Wrocław, 19 marca 2024

Platformy **P**rogramistyczne `.NET` i `Java`

Laboratorium 2

Projekt aplikacji bazodanowej `.Net`

prowadzący: *Dr inż. Radosław Idzikowski, mgr inż. Michał Jaroszczuk*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami projektowania aplikacji bazodanowych oraz korzystania z interfejsu programowania aplikacji (ang. application programming interface, `API`). W ramach zajęć należy stworzyć program bazodanowy w języku `C#`, który będzie rozbudowywany o kolejne funkcjonalności. Praca będzie oceniania na bieżąco podczas zajęć. **Ukończenie każdego etapu powinno być zgłoszone prowadzącemu w celu akceptacji i odnotowania postępów**. Sam program należy umieścić na repozytorium `github` i wysłać zaproszenie do prowadzącego. Czas na wykonanie zadania to dwa zajęcia laboratoryjne. Podczas pierwszego spotkania trzeba wykonać co najmniej jedno zadanie zdefiniowane w Rozdziale 3.

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Pobranie i deseralizacja danych z zewnętrznego `API`.
2. Obsługa bazy danych.
3. Zaawansowane GUI i obróbka danych.

Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Link do repozytorium należy przesłać dopiero po oddaniu i ocenieniu pracy na laboratorium.

3 Opis Zadań

W tej sekcji zostaną kolejno omówione wszystkie zadania do wykonania podczas laboratorium. Komputery w laboratorium zostały przygotowane do przeprowadzenia zajęć.

3.1 Zadanie 1

W ramach zadania należy wykonać aplikację w technologii `.NET 8.0`, która pozwoli na pobranie oraz wyświetlenie danych z wybranego `API`. Na tym etapie aplikacja może działać w konsoli, jednak zalecane jest tworzenie docelowo aplikacji okienkowej (np.: `Windows Forms`). Wybór odpowiedniego `API` jest dowolny i bezpośrednio nie ma wpływu na ocenę, jednak nie może być on wulgarny oraz musi zostać zaakceptowany przez prowadzącego. Ponadto istotne jest, żeby interfejs pozwalał na podawanie argumentów, ponieważ ich brak może uniemożliwić uzyskanie maksymalnej oceny za ten etap. Poniżej zaproponowane są trzy sprawdzone interfejsy:

1. Open Wheather Api - serwis pogodowy (miasto jako parametr),
2. Open Exchange Rates - serwis z kursem walut (data jako parametr),
3. VIES - VAT Information Exchange System - serwis zwracający dane podatnika (kraj oraz numer VAT ID jako parametry). Warto użyć poniższego linku jako endpoint'a: (dokumentacja na stronie nie jest zaktualizowana):
   https://ec.europa.eu/taxation_customs/vies/rest-api/ms/PL/vat/5260309174

Zaproponowane serwisy posiadają wersję darmową z ograniczoną liczbą zapytań na minutę oraz limitem miesięcznym wystarczającym na potrzeby zajęć. W serwisie pogodowym ponadto będziemy ograniczeni do aktualnej pogody, ale między zajęciami można zebrać kilka próbek. W przypadku serwisu z kursem walut jesteśmy ograniczeni do waluty "USD" jako bazowej. Mile widziane są własne propozycje.

Podłączenie do `API`

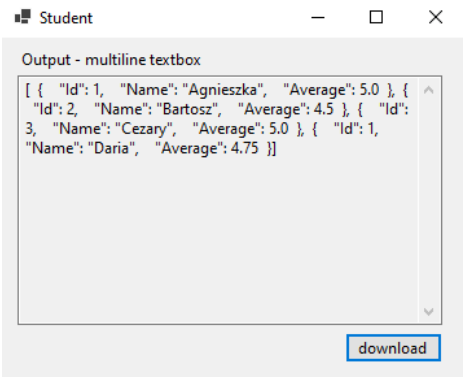
Głównym celem zadania jest pobranie danych z `API` z poziomu programu. Należy zacząć od utworzenia obiektu klasy `HttpClient`, który jest niezbędny do komunikacji z wykorzystaniem protokołu `http`. Żądanie typu pobierz można wywołać za pomocą metody `GetStringAsync()` lub `GetAsync()`. Należy pamiętać, że wymienione metody są asynchronicznie, w związku czym należy wymusić czekanie odpowiedź operatorem `await`, który czeka na zakończenie fragmentu kod wykonywanego asynchronicznie. Metoda nadrzędna również musi być asynchroniczna. Poniżej przedstawiono przykładowy kod do obsługi prostego interfejsu udostępniającego listę studentów:

```
namespace StudentAPI
{
    public partial class FormStudent : Form
    {
        private HttpClient client;
        public Form1()
        {
            InitializeComponent();
            client = new HttpClient();
        }
        private async void buttonDownload_ClickAsync(object sender, EventArgs e)
        {
            string call = "http://radoslaw.idzikowski.staff.iiar.pwr.wroc.pl/
    instruction/students.json";
            string response = await client.GetStringAsync(call);
            textBoxResponse.Text = response;
        }
    }
}
```

