Librus-Scrapper

Dokumentacja Projektu

Bartosz Ujma

Spis treści

1. Opis projektu

Projekt ma na celu automatyczne pobieranie danych z systemu Librus Skrypt napisany w języku JavaScript i uruchamiany przez rozszerzenie Tampermonkey w przeglądarce, następnie zbierane dane są przetwarzane i wysyłane do REST API gdzie są walidowane i przygotowywane do wysłania do bazy danych .

Projekt został stworzony jako zadanie zaliczeniowe do szkoły.

2. Technologie

JavaScript

JavaScript to najważniejszy język programowania używany w tym projekcie. Dzięki swojej elastyczności i możliwości działania bezpośrednio w przeglądarce, jest idealnym wyborem do wyciągania danych z kodu HTML(Web Scrappingu) w połączeniu z rozszerzeniem Tampermonkey. JavaScript jest językiem skryptowym, który działa po stronie klienta, co oznacza, że jest wykonywany bezpośrednio na komputerze użytkownika za pomocą przeglądarki.

JSON

JSON (JavaScript Object Notation) to format danych używany w projekcie do przesyłania informacji między skryptem a API. Jest to lekki format wymiany danych, który jest łatwy do odczytu i późniejszej walidacji podczas operacji na danych.

Tampermonkey

Tampermonkey to rozszerzenie przeglądarki, które umożliwia użytkownikom tworzenie i uruchamianie własnych skryptów JavaScript na stronach internetowych. W tym projekcie rozszerzenie to odgrywa kluczową rolę, ponieważ to dzięki niemu możliwe jest automatyczne uruchamianie skryptu na stronach internetowych , co pozwala na pobranie danych bez potrzeby ingerowania w serwer Librus, czy też prób przechwytywania pakietów wysyłanych do klienta.

REST API

REST API (Representational State Transfer Application Programming Interface) to interfejs, który pozwala na komunikację między skryptem a zewnętrzną bazą danych za pomocą endpointów, komunikacja odbywa się z pomocą protokołu HTTP/S. REST API w tym projekcie jest używane do przesyłania danych pobranych z systemu Librus do bazy danych.

ASP.NET

ASP.NET to platforma stworzona przez Microsoft do budowy aplikacji internetowych i API opierająca się o język .NET(C#). W tym projekcie język ten zostanie wykorzystany do stworzenia REST API i relacyjnej bazy danych MySql poprzez automatyczną migracje czyli generowanie bazy danych na podstawie schematu danych używanych w API, dzięki czemu mamy pewność że baza będzie w pełni kompatybilna z REST API.

Entity Framework Core (EF Core)

EF Core to narzędzie ORM (Object-Relational Mapping) które pozwala pracować z bazami danych w sposób obiektowy z poziomu kodu .NET. Zamiast pisać bezpośrednio zapytania SQL, programiści mogą operować na obiektach, które są mapowane na tabele w bazie danych, z jego pomocą zostanie zrobiona migracja(generowanie) bazy danych.

Baza danych MySql

Baza danych zostanie wygenerowana za pomocą systemu migracji na platformie ASP.NET z użyciem EF Core. W projekcie baza danych nie będzie łączyć się bezpośrednią z skryptem JavaScript, komunikacja będzie się odbywać za pomocą wcześniej wspomnianego REST API.

GIT

GIT rozproszony system kontroli wersji, który umożliwia śledzenie zmian w plikach projektu oraz współpracę w zespole nad wspólnym kodem. Jako serwis hostingowy dla repozytorium tego projektu został wybrany serwis GitHub.

3. Architektura rozwiązania

Projekt składa się z 3 głównych modułów

Moduł Web Scrappingu

Moduł Web Scrappingu opracowany w języku JavaScript, skrypty te będą obsługiwane przez rozszerzenie do przeglądarki internetowej tampermonkey. Język JS udostępnia wiele prostych funkcjonalności takich jak obiekt *DOM* za pomocą którego z łatwością można manipulować kodem strony HTML jak i co w tym przypadku ważniejsze ekstraktować z niego informacje.

Moduł REST API

Moduł ten jest opracowany w języku .NET na platformie ASP, będzie on odpowiedzialny za komunikacje z bazą jak i jej utworzenie na podstawie struktury danych zawartych w modelach danych. W tym module będzie odbywać się cała walidacja danych odebranych z pomocą protokołu HTTP/S wysłanych przez skrypty uruchamiane w przeglądarce

Moduł Bazy danych

Moduł relacyjnej bazy danych MySQL, w całości wygenerowany z pomocą REST API. Baza podczas procesu Developerskiego będzie działać lokalnie z pomocą narzędzia XAMPP

4. Struktura Bazy Danych/Modeli DBO

Struktura bazy danych może ulec zmianie podczas realizacji projektu z racji na czynniki które nie zostały przemyślane w okresie pisania dokumentacji i co ważniejsze, przyszłe zmiany w strukturze mogą być spowodowane jakością wykonania serwisu Librus.

