Collegium Witelona Uczelnia Państwowa w Legnicy
Wydział Nauk Technicznych i Ekonomicznych
Kierunek: Informatyka

Projekt z przedmiotu "Projektowanie i programowanie systemów internetowych I"

Temat: Grader.NET.

Autorzy
Denys Zachynaiev, nr. indeksu: 44056
Jan Rojek, nr. indeksu: 43881
Wiktor Szuwart, nr. indeksu: 43889

Prowadzący przedmiot
mgr inż. Krzysztof Rewak

Legnica, 2025

1 Spis treści

2 Cel i ogólna charakterystyka projektu

Projekt Grader.NET to webowa aplikacja przygotowana jako projekt zaliczeniowy modułu PPSI na drugim roku studiów.

2.1 Główne cele projektu

Celem projektu jest stworzenie funkcjonalnego elektronicznego dziennika.

2.2 Funkcje

Aplikacja umożliwia użytkownikom (w zależności od ich uprawnień):

- Przeglądanie ocen, zadań, sprawdzianów i harmonogramu klas.
- Dodawanie uczniów, ocen, zadań, sprawdzianów, lekcji.
- Przeglądanie kalendarza z dniami wolnymi od szkoły.

3 Architektura

Aplikacja Grader.NET została zaprojektowana w architekturze warstwowej z wykorzystaniem wzorca Model-View-Controller (MVC). Warstwa danych oparta jest o Entity Framework Core i bazę danych SQLite. Logika biznesowa została zaimplementowana w kontrolerach, a interfejs użytkownika oparty jest o technologię Razor Pages z wykorzystaniem frameworka Bootstrap 5.

4 Technologie i Frameworki

Technologia	Opis
.NET	v8.0
ASP.NET Core	v8.0
Bootstrap	Framework CSS v5.3
MailKit	biblioteka klienta pocztowego v4.12.0
jQuery	biblioteka JS v3.7.1
EntityFrameworkCore	Framework ORM do bazy danych v9.0.4
SQLite	System bazy danych

5 Opis działania komponentów

5.1 Kontrolery

- **AccountController** – autoryzacja i uwierzytelnianie użytkownika.
- **AdminController** – panel administratora, zarządzanie użytkownikami.
- **HomeController** – wyświetlanie strony domowej, obsługa błędów.
- **LanguageController** – zmiana języka interfejsu (pl, en).
- **StudentController** – zarządzanie uczniami i przypisaniami do klas.
- **PublicHolidaysController** – integracja z zewnętrznym API świąt.
- **APIController** – publikacja własnego API (lista studentów).

5.2 Modele

- **AppUser** – model użytkownika.
- **Class** – model klasy.
- **ClassStudent** – powiązanie ucznia z klasą.
- **Student** – model ucznia.
- **PublicHolidays** – model świąt wolnych od szkoły.
- **ErrorViewModel** – model obsługi błędów.

5.3 Widoki

- **Account** – logowanie, rejestracja, reset hasła.
- **Admin** – panel zarządzania użytkownikami.
- **Home** – strona domowa, wyświetlanie świąt.
- **Student** – zarządzanie uczniami i klasami.
- **Shared** – layout i obsługa błędów.

5.4 System logowania

Autoryzacja zrealizowana przy użyciu ASP.NET Core Identity.

5.5 System mailingu

Wysyłanie wiadomości e-mail (rejestracja, reset hasła) za pomocą MailKit.

5.6 Routing

Routing realizowany standardowym mechanizmem ASP.NET Core MVC.

5.7 Komponenty jQuery

Wykorzystane m.in. do asynchronicznego pobierania danych (moduł klas).

6 Analiza wdrożonych systemów

1. **Framework MVC** – ASP.NET Core MVC.
2. **Framework CSS** – Bootstrap 5.
3. **Baza danych** – SQLite + Entity Framework Core.
4. **Cache** – IMemoryCache, np. lista studentów w StudentController.
5. **Dependency manager** – NuGet.
6. **HTML** – Razor Pages.
7. **CSS** – Bootstrap + własne modyfikacje w site.css.
8. **JavaScript** – jQuery (np. sortowanie tabel, AJAX).
9. **Routing** – ASP.NET Core MVC (pretty URLs).
10. **ORM** – Entity Framework Core.
11. **Uwierzytelnianie** – ASP.NET Core Identity.
12. **Lokalizacja** – ASP.NET Localizer + pliki .resx.
13. **Mailing** – MailKit, realizacja w AccountController.
14. **Formularze** – Razor Pages + formularze HTML.
15. **Asynchroniczne interakcje** – jQuery + AJAX (moduł klas).
16. **Konsumpcja API** – API świąt w PublicHolidaysController.
17. **Publikacja API** – lista studentów w APIController.
18. **RWD** – Bootstrap 5, responsywne widoki dostosowane do urządzeń mobilnych.
19. **Logger** – Microsoft.Extensions.Logging, logi w katalogu /Logs.
20. **Deployment** – platforma Render.

7 Uruchomienie aplikacji

7.1 Wymagania systemowe

- Windows 10/11 lub Linux.
- .NET SDK 8.0 lub wyższy.
- Visual Studio 2022 lub Visual Studio Code.
- Git (opcjonalnie).
- Przeglądarka internetowa.

7.2 Instrukcja lokalnego uruchomienia

1. Sklonować repozytorium:
   ```
   git clone <adres_repozytorium>
   ```
2. Otworzyć projekt w Visual Studio / VS Code.
3. Wykonać migracje bazy danych:
   ```
   dotnet ef database update
   ```
4. Uruchomić aplikację:
   ```
   dotnet run
   ```
5. Otworzyć w przeglądarce: `https://localhost:5001`.

7.3 Instrukcja zdalnego uruchomienia

- Deployment zrealizowany na platformie Render.
- Konfiguracja parametrów środowiskowych w appsettings.json.
- Obsługa automatycznego deploymentu możliwa z GitHub Actions.

8 Wnioski projektowe

Projekt Grader.NET spełnił założone cele i dostarczył funkcjonalny system typu eDziennik. Udało się wdrożyć wszystkie wymagane elementy technologiczne i funkcjonalne.

8.1 Napotykanie trudności

- Integracja modeli i relacji w bazie danych.
- Synchronizacja danych uczniów i klas.
- Implementacja dynamicznych widoków (harmonogram, klasy).

8.2 Sugestie rozwoju

- Dodanie testów jednostkowych.
- Historia zmian ocen.
- Eksport danych do PDF/CSV.
- Rozszerzenie API REST.

8.3 Podsumowanie

Projekt pozwolił zespołowi na praktyczne wykorzystanie technologii .NET, wzorca MVC oraz integracji z bazą danych i zewnętrznymi API. Grader.NET jest kompletnym i stabilnym systemem, który może być rozwijany w kolejnych iteracjach.