

# Dokumentacja techniczna aplikacji Lighter









## 1. Instrukcja pierwszego uruchomienia

Wymagane oprogramowanie:

- .Net SDK – najlepiej wersja 8.0 .NET SDK
- Visual Studio 2022
- PgAdmin 4
- PostgreSQL - zalecana baza danych. W przypadku uruchomienia na innej bazie wymagane zmiana appsettings.json.

## 2. Kroki instalacji (konsola menadżera pakietów)

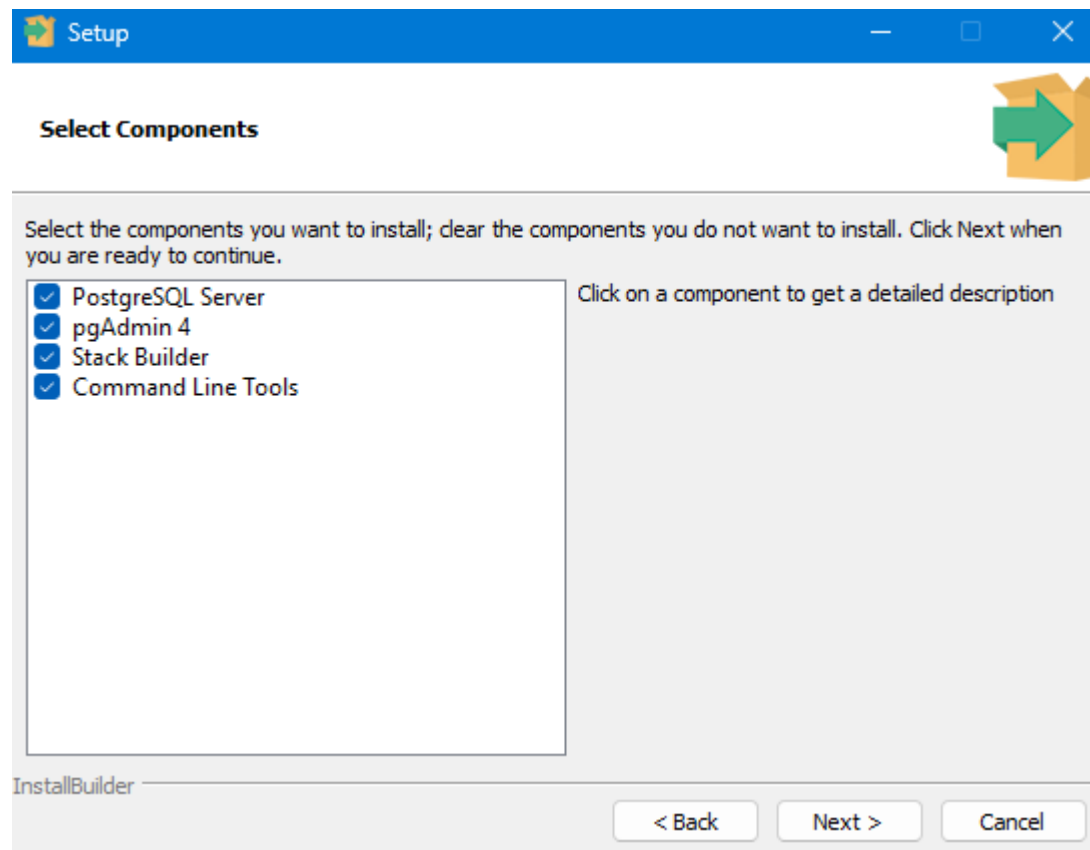
- Otwórz projekt w Visual Studio i sklonuj projekt z podanego linku.
- Otwórz pgAdmin 4
- Ustawienie hasła w bazie danych PostgreSQL/pgAdmin 4 na "123". W przypadku ustawienia innego hasła należy zmienić parametry 'Password' w appsettings.json.
- Otwórz Konsolę menadżera pakietów: Narzędzia - Menedżer pakietów NuGet - Konsola Menadżera Pakietów
- Przywróć wymagane pakiety NuGet: Restore-Package