Przykładowa struktura bazy danych:

Students:

- student_id: unikalny identyfikator
- first_name: imię ucznia
- last_name: nazwisko ucznia
- class: klasa, do której należy uczeń

Grades:

- grade_id: unikalny identyfikator oceny
- student_id relacja z tabelą Students
- subject: przedmiot, z którego została wystawiona ocena
- grade: ocena (np. 5, 4, 3, itd.)
- date: data wystawienia oceny

Attendance:

- attendance_id: unikalny identyfikator obecności
- student_id: odniesienie do tabeli Students
- date: data obecności
- status: status (obecny, nieobecny, spóźniony)
- lesson_nr: numer lekcji z którego wystawiona została obecność

5. Struktura REST API

Kontrolery

Kontrolery to podstawowy punkt wejścia(endpointy) dla każdego żądania HTTP w architekturze REST API. W .NET, każdy kontroler to klasa odpowiadająca za odbieranie różnych operacji CRUD(GET, POST, PUT i DELETE). Każdy kontroler jest powiązany z konkretnym modelem danych i modułem serwisowym do którego wysyłane są dane w celu walidacji i ich przetworzenia

Przykładowy kod kontrolera:

```
[ApiController]
[Route("api/[controller]")]
public class GradesController : ControllerBase
{
   private readonly IGradeService _gradeService;
   public GradesController(IGradeService gradeService)
   {
      _gradeService = gradeService;
   }
   [HttpGet]
   public async Task<IActionResult> GetAllGrades()
   {
      var grades = await _gradeService.GetAllGradesAsync();
      return Ok(grades);
   }
}
```

Serwisy

Serwisy zawierają parsing i walidację danych przyjętych od kontrolera, a także koordynują współpracę między repozytoriami a kontrolerami. Dzięki użyciu serwisów, kontrolery są tylko endpointami do odbierania danych ponieważ nie muszą zawierać żadnej logiki, rozbija to strukturę projektu co ułatwia jego późniejsze debugowanie. Dane później przesyłane są do repozytorium odpowiadającego za konkretne modele danych.

Przykładowy kod serwisu:

```
public class GradeService : IGradeService
{
    private readonly IGradeRepository _gradeRepository;


    public GradeService(IGradeRepository gradeRepository)
    {
        _gradeRepository = gradeRepository;
    }
    public async Task<ServiceResponse<GradeDTO>> AddGradeAsync(GradeDTO gradeDto)
    {
        var grade = new Grade
        {
            StudentId = gradeDto.StudentId,
            Subject = gradeDto.Subject,
            Value = gradeDto.Value,
            Date = gradeDto.Date
        };
        await _gradeRepository.AddGradeAsync(grade);
        return new ServiceResponse<GradeDTO>
    }
}
```

Repozytoria

Repozytoria są warstwą pośredniczącą pomiędzy serwisami a bazą danych. Repozytoria mają za zadanie odbierać dane z serwisów i komunikować się z pomocą obiektu DbContext dostarczonego przez EFCore z bazy danych.

Przykładowy kod repozytorium:

```
public class GradeRepository : IGradeRepository
{
    private readonly ApplicationDbContext _context;
    public GradeRepository(ApplicationDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Grade>> GetAllGradesAsync()
    {
        return await _context.Grades.ToListAsync();
    }
    public async Task<Grade> GetGradeByIdAsync(int id)
    {
        return await _context.Grades.FindAsync(id);
    }
    public async Task AddGradeAsync(Grade grade)
    {
        await _context.Grades.AddAsync(grade);
    }
}
```

Contexty

Entity Framework Core w ASP.NET wykorzystuje klasy DbContext do zarządzania dostępem do bazy danych. DbContext to klasa, która pozwala na interakcję z bazą danych, a także umożliwia mapowanie modeli DBO na tabele w bazie. Każdy kontekst jest odzwierciedleniem bazy danych i zawiera on modele danych odzwierciedlające tabele w tej bazie.

