**Healthcare management Web Application**

**Software Design Document**

Introduction

Software Architecture

Class Diagram

UI Design

Database Design

Security Design

Business Logic and/or Key Algorithms

Design Patterns

Models and Tools

References

Glossary

# 

Introduction

Develop a healthcare-focused web application using .NET Core 8, Angular 18, and Entity Framework core 8 to manage patient recommendations. The system should allow users to log in, browse a list of patients, and view recommendations such as allergy checks and screenings.

## Project Goals

1. **Authentication & Authorization:**
   * Implement secure login functionality.
   * Role-based access control (e.g., Admin, Healthcare Professional).
2. **Patient Management:**
   * Display a list of patients with pagination and filtering options.
   * Search functionality by name or ID.
   * View detailed patient information.
3. **Recommendations:**
   * Display a list of recommendations (Allergy check, screenings, follow-ups) associated with a patient.
   * Allow users to mark recommendations as completed.
4. **Security Considerations:**
   * Implement OWASP security best practices (CSP, anti-CSRF, secure headers).
   * Protect against SQL Injection, XSS, and authentication-related attacks.
5. **Technical Requirements:**
   * Backend: .NET Core 6+ Web API.
   * Frontend: Angular 17+ (using Angular Material components).
   * Database: SQL Server (Entity Framework is optional).
   * API should follow RESTful principles.
6. **Infrastructure:**
   * Deployment-ready configuration (Docker support is a plus).
   * Local development setup instructions

## Project Deliverables:

1. **Source Code:**
   * Well-structured code repository (GitHub/GitLab/Bitbucket).
   * Adherence to clean architecture principles.
2. **Documentation:**
   * Setup guide for running the project locally.
   * Brief overview of the architecture.
   * API documentation using Swagger/OpenAPI.
3. **Bonus Considerations:**
   * Implement audit logging for patient interactions.
   * Responsive design for mobile and tablet devices.

## **Technical Requirements**

* **IDE:** Visual Studio 2022
* **Backend:** .NET Core 8+ Web API
* **Frontend:** Angular 18 (utilizing Angular Material components)
* **Database:** SQL Server Management Studio and Entity Framework Core 8
* **API Design:** Follow RESTful principles

