Online Legal Advisory System with ASP.NET Core

Objective:

Develop a robust, scalable, and maintainable web platform that connects clients seeking legal assistance with qualified lawyers, using ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity (for authentication), Stripe (for payment integration), and Angular.

Description:

The **Online Legal Advisory System** is a platform designed to facilitate legal consultations and services by allowing clients to browse lawyer profiles, purchase predefined legal services (gigs), and securely communicate and transact. The platform also features a comprehensive admin panel for managing lawyers, clients, and transactions. The system will use the listed technologies to deliver a streamlined, maintainable, and scalable solution.

Technologies to Use:

Architectural and Design patterns:

- **N-Tier Architecture:** Separation of the application into Presentation (UI), Business Logic (Service Layer), and Data Access (Repository/Entity Framework) layers.
- **Repository Pattern:** A layer that abstracts database interactions, providing a clean API for querying and persisting data.
- **Unit of Work Pattern:** Ensures that multiple related changes are grouped together in a single transaction, reducing the risk of data inconsistencies.
- **Dependency Injection:** Inject services (repositories, business logic) into controllers to promote testability and loose coupling.
- **ASP.NET Core Identity:** Provides robust authentication and authorization, ensuring secure user management.
- **Automated Database Migrations:** Facilitates easy setup and updates with integrated seed database migrations.

Front-End:

- **HTML** : Structures the web pages for content presentation.
- **CSS** : Styles and designs the user interface.
- **Bootstrap** : A CSS framework that helps build responsive and fast-loading interfaces.
- **Angular :** for a responsive, interactive front-

end. Back-End:

- **C#:** The programming language used for developing the application logic
- **ASP.NET Core MVC:** The framework used for building the web application.
- **Entity Framework Core:** Facilitates object-relational mapping (ORM) for interaction with the database.
- **LINQ:** Used for querying data from collections and databases in a concise manner.
- **Stripe** for payment

integration. Database:

- **SQL Server:** The database used to store application data

---

Weekly Development Plan

Week 1: Initial Setup and Lawyer Listings

1. **N-Tier Architecture:**
   - **Presentation Layer (UI):**
     - Set up the **ASP.NET Core MVC** project with views and controllers to handle user interactions.
     - Use **HTML, CSS, Bootstrap**, and **Angular** for the front-end, ensuring a responsive, dynamic, and user-friendly UI.
   - **Business Logic Layer (Service Layer):**
     - Implement **services** that handle the core business logic, such as managing lawyer profiles, handling transactions, and user authentication.
   - **Data Access Layer (Repository):**
     - Set up **Entity Framework Core** to interact with the database using the **Repository Pattern**.
     - Create repositories for managing entities
     - The repositories will abstract database operations (CRUD) and provide a clean API for accessing and persisting data.

2. **User Authentication Setup (ASP.NET Identity):**

   - Set up **ASP.NET Identity** for handling user registration, login, and role-based access.
   - Implement **role-based authorization** with different user roles (e.g., Lawyer, Client, Admin).
   - Use **Dependency Injection** (DI) to inject the **Authentication Service** into controllers and ensure loose coupling.

3. **Repository Layer and Database Setup:**

   - Set up the **Repository Pattern** to manage data access through services, making database operations easier to test and maintain.
   - Use **Entity Framework Core** to configure the database schema and ensure data consistency.
   - Implement the **Unit of Work Pattern** to manage transactions. This ensures that any changes to related entities (e.g., creating a new lawyer and assigning them gigs) are grouped in a single transaction to avoid data inconsistencies.
   - Run **database migrations** to set up the initial database schema and models.

4. **Lawyer Listings and Filters:**

   - Implement a **lawyer profile listing page** where clients can browse lawyer profiles.
   - Include filters (e.g., expertise, location) to allow users to narrow down search results.
   - Implement sorting and pagination for easy navigation.

5. **Dependency Injection (DI):**

   - Use **Dependency Injection** to inject services such as the **LawyerService**, **TransactionService**, and **AuthenticationService** into controllers. This ensures testability, flexibility, and separation of concerns.
   - Make sure that services, repositories, and business logic components are loosely coupled, making the code more modular and easier to maintain.

