

IT IS 6120/8120 - Project 2 Report

# Electronic Medical Record (EMR) System - Project 2

Robust Medical Database System

Prakasam Venakatachalam (SID: 8013749080), Keerthi  
Erukulla (SID: 801452386), Siddarth Gabburi (SID: )  
Rakesh Kumar Reddy Dasari Ganga Sai (SID: 801461580)

12-4-2025

# Table of Contents

1.	Introduction .....	3
1.1	Project Description.....	3
1.2	Purpose.....	3
1.3	Architecture .....	4
1.4	System Overview .....	5
1.5	Stakeholders and User Roles .....	5
1.6	References .....	6
2.	Scope.....	7
2.1	In-Scope .....	7
2.2	Out-of-Scope.....	7
3.	Functional Requirements (FR) .....	8
3.1	Authentication and User Management .....	8
3.2	Role-Based Access Control (RBAC) .....	8
3.3	Department and Staff Management .....	8
3.4	Patient Registration and Demographics.....	9
3.5	Visits and Clinical Documentation .....	9
3.6	Prescription Management.....	9
3.7	Pharmacy Dispensing Workflow.....	9
3.8	Patient Portal.....	10
3.9	Audit Logging .....	10
3.10	Stored Procedure–Based Access .....	10
4.	Non-Functional Requirements.....	12
4.1	Security .....	12
4.2	Performance.....	12
4.3	Reliability and Data Integrity .....	12
4.4	Usability .....	12
4.5	Maintainability and Extensibility .....	13
5.	Data and Database Requirements.....	14
5.1	Core Entities .....	14

5.2	Data Model and Normalization .....	14
5.3	Views, Triggers, and Indexes.....	14
5.4	Test Data .....	15
6.	Interface Requirements .....	16
6.1	Application to Database .....	16
6.2	Presentation Layer (for demo) .....	16
7.	Changes to Project 1 Database Design .....	17
7.1	New User & Security Tables .....	17
7.2	Department Structure .....	17
7.3	Patient & Visit Enhancements .....	17
7.4	Audit Trail .....	17
7.5	Stored Procedures & Views.....	18
7.6	Pharmacy Module .....	18
7.7	Performance Improvements.....	18
8.	How to Install and Run the Student Clinic EMR Application .....	19
8.1	Prerequisites.....	21
8.2	Source Code Structure .....	21
8.2.1	Step 1 – Set Up the Database .....	21
8.2.2	Step 2 – Configure and Run the API (StudentClinicAPI) .....	22
8.2.3	Step 3 – Configure and Run the Angular App (EMRApp) .....	23
8.2.4	Verifying the System End-to-End .....	25
8.2.5	Common Issues & Quick Fixes.....	25
9.	Demo Screenshots .....	27

# 1. Introduction

## 1.1 Project Description

The Student Clinic EMR System is a full-stack Electronic Medical Records app designed for a university clinic to manage patient records, visits, and prescriptions securely. The backend uses ASP.NET Core 7 Web API with Entity Framework Core and Dapper, while the frontend runs on Angular 17 and Bootstrap 5. This setup gives clinical staff and patients a modern, responsive interface. The database uses MySQL 8.0 and includes over 22 stored procedures, seven audit triggers, more than 14 performance indexes, and 7 security views to ensure efficient data access, strong auditing, and data isolation by department.

Security and governance are key parts of the system. The app uses JWT authentication, SHA-256 password hashing, and role-based access control for eight user roles: Doctor, Nurse, Hospital Support, Manager, IT Admin, Pharmacist, Pharmacy Manager, and Patient. There are over 35 RESTful API endpoints that handle tasks like patient registration, visit tracking, prescription management, pharmacy dispensing, user management, and audit log review. The Angular client uses HTTP interceptors and route guards to protect sensitive pages and automatically manage tokens. This project offers an enterprise-level EMR solution focused on security, performance, auditing, and real healthcare workflows.

The Student Clinic EMR System aims to replace paper and spreadsheet workflows with a secure, integrated digital platform in a university clinic. Handling sensitive data, frequent visits, and high staff turnover makes manual processes error-prone, slow, and hard to audit. Building a full-stack EMR with strong authentication, role-based access, and audit trails, the project seeks to enhance care coordination, protect privacy, and give clinicians quick, reliable access to records, prescriptions, and history—similar to real healthcare software.

