ELDER CARE SERVER DOCUMENTATION

Lead Backend Developer: *Ryan Munga*.

Overview

This project is a *medical records and appointment management system* designed to help healthcare providers manage patient information, track medical records, and streamline scheduling.

Core Features

1. Patient Management
2. Appointment Scheduling and Tracking
3. Medical Records
4. Medication and Prescription Tracking
5. Progress Reports
6. User Access and Roles

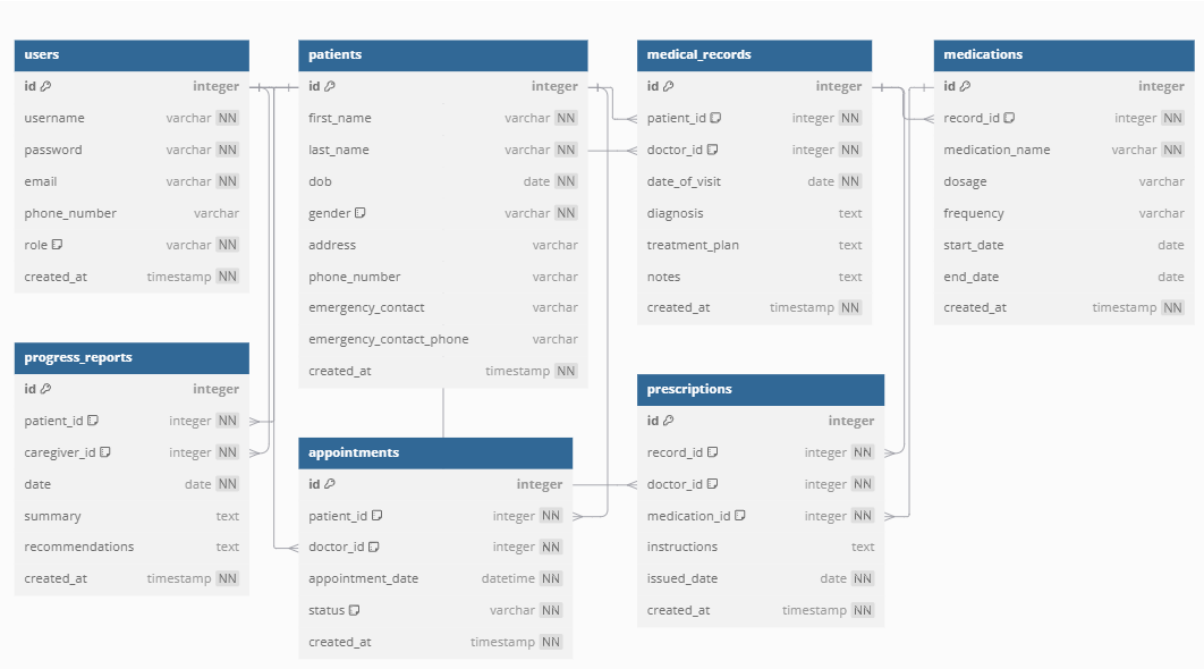
Technical Stack and Structure

The system uses Java with Spring Boot as the backend framework, employing JPA (Java Persistence API) repositories for data access. The backend is structured with various JPA repositories that allow complex queries maintaining a scalable and maintainable codebase. Spring Services and Controllers drive all of this to allow access to the backend.

API Layer	 Spring Web REST controllers
Service Layer	Spring Services. Manages the Business Logic
Data Access Layer	 Spring Data JPA, built on Jakarta Hibernate. Manages CRUD operations by directly interfacing with the Database
Database Layer	

Entities

An overview of the schema:



All entities are defined under the *"entity"* directory in the main source files.

Appointment Entity

The Appointment entity represents a scheduled meeting between a patient and a doctor.

Fields:

- ***id (Long):*** Primary key.
- ***patient (Patient):*** Foreign key reference to the Patient entity.
- ***doctor (User):*** Foreign key reference to the User entity representing the doctor.
- ***appointmentDate (LocalDateTime):*** Date and time of the appointment.
- ***location (String):*** Location of the appointment.
- ***status (String):*** Status of the appointment (e.g., scheduled, completed).
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

MedicalRecord Entity

The MedicalRecord entity stores a patient's medical history related to a specific doctor visit.

Fields:

- ***id (Long):*** Primary key.
- ***patient (Patient):*** Foreign key reference to the Patient entity.
- ***doctor (User):*** Foreign key reference to the User entity representing the doctor.
- ***dateOfVisit (LocalDateTime):*** Date of the medical visit.
- ***location (String):*** Location where the visit occurred.
- ***diagnosis (String):*** Diagnosis given by the doctor.
- ***treatmentPlan (String):*** Outline of the proposed treatment.
- ***notes (String):*** Additional notes related to the visit.
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

Medication Entity

The Medication entity tracks individual medications prescribed as part of a medical record.