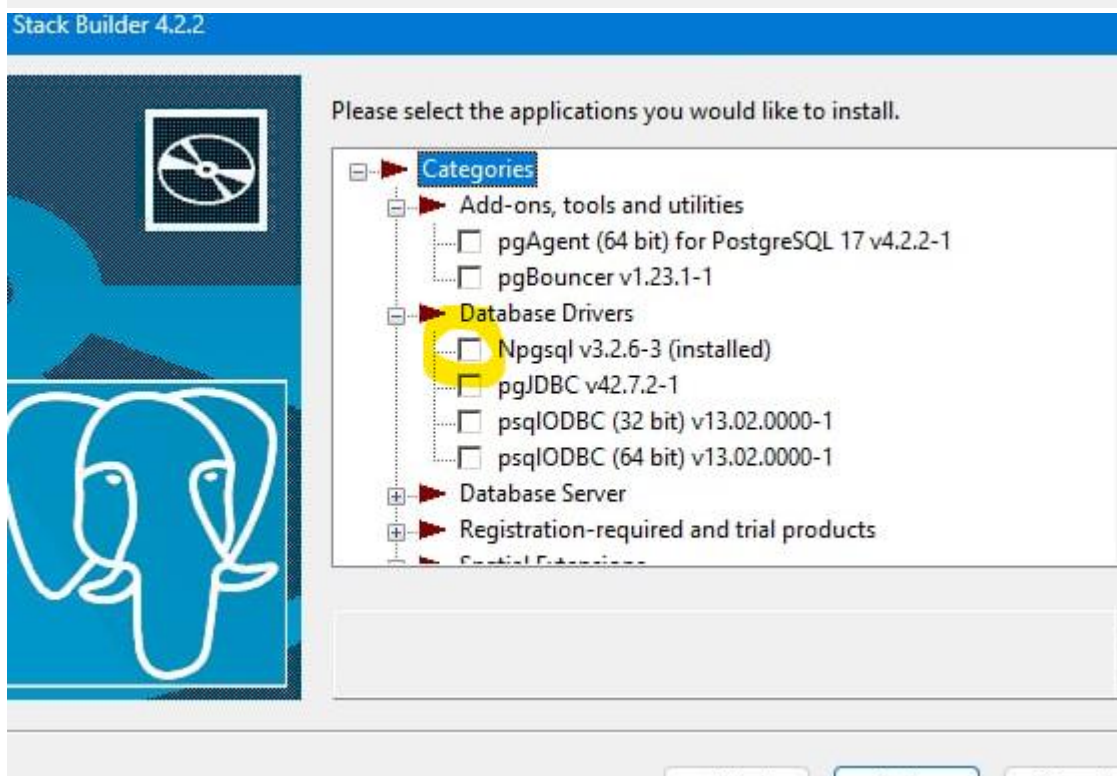
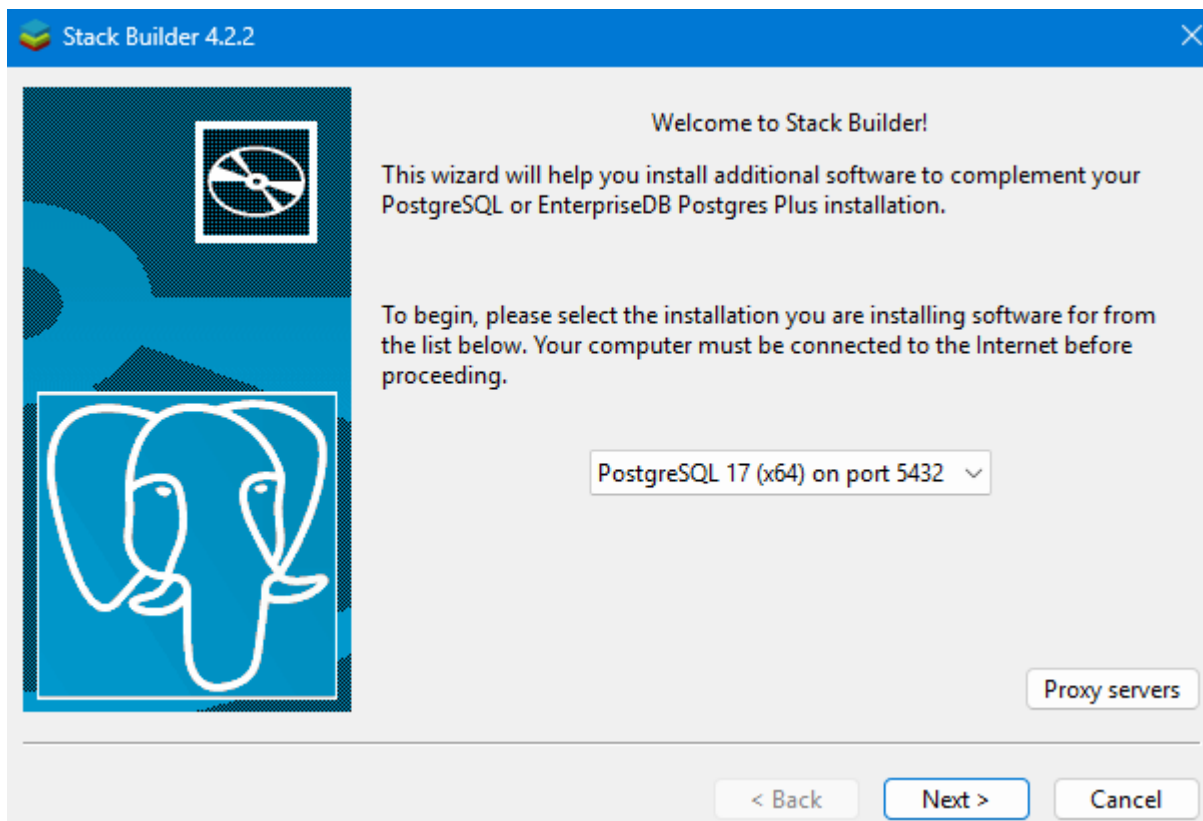
Pakiety najwyższego poziomu (8)			
	<b>Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore</b> przez: Microsoft	8.0.10	
	ASP.NET Core middleware for Entity Framework Core error pages. Use this middleware to detect and diagnose errors with Entity Framework Core migrations.	9.0.1	
	<b>Microsoft.AspNetCore.Identity.EntityFrameworkCore</b> przez: Microsoft	8.0.11	
	ASP.NET Core Identity provider that uses Entity Framework Core.	9.0.1	
	<b>Microsoft.AspNetCore.Identity.UI</b> przez: Microsoft	8.0.10	
	ASP.NET Core Identity UI is the default Razor Pages built-in UI for the ASP.NET Core Identity framework.	9.0.1	
	<b>Microsoft.EntityFrameworkCore</b> przez: Microsoft	8.0.11	
	Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	9.0.1	
	<b>Microsoft.EntityFrameworkCore.SqlServer</b> przez: Microsoft	8.0.10	
	Microsoft SQL Server database provider for Entity Framework Core.	9.0.1	
	<b>Microsoft.EntityFrameworkCore.Tools</b> przez: Microsoft	8.0.11	
	Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	9.0.1	
	<b>Microsoft.VisualStudio.Web.CodeGeneration.Design</b> przez: Microsoft	8.0.7	
	Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	9.0.0	
	<b>Npgsql.EntityFrameworkCore.PostgreSQL</b> przez: Shay Rojansky, Austin Drenski, Yoh Deadfall	8.0.11	
	PostgreSQL/Npgsql provider for Entity Framework Core.	9.0.3	