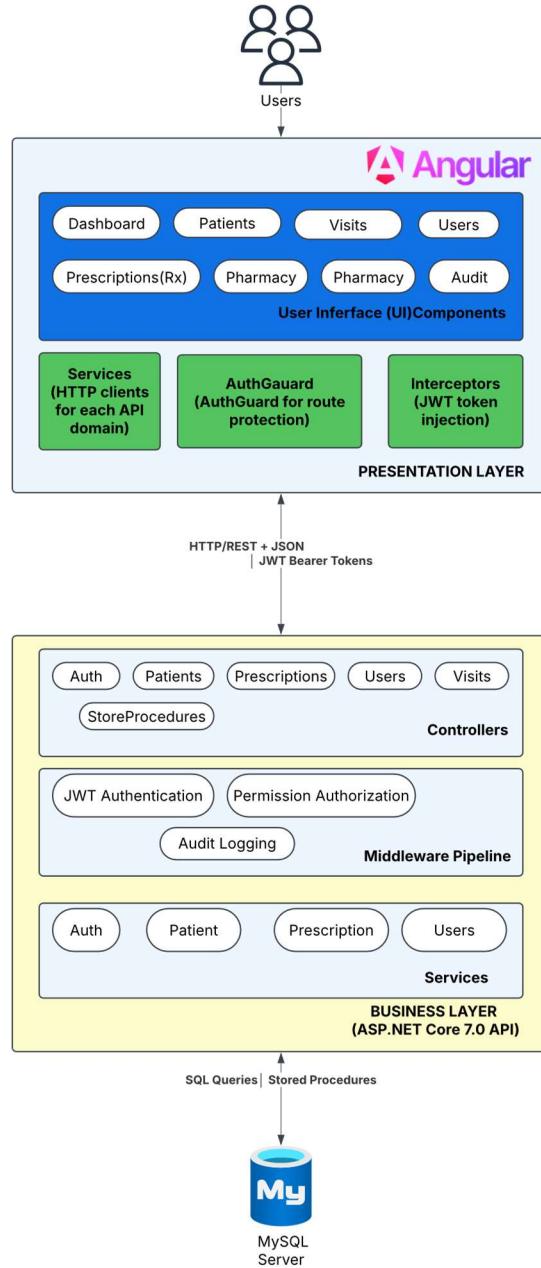
## 1.2 Purpose

This document outlines the functional and non-functional requirements for the Student Clinic EMR System, covering the database, API, and user interfaces (staff UI and patient portal). It is meant for instructors, developers, and testers assessing the project implementation.

## 1.3 Architecture

The EMR system at the Student Clinic is built with a three-layer architecture:

- Angular presentation layer
- ASP.NET Core business layer
- MySQL data layer with SPs, triggers, indexes, and views.



## 1.4 System Overview

The Student Clinic EMR System is a small-scale electronic medical record application for a university clinic. It will:

- Store and manage patient demographics, visits, diagnoses, prescriptions, and pharmacy dispensing.
- Support multiple user roles (Doctor, Nurse, Pharmacist, Front Desk, Manager, IT Admin, Patient).
- Provide a secure patient portal so students can view their own clinical and billing information.
- Enforce role-based access control (RBAC) and department-based visibility.
- Maintain an audit trail for changes to critical clinical data.

The back end is implemented with MySQL (database), stored procedures, triggers, views, and indexes, and an application layer (e.g., text-based or web-based UI) that calls these database objects.

## 1.5 Stakeholders and User Roles

- **Clinic Staff & Providers**
  - Doctor
  - Nurse
  - Hospital Support / Front Desk
  - Pharmacist
  - Pharmacy Manager
  - Manager (Clinic Admin)
- **IT Admin** – Manages users, roles, permissions, and technical configuration.
- **Patient** – Student who logs into the portal to view their own records.

## 1.6 References

- Project assignment description and deliverables.
- Project 1 database schema and scripts (baseline design).
- Updated EMR schema and SQL scripts included in this submission.

## **2. Scope**

### **2.1 In-Scope**

- Patient registration and demographic management.
- Visit and basic clinical documentation.
- Prescription creation and pharmacy dispensing workflow.
- Patient portal views for visits, diagnoses, prescriptions, results, and billing.
- Centralized user and role management with RBAC.
- Audit logging of changes to key clinical tables.
- Performance optimization through indexes and stored procedures.

### **2.2 Out-of-Scope**

- Full billing/invoicing integration with external systems.
- Insurance claims processing.
- Advanced analytics or dashboards.
- Full pharmacy inventory management (only integration points reserved).

### 3. Functional Requirements (FR)

#### 3.1 Authentication and User Management

**FR-1.1** The system shall allow staff and patients to log in using a **unique email** and **password**.

**FR-1.2** The system shall store **password hashes** (e.g., using SHA2()) in MySQL instead of plain-text passwords.

**FR-1.3** The system shall separate user profile data (staff\_user) from authentication credentials (user\_auth).

**FR-1.4** The system shall allow IT Admins to **create, activate/deactivate, and soft-delete** user accounts.

#### 3.2 Role-Based Access Control (RBAC)

**FR-2.1** The system shall support at least the following roles:

Doctor, Nurse, Hospital Support, Manager, IT Admin, Pharmacist, Pharmacy Manager, Patient.

**FR-2.2** The system shall allow each user to be assigned **one or more roles** (user\_role).

**FR-2.3** The system shall represent permissions as **resource-action pairs** (e.g., patient:SELECT, prescription:INSERT) in a permission table.

**FR-2.4** The system shall map roles to permissions using a **many-to-many** relationship (role\_permission).

**FR-2.5** All database operations from the application shall be executed through stored procedures and/or views that enforce the user's permissions.