Chcąc użyć API w aplikacji konsolowej należy wykorzystać klasę `Task` pozwalającą na asynchroniczne wywołanie zapytania do API. Podobnie jak w przypadku aplikacji okienkowej, należy w funkcji o typie `Task` użyć obiektu klasy `HttpClient` i z wykorzystaniem metody `GetStringAsync` pobrać dane z zewnętrznego API. Samą funkcję o typie `Task` należy wywołać w głownym programie z metodą `Wait()`, która sprawia iż funkcja będzie oczekiwać na otrzymanie odpowiedzi z API.

```
public class APITest {
    public HttpClient client;
    public async Task GetData(){
        client = new HttpClient();
        string call = "http://radoslaw.idzikowski.staff.iiar.pwr.wroc.pl/
  instruction/students.json";
        string response = await client.GetStringAsync(call);
        Console.WriteLine(response);
    }
}
internal class Program
{
    static void Main(string[] args)
    {
    APITest t = new APITest();
    t.GetData().Wait();
    }
}
```

W przypadku bardziej rozbudowanych żądań należy edytować samo zapytanie, w opisanym przypadku zmienna `call`. Najprościej poprzez sklejanie odpowiednich fragmentów typu `string`. Efekt programu pokazano na Rysunku 1. Łatwo zaważyć, że odpowiedź jest w formacie `json`.



Rysunek 1: Dane pobrane z uproszczonego interfejsu student

Deserializacja

Od `.NET 5.0` do serializacji i deserializacji danych, m. in. w formacie `json`, nie trzeba używać zewnętrznych pakietów `NuGet`. W danych pobranych z API pokazanym w poprzednim etapie, znajduje się **lista**, składająca się z 4 rekordów z danymi o studentach (numer identyfikacyjny, imię oraz średnia ocen). Przed przystąpieniem do deseralizacji, należy stworzyć klasę `Student`, która posłuży w pierwszej kolejności do zmapowania pliku, a następnie do przechowywania informacji.

```
internal class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public float Average { get; set; }

    public override string ToString()
    {
        return $"Id: {Id},\tName: {Name,-15}\t,Average: {Average:0.00}";
    }
}
```

W przypadku każdej klasy warto przeciążyć metodą `ToString()` w celu łatwiejszego wyświetlania danych na dalszych etapach. Po przygotowaniu klasy do samej deseralizacji należy skorzystać z funkcji `Deserialize<typ>(string)`, która jako argument przyjmuje `string` do zmapowania oraz należy określić typ danych, w omowianym przypadku jest to **lista** studentów, ale nie zawsze odpowiedź z `API` będzie zwracana w formie listy. Po liście w łatwy sposób można iterować i wrzucić obiekty przykładowo do `listBox` lub osobno wyświetlić pole z klasy `Student` (np. same imiona).

```
private async void bDownload_ClickAsync(object sender, EventArgs e)
{
    string call = "http://radoslaw.idzikowski.staff.iiar.pwr.wroc.pl/instruction/
    students.json";
    string response = await client.GetStringAsync(call);
    List<Student> students = JsonSerializer.Deserialize<List<Student>>(response);
    foreach (var student in students) listBox1.Items.Add(student.ToString());
}
```

3.2 Zadanie 2

Celem zadania jest rozbudowanie programu z zadania pierwszego o prostą bazę danych. Program musi pozwalać na podstawową obsługę bazy (dodawanie, filtrowanie czy sortowanie danych). Ponadto, aby uzyskać maksymalną ocenę za ten etap, należy pobierać dane z `API` tylko w przypadku ich braku w bazie danych (ma to na celu uniknięcie ponownego pobierania kilkukrotnie tych samych danych). W zadaniu zostanie wykorzystanie mapowanie obiektowo-relacyjne (*Object-Relational Mapping*, `ORM`) z uzyciem `Enity Framework`. Struktura klasy, która posłuży do przechowywania danych w bazie może się różnić od klasy użytej do deserializacji odpowiedzi z `API`.