Fields:

- ***id (Long):*** Primary key.
- ***medicalRecord (MedicalRecord):*** Foreign key reference to the MedicalRecord entity.
- ***medication Name (String):*** Name of the medication.
- ***dosage (String):*** Dosage information.
- ***frequency (String):*** Frequency of dosage.
- ***startDate (LocalDateTime):*** Start date for the medication.
- ***endDate (LocalDateTime):*** End date for the medication.
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

Patient Entity

The Patient entity represents an individual receiving medical care.

Fields:

- ***id (Long):*** Primary key.
- ***firstName (String):*** Patient's first name.
- ***lastName (String):*** Patient's last name.
- ***dob (LocalDate):*** Date of birth.
- ***gender (String):*** Gender of the patient.
- ***address (String):*** Address of the patient.
- ***phoneNumber (String):*** Contact number.
- ***emergencyContact (String):*** Emergency contact's name.
- ***emergencyContactPhone (String):*** Emergency contact's phone number.
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

Prescription Entity

The Prescription entity contains information about medication prescribed to a patient.

Fields:

- ***id (Long):*** Primary key.
- ***medicalRecord (MedicalRecord):*** Foreign key reference to the MedicalRecord entity.
- ***medication (Medication):*** Foreign key reference to the Medication entity.
- ***doctor (User):*** Foreign key reference to the prescribing doctor.
- ***instructions (String):*** Instructions for taking the medication.
- ***issuedDate (LocalDateTime):*** Date the prescription was issued.
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

ProgressReport Entity

The ProgressReport entity provides updates on a patient's health over time.

Fields:

- ***id (Long):*** Primary key.
- ***patient (Patient):*** Foreign key reference to the Patient entity.
- ***caregiver (User):*** Foreign key reference to the User entity representing the caregiver.
- ***date (LocalDateTime):*** Date of the report.
- ***summary (String):*** Summary of the patient's condition and progress.

- ***recommendations (String):*** Recommendations for future care.
- ***createdAt (LocalDateTime):*** Timestamp for when the record was created.

User Entity

The User entity represents medical personnel, such as doctors and caregivers.

Fields:

- ***id (Long):*** Primary key.
- ***username (String):*** Unique username.
- ***password (String):*** User's password.
- ***email (String):*** Email address.
- ***primaryLocation (String):*** Primary work location.
- ***secondaryLocation (String):*** Secondary work location (optional).
- ***phoneNumber (String):*** Contact number.
- ***role (String):*** Defines the user's role, which can be **"doctor," "nurse," or "patient."**
- ***privileges (String):*** Specifies the user's access level within the system. Options include **"overseer," "admin," "supervisor," "editor," or "viewer."**
- ***createdAt (LocalDateTime):*** Timestamp for when the user account was created.

Note on User Constraints:

Configuration for a User's role & privileges can be found under the "*config*" directory in the main source files

accessConfig.java

```java
package com.app4080.eldercareserver.config;

import java.util.HashMap;

public class accessConfig {  5 usages  Ryan Munga
    static HashMap<String, Integer> tier = new HashMap<>();  6 usages

    static {
        tier.put("viewer", 1);
        tier.put("editor", 2);
        tier.put("supervisor", 3);
        tier.put("admin", 4);
        tier.put("overseer", 5);
    }

    public static int getTier(String role) {  3 usages  Ryan Munga
        return tier.getOrDefault(role,  defaultValue: -1);
    }
}

```

roleClusterConfig.java

```java
package com.app4080.eldercareserver.config;

import java.util.List;

public class roleClusterConfig {  8 usages  Ryan Munga
    static List<String> all_roles;  2 usages
    static List<String> role_nurse;  2 usages
    static List<String> role_doctor;  2 usages
    static List<String> staff;  2 usages

    static {
        all_roles = List.of("patient", "doctor", "nurse");
        role_nurse = List.of( e1: "nurse");
        role_doctor = List.of( e1: "doctor");
        staff = List.of("doctor", "nurse");
    }
```

Data Access (JPA Repositories)

The application uses repositories from Spring Data JPA to manage Data Access. This notably includes CRUD (Create, Read, Update, Delete) functionality, as well as bespoke Queries.

