User Management System Documentation

---

1. Introduction

The User Management System is designed to handle user-related operations such as registration, login, and updating user profiles. The system is built using the Microsoft .NET stack on the backend and Angular on the frontend, adhering to established software design principles and architectural patterns to ensure scalability, maintainability, and security.

---

2. System Architecture

The architecture of the User Management System follows a layered approach, separating concerns into distinct layers: the Presentation Layer, Business Logic Layer, Data Access Layer, and Database. Each layer has a specific responsibility, promoting code modularity and reusability.

- **Presentation Layer (Frontend)**
    - Built using Angular, this layer handles user interaction. It includes components for login, registration, and updating user information.
- **Business Logic Layer (Service Layer)**
    - This layer contains the core business logic of the application. It processes data received from the Presentation Layer and communicates with the Data Access Layer.
- **Data Access Layer (Repository Layer)**
    - The repository pattern is employed here to abstract data access operations. This layer interacts with the database using Entity Framework Core, providing an interface to perform CRUD operations.
- **Database**
    - The system uses SQL Server to persist user data, including authentication credentials and user profiles.

---

3. Design Principles

The design of the User Management System is guided by several key principles to ensure that the system is robust and maintainable:

- **Single Responsibility Principle (SRP)**
    - Each class and module has a single responsibility. For example, the service layer handles business logic, while the repository layer manages data access. This separation allows for easier testing and maintenance.
- **Dependency Injection**

    - The system utilizes dependency injection to manage dependencies between classes, improving modularity and making it easier to swap out implementations, such as replacing the data source or changing how services are implemented.

- **Repository Pattern**
    - The repository pattern is used to abstract the data access logic from the business logic. This provides a clean API for the service layer to interact with the database and makes the data access layer easily replaceable.
- **RESTful API Design**
    - The backend services are designed following RESTful principles, providing a clear and stateless interface for the frontend to interact with.
- **DRY (Don't Repeat Yourself)**
    - The DRY principle is applied to reduce code duplication. Common functionalities are abstracted into reusable components and services.

---

4. Frontend Implementation

The frontend is implemented using Angular and includes the following components:

- **Login Component**
    - Allows users to authenticate by entering their credentials. It interacts with the backend via a service that sends a POST request to the login API.
- **Register Component**
    - Handles user registration, collecting necessary information such as username, email, and password. It sends a POST request to the registration API on the backend.
- **Update User Component**
    - Enables authenticated users to update their profile information. This component fetches the current user data from the backend and allows for modifications, which are then sent back to the backend for processing.

---

5. Backend Implementation

The backend is developed using .NET Core, structured around the following layers:

- **Controllers**
    - Each controller handles HTTP requests for a specific resource. For example, the UserController handles requests related to user operations like login, registration, and updating user profiles.
- **Services**
    - The service layer contains business logic. For instance, the UserService processes the registration logic, handles login verification, and updates user information.

- **Repositories**
    - The repository layer interacts with the database. For example, the UserRepository provides methods to query user data, save new users, and update existing users.
- **Database**
    - The system uses SQL Server, and Entity Framework Core handles ORM tasks. Migrations are used to manage database schema changes.

---

6. Security Considerations

The system implements various security measures to protect user data:

- **Password Hashing**
    - User passwords are hashed before being stored in the database to ensure they are not stored in plain text.
- **JWT Authentication**
    - JSON Web Tokens (JWT) are used for user authentication. Upon successful login, a JWT is generated and sent to the client, which is then used for authenticating subsequent requests.
- **Input Validation**
    - Both frontend and backend layers validate user inputs to prevent SQL injection, XSS attacks, and other common vulnerabilities.

---

7. Conclusion

The User Management System is designed with a focus on modularity, scalability, and security. By following best practices and adhering to solid design principles, the system is maintainable and ready for future enhancements. The separation of concerns between layers ensures that each component can be developed, tested, and maintained independently, contributing to the overall robustness of the application.

This document serves as an overview of the architecture and design principles used in the User Management System. It provides insights into the system's structure and the rationale behind the design choices made during development.