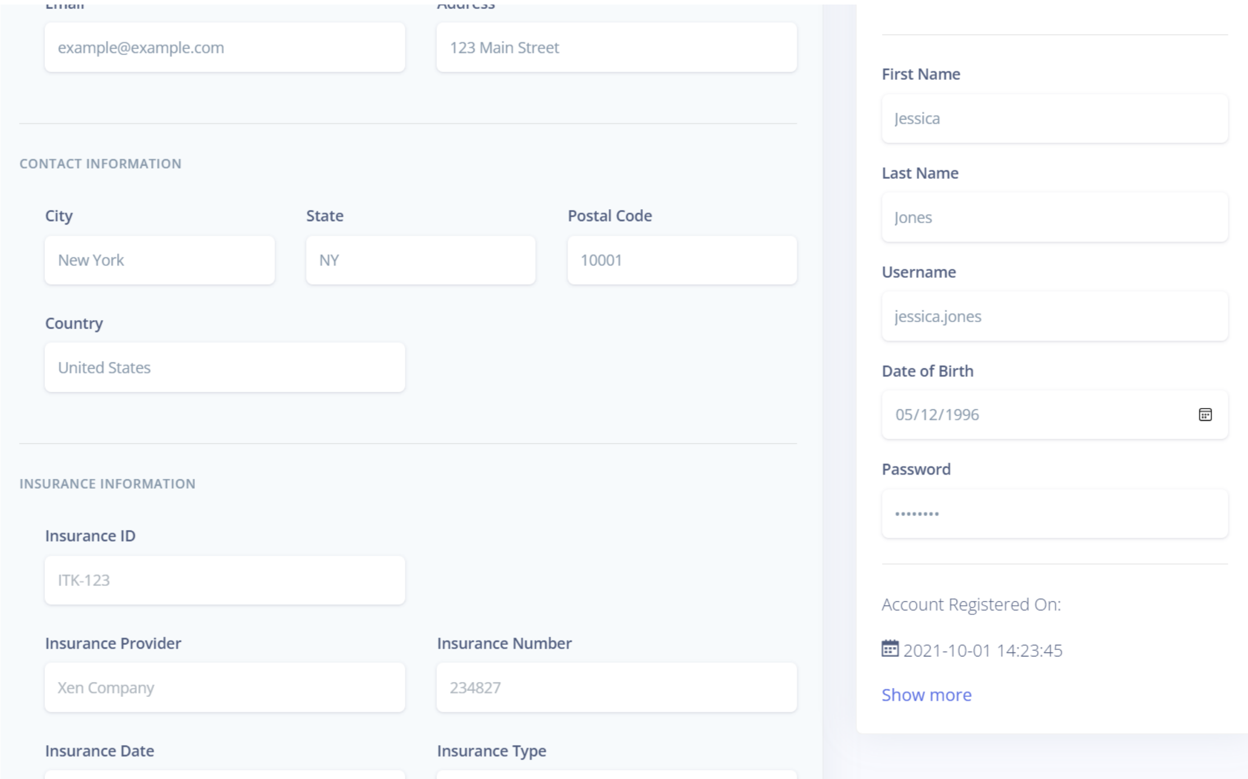
GENDER

Female

First Name

Jessica

Last Name



Email

example@example.com

Address

123 Main Street

CONTACT INFORMATION

City

New York

State

NY

Postal Code

10001

Country

United States

INSURANCE INFORMATION

Insurance ID

ITK-123

Insurance Provider

Xen Company

Insurance Number

234827

Insurance Date

Insurance Type

First Name

Jessica

Last Name

Jones

Username

jessica.jones

Date of Birth

05/12/1996

Password

••••••••

Account Registered On:

2021-10-01 14:23:45

Team Contributions:

Katha Patel

- Developed the Doctor's Dashboard.
- Designed and implemented the Patient Information Profile for doctors.
- Integrated APIs for medical history and patient insurance into the frontend.

Yashika Lodh

- Designed the Staff Appointment Requests interface.
- Developed the Patient Dashboard.
- Created functionality to display all patient appointments.

Nidhi Mehta

- Designed and implemented the Login Page.
- Developed the Doctor's "My Profile" page.
- Built the Staff's Scheduled Appointments interface.

Saniya Azmat

- Created the Staff Dashboard and Staff Profile.
- Integrated Prescription Information APIs with the frontend.
- Integrated APIs for patient basic information into the frontend.