Powyższe pakiety muszą być zainstalowane i załadowane do projektu

- Dodaj migracje dla bazy danych: Add-Migration LighterCreate
- Wykonaj migracje Entity Framework, aby utworzyć bazę danych: Update-Database
- Sprawdź czy w pgAdmin pojawiła się nowa baza LighterDB (może wymagane być odświeżenie bazy)
- Uruchom aplikację

## Szczegóły instalacji PostgreSQL:





Password: 123

## 3.Opis backendu aplikacji

### Kontrolery:

#### 1. HomeController:

- Odpowiada za widoki aplikacji takie jak Najnowsze posty, Popularne, itd.
- Wyświetla widoki z folderu Views/Home

#### 2. PostsController:

- Kontroler PostsController zarządza postami. Obsługuje operacje związane z postami, komentarzami, polubieniami oraz różnymi widokami postów, jak popularne, najnowsze, czy użytkownika.

**Konstruktor `public PostsController(ApplicationDbContext context)`** - to instancja bazy umożliwiająca dostęp do tabel aplikacji i inicjalizuje kontroler z kontekstem LighterDB.

```
Odwolania: 0
public PostsController(ApplicationDbContext context)
{
    _context = context;
}
```

**Create (GET)** - wyświetla formularz tworzenia nowego posta i zwraca widok formularza.

```
Odwolania: 0
public IActionResult Create()
{
}
```

**Create (POST)** - który tworzy nowy post - posiada parametry post: które z modelu posta zawierają jego treść.

```
Odwolania: 0
public async Task<IActionResult> Create([Bind("Content")] Post post)
{
}
```

W walidacji sprawdzamy treść posta, która nie może być pusta a jego maksymalna długość to 380 znaków.

Autor posta jest generowany na podstawie e-maila użytkownika postującego

**Details** - wyświetla szczegóły posta w tym jego komentarze i daje możliwość polubienia. Pobiera identyfikator posta (id) i zwraca widok szczegółów posta.

**Index** – Przekierowuje użytkownika na widok latest wyświetlający najnowsze posty

```
1 odwołanie  
public async Task<IActionResult> Index()  
{
```

Popular, Latest i MyPosts wyświetlają analogicznie wybrany tryb sortowania i widok postów.

#### Metody w kontrolerze:

**AddComment** - Dodaje komentarz do wybranego posta z parametrami id posta i treścią komentarza dodanego przez użytkownika. Komentarz nie może być pusty a jego maksymalna długość to 150 znaków. Dodatkowo została zaimplementowana funkcjonalność odpowiadania na komentarze.

```
[HttpPost]  
Odwołania: 0  
public async Task<IActionResult> AddComment(int postId, string content)  
{
```

**ToggleLike** - Dodaje lub usuwa polubienie dla posta. Jeśli użytkownik zalogowany poprzednio polubił dany post to polubienie jest cofnięte. Zwraca JSON z aktualną liczbą polubień posta.

```
[HttpPost]  
//Lajki  
Odwołania: 0  
public async Task<IActionResult> ToggleLike(int postId)  
{
```

#### Routing:

Index - Przekierowuje na Latest.

Details - Wyświetla szczegóły posta.

Create - Formularz tworzenia posta.

Popular - Lista popularnych postów.

Latest - Lista najnowszych postów.

MyPosts - Posty aktualnie zalogowanego użytkownika.

Metody zwracają widaki i dane JSON w zależności od typu żądania

## 4.Opis modeli w aplikacji

### Model Post -

Reprezentuje posty w aplikacji.

```
Odwołania: 4
public class Comment
{
    Odwołania: 0
    public int Id { get; set; }

    [Required(ErrorMessage = "Treść komentarza jest wymagana.")]
    [StringLength(150, ErrorMessage = "Komentarz nie może przekraczać 150 znaków.")]
    Odwołania: 2
    public string Content { get; set; }

    [Required(ErrorMessage = "Autor komentarza jest wymagany.")]
    Odwołania: 3
    public string Author { get; set; }

    1 odwołanie
    public int PostId { get; set; }
    Odwołania: 0
    public Post Post { get; set; }

    1 odwołanie
    public DateTime CreatedAt { get; private set; } = DateTime.UtcNow;

    1 odwołanie
    public string ReplyTo { get; set; }
}
```

Model służy do obsługi komentarzy w aplikacji, przechowując dane takie jak treść, autor, data utworzenia, a także możliwość śledzenia odpowiedzi na inne komentarze.