AppointmentRepository

Manages CRUD operations for Appointment entities and provides custom queries:

- ***findByPatientId(Long patientId):*** Retrieves all appointments for a specific patient.
- ***findByDoctorId(Long doctorId):*** Retrieves all appointments for a specific doctor.
- ***findByStatus(String status):*** Finds appointments by their status **(active, cancelled, or completed).**
- ***findByPatientIdAndDoctorId(Long patientId, Long doctorId):*** Retrieves appointments for a specific patient-doctor pair.
- ***findByLocationAndDoctorId(String location, Long doctorId):*** Finds appointments at a specific location for a doctor.
- ***findByLocation(String location):*** Finds appointments by location.
- ***findByPatientIdAndStatus(Long patientId, String status): Retrieves*** appointments for a patient with a specific status.
- ***findByDoctorIdAndStatus(Long doctorId, String status):*** Retrieves appointments for a doctor with a specific status.
- ***findByAppointmentDateBetween(LocalDateTime startDate, LocalDateTime endDate):*** Finds appointments within a specified date range.
- ***findUpcomingAppointments():*** Finds upcoming scheduled appointments.
- ***findOverlappingAppointments(Long doctorId, LocalDateTime startTime, LocalDateTime endTime):*** Checks for overlapping appointments for a doctor within a time range.

MedicalRecordRepository

Manages CRUD operations for MedicalRecord entities and custom queries:

- ***findByPatientId(Long patientId):*** Retrieves all medical records for a specific patient.
- ***findByDoctorId(Long doctorId):*** Retrieves all medical records created by a specific doctor.

- ***findByPatientIdAndDoctorId(Long patientId, Long doctorId):*** Finds records for a patient-doctor pair.
- ***findByDoctorIdAndDateOfVisit(Long doctorId, LocalDateTime dateOfVisit):*** Finds records for a doctor on a specific visit date.
- ***findAll():*** Retrieves all records.
- ***findByLocation(String location):*** Finds records based on location.
- ***findByDateOfVisitBetween(LocalDateTime startDate, LocalDateTime endDate):*** Finds records within a date range.
- ***searchRecords(String searchTerm):*** Searches records by diagnosis or treatment.

MedicationRepository

Manages CRUD operations for Medication entities and provides:

- ***findByMedicalRecord(MedicalRecord mr):*** Finds medications related to a specific medical record.
- ***findActiveMedications():*** Retrieves medications still active based on endDate.
- ***findByMedicationNameContainingIgnoreCase(String name):*** Finds medications by name (case-insensitive).
- ***findMedicationsExpiringSoon(LocalDateTime date):*** Retrieves medications expiring soon within a specified date.

PatientRepository

Manages CRUD operations for Patient entities and includes:

- ***findByLastName(String lastName):*** Finds patients by last name.
- ***findByFirstNameAndLastName(String firstName, String lastName):*** Finds patients by first and last name.
- ***searchPatients(String searchTerm):*** Searches patients by name, phone number, or emergency contact.
- ***findByAgeRange(int minAge, int maxAge):*** Retrieves patients within a specified age range.

PrescriptionRepository

Manages Prescription entities with the following:

- ***findByMedicalRecord(MedicalRecord mr):*** Finds prescriptions for a medical record.
- ***findByDoctorId(Long doctorId):*** Finds prescriptions issued by a doctor.
- ***findByMedicationId(Long medicationId):*** Finds prescriptions for a specific medication.
- ***findActivePrescriptions():*** Retrieves active prescriptions based on medication end date.
- ***findByIssuedDateBetween(LocalDateTime startDate, LocalDateTime endDate):*** Finds prescriptions within a date range.
- ***findByPatientId(Long patientId):*** Finds prescriptions for a specific patient.

ProgressReportRepository

Manages ProgressReport entities and provides:

- ***findByPatientId(Long patientId):*** Finds progress reports for a patient.
- ***findByCaregiverId(Long caregiverId):*** Finds reports created by a caregiver.
- ***findByDateBetween(LocalDateTime startDate, LocalDateTime endDate):*** Finds reports within a date range.
- ***findLatestReportsForAllPatients():*** Retrieves the latest report for each patient.
- ***searchReports(String searchTerm):*** Searches reports by summary or recommendations.

UserRepository

Manages User entities and includes:

- ***findByUsername(String username):*** Finds a user by username.
- ***findByRole(String role):*** Finds users by role (e.g., doctor, caregiver).
- ***findByPrivileges(String privileges):*** Finds users by specific privileges.
- ***findByEmail(String email):*** Finds users by email.
- ***findByPrimaryLocation(String primaryLocation):*** Finds users by primary location.
- ***findBySecondaryLocation(String secondaryLocation):*** Finds users by secondary location.
- ***existsByUsername(String username):*** Checks if a username exists.

- ***existsByEmail(String email):*** Checks if an email exists.
- ***searchUsers(String searchTerm):*** Searches users by username, email, or phone number.

Business Logic (Service Layer)

This is the core of the application. An intermediary between the API and Data Access layer, the Service layer manages the day to day operations of the system. It is built using Services defined by Spring Boot. These Services can be found under the "*service*" directory in the source files.

User Service

Provides core functionality for managing User accounts including methods for access control, CRUD operations, and various relevant queries.