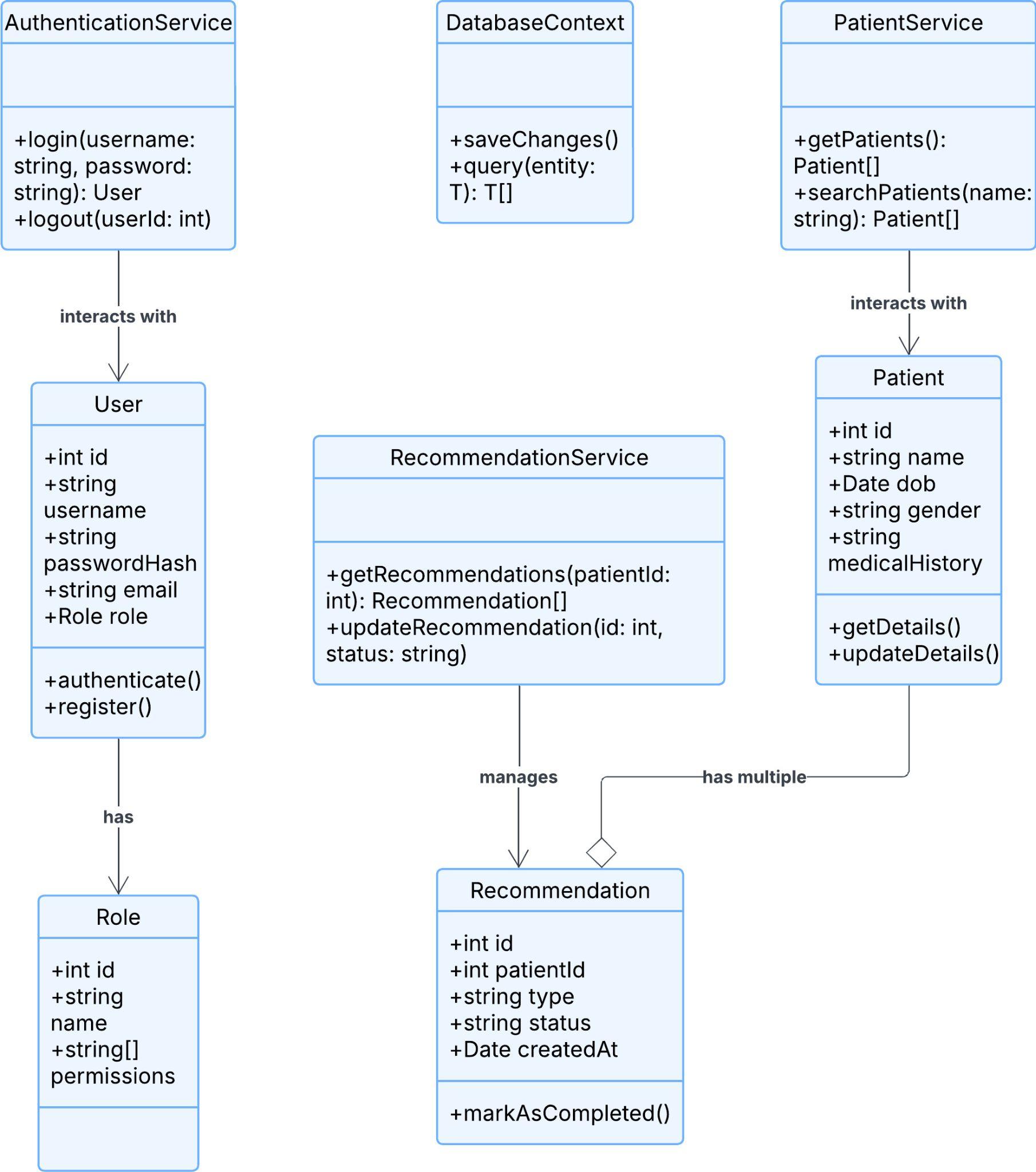
# Software Architecture

The application follows a layered architecture consisting of:

* **Presentation Layer:** Angular 18 frontend using Angular Material for UI components.
* **Business Logic Layer:** .NET Core 8 Web API handling application logic and user interactions.
* **Data Access Layer:** Entity Framework Core 8 for database operations.
* **Database Layer:** SQL Server storing patient data and recommendations.
* **Security Layer:** Implements authentication, authorization, and OWASP security practices.

Data flow is managed via RESTful APIs, ensuring a seamless exchange between the frontend and backend while maintaining security and scalability.

# **Class Diagram**



Application Class Diagram

* User (Represents application users)
  + id: int
  + username: string
  + passwordHash: string
  + email: string
  + role: Role
  + + authenticate()
  + + register()
* Role (Defines user roles)
  + id: int
  + name: string
  + permissions: string[]
* Patient (Represents patients)
  + id: int
  + name: string
  + dob: Date
  + gender: string
  + medicalHistory: string
  + + getDetails()
  + + updateDetails()
* Recommendation (Represents recommendations for patients)
  + id: int
  + patientId: int
  + type: string
  + status: string
  + createdAt: Date
  + + markAsCompleted()
* AuthenticationService (Handles authentication & authorization)
  + + login(username: string, password: string): User
  + + logout(userId: int)
* PatientService (Manages patient-related operations)
  + + getPatients(): Patient[]
  + + searchPatients(name: string): Patient[]
* RecommendationService (Handles patient recommendations)
  + + getRecommendations(patientId: int): Recommendation[]
  + + updateRecommendation(id: int, status: string)
* DatabaseContext (Manages database interactions using Entity Framework Core)
  + + saveChanges()
  + + query<T>(entity: T): T[]

Relationships:

* User has a Role
* Patient has multiple Recommendations
* AuthenticationService interacts with User
* PatientService interacts with Patient
* RecommendationService manages Recommendation

**Notes:**

* The SecurityManager handles user authentication and authorization.
* DatabaseContext is responsible for Entity Framework core's interactions with SQL Server.
* Admin and HealthcareProfessional roles will have different access levels, controlled by Role-based Access Control.
* Recommendation and Allergy models represent the medical suggestions or alerts for a patient.

# 

# **Database Design**

The database consists of the following tables:

Tables and Schema

1. Users (Stores user authentication and role data)
   * id (Primary Key, int, Auto-increment)
   * username (nvarchar(50), Unique, Not Null)
   * passwordHash (nvarchar(255), Not Null)
   * email (nvarchar(100), Unique, Not Null)
   * roleId (Foreign Key -> Roles.id, Not Null)
2. Roles (Defines user roles)
   * id (Primary Key, int, Auto-increment)
   * name (nvarchar(50), Unique, Not Null)
   * permissions (nvarchar(255), Not Null)
3. Patients (Stores patient details)
   * id (Primary Key, int, Auto-increment)
   * name (nvarchar(100), Not Null)
   * dob (Date, Not Null)
   * gender (nvarchar(10), Not Null)
   * medicalHistory (text, Nullable)
4. Recommendations (Stores recommendations for patients)
   * id (Primary Key, int, Auto-increment)
   * patientId (Foreign Key -> Patients.id, Not Null)
   * type (nvarchar(50), Not Null)
   * status (nvarchar(50), Not Null)
   * createdAt (DateTime, Default: GETDATE())
5. AuditLogs (Tracks changes and security events)
   * id (Primary Key, int, Auto-increment)
   * userId (Foreign Key -> Users.id, Nullable)
   * action (nvarchar(255), Not Null)
   * timestamp (DateTime, Default: GETDATE())

Relationships:

* Users have a Role (roleId Foreign Key).
* Patients have multiple Recommendations (patientId Foreign Key).
* AuditLogs track actions related to users and system changes.

# **Security Design**

The application incorporates multiple security measures to ensure data protection and user authentication:

#### **Authentication & Authorization**

* Implement JWT-based authentication.
* Enforce role-based access control (RBAC) with predefined roles (Admin, Healthcare Professional).
* Secure API endpoints with authorization policies.