Deliverables for week 1:

- Set up the **ASP.NET Core MVC** project following **N-Tier Architecture**.
- Develop the **lawyer profile listing page** with basic filters and pagination.
- Implement **user authentication** with **ASP.NET Identity**, including role-based access control.
- Integrate **Entity Framework Core** with **Repository Pattern** for data access and **Unit of Work Pattern** for transaction management.
- Set up **Dependency Injection** for injecting services into controllers.
- Run **database migrations** to initialize the schema
- User authentication integrated using ASP.NET Identity.

Week 2: Gig Creation, Role-Based Access Control, and Admin Panel

1. **Gig Management:**
    - Allow lawyers to create **gig categories**, such as legal advice, contract review, legal documents, etc.
    - Enable lawyers to specify **pricing**, **service descriptions**, and **duration** for each gig.
    - Implement **CRUD operations** for gigs (create, read, update, delete).
    - Add **gig approval** functionality so admins can review and approve gigs before they go live.
2. **Role-Based Access Control (RBAC):**
    - Use **ASP.NET Core Identity** to define roles like **Client**, **Lawyer**, and **Admin**.
    - Implement **custom role-based authorization** to control access to different parts of the application. For example:
        - **Lawyers** can create/manage gigs, view their own transactions, and respond to client reviews.
        - **Clients** can browse available gigs, book services, and view their transaction history.
        - **Admins** have full control over users, gigs, and platform data.
3. **Admin Dashboard:**
    - Create an **admin dashboard** that includes:
        - A list of **lawyer registrations** for approval.
        - **Gig management** interface to approve, reject, or edit lawyer-created gigs.
        - **Reviews moderation** to allow admins to filter inappropriate comments.
    - Add **search and filter options** for managing data more efficiently.

Deliverables for week 2:

- Functional **gig creation and management** features for lawyers.
- **Admin panel** with features for managing users, gigs, and reviews.
- Fully implemented **role-based access control (RBAC)** with tested permissions.

Week 3: Payments, Transactions, and Client Profiles

1. **Stripe Payment Integration:**
    - Integrate **Stripe API** to securely handle payments for purchased gigs.
    - Implement **one-time payment** or **subscription models** based on gig types.
    - Store transaction details (e.g., client, lawyer, gig, amount, timestamp) in the database.
    - **Receipt generation**: Send a receipt via email or show it on the UI after a successful payment.
2. **Client Profile Management:**
    - Allow **clients to create and update profiles** with personal details and contact information.
    - Enable clients to **view their transaction history**, including past payments for gigs and status.
    - Add a **dashboard for clients** to track their active gigs, payments, and communication with lawyers.
    - Consider adding a **review system** where clients can rate the lawyers' services, contributing to the overall platform reputation.

Deliverables for week 3:

- **Stripe payment gateway** integrated for gig purchases.
- Fully functional **client profile** management with transaction history.
- Tested **payment flow** and client-dashboard features.

Week 4: Final Testing, UI Enhancements, and Deployment

1. **UI Refinement:**
    - Improve **UI/UX** with **Angular** to ensure a seamless experience across different devices.
    - Add **responsive design enhancements** using **CSS media queries** and **Angular components** to support mobile, tablet, and desktop views.
    - Polish UI elements such as **buttons**, **navigation**, and **forms** to improve accessibility and visual appeal.
2. **Comprehensive Testing:**
    - Test all workflows, including lawyer registration, gig management, and payments.
3. **Deployment:**
    - Deploy the application on a platform like **Azure.**
    - **Prepare project documentation that includes:**
        - **System architecture** explanation.
        - **User manual** for clients and lawyers.
        - **Admin manual** for managing the platform.

Deliverables for week 4:

- **Responsive UI** with enhancements for accessibility and aesthetics.
- **Tested workflows**, including lawyer registration, gig management, and payments.
- Finalized **project documentation** for user and admin instructions.

Final Deliverables

- Fully functional Online Legal Advisory System.
- Lawyer gig-based service model with secure payments.
- Admin panel for platform management.
- User documentation and architecture explanation.

Team

- **Kerolos Amiel Wassef**
- **Peter Micheal Ghaly**
- **Mina Gerges Sand**
- **Mohamed Mahmoud El Said**
- **Zena Mohamed Saeed**
- **Sama Mohamed abdelaziz**