### Model Like:

Reprezentuje reakcje na posty użytkowników (polubienia)

```
public class Like
{
    public int Id { get; set; }

    public int PostId { get; set; }
    Odwołania: 0
    public Post Post { get; set; }

    Odwołania: 2
    public string UserId { get; set; }
}
```

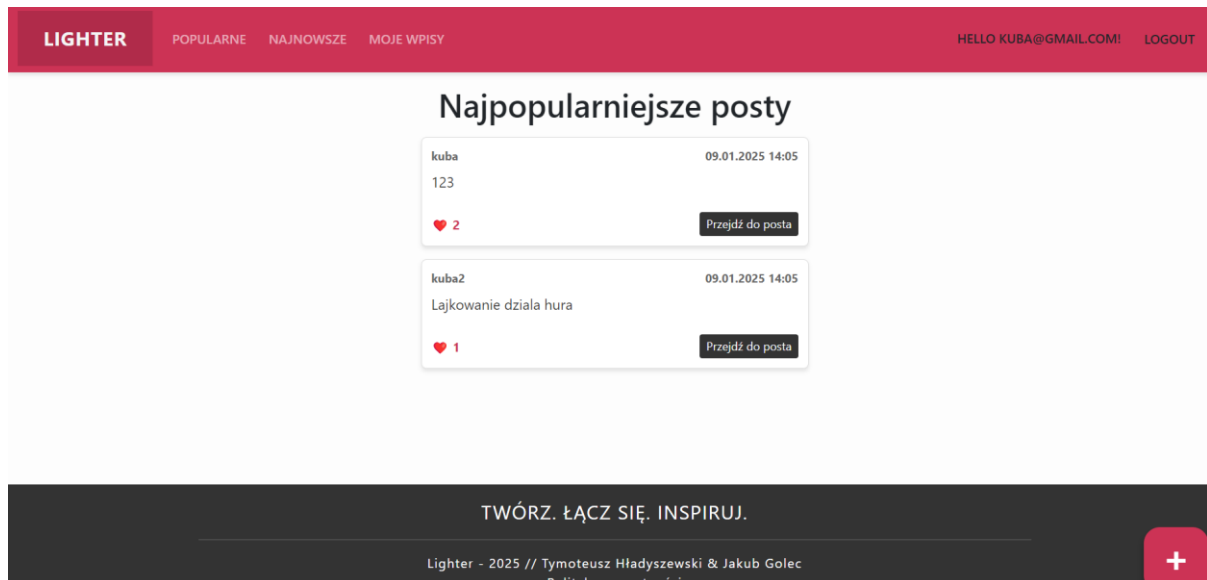
Model służy do przechowywania informacji o polubieniach postów. Przechowuje UserId użytkownika który polubił dany post (PostID). Zarządza liczy polubieniami postów w aplikacji.