#### 3.3 Department and Staff Management

**FR-3.1** The system shall maintain a list of **departments** (e.g., Front Desk, Pharmacy, IT, Internal Medicine).

**FR-3.2** The system shall allow staff and providers to be assigned to one or more departments (staff\_department, provider\_department).

**FR-3.3** The system shall restrict staff access so they only see **patients and visits** for their department(s), enforced via department-scoped views.

## 3.4 Patient Registration and Demographics

**FR-4.1** The system shall allow authorized staff (Front Desk, Manager) to **create** and **update** patient records.

**FR-4.2** A patient record shall include:

- Name, date of birth, gender
- Phone, email
- Full mailing address
- Emergency contact information

**FR-4.3** The system shall assign each patient a **unique identifier** (e.g., MRN).

## 3.5 Visits and Clinical Documentation

**FR-5.1** The system shall allow staff to create and update visit records linked to a patient and provider.

**FR-5.2** Each visit shall include date/time, department, provider, visit status, and **chief complaint**.

**FR-5.3** The system shall allow diagnoses and other clinical information to be associated with a visit.

## 3.6 Prescription Management

**FR-6.1** The system shall allow doctors to create prescriptions linked to a visit and patient.

**FR-6.2** Each prescription shall store: medication, dosage, frequency, duration, instructions, and status.

**FR-6.3** The system shall maintain prescription status values such as **Pending**, **Dispensed**, and **Cancelled**.

## 3.7 Pharmacy Dispensing Workflow

**FR-7.1** The system shall provide a **pharmacy queue** view of prescriptions that need to be dispensed.

**FR-7.2** The system shall allow pharmacy staff to record **dispensing events** in `pharmacy_dispense`, including dispenser, date/time, and quantity.

**FR-7.3** The system shall automatically update the prescription status from **Pending** to **Dispensed** when the dispensing procedure completes.

## 3.8 Patient Portal

**FR-8.1** The system shall provide a **patient portal** where a student can log in to view their own EMR data.

**FR-8.2** Each portal user shall be linked to exactly one patient record via `user_patient_link`.

**FR-8.3** Using secure views (e.g., `v_my_visits`, `v_my_prescriptions`), the portal shall allow patients to view only:

- Their own visits
- Their own diagnoses
- Their own prescriptions
- Their own results
- Their own billing information

## 3.9 Audit Logging

**FR-9.1** The system shall log all **INSERT**, **UPDATE**, and **DELETE** operations on critical tables (e.g., patient, visit, prescription, pharmacy dispense) into an `audit_log` table.

**FR-9.2** Each audit record shall include: table name, record key, action type, acting user, timestamp, and JSON snapshots of old/new values.

**FR-9.3** The system shall capture user context (e.g., `@app_user_id`, IP address, user agent) provided from the application layer.

## 3.10 Stored Procedure-Based Access

**FR-10.1** The application shall use **stored procedures** for CRUD operations on patient, visit, prescription, and dispensing entities.

**FR-10.2** The system shall implement procedures for:

- Patient CRUD

- Visit CRUD
- Prescription CRUD and dispensing

**FR-10.3** Direct ad-hoc SQL from the application to core tables shall not be used; all access goes through stored procedures and security views.

## 4. Non-Functional Requirements

### 4.1 Security

**NFR-1.1** All authentication and authorization checks shall be enforced in both the application and database layers.

**NFR-1.2** Passwords shall be stored only as hashes and never in plain text.

**NFR-1.3** The database design (procedures, views) shall minimize exposure to **SQL injection** by using parameters and not concatenating raw input.

### 4.2 Performance

**NFR-2.1** Typical data retrieval operations (e.g., viewing a patient record, daily visit list) shall respond within 2–3 seconds for the anticipated clinic load.

**NFR-2.2** Frequently queried fields (e.g., patient ID, email, visit date, foreign keys) shall be indexed to support acceptable query performance.

### 4.3 Reliability and Data Integrity

**NFR-3.1** The database shall enforce **referential integrity** using primary and foreign keys.

**NFR-3.2** Deletion of logically essential records (e.g., patients, visits) shall preferably be handled via **soft delete** or status flags rather than hard deletion.

### 4.4 Usability

**NFR-4.1** The staff interface (or text-based menu) shall clearly separate workflows: registration, clinical documentation, and pharmacy.

**NFR-4.2** Patient portal views shall present information in simple, student-friendly language.

## 4.5 Maintainability and Extensibility

**NFR-5.1** The schema shall be normalized (3NF or better) to reduce redundancy and simplify maintenance.

**NFR-5.2** The design shall allow new modules (e.g., labs, inventory) to be added without major schema changes.

**NFR-5.3** New roles and permissions shall be addable via data changes (in role and permission tables) without restructuring the schema.