#### **Data Security**

* Encrypt sensitive data (e.g., passwords using bcrypt).
* Implement database encryption for stored patient information.
* Utilize HTTPS for all communications.

#### **Application Security Measures**

* Prevent SQL Injection using parameterized queries.
* Implement Cross-Site Request Forgery (CSRF) protection.
* Secure headers with Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS).

#### **Logging & Monitoring**

* Maintain audit logs for critical user actions (e.g., login attempts, data modifications).
* Implement real-time security monitoring and alerting for suspicious activities.
* Use rate-limiting and IP throttling to prevent brute-force attacks.

# **Business Logic and/or Key Algorithms**

The application enforces key business logic, including:

* **User Authentication & Authorization:** JWT-based authentication with role-based access control.
* **Patient Search & Filtering:** Optimized search queries with pagination and filters.
* **Recommendation Status Management:** Users can update the status of recommendations, triggering audit logs.
* **Security Compliance Checks:** Ensuring compliance with OWASP and HIPAA guidelines through automated validation.
* **Audit Logging:** Automatic logging of important events, such as user logins and data changes.

# 

# 

# **Design Patterns**

The application implements the following design patterns:

1. **Repository Pattern** - To abstract database access and ensure a clean separation of concerns between business logic and data persistence.
2. **Factory Pattern** - For creating different types of recommendations dynamically.
3. **Singleton Pattern** - Ensuring a single instance of important services, such as logging and configuration management.
4. **Observer Pattern** - Used for notification services, such as sending alerts when a patient’s recommendation status changes.
5. **Decorator Pattern** - For extending functionalities, such as adding logging or caching to service layers without modifying their code.

# **Model and Tools**

### **Models, Frameworks, Libraries, and Tools Utilized**

#### **Models**

* **Patient**: Represents a patient's personal information and medical history.
* **Recommendation**: Represents a patient's healthcare recommendation (e.g., allergy check, screenings, follow-ups).
* **User**: Represents a system user, including roles and authentication information.
* **AuditLog**: Tracks changes and actions performed by users in the system.

#### **Backend Technologies (Tools & Frameworks)**

* **.NET Core 8**: Framework for building the backend API (Web API) to handle requests from the frontend.
* **Entity Framework Core 8**: ORM (Object-Relational Mapper) for database interactions, enabling CRUD operations and database management.
* **SQL Server**: Relational database used for storing data such as patient details, recommendations, users, and logs.
* **Swashbuckle (Swagger)**: Used for auto-generating OpenAPI documentation and providing an interactive UI to test API endpoints.
* **JWT (JSON Web Token)**: For handling user authentication and authorization with role-based access control (RBAC).
* **ASP.NET Core MVC/Web API**: Used to build the web API with controllers to manage patient and recommendation data.

#### **Frontend Technologies (Tools & Frameworks)**

* **Angular 18**: Framework for building the frontend, managing UI components and state.
* **Angular Material**: A set of UI components to create a responsive and polished UI.
* **RxJS**: For reactive programming in Angular, handling asynchronous operations like HTTP requests.
* **TypeScript**: Superset of JavaScript used to build the frontend logic with type safety.
* **Bootstrap**: Optional, used for quick UI styling and layout if needed.

#### **Authentication & Security Tools**

* **JWT (JSON Web Token)**: Used to secure API requests and manage user authentication.
* **OAuth 2.0 / OpenID Connect**: For external authentication using services like Google, GitHub, or other OAuth providers (optional, if needed).
* **CSP (Content Security Policy)**: A browser security feature to help prevent XSS attacks.
* **Anti-CSRF Tokens**: To protect against Cross-Site Request Forgery attacks.
* **Bcrypt**: For hashing and securing user passwords.
* **HTTPS**: Used for securing the communication between the client and server.

#### **Database Tools**

* **SQL Server Management Studio (SSMS)**: Tool used to manage the SQL Server instance and database.
* **Entity Framework Core 8**: ORM (Object-Relational Mapper) for handling database migrations, schema generation, and queries.

#### **Development & CI/CD Tools**

* **Visual Studio 2022**: IDE for backend development, debugging, and running the .NET Core application.
* **VS Code**: IDE used for frontend development with Angular.
* **Docker**: Containerization tool for deploying the application and its dependencies, ensuring consistency across different environments.
* **Git**: Version control system for managing code changes.
* **GitHub**: For version control, collaboration, and continuous integration workflows.

#### **Testing Tools**

* **xUnit**: Framework used for unit and integration testing of the .NET Core application.
* **Karma**: A test runner for running frontend unit tests in the Angular application.
* **Jasmine**: Testing framework for Angular to write unit and integration tests.

#### **API Documentation and Testing Tools**

* **Swagger/OpenAPI**: Automatically generates API documentation, with the ability to test the API directly from the browser (via Swagger UI).
* **Postman**: Optional tool for API testing and interacting with endpoints during development.

#### **Other Tools**

* **Webpack**: Used to bundle and optimize frontend assets for production.
* **Nginx**: If used, serves as a reverse proxy for hosting the frontend Angular application.