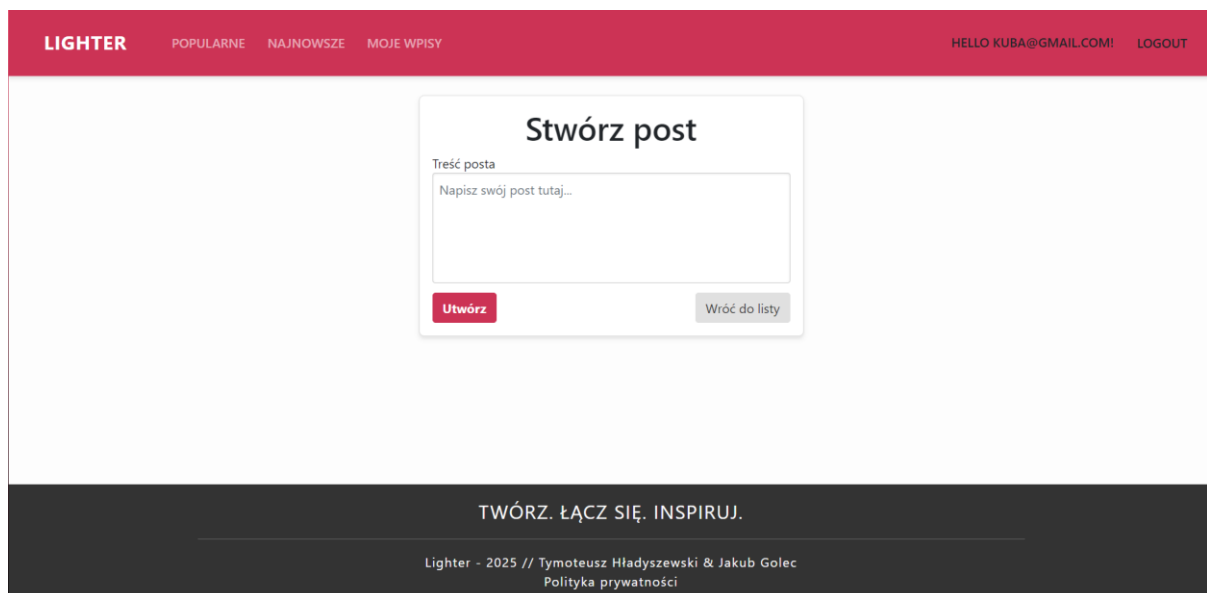
## 5. Widoki

Widoki znajdują się w folderze Views. Każdy widok jest powiązany z odpowiednim kontrolerem i wyświetla dane dla użytkownika.