## 5. Data and Database Requirements

### 5.1 Core Entities

The database shall include, at least, tables for:

- patient, visit, diagnosis, prescription, pharmacy\_dispense
- staff\_user, user\_auth, user\_role, user\_patient\_link
- role, permission, role\_permission
- department, staff\_department, provider\_department
- audit\_log

### 5.2 Data Model and Normalization

- The UML data model and ER diagram shall reflect a **normalized** schema (3NF or BCNF where reasonable).
- Many-to-many relationships (e.g., users–roles, roles–permissions, staff–departments) shall be implemented via junction tables.

### 5.3 Views, Triggers, and Indexes

- Security views (e.g., v\_support\_patient\_schedule, v\_pharmacy\_prescriptions, v\_my\_\*) shall be defined to implement **row-level security**.
- Triggers shall be defined to populate the audit\_log table for critical operations.
- Indexes shall be created on frequently used lookup and join columns to support performance.

## 5.4 Test Data

- The database creation script shall include **sample test data** for:
  - Multiple patients, visits, prescriptions.
  - At least one user for each role.
  - At least one record in each department and mapping table.

## 6. Interface Requirements

### 6.1 Application to Database

- The application (presentation layer or text interface) shall connect to the MySQL database using a configured database user account.
- All operations shall call **stored procedures** and **views**, not core tables directly.

### 6.2 Presentation Layer (for demo)

The system shall demonstrate:

- Login and role-based menus (While running as a Local Host application in presentation).
- Basic flows: patient registration, creating a visit, creating a prescription, dispensing in pharmacy, and viewing patient portal data.

## 7. Changes to Project 1 Database Design

This section clearly addresses the requirements. Compared to the original Project 1 design, the following major modifications were implemented:

### 7.1 New User & Security Tables

- Added staff\_user, user\_auth, role, permission, role\_permission, user\_role, user\_patient\_link for unified user management and RBAC.
- Reason: Move from provider-only login to a full multi-role architecture with proper separation of identity and credentials.

### 7.2 Department Structure

- Added department, staff\_department, provider\_department to support department-based filtering.
- Reason: Enforce that staff can only see clinic data belonging to their department(s).

### 7.3 Patient & Visit Enhancements

- Extended patient to include full address and emergency contacts.
- Extended visit to include chief\_complaint.
- Reason: Support more realistic clinical documentation and intake requirements.

### 7.4 Audit Trail

- Added audit\_log table and multiple triggers on patient, visit, prescription, and related tables.

- Reason: Provide a complete audit history for compliance and debugging (who changed what and when).

## 7.5 Stored Procedures & Views

- Replaced direct SQL with **22+ stored procedures** for patient, visit, and prescription workflows.
- Added **security views** (e.g., v\_my\_visits, v\_my\_prescriptions, v\_support\_patient\_schedule, v\_pharmacy\_prescriptions).
- Reason: Centralize business logic, reduce SQL injection risk, and enforce row-level security at the database level.

## 7.6 Pharmacy Module

- Added pharmacy\_dispense table and sp\_dispense\_prescription procedure.
- Reason: Implement a full prescription-to-dispensing workflow instead of just static prescription records.

## 7.7 Performance Improvements

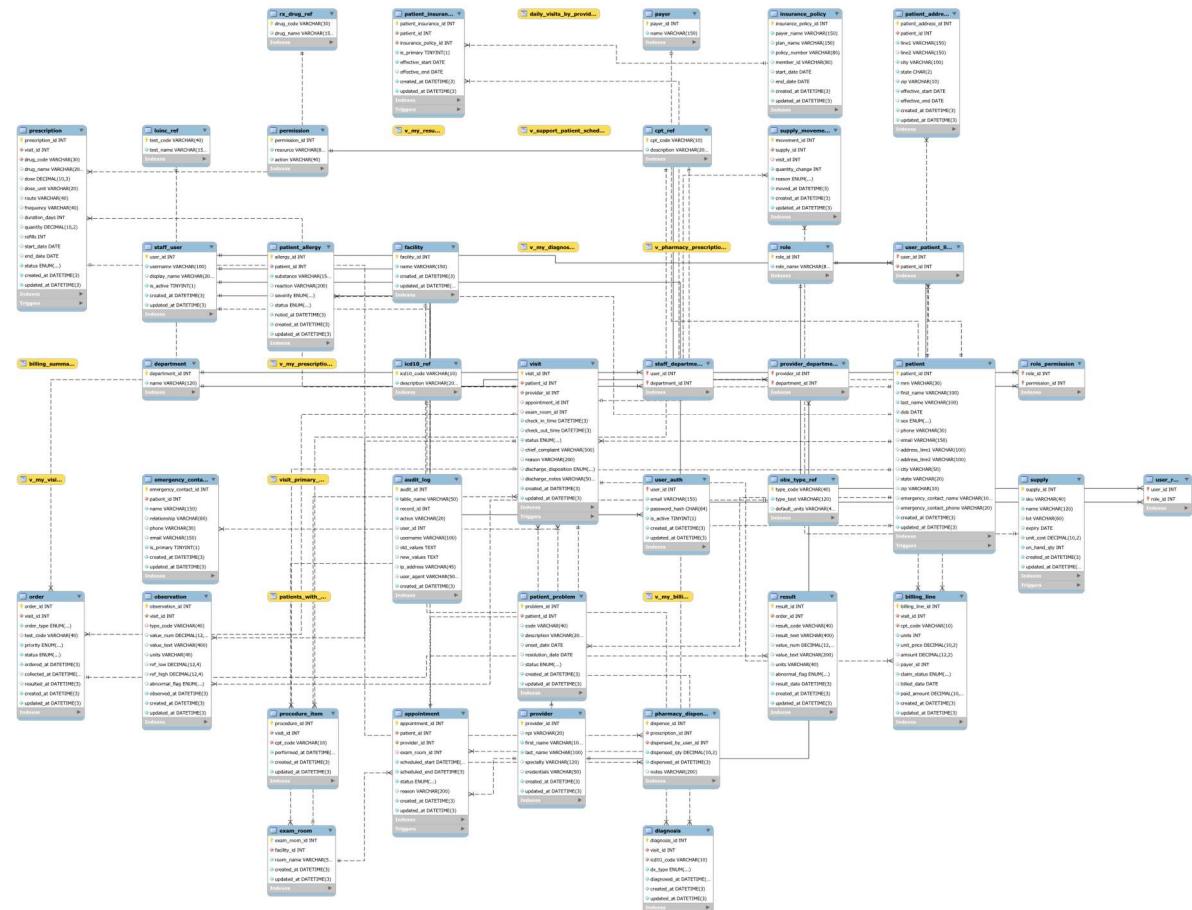
- Added **14+ indexes** on frequently queried columns beyond primary/foreign keys.
- Reason: Improve query performance for common UI and portal operations.

These updates transformed the earlier Project 1 database into a **secure, auditable, performance-optimized EMR backend** ready for demonstration.

# 8. Diagram

## 8.1 ER Diagram

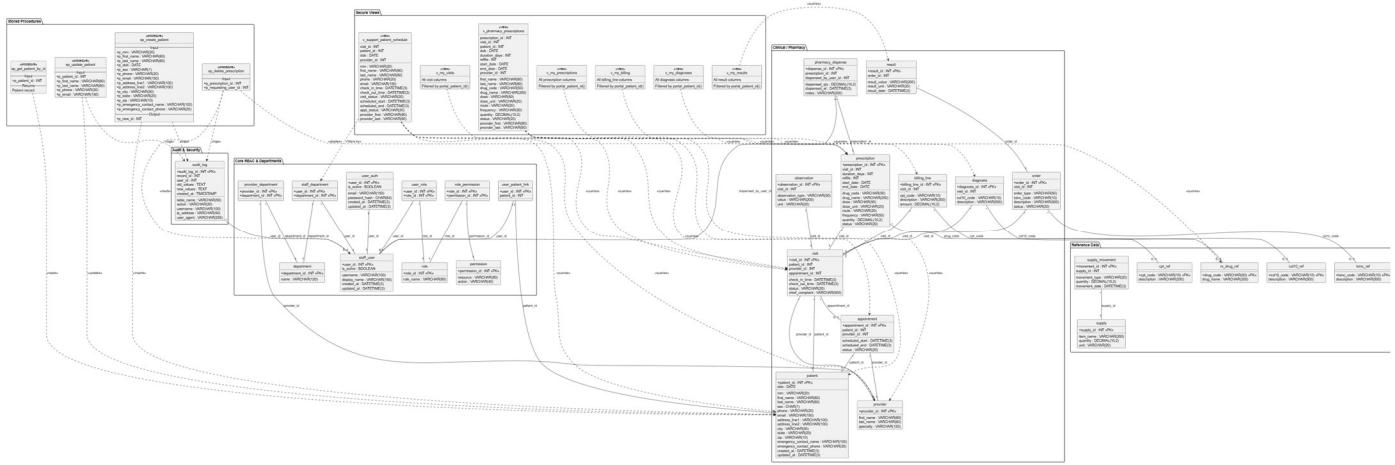
Better view, please open in [https://github.com/prakasamy-uncc/studentClinicAPI/blob/main/Diagram's/ER\\_Diagram.svg](https://github.com/prakasamy-uncc/studentClinicAPI/blob/main/Diagram's/ER_Diagram.svg)



## 8.2 UML Diagram

Open file in VS Code to Preview Diagram : [https://github.com/prakasamv-uncc/studentClinicAPI/blob/main/Diagram's/Student\\_Clinic\\_EMR\\_Complete.plantuml](https://github.com/prakasamv-uncc/studentClinicAPI/blob/main/Diagram's/Student_Clinic_EMR_Complete.plantuml)

To install



## 9. How to Install and Run the Student Clinic EMR Application

### 9.1 Prerequisites

Make sure the following are installed on your machine:

