Dokumentacja techniczna

Spis treści

[Opis projektu 1](#_Toc187441725)

[Wymagania techniczne 1](#_Toc187441726)

[Architektura i użyte technologie 1](#_Toc187441727)

[Instrukcja uruchomienia 2](#_Toc187441728)

[Struktura katalogów 2](#_Toc187441729)

[Opis głównych elementów (kontrolery, modele, widoki) 4](#_Toc187441730)

# Opis projektu

**Cel**: Prosta aplikacja webowa z obsługą rejestracji i logowania użytkowników, prywatnymi podstronami (dla zalogowanych), bazą danych (SQLite) do przechowywania danych oraz możliwością zapisywania danych związanych z konkretnym użytkownikiem.

W ramach projektu przygotowano:

1. **Moduł kont użytkowników** (Account) – rejestracja, logowanie, wylogowanie, w oparciu o ASP.NET Core Identity.
2. **Moduł zarządzania zadaniami** (Tasks) – tworzenie i edycja zadań przypisanych do użytkownika zalogowanego.
3. **Formularz kontaktowy** (Contact) – przykład publicznego formularza, którego dane można zapisywać w bazie.
4. **Stronę główną** (Home), z prostą nawigacją i widokami.

# Wymagania techniczne

**.NET SDK** w wersji 8.0 (lub wyższej, jeśli finalna).

S**QLite** – zainstalowany pakiet Microsoft.EntityFrameworkCore.Sqlite.

**Dowolny edytor**/IDE obsługujący .NET (VS Code, Visual Studio, Rider).

**Pakiet** Microsoft.AspNetCore.Identity.EntityFrameworkCore (do obsługi tożsamości) i pokrewne pakiety EF Core (Microsoft.EntityFrameworkCore.Tools, etc.).

# Architektura i użyte technologie

* **ASP.NET Core MVC** – Wzorzec Model-View-Controller.
* **Identity** – Mechanizm autoryzacji i autentykacji (logowanie, rejestracja, role).
* **Entity Framework Core** – Warstwa dostępu do danych (ORM).
* **SQLite** – Baza danych (pliki .db).
* **Razor** – Silnik widoków (pliki .cshtml).
* **Bootstrap** – Biblioteka CSS i JS do stylowania interfejsu (opcjonalnie).

Aplikacja podzielona jest logicznie na warstwy:

* **Models** – klasy reprezentujące dane i konteksty bazy.
* **Controllers** – klasy kontrolerów obsługujące logikę (Account, Tasks, Home, Contact).
* **Views** – folder z widokami Razor przypisanymi do poszczególnych kontrolerów.
* **wwwroot** – zasoby statyczne (CSS, JS, grafiki).

# Instrukcja uruchomienia

Upewnij się, że plik appsettings.json zawiera sekcję:

"ConnectionStrings": {

"DefaultConnection": "Data Source=ZaliczeniowyProjekt.db"

}

Wykonaj migracje bazy (tylko przy pierwszym uruchomieniu lub po zmianach w modelach):

dotnet ef migrations add InitialCreate

dotnet ef database update

Zbuduj projekt: dotnet build

Uruchom projekt: dotnet run

# Struktura katalogów

ZaliczeniowyProjekt

│ ZaliczeniowyProjekt.csproj

│ Program.cs

│ appsettings.json

│ appsettings.Development.json

│ ...

├───Controllers

│ HomeController.cs

│ AccountController.cs

│ TasksController.cs

│ ContactController.cs (opcjonalnie)

│

├───Models

│ ApplicationDbContext.cs

│ ApplicationUser.cs

│ TaskItem.cs

│ ContactMessage.cs (opcjonalnie)

│

├───ViewModels

│ RegisterViewModel.cs

│ LoginViewModel.cs

│ ...

│

├───Views

│ ├───Home

│ │ Index.cshtml

│ │ Privacy.cshtml

│ ├───Account

│ │ Login.cshtml

│ │ Register.cshtml

│ ├───Tasks

│ │ Index.cshtml

│ │ Create.cshtml

│ │ Edit.cshtml

│ │ Delete.cshtml

│ ├───Contact

│ │ Index.cshtml

│ │ ThankYou.cshtml

│ └───Shared

│ \_Layout.cshtml

│ Error.cshtml

│

└───wwwroot

├───css

├───js

└───lib

└── bootstrap

# Opis głównych elementów (kontrolery, modele, widoki)

**Program.cs**

Zawiera konfigurację usług i middleware:

* Dodanie DbContext (ApplicationDbContext) z SQLite,
* Dodanie Identity (z ApplicationUser),
* Konfiguracja ścieżek cookie (opcjonalnie options.LoginPath),
* Middlewares: UseAuthentication(), UseAuthorization(),
* Domyślny routing: MapControllerRoute(...).

**ApplicationDbContext.cs**

Dziedziczy po IdentityDbContext<ApplicationUser> i zawiera DbSet<TaskItem> (oraz ewentualne inne tabele, np. ContactMessage). Dzięki temu EF Core wie, jak mapować klasy (modele) na tabele w bazie SQLite.

**ApplicationUser.cs**

Reprezentuje encję użytkownika w Identity (dziedziczy po IdentityUser). Można tu dodać dodatkowe pola, np. FirstName, LastName.

**AccountController.cs**

Obsługuje proces:

* **Rejestracja** (Register) – tworzy konto użytkownika, loguje go od razu, ustawia komunikat w TempData, przekierowuje do Home/Index.
* **Logowanie** (Login) – próbuje \_signInManager.PasswordSignInAsync(...); po sukcesie ustawia TempData["SuccessMessage"].
* **Wylogowanie** (Logout) – wylogowuje \_signInManager.SignOutAsync().

**TasksController.cs**

Podstawowy CRUD z [Authorize] (tylko dla zalogowanych). Metody:

* **Index()** – pokazuje listę zadań przypisanych do danego użytkownika (UserManager.GetUserId(User)).
* **Create()** – tworzenie nowego zadania.
* **Edit()** – edycja zadania.
* **Delete()** – usunięcie zadania.

W widokach używane są klasy Bootstrapa (form-control, btn, table, card), by ułatwić stylowanie.

**Widoki**

* **\_Layout.cshtml** – główny layout, zawiera pasek nawigacji (navbar), sprawdzający, czy użytkownik jest zalogowany. W razie zalogowania – przycisk „Wyloguj” i link do „Moje zadania”; w razie niezalogowania – linki „Zaloguj” i „Rejestracja”.
* **Index.cshtml** (Home) – strona główna, może zawierać krótką prezentację projektu.
* **Login.cshtml, Register.cshtml** (Account) – formularze do logowania/rejestracji, stylowane przez Bootstrap.
* **Index.cshtml, Create.cshtml, Edit.cshtml, Delete.cshtml** (Tasks) – CRUD dla zadań.
* **Index.cshtml (Contact)** – formularz kontaktowy (jeśli zaimplementowany).
* **ThankYou.cshtml** (Contact) – strona z podziękowaniem za wysłanie wiadomości.