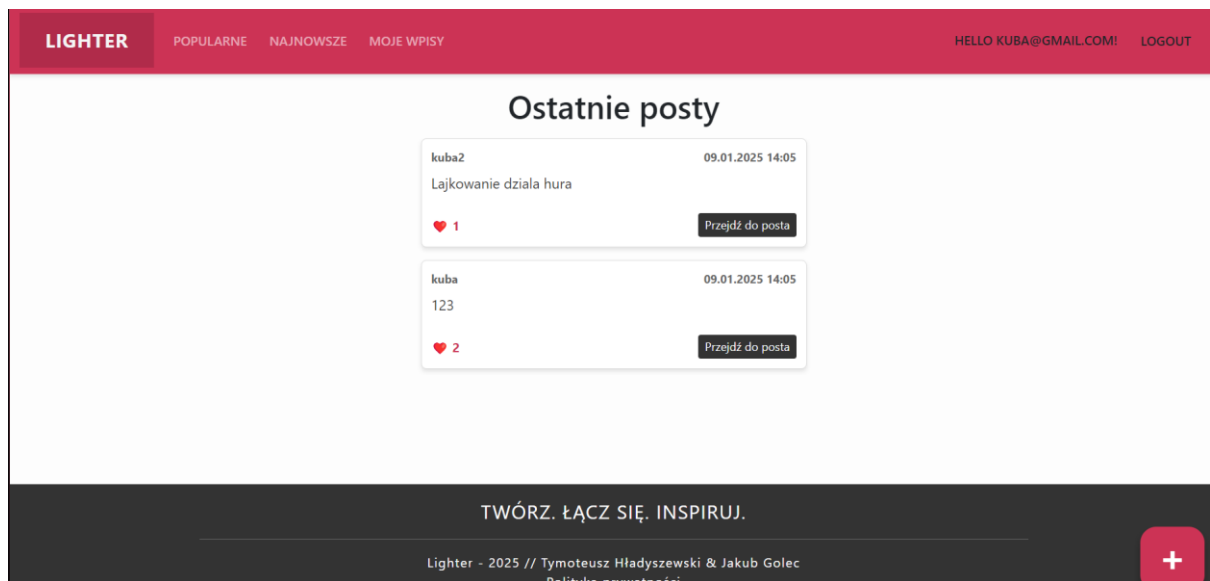
### Popular.cshtml



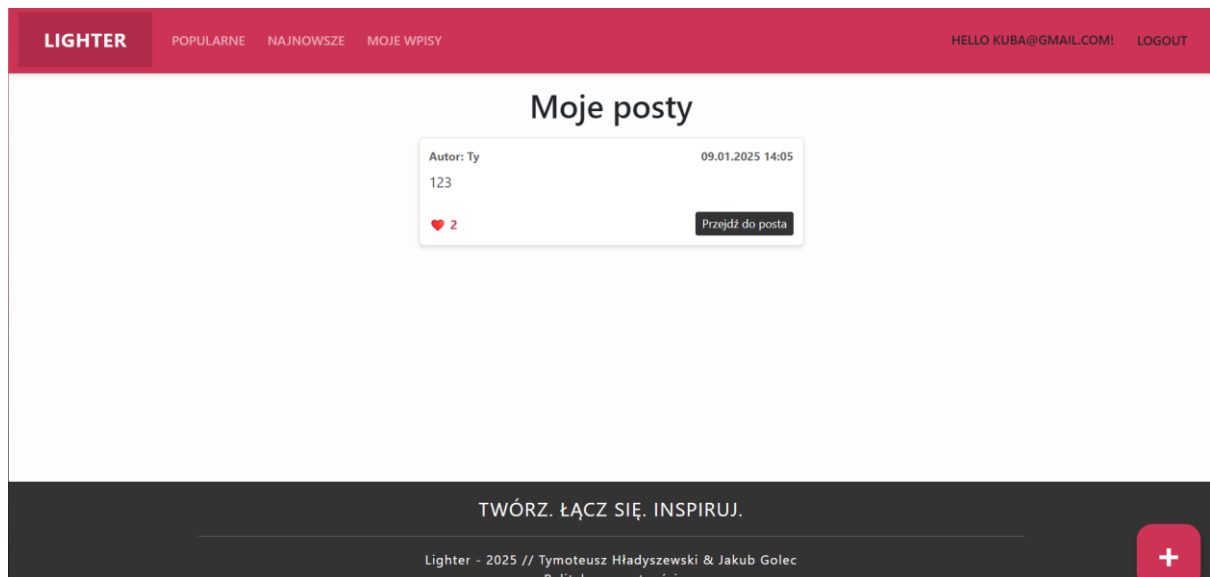
### Create.cshtml



### Latest.cshtml



## MyPosts.cshtml



## 6.Layout

Jest zorganizowany w klasyczny sposób dla MVC, główny układ jest w pliku `_Layout.cshtml` w folderze `Views/Shared`. Zawiera elementy interfejsu takie jak pasek nawigacji z poszczególnymi widokami, stopkę. Strony widoków są osadzone w głównym układzie korzystając z `@RenderBody()`.



W aplikacji wykorzystaliśmy Entity Framework zapewnia ochronę przed SQL Injection dzięki wykorzystaniu zapytań LINQ i parametrów wbudowanych w framework. Wszystkie operacje na bazie danych są zabezpieczone przed wstrzyknięciem kodu SQL.

Projekt wykorzystuje moduł autoryzacji Microsoft do zarządzania dostępem użytkowników. Daje możliwość logowania się za pomocą kont, może wykorzystywać OAuth2 do autoryzacji, a dane konfiguracyjne przechowywane są w `appsettings.json`

Dokumentacje wykonali Jakub Golec i Tymoteusz Hładyszewski