- **MySQL 8.0+**
- **.NET 7 SDK** (for the StudentClinicAPI)
- **Node.js 18+** (or at least 16+)
- **Angular CLI 17+** (install via npm install -g @angular/cli)
- A SQL client (e.g., **MySQL Workbench**)
- An editor/IDE (e.g., Visual Studio / VS Code) – optional but helpful

### 9.2 Source Code Structure

After unzipping the project ZIP:

- **StudentClinicAPI/** – ASP.NET Core 7 Web API
- **EMRApp/** – Angular 17 front-end
- **/sql/** - Database creation + seed scripts (schema + test data)- SQL query  
Attached in zip

#### 9.2.1 Step 1 – Set Up the Database

1. **Start MySQL Server** (e.g., from services or XAMPP).
2. Open **MySQL Workbench** (or your SQL client).
3. Create the database (if not created in the script), for example:

4. CREATE DATABASE student\_clinic\_emr CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;
5. Run the provided SQL scripts in this order (folder names may vary slightly):
  - o 01\_create\_tables.sql (or similar) – creates all tables, views, indexes.
  - o 02\_seed\_data.sql – inserts sample roles, users, departments, patients, etc.
6. Verify that key tables contain data, e.g.:
7. USE student\_clinic\_emr;
8. SELECT \* FROM patient LIMIT 5;
9. SELECT \* FROM role LIMIT 5;
10. SELECT \* FROM staff\_user LIMIT 5;

**Note:** The seed script usually creates **demo users** (doctor, nurse, pharmacist, patient, etc.).

Use those email/password pairs when you log into the UI.

### 9.2.2 Step 2 – Configure and Run the API (StudentClinicAPI)

1. Go to the API project folder:
  2. cd StudentClinicAPI
  3. Open appsettings.json and check the **connection string**:
  4. "ConnectionStrings": {  
    "DefaultConnection":  
        "Server=127.0.0.1;Port=3306;Database=student\_clinic\_emr;User=root;Password=root;  
        AllowUserVariables=True;UseAffectedRows=False;"  
}
- o Make sure:
    - Server and Port match your MySQL server.

- Database matches the DB you created.
- User and Password are your MySQL credentials.

5. Restore NuGet packages and run the API:

6. dotnet restore

7. dotnet run

8. The API will start at:

- **Base URL:** http://localhost:5000
- **Swagger UI:** http://localhost:5000/swagger

9. Open http://localhost:5000/swagger in a browser to confirm:

- You should see all endpoints (Auth, Patients, Visits, Prescriptions, Pharmacy, Users, etc.).
- Try a simple GET, e.g., list patients or roles, to verify the DB connection.

### 9.2.3 Step 3 – Configure and Run the Angular App (EMRAApp)

1. In a new terminal, go to the Angular app folder:

2. cd EMRAApp

3. Install NPM dependencies:

4. npm install

5. Check the API URL in src/environments/environment.ts:

6. 

```
export const environment = {
```

```
  production: false,
```

```
  apiUrl: 'http://localhost:5000/api'
```

```
};
```

## 9.2.4 Step 4 – Install the PlantUML extension in VS Code

1. Open **Visual Studio Code**.
2. Click the **Extensions** icon on the left sidebar (or press Ctrl+Shift+X).
3. In the search box, type: **PlantUML**
4. Install the extension named “**PlantUML**” (usually by *jebbs*).
5. Reload VS Code if it asks.

## 9.2.5 Step 5 Preview the diagram

Open file in VS Code : [https://github.com/prakasamv-uncc/studentClinicAPI/blob/main/Diagram's/Student\\_Clinic\\_EMR\\_Complete.plantuml](https://github.com/prakasamv-uncc/studentClinicAPI/blob/main/Diagram's/Student_Clinic_EMR_Complete.plantuml)

- Option A: **Right-click** in the editor → click “**Preview Current Diagram**”
- Option B: Press **Ctrl+Shift+P** → type “**PlantUML: Preview Current Diagram**” → Enter

A preview pane will open showing the rendered UML.

This should match the API URL you saw earlier (<http://localhost:5000>).

7. Start the Angular development server:
8. `npm start`
9. # or
10. `ng serve`

11. Open the front-end in a browser:

- **URL:** `http://localhost:4200`

12. Log in with one of the **seed users** created by your SQL script (e.g., doctor, pharmacist, patient).

- If you're not sure of the credentials, open your seed SQL and look at the `user_auth` / `staff_user` inserts.

## 9.2.6 Verifying the System End-to-End

Once both the API and the front-end are running:

1. **Login** as a staff user (e.g., Doctor / Nurse):

- URL: `http://localhost:4200`
- Use demo email/password from your seed data.

2. **Test core flows:**

- Create a **patient** in the Patients screen.
- Create a **visit** and record chief complaint.
- Add a **prescription** for that visit.
- Switch to the **Pharmacy** screen to see the prescription queue and mark it as **Dispensed**.

3. **Login as a Patient:**

- Use a portal user created in the seed script.
- Navigate to “**My Visits**” / “**My Prescriptions**” to confirm row-level security is working.