Przykładowy kod contextu:

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    public DbSet<Grade> Grades { get; set; }
    public DbSet<Student> Students { get; set; }
}
```

Modele DBO i DTO

Modele DBO są bezpośrednim odwzorowaniem tabel z bazy danych, i na ich podstawie jest ona generowana. Reprezentują one strukturę danych, która zostanie zapisana w bazie. Zazwyczaj posiadają takie elementy jak klucze główne (Primary Key), relacje między tabelami czy inkrementacje identyfikatorów.

Modele DTO są używane do wymiany danych pomiędzy API a klientem. Zazwyczaj są uproszczoną wersją modeli DBO, zawierając tylko te pola, które są niezbędne do wymiany danych w danym kontekście. DTO są szczególnie przydatne w przypadkach, gdy chcemy ukryć szczegóły implementacji, czy też ukrycia danych z bazy przed klientem.

Oba przypadki są implementowane w kodzie jako zwykła klasa.

6. Instalacja i konfiguracja

Wymagania

- Przeglądarka obsługująca Tampermonkey (np. Google Chrome, Firefox)
- Konto w systemie Librus
- Klucz API do bazy danych

Instalacja

Pobierz i zainstaluj rozszerzenie Tampermonkey dla swojej przeglądarki:

- Wersja dla przeglądarek opartych na chromium (np. Chrome, Opera, Brave) https://chromewebstore.google.com/detail/tampermonkey/dhdgffkkebhmkfjojejmpbldmpobfkfo?hl=pl&pli=1
- Wersja dla przeglądarki Firefox https://addons.mozilla.org/pl/firefox/addon/tampermonkey/

Dodanie skryptu do Tampermonkey:

Sposób pierwszy:

- Otwórz Tampermonkey klikając na jego ikonę w pasku narzędzi przeglądarki.
- Wybierz opcję "Utwórz nowy skrypt".
- Usuń przykładowy kod w edytorze i wklej swój skrypt.

Sposób drugi:

- Otwórz Tampermonkey klikając na jego ikonę w pasku narzędzi przeglądarki.
- W panelu Tampermonkey wybierz "Narzędzia".
- Kliknij przycisk "Import".
- wybierz plik JS skryptu z komputera.

7. Proces działania

Inicjalizacja skryptu JS

Cały proces rozpoczyna się od uruchomienia skryptu web scrapingu w przeglądarce, wykorzystując rozszerzenie Tampermonkey. Skrypt jest automatycznie wykonywany po otwarciu strony systemu Librus, a jego celem jest pobranie odpowiednich danych z serwisu.

Skrypt identyfikuje odpowiednie elementy strony internetowej (np. oceny), wykorzystując id i classy elementów HTML. Dane są odczytywane z DOM (Document Object Model) i przetwarzane do formatu JSON

Wysyłanie danych do REST API

Po zakończeniu skryptu wyciągającego dane, wykorzystuje funkcję fetch() do wysłania przetworzonych danych do serwera, na którym działa aplikacja REST API. Dane są wysyłane w formacie JSON w żądaniach HTTP typu POST.

W przypadku poprawnego odbioru danych serwer zwraca odpowiedź potwierdzającą sukces (status HTTP 200 lub 201).

Jeśli wystąpią problemy (np. błędy sieciowe lub niepoprawne dane), odpowiednie błędy są obsługiwane i użytkownik zostaje o nich poinformowany (np. przez alert w przeglądarce).

Przetwarzanie danych przez REST API

Po stronie serwera dane przychodzące od klienta są przetwarzane i mapowane na odpowiednie modele DTO(Data Transfer Object), po czym są one przekazywane z kontrolera do serwisu w którym przechodzą one walidację pod kątem ich poprawności (np. sprawdzanie, czy wszystkie wymagane pola są obecne, czy wartości są w dopuszczalnym zakresie)

Zweryfikowane dane są zapisywane w bazie danych z użyciem Entity Framework Core.

8. Struktura repozytorium projektu

Poniższy diagram pokazuje uproszczoną strukturę repozytorium projektu, używającego technologii kontroli wersji GIT:

https://github.com/YOURIxYOURI/Librus-Scrapper

```
Librus-Scrapper/
├── API/
│   ├── Controllers/
│   ├── Models/
│   ├── DTOs/
│   ├── DBOs/
│   ├── Repositories/
│   ├── Services/
│   ├── Migrations/
│   ├── ApplicationDbContext.cs
│   ├── Startup.cs
│   └── Program.cs
├── ScrapperScript/
│   ├── (Skrypty JS do Tampermonkey)
├── Dokumentacja Projektu.pdf
├── LICENSE
└── README.md
```

przedstawienie struktury wygenerowane przez AI

9. Wymagania systemowe

- Przeglądarka internetowa z zainstalowanym Tampermonkey
- Połączenie internetowe
- Dostęp do API bazy danych

10. Autorzy

Projekt został opracowany przez: Bartosz Ujma.

11. Licencja