Project Overview

The solution consists of two main projects:

1. **PinewoodTechnologies.API**: A backend Web API project that provides data access and operations using ASP.NET Core Web API, Entity Framework Core, and SQLite database.
2. **PinewoodTechnologies.UI**: A frontend Razor Pages (this was chosen because it was mentioned on the job description) project that serves as the user interface, allowing users to interact with the application through web pages.

This layered architecture separates concerns, improves maintainability, and allows for scalability. Below is a detailed breakdown of each project to help developers understand the codebase.

Project Set Up

Tools used for development:-

- Visual Studio 2022
- .Net core v8.0
- ASP.Net Web API
- ASP.Net Razor Pages
- GitHub
- Docker Desktop (for containerisation)

Note: I've tested the application against the "**https**" launch profile. Separately, I have created a "**docker**" launch profile which require some tweaks to fully get working redirecting the API endpoint to the API docker image.

PinewoodTechnologies.API

Project Purpose

The API project serves as the backend of the application, handling data storage, retrieval, and business logic. It exposes RESTful endpoints that the UI project consumes to perform CRUD (Create, Read, Update, Delete) operations on customers.

- **LaunchURL**: https://localhost:7296

Project Structure

- **Controllers**
    - CustomersController.cs: Handles HTTP requests related to customer operations.

      Exposes RESTful endpoints for customer operations.

        - **GET** `/api/customers`: Retrieves all customers.
        - **GET** `/api/customers/{id}`: Retrieves a specific customer.
        - **POST** `/api/customers`: Creates a new customer.
        - **PUT** `/api/customers/{id}`: Updates an existing customer.
        - **DELETE** `/api/customers/{id}`: Deletes a customer.
- **Models**
    - Customer.cs: Defines the data structure of a customer.

      A customer has the following properties

        - Firstname
        - Lastname
        - DoB
        - AddressLine1
        - AddressLine2
        - AddressLine3
        - Postcode
        - TelephoneNumber
- **Data**
    - CustomerContext.cs: Manages database interactions using Entity Framework Core.
    - CustomerContextFactory.cs: Provides a design-time context for migrations.

- **Middleware**
    - ApiKeyMiddleware.cs: Validates API key for incoming requests. For demonstration, the REST API requires an APIKey for access, defined in the appsettings.json, however in production we would use something more robust and secure.
- **Program.cs**: Configures services and middleware.
- **appsettings.json**: Stores configuration settings like API key and connection string (SQLite).
- **Customer**.**db**: SQLite database with one table called Customers. We would an external SQL Server in production.

PinewoodTechnologies.UI

Project Purpose

The UI project is a Razor Pages application that provides the frontend user interface. It allows users to log in, and perform CRUD operations on customers by interacting with the API.

- **LaunchURL**: https://localhost:7253

Project Structure

- **Pages**
  - **Base**
    - BasePageModel.cshtml and BasePageModel.cshtml.cs: A parent page model that all page must inherit for common functionality. The page checks if the current user is logged on and redirects the user to a login page if a session is not found. In production, you would use something more robust.
  - **Account**
    - Login.cshtml and Login.cshtml.cs: Handles user authentication.
    - Logout.cshtml and Logout.cshtml.cs: Handles user logout.
  - **Customers**
    - Index.cshtml and Index.cshtml.cs: Lists all customers.
    - Create.cshtml and Create.cshtml.cs: Adds a new customer.
    - Edit.cshtml and Edit.cshtml.cs: Edits an existing customer.
    - Delete.cshtml and Delete.cshtml.cs: Deletes a customer.
- **Models**
  - References the model from **PinewoodTechnologies.API**.
- **Shared**
  - _Layout.cshtml: The main layout for the application.
  - _ValidationScriptsPartial.cshtml: Includes client-side validation scripts.
- **Program.cs**: Configures services and middleware.

- **appsettings.json**: Stores configuration settings like API URL, API key, and login credentials. **Note**: The API URL must be set to the **PinewoodTechnologies.API** endpoint**.**

Interaction Between UI and API

API Consumption

- The UI project consumes the API by making HTTP requests using HttpClient.
- The API key is included in the request headers for authentication.

Security Measures

API Key Authentication

- The API requires an ApiKey header in requests.
- The API key is stored in appsettings.json for demo purpose. **<u>NOTE</u>**: It should be secured in production environments.

Session Authentication in UI

- User credentials are stored in appsettings.json for demonstration purposes.
- The session variable "IsAuthenticated" is set upon successful login.

Data Validation

- The Customer model in the UI includes data annotations for validation.
- Client-side validation scripts are included to provide immediate feedback.

Database and Data Layer

Entity Framework Core with SQLite

- The API uses Entity Framework Core to interact with a SQLite database.
- The CustomerContext manages database operations.
- Migrations are used to create and update the database schema

Ideas for future improvements

The technical task was implemented quickly in the given timescale but further enhancements might include: -

- **Enhance Security**: User Identification and Role based Access Control.
- **Improve User Experience**. Make use of UI components to be re-used on page. Use commercial UI component instead of bootstrap html/css.
- **Expand Functionality**: Advanced Customer Feature such as Search and Filtering, pagination, and export data.
- **Testing**: Unit testing and Integration testing.
- **Performance Optimisation**: Caching.
- **Logging**: Improvement logging and monitoring.
- **Containerisation**: Package your API and UI into Docker containers. I have dockerfile for each project but I have **not tested** against docker environment.
- **Internationalisation and Localisation**: Resource files and Culture specific date, number formatting.