4. Optionally, in MySQL:

- Check `audit_log` to see entries for INSERT/UPDATE/DELETE operations.

## 9.2.7 Common Issues & Quick Fixes

• **API cannot connect to the database**

- Check `appsettings.json` connection string.
- Confirm MySQL is running and DB name/credentials are correct.

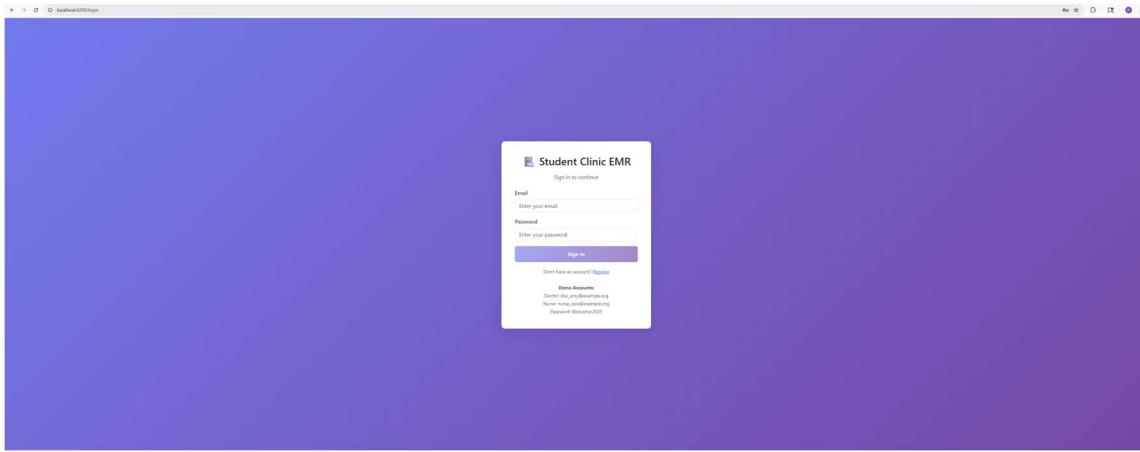
• **Angular app gets CORS errors**

- Ensure the API is running *before* starting Angular.

- CORS is already enabled in the API; if you changed ports/URLs, update the config.
- **Login fails**
  - Double-check the seed SQL for the correct email/password.
  - Ensure passwords weren't changed manually in the DB (they're stored hashed).

## 10. Demo Screenshots

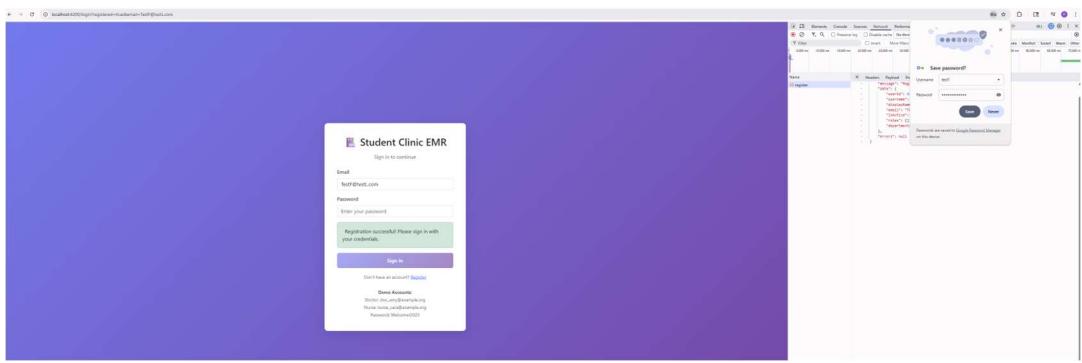
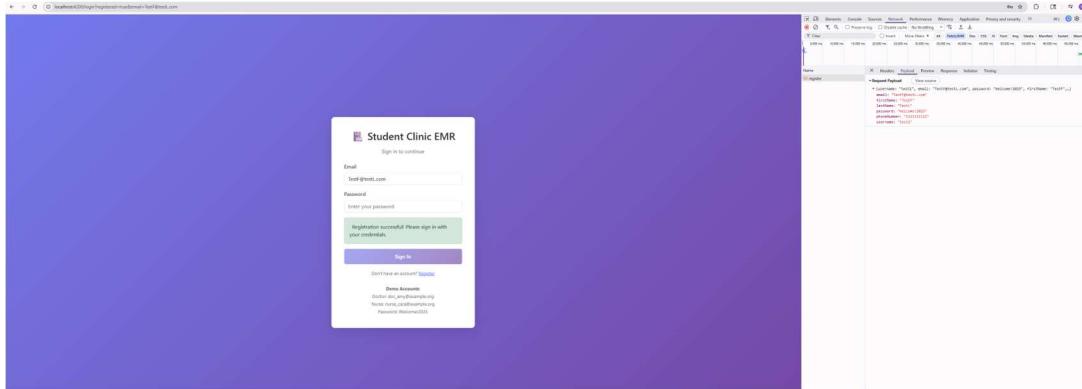
- Login Page



