COLLEGIUM WITELONA
Uczelnia Państwowa

Sprawozdanie z projektu z przedmiotu Projektowanie i programowanie systemów internetowych I

System z ogłoszeniami, gdzie każdy może wystawiać i odpowiadać na ogłoszenia z ujęciem kategorii i lokalizacji

Autorzy:

Adam Rudziewicz 43882 s2PAM1(2)
Sebastian Waga 43894 s2PAM1(2)
Szymon Rogula 43880 s2PAM1(2)

Prowadzący: mgr inż. Krzysztof Rewak

9 czerwca 2025

Spis treści

1 Opis Funkcjonalny Systemu

Nasz projekt to nowoczesna, w pełni responsywna aplikacja internetowa, która pełni funkcję platformy do publikowania i przeglądania ogłoszeń. Została zaprojektowana z myślą o intuicyjności i łatwości obsługi, oferując bogaty zestaw funkcji dla użytkowników.

1.1 Główne moduły i funkcjonalności systemu

1.1.1 Przeglądanie i Wyszukiwanie Ogłoszeń

- **Strona główna:** Prezentuje najnowsze ogłoszenia oraz wizualne odnośniki do głównych kategorii, zachęcając użytkowników do eksploracji.
- **Zaawansowana wyszukiwarka:** Umożliwia filtrowanie ogłoszeń na podstawie słów kluczowych, wybranej kategorii oraz lokalizacji.
- **Lista ogłoszeń:** Wyniki wyszukiwania i przeglądania kategorii są prezentowane w formie czytelnej listy z paginacją, co ułatwia nawigację przy dużej liczbie wyników.

1.1.2 Szczegóły Ogłoszenia

- **Kompleksowy widok:** Każde ogłoszenie posiada dedykowaną stronę, na której wyświetlane są szczegółowe informacje: pełny tytuł, opis, cena, galeria zdjęć, dane autora, data publikacji i ważności oraz liczba wyświetleń.
- **Galeria zdjęć:** Użytkownicy mogą przeglądać zdjęcia w formie siatki, a kliknięcie na miniaturę otwiera je w powiększonym modalnym oknie.
- **Dynamiczne widżety:** Strona szczegółów została wzbogacona o interaktywne elementy, takie jak widżet pogodowy i lista podobnych ogłoszeń.

1.1.3 Zarządzanie Kontem Użytkownika

- **Rejestracja i Logowanie:** System oferuje bezpieczny proces tworzenia konta i logowania, z walidacją danych po stronie serwera i klienta.
- **Panel Użytkownika:** Zalogowany użytkownik ma dostęp do sekcji "Moje ogłoszenia", gdzie może zarządzać swoimi publikacjami.

1.1.4 Zarządzanie Ogłoszeniami (dla zalogowanych)

- **Tworzenie ogłoszenia:** Intuicyjny, wieloetapowy formularz pozwala na dodanie nowego ogłoszenia.
- **Edycja ogłoszenia:** Użytkownik może modyfikować wszystkie dane swojego ogłoszenia.
- **Usuwanie ogłoszenia:** Możliwość trwałego usunięcia ogłoszenia z systemu.

1.1.5 Interakcja Społeczna

- **System odpowiedzi:** Zalogowani użytkownicy mogą dodawać publiczne odpowiedzi (komentarze) do ogłoszeń w sposób asynchroniczny (bez przeładowania strony).

2 Opis Technologiczny

Aplikacja została zrealizowana w oparciu o nowoczesny stos technologiczny firmy Microsoft, z wykorzystaniem platformy .NET i języka C#. Architektura systemu bazuje na sprawdzonych wzorcach projektowych, co zapewnia jej skalowalność, wydajność i łatwość w utrzymaniu.

