### **Architektura MVC**

W projekcie występują następujące kontrolery:

* **BooksController.cs**Obsługa CRUD dla książek.  
  **Funkcje:** dodawanie, edytowanie, usuwanie, szczegóły i lista książek.
* **UsersController.cs**Obsługa CRUD dla użytkowników.  
  **Funkcje:** dodawanie, edytowanie, usuwanie użytkowników.
* **LoansController.cs**Obsługa CRUD dla wypożyczeń.  
  **Funkcje:** zapewnia dodawanie wypożyczeń oraz zarządzanie relacjami między książkami i użytkownikami.
* **HomeController.cs  
  Funkcje:** zarządzanie stroną główną aplikacji oraz obsługą błędów.  
  Implementacja middleware’a pozwala na wyświetlenie informacji o ostatniej wizycie użytkownika.

### **Partial View**

* **\_ViewImports.cshtml** i **\_ViewStart.cshtml**Wspierają współdzielenie kodu dla widoków.
* **\_BookList.cshtml**Partial View, który renderuje dynamiczną tabelę z listą książek.  
  **Struktura tabeli:**Kolumny: Title, Author, AvailableCopies oraz przyciski akcji dla każdej książki (np. edytuj, usuń).

### **WebAPI w Projekcie**

W projekcie zaimplementowano kontrolery WebAPI, które umożliwiają operacje CRUD (Create, Read, Update, Delete) na encjach **Books**, **Users** oraz **Loans**.  
Każdy kontroler API znajduje się w folderze Controllers/Api i wykorzystuje mechanizm routingu api/[controller] w ASP.NET Core.

* **BooksApiController.cs  
  Cel:** Zarządzanie danymi książek w formacie REST API.  
  **Bezpieczeństwo:** Wszystkie metody są chronione mechanizmem autoryzacji [Authorize].  
  **Przykładowe metody:**
  + **GET: api/BooksApi**Pobiera listę wszystkich książek z bazy danych i zwraca je w formacie JSON.
  + **GET: api/BooksApi/{id}**Pobiera książkę o określonym identyfikatorze. Jeśli książka nie istnieje, zwraca status **404 Not Found**.
  + **POST: api/BooksApi**Dodaje nową książkę do bazy danych. Weryfikuje poprawność danych przesłanych w treści żądania za pomocą **ModelState**.
  + **PUT: api/BooksApi/{id}**Aktualizuje dane książki na podstawie jej identyfikatora.
  + **DELETE: api/BooksApi/{id}**Usuwa książkę z bazy danych.
* **Kontrolery LoansApiController.cs oraz UsersApiController.cs** działają na tej samej zasadzie dla danych wypożyczeń oraz użytkowników.

### **Middleware**

Zaimplementowane customowe middleware w pliku **Middleware.cs**:

* Middleware zapisuje informacje o ostatniej wizycie użytkownika i dodaje je do nagłówka aplikacji.
* Informacje o ostatniej wizycie wyświetlane są na stronie głównej aplikacji.

### **Uwierzytelnianie Identity**

W projekcie wykorzystano **ASP.NET Identity** do obsługi logowania, rejestracji oraz zarządzania kontami użytkowników.  
Po otwarciu aplikacji można:

* Zarejestrować nowe konto.
* Zalogować się przy użyciu predefiniowanego konta:  
  **Email:** user@gmail.com  
  **Hasło:** Password1

### **Repozytorium danych**

W projekcie wykorzystano **Entity Framework Core** do komunikacji z bazą danych **SQLite**.

* **Plik ApplicationDbContext.cs**Definiuje schemat bazy danych.
* **Plik library.db**Zawiera wypełnioną bazę danych z przykładowymi danymi.

### **Testy jednostkowe**

* **Folder Library.Tests**Zawiera testy jednostkowe dla kontrolerów:
  + **BooksController:** testuje operacje CRUD dla książek.
  + **LoansController:** testuje operacje CRUD dla wypożyczeń.
  + **HomeController:** testuje poprawność strony głównej.
  + **DummyTests:** pomocnicze testy w fazie wczesnego rozwoju projektu.
* Testy zostały napisane z użyciem **NUnit**.

### **Wymagane Paczki**

Aby uruchomić projekt, należy zainstalować następujące paczki NuGet:

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Sqlite
* Microsoft.EntityFrameworkCore.Tools
* Microsoft.AspNetCore.Identity.EntityFrameworkCore
* Microsoft.AspNetCore.Identity.UI
* Microsoft.AspNetCore.Authentication
* NUnit
* NUnit3TestAdapter
* Microsoft.NET.Test.Sdk

### **Trudności i uwagi**

Jedną z największych trudności podczas realizacji projektu było zaimplementowanie bazy danych, którą dodałem w późniejszej fazie, co znacząco utrudniło cały proces implementacji. Mimo to, udało się ją skutecznie zintegrować. Problemy wystąpiły również przy edytowaniu stron logowania i rejestracji, które były jedynie szkieletowe. Z nie do końca jasnych powodów generowały one błędy, co dodatkowo komplikowało ich poprawną implementację.  
  
  
Jakub Furmański  
Informatyka Stosowana 13644