- Successful Login

A screenshot of the 'Student Clinic EMR' dashboard. The top navigation bar shows the user 'doc\_amy' is logged in. The dashboard features a dark sidebar on the left with icons for 'Dashboard', 'Patients', 'Visits', 'Prescriptions', 'Users', and 'Audit Logs'. The main area has a 'Welcome, doc\_amy!' message and four summary cards: '45 Total Patients', '128 Total Visits', '89 Prescriptions', and '12 Pending Dispense'. Below these are 'Quick Actions' buttons for 'New Patient', 'New Visit', 'New Prescription', 'Pharmacy Queue', 'Manage Users', and 'Audit Logs'. To the right, there is a 'Your Profile' section with details: Username 'doc\_amy', Email 'doc\_amy@example.org', Roles 'Doctor', and User ID '5'. On the far right, a developer tool's 'Elements' tab is open, showing the HTML structure of the 'Sign In' button from the previous screenshot. The browser's address bar shows 'localhost:4200'.

- Registration Page



The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema "student\_clinic\_emr" with its tables: appointment, audit\_log, billing\_line, cpt\_ref, department, diagnosis, emergency\_contact, exam\_room, facility, icd10\_ref, insurance\_policy, loan\_ref, observation, obx\_type\_ref, order, patient, patient\_address, patient\_allergy, patient\_insurance, patient\_problem, and naver. The central area shows a SQL editor with the query: `SELECT * FROM student_clinic_emr.user_auth WHERE email='testF@testL.com';`. Below the query is a "Result Grid" table with one row of data:

user_id	email	password_hash	is_active	created_at	updated_at
67	TestF@testL.com	ce50fe604343236fe0979e3b0081999280d68ca...	1	2025-12-05 00:12:49.706	2025-12-05 00:12:49.706

- API Swagger : <http://localhost:5000/swagger/index.html>

The screenshot shows the **Student Clinic EMR API v1** documentation generated by SWAGGER. The top navigation bar includes a logo, a search bar labeled "Select a definition" with "StudentClinicAPI v1", and a help icon.

**AuditLog**

- GET** /api/AuditLog
- GET** /api/AuditLog/{id}

**Auth**

- POST** /api/Auth/login
- POST** /api/Auth/register
- POST** /api/Auth/change-password
- GET** /api/Auth/me

**Departments**

- GET** /api/Departments
- GET** /api/Departments/{id}

**DevAuth**

The screenshot shows the **Student Clinic EMR API v1** documentation generated by SWAGGER. The top navigation bar includes a logo, a search bar labeled "Select a definition" with "StudentClinicAPI v1", and a help icon.

**Patients**

- GET** /api/Patients
- POST** /api/Patients
- GET** /api/Patients/{id}
- PUT** /api/Patients/{id}
- DELETE** /api/Patients/{id}
- GET** /api/Patients/mrn/{mrn}
- GET** /api/Patients/search/{searchTerm}

**Pharmacy**

- GET** /api/Pharmacy/pending
- POST** /api/Pharmacy/dispense
- GET** /api/Pharmacy/dispense-history/{prescriptionId}

**Prescriptions**

- GET** /api/Prescriptions
- POST** /api/Prescriptions
- GET** /api/Prescriptions/{id}
- PUT** /api/Prescriptions/{id}

localhost:5000/swagger/index.html

## Prescriptions

GET /api/Prescriptions

POST /api/Prescriptions

GET /api/Prescriptions/{id}

PUT /api/Prescriptions/{id}

DELETE /api/Prescriptions/{id}

GET /api/Prescriptions/visit/{visitId}

GET /api/Prescriptions/patient/{patientId}

## StoredProcedures

GET /api/sp/patients/{id}

PUT /api/sp/patients/{id}

POST /api/sp/patients

DELETE /api/sp/prescriptions/{id}

## StudentClinicAPI

GET /

GET /test

## Users

localhost:5000/swagger/index.html

## Users

GET /api/Users

GET /api/Users/{id}

PUT /api/Users/{id}

DELETE /api/Users/{id}

POST /api/Users/{userId}/roles/{roleId}

DELETE /api/Users/{userId}/roles/{roleId}

POST /api/Users/{userId}/departments/{departmentId}

DELETE /api/Users/{userId}/departments/{departmentId}

## Visits

GET /api/Visits

POST /api/Visits

GET /api/Visits/{id}

PUT /api/Visits/{id}

DELETE /api/Visits/{id}

GET /api/Visits/patient/{patientId}

## Schemas

localhost:5000/swagger/index.html

Schemas

- AuditLogDto >
- AuditLogDtoApiResponse >
- AuditLogDtoListApiResponse >
- BooleanApiResponse >
- ChangePasswordRequest >
- DevTokenCredentialRequest >
- DevTokenRequest >
- LoginRequest >
- LoginResponse >
- LoginResponseApiResponse >
- ObjectApiResponse >