Komponent	**Technologia/Narzędzie**	**Opis i Zastosowanie**
Backend	ASP.NET Core MVC	Podstawowy framework aplikacji, implementujący wzorzec Model-View-Controller. Odpowiada za obsługę żądań HTTP, logikę biznesową oraz renderowanie widoków.
Baza Danych	SQLite	Lekki, plikowy system zarządzania bazą danych, używany do przechowywania wszystkich danych aplikacji.
ORM	Entity Framework Core	Narzędzie do mapowania obiektowo-relacyjnego (ORM), które umożliwia interakcję z bazą danych przy użyciu obiektów C# i zapytań LINQ.
API	ASP.NET Core Web API	Aplikacja wystawia własne, REST-owe API do obsługi interakcji asynchronicznych (AJAX) oraz konsumuje zewnętrzne API (OpenWeatherMap).
Uwierzytelnianie	ASP.NET Core Identity	Kompletny system do zarządzania użytkownikami, uwierzytelniania i autoryzacji.
Frontend	HTML5, CSS3, JavaScript	Podstawowe technologie webowe. Wykorzystano semantyczny HTML, nowoczesny CSS z trybem ciemnym oraz JavaScript (z jQuery) do dynamicznej manipulacji DOM.
Framework CSS	Bootstrap 5	Zapewnia responsywny system siatki, gotowe komponenty oraz spójny wygląd aplikacji.
Logowanie	Serilog	Zaawansowana biblioteka do logowania zdarzeń systemowych, błędów i informacji diagnostycznych.
Optymalizacja	In-Memory Caching	Wbudowany w ASP.NET Core mechanizm buforowania w pamięci, wykorzystywany do przechowywania często odpytywanych danych.
Wzorce	Dependency Injection (DI)	Centralny mechanizm architektury ASP.NET Core, używany do zarządzania cyklem życia serwisów i odwracania zależności.
Zarządzanie	NuGet	Menedżer pakietów dla platformy .NET, używany do zarządzania wszystkimi zewnętrznymi bibliotekami.

3 Analiza Zagadnień Kwalifikacyjnych

Poniżej znajduje się szczegółowa lista 20 zagadnień wraz z opisem implementacji i konkretnymi dowodami w kodzie projektu.

1. **Framework MVC**

   Opis Implementacji

   Aplikacja została zbudowana w oparciu o architekturę Model-View-Controller (MVC) na platformie ASP.NET Core. Logika biznesowa jest oddzielona od prezentacji.

   Dowody w Kodzie

   - `Program.cs`: Konfiguracja usług MVC (`builder.Services.AddControllersWithViews();`).
   - `Controllers/HomeController.cs`: Definicje kontrolerów z akcjami.
   - `Models/Listing.cs`: Przykłady klas modeli.
   - `Views/Home/Index.cshtml`: Przykłady widoków Razor.

2. **Framework CSS**

   Opis Implementacji

   Frontend aplikacji jest zbudowany z wykorzystaniem frameworka Bootstrap 5. Użyto jego systemu siatki, komponentów i klas użytkowych. Całość została rozszerzona w pliku `site.css`.

   Dowody w Kodzie

   - `site.css`: Rozbudowany plik stylów z customizacją Bootstrapa, zmiennymi CSS i trybem ciemnym (`@media (prefers-color-scheme: dark)`).
   - `Views/**/*.cshtml`: Powszechne użycie klas Bootstrapa, np. `.container`, `.row`, `.card`.

3. **Baza Danych**

   Opis Implementacji

   Projekt wykorzystuje system bazy danych SQLite, zarządzany za pomocą ORM Entity Framework Core.

   Dowody w Kodzie

   - `Program.cs`: Konfiguracja połączenia z bazą SQLite (`options.UseSqlite(...)`).
   - `Data/ApplicationDbContext.cs`: Definicja kontekstu bazy danych z DbSetami.

4. **Cache**

   Opis Implementacji

   W aplikacji zaimplementowano mechanizm buforowania w pamięci (in-memory cache) w celu optymalizacji wydajności, redukując liczbę zapytań do bazy danych.

   Dowody w Kodzie

   - `Program.cs`: Rejestracja usługi cache (`builder.Services.AddMemoryCache();`).
   - `Services/ListingService.cs`: Logika sprawdzania cache (`_cache.TryGetValue`), zapisywania (`_cache.Set`) oraz usuwania z cache.

5. **Dependency Manager**

   Opis Implementacji

   Projekt korzysta z NuGet jako menedżera pakietów oraz z wbudowanego kontenera Dependency Injection (DI) do zarządzania zależnościami wewnątrz aplikacji.

   Dowody w Kodzie

   - `Program.cs`: Rejestracja serwisów w kontenerze DI, np. `builder.Services.AddScoped<IListingService, ListingService>();`.
   - `Controllers/HomeController.cs`: Wstrzykiwanie zależności przez konstruktor.