Przygotowanie bazy danych

W pierwszej kolejności należy dodać odpowiednie pakiety do naszego projektu z wykorzystaniem menadżera paczek `NuGet`, który znajduje się w karcie `Tools` (`narzędzia`). Podstawowym pakietem do zainstalowania jest `Microsoft.EntityFrameworkCore` w najnowszej wersji `8.0.3` (zgodność wersji jest bardzo istotna). Oprócz wybranego pakietu, powinny zainstalować się również inne wymagane pakiety. W ramach zadania wystarczy użycie bazy danych przechowywanej w pliku, więc niezbędny będzie jeszcze pakiet `Microsoft.EntityFrameworkCore.Sqlite`. Nie jest to jedyna możliwość, ponieważ `Microsoft` dostarcza również własne rozwiązanie czy wsparcie dla `PostgreSQL`. Na koniec należy jeszcze zainstalować narzędzia do obsługi bazy danych `Microsoft.EntityFrameworkCore.Tools`.

Wracając do przykładu studentów, w pierwszej kolejności należy przygotować klasę do przechowywania informacji o studentach. Można użyć tej samej klasy co poprzednio, ale w większości przypadków klasa będzie zawierać trochę inne pola (cześć poprzednich może być niepotrzebnych, ale też mogą dojść nowe). Ponadto warto oznaczyć pola wymagane `required` oraz te, które mogą przyjąć wartość `null` (należy wykorzystać operator `?`). Ponadto `Entity Framework` wymaga, aby pojawiło się pole `Id` (lub `NameClassId`), które będzie kluczem głównym. W przypadku pól złożonych (reprezentowanych) przez inne klasy możemy stworzyć osobne klasy, które również są identyfikowane po kluczu (wymaga zdefiniowania osobnej tabeli) lub od `.NET 8.0` bez klucza (nie wymaga zdefiniowania osobnej tabeli).

```
internal class Student
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required float Average { get; set; }
    public string? Specialty { get; set; }
    public override string ToString()
    {
        return $"Id: {Id},\tName: {Name,-15}\t,Average: {Average:0.00}, Specialty: {
    Specialty,-3}";
```

```
    }
}
```

Kolejnym krokiem jest zdefiniowanie klasy odpowiedzialnej za kontekst bazy danych. Ma ona za zadanie zarządzanie wszystkimi tabelami. Klasa nie może być publiczna. W bieżącym przykładzie wystarczy jedna tabela przechowująca dane o studentach. Następnie tworzymy konstruktor domyślny, który ma za zadanie utworzenie obiektu bazy danych. Z powodu użycia `SQLite`, trzeba jeszcze przeciążyć metodę konfiguracyjną i wskazać plik do przechowywania danych. Na koniec możemy jeszcze wstępnie uzupełnić bazę danych kilkoma rekordami początkowymi przy jej tworzeniu przeciążając metodę `OnModelCreating` (rozwiązanie opcjonalne). Przy dodawaniu nowych rekordów do bazy danych, należy pamiętać że klucz główny (w tym przypadku Id) nie może się powtarzać

```
internal class University : DbContext
{
    public DbSet<Student> Students { get; set; }
    public University()
    {
        Database.EnsureCreated();
    }
    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        options.UseSqlite(@"Data Source=Univ.db");
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Student>().HasData(
            new Student() { Id = 1, Name = "Agnieszka", Average = 5.5f },
            new Student() { Id = 2, Name = "Bartosz", Average = 4.5f },
            new Student() { Id = 3, Name = "Czarek", Average = 5.0f }
            );
    }
}
```

Przed przystąpieniem do wykonywania operacji na bazie danych, trzeba ją w pierwszej kolejności utworzyć. W tym celu należy przejść do konsoli zarządzania pakietami (`Package Manager Console`, domyślnie powinna być uruchomiana, widoczna obok Danych wyjściowych – `Output` oraz Listy błędów `Error List`. Jeśli nie, to jest dostępna w Narzędziach – `Tools`). W pierwszej kolejności należy utworzyć pierwszy punkt migracji bazy danych np.: `Add-Migