**Dokumentacja Projektu PostoPedia**

**1. Tytuł Projektu:**

**PostoPedia**

**2. Krótki opis działania projektu:**

PostoPedia to internetowa platforma blogowa stworzona z myślą o wymianie myśli i publikacji artykułów na różnorodne tematy. Użytkownicy mogą tworzyć posty, komentować je, wysyłać wiadomości prywatne oraz zarządzać swoim profilem. Administratorzy mają pełną kontrolę nad użytkownikami i ich twórczością, w tym możliwość ich edycji iusuwania . System logowania i rejestracji oparty jest na mechanizmach ASP.NET Identity, zapewniając bezpieczną autentyfikację oraz zarządzanie użytkownikami.

**3. Autorzy:**

Robert Synak, Radosław Wróbel

**4. Instrukcja uruchamiania projektu:**

* **Wykorzystane technologie:**
* **Język programowania:** C#
* **Framework:** .NET 8
* **Wykorzystana baza danych:** MS SQL SERVER
* **ORM:** Entity Framework Core
* **Frontend:** Razor Pages, HTML + CSS, JavaScript,
* **Wymagania:**
* **System operacyjny:** Windows 10/11 lub Linux z obsługą .NET 8
* **.NET SDK:** Wersja 8.0.
* **Serwer bazy danych:** MS SQL Server (preferowany) lub SQLite (dostosowanie projektu SQLite wymaga dodatkowej konfiguracji)
* **IDE:** Visual Studio 2022 (lub nowszy) z ASP .NET 8
* **Opcja 1: Nowa baza danych**

1. Otwórz kod źródłowy projektu w Visual Studio
2. Otwórz **Package Manager Console** w Visual Studio

* W menu wybierz **Tools > NuGet Package Manager > Package Manager Console**

1. Wykonaj polecenie, aby zastosować migracje i utworzyć nowa bazę danych

Upadate - Database

1. Uruchom aplikację za pomocą terminala

dotnet run

* Lub bezpośrednio w Visual Studio, klikając przycisk **Uruchom**

1. Domyślnie aplikacja zostanie uruchomiona pod jednym z poniższych adresów:

* https://localhost :7291

1. Otwórz wybrany adres w przeglądarce, aby rozpocząć korzystanie z aplikacji

* **Opcja 2: Gotowa baza danych**

1. Do projektu dołączona jest wstępnie wypełniona baza danych. Znajdziesz ją w plikach projektu
2. Aby podłączyć bazę danych:
   * + Otwórz **SQL Server Object Explorer** w Visual Studio
     + Wykonaj poniższe zapytanie:
3. Gdzie ‘’<SCIEŻKA\_DO\_PROJEKTU>’’ wskaż lokalizację pliku .mdf z kopią bazy danych znajdującego się w folderze projektu:
4. Przejdź do pliku appsettings.json i zmień connection string na ten z od komentowanym parametrem DefaultConnection:

* **Konto administratora**

Aplikacja automatycznie tworzy konto administratora oraz role użytkowników podczas pierwszego uruchomienia. Dane logowania do konta administratora to:

* + **Login:** admin@admin.com
  + **Hasło:** Admin123#

Kod odpowiedzialny za utworzenie konta administratora znajduje się w pliku Program.cs:

5. Opis struktury projektu:

Projekt opiera się na wzorcu architektonicznym **Model-View-Controller (MVC)**

**Controllers** – zawiera logikę obsługi żądań http i przetwarzania danych dla widoków

**Models** – odpowiada za dane aplikacji i definicje encji bazy danych

**Views** – Przechowuje pliki widoków generowanych przez Razor

**Data** – Zawiera plik ApplicationDbContext, który definiuje konfiguracje bazy danych i powiązania między modelami oraz kopię bazy danych,

**Migrations** – zawiera automatycznie wygenerowane pliki migracji dla bazy danych

**wwwroot** – Przechowuje pliki Css, JavaScript i grafiki

6. Modele wykorzystane w projekcie:

**AdminDashboardViewModel**  
Model służący do prezentacji informacji na dashboardzie administratora, takich jak liczba użytkowników i postów

Pola:

TotalUsers

TotalPosts

TotalComments

NewUsersToday

NewUsersThisWeek

PostsToday

PostsThisWeek

**ApplicationUser**  
Model użytkownika oparty na klasie IdentityUser, rozszerzony o dodatkowe właściwości, takie jak Nickname, RegistrationDate, LoginCount, IsBanned, IsDeleted, itp.

Pola wymagane:

Nickname – nazwa użytkownika, string

RegistrationDate – data rejestracji, DateTime

LastLoginDate – data ostatniego logowania, DateTime

LoginCount – Ilość logowań, int

AboutMe – opis użytkownika, maks. długość 2000 znaków, string

**Comment**  
Model reprezentujący komentarze użytkowników na postach. Zawiera informacje o treści komentarza, użytkowniku oraz powiązane komentarze (odpowiedzi).

Pola:

Content – komentarz, string, wymagane

CreatedAt -data utworzenia, DateTime. wymagane

UserID – ID użytkownika, string, wymagane

User – nazwa użytkownika, application user, wymagane

ParentCommentID – komentarz do którego odwołuje się post, ID

ICollection – odpowiedzi, virtual

Visibility – widoczność komentarza, bool, wymagane

**EditPostViewModel**  
Model wykorzystywany przy edycji postów, zawiera pola takie jak Title, Content oraz UserId.

Pola wymagane:

Title – tytuł, maks. długość 40 znaków, string

Content – zawartość, maks. długość 8000 znaków, string

UserID – ID użytkownika, string

**EditProfileViewModel**  
Model wykorzystywany do edycji profilu użytkownika, w tym zmiany danych osobowych i ustawień konta.

Pola wymagane:

Nickname – nazwa użytkownika, maks. długość 30 znaków, string

AboutMe – opis użytkownika, string

**EditUserViewModel**  
Model wykorzystywany przez administratora do edytowania danych użytkownika, takich jak Nickname, Email oraz przypisane role.

Pola wymagane:

Nickname – nazwa użytkownika, maks. długość 30 znaków, string

Email – adres e-mail użytkownika, walidacja zapisu: [x@x.x](mailto:x@x.x) ,string

**ErrorViewModel**  
Model reprezentujący dane o błędzie, takie jak kod błędu oraz jego opis, wykorzystywany w przypadku wystąpienia błędów w aplikacji.

Pola:

RequestID – ID wniosku, string

ShowRequest ID – pokazuje ID wnioski, walidacja: nie może być pusty, bool

**Post**  
Model reprezentujący posty użytkowników, zawiera informacje o tytule, treści, dacie utworzenia oraz powiązanych komentarzach. Dodatkowo zawiera flagi dotyczące widoczności i zatwierdzenia postu.

Pola wymagane:

Content – komentarz, string, wymagane

CreatedAt -data utworzenia, DateTime. wymagane

UserID – ID użytkownika, string, wymagane

Visibility – widoczność komentarza, bool, wymagane

Approved -

**PrivateMessage**  
Model reprezentujący wiadomości prywatne między użytkownikami. Zawiera właściwości związane z nadawcą, odbiorcą oraz treścią wiadomości.

**ProfileViewModel**  
Model wykorzystywany do prezentacji profilu użytkownika, zawiera dane takie jak Nickname, RegistrationDate, LastLoginDate, PostCount, itp.

**SendMessageViewModel**  
Model służący do wysyłania wiadomości prywatnych między użytkownikami. Zawiera dane takie jak odbiorca, tytuł i treść wiadomości.

Pola Wymagane:

RecpipientID – ID odbiorcy, string

Content – zawartośc wiadomości, maks. dlugosc 4000 znaków, string