6. **HTML**

   Opis Implementacji

   Struktura stron została zdefiniowana przy użyciu standardu HTML5 w plikach widoków Razor (`.cshtml`), z wykorzystaniem semantycznych tagów.

   Dowody w Kodzie

   - Wszystkie pliki `.cshtml` w folderze `Views`, np. `Home/Index.cshtml`, zawierają kod HTML.

7. **CSS**

   Opis Implementacji

   Aplikacja posiada zaawansowany, niestandardowy arkusz stylów CSS, który wykorzystuje zmienne CSS, `backdrop-filter`, gradienty, cienie i animacje.

   Dowody w Kodzie

   - `wwwroot/css/site.css`: Główny plik ze stylami, zawierający definicje zmiennych w `:root`, style dla trybu ciemnego oraz animacje `@keyframes`.

8. **JavaScript**

Opis Implementacji

Interaktywność aplikacji została wzbogacona przez użycie JavaScript i jQuery (asynchroniczne dodawanie odpowiedzi, podgląd zdjęć, walidacja, widżety).

Dowody w Kodzie

- `wwwroot/js/site.js`: Logika dla formularzy, powiadomień, widżetu pogody.
- `wwwroot/js/similar-listings.js`: Logika do asynchronicznego ładowania podobnych ofert.

9. **Routing**

Opis Implementacji

Aplikacja wykorzystuje routing ASP.NET Core do tworzenia przyjaznych dla użytkownika adresów URL (tzw. "pretty URLs").

Dowody w Kodzie

```
app.MapControllerRoute(
    name: "listingDetails",
    pattern: "listing/{id:int}/{slug?}",
    defaults: new { controller = "Listing", action = "Details" });

app.MapControllerRoute(
    name: "categoryBrowse",
    pattern: "category/{kategoria}/{lokalizacja?}",
    defaults: new { controller = "Home", action = "Index" });

```

10. **ORM**

Opis Implementacji

Zastosowano Entity Framework Core jako narzędzie ORM do interakcji z bazą danych przy użyciu obiektów C# i zapytań LINQ.

Dowody w Kodzie

- `Data/ApplicationDbContext.cs`: Definicja relacji w metodzie `OnModelCreating`.
- `Services/ListingService.cs`: Przykłady zapytań LINQ, np. `_context.Ogloszenia.Include(...).Where(...).ToListAsync()`.

11. **Uwierzytelnianie**

    Opis Implementacji

    System uwierzytelniania oparty jest na ASP.NET Core Identity, umożliwiając rejestrację, logowanie oraz integrację z zewnętrznymi dostawcami.

    Dowody w Kodzie

    - `Program.cs`: Konfiguracja `AddDefaultIdentity<ApplicationUser>(...)`.
    - `Models/ApplicationUser.cs`: Rozszerzenie modelu użytkownika.
    - `Views/Listing/Details.cshtml`: Warunkowe wyświetlanie treści (`@if (User.Identity.IsAuthenticated)`).

12. **Lokalizacja**

    Opis Implementacji

    Aplikacja nie jest przygotowana do obsługi wielu języków, zawiera jedynie opcje wyboru jezyka.

    Dowody w Kodzie

    Brak, zagadnienie niewykonane.

13. **Mailing**

    Opis Implementacji

    W projekcie przewidziano funkcjonalność wysyłania e-maili przez interfejs `IEmailService`, który jednak nie został skonfigurowany, ani nigdzie użyty.

    Dowody w Kodzie

    Brak, zagadnienie niewykonane.

14. **Formularze**

    Opis Implementacji

    Aplikacja intensywnie wykorzystuje formularze HTML do tworzenia/edycji ogłoszeń, logowania, rejestracji i wyszukiwania. Są one walidowane po stronie serwera i klienta.

    Dowody w Kodzie

    - `Views/Listing/Create.cshtml`: Formularz z obsługą `enctype="multipart/form-data"`.
    - `Views/Listing/Details.cshtml`: Formularz do dodawania odpowiedzi, obsługiwany przez AJAX.

15. **Asynchroniczne Interakcje**

    Opis Implementacji

    Aplikacja wykorzystuje AJAX do asynchronicznej komunikacji z serwerem (dodawanie komentarzy, ładowanie podobnych ogłoszeń).

    Dowody w Kodzie

    - `wwwroot/js/site.js`: Funkcje `$.ajax` do komunikacji z API.
    - `Controllers/ApiController.cs`: Endpointy API, które obsługują te zapytania.

16. **Konsumpcja API**

    Opis Implementacji

    Aplikacja integruje się z zewnętrznym API OpenWeatherMap w celu pobierania danych pogodowych.

    Dowody w Kodzie

    - `Services/ApiService.cs`: Metoda `GetWeatherAsync` wysyła zapytanie `HttpClient` i przetwarza odpowiedź JSON.

17. **Publikacja API**

    Opis Implementacji

    Aplikacja wystawia własne publiczne API w formacie JSON, udostępniając podobne ogłoszenia.

    Dowody w Kodzie

    - `Controllers/ApiController.cs`: Kontroler z atrybutem `[ApiController]`, który definiuje endpointy, np. `[HttpGet("listing/similar/{id}")]`.

18. **RWD (Responsive Web Design)**

    Opis Implementacji

    Interfejs użytkownika jest w pełni responsywny dzięki systemowi siatki Bootstrap oraz dedykowanym regułom `@media` w CSS.

    Dowody w Kodzie

    - `wwwroot/css/site.css`: Zapytania `@media (max-width: ...)`.
    - Użycie klas responsywnych Bootstrapa (`.col-md-6`, `.col-lg-4`) w plikach `.cshtml`.

19. **Logger**

Opis Implementacji

W projekcie zaimplementowano system logowania zdarzeń przy użyciu biblioteki Serilog, z zapisem do konsoli i plików tekstowych.

Dowody w Kodzie

```
var logger = new LoggerConfiguration()
  .ReadFrom.Configuration(builder.Configuration)
  .Enrich.FromLogContext()
  .WriteTo.Console()
  .WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)
  .CreateLogger();
builder.Logging.AddSerilog(logger);

```

20. **Deployment**

Opis Implementacji

Aplikacja nie została wdrożona.

Dowody w Kodzie

Brak, zagadnienie niewykonane.

4 Instrukcja Uruchomienia Systemu

4.1 Uruchomienie Lokalne (Deweloperskie)

Wymagania wstępne

- .NET SDK (wersja 8.0 lub nowsza).
- Edytor Kodu/IDE: Visual Studio 2022, VS Code lub JetBrains Rider.
- Klucz API OpenWeatherMap (opcjonalnie).

Kroki uruchomienia

1. **Klonowanie Repozytorium:**

```
git clone https://github.com/adamrudziewicz/projekt_S4-PPSI_1
cd OgloszeniaSytem

```

2. **Konfiguracja Klucza API (opcjonalnie):** W pliku `appsettings.json` dodaj swój klucz API.

```
"WeatherApi": {
  "ApiKey": "TWOJ_KLUCZ_API"
}

```

3. **Budowanie i Uruchomienie (z linii poleceń):**

```
dotnet restore
dotnet build
dotnet run
```

4. **Dostęp do Aplikacji:** Po uruchomieniu aplikacja będzie dostępna w przeglądarce pod adresem wskazanym w konsoli (u nas `https://localhost:5025`).

5 Wnioski Projektowe

Realizacja tego projektu pozwoliła na praktyczne zastosowanie i pogłębienie wiedzy z zakresu tworzenia nowoczesnych aplikacji internetowych.

- **Architektura MVC to solidny fundament:** Wybór wzorca MVC w ASP.NET Core ułatwił organizację kodu i jego utrzymanie.
- **Znaczenie nowoczesnego frontendu:** Połączenie Bootstrap 5 z niestandardowymi stylami i JavaScriptem pozwoliło na stworzenie estetycznego i funkcjonalnego interfejsu.
- **Asynchroniczność poprawia doświadczenie użytkownika:** Zastosowanie AJAX do dynamicznego ładowania treści jest kluczowe dla nowoczesnych aplikacji, zapewniając płynność i szybkość działania.
- **Kompleksowość ekosystemu .NET:** Narzędzia takie jak Entity Framework Core czy ASP.NET Core Identity pozwoliły na szybkie zaimplementowanie zaawansowanych funkcjonalności.

Podsumowując, projekt zakończył się sukcesem, a stworzona aplikacja jest w pełni funkcjonalnym i nowoczesnym systemem, który z powodzeniem realizuje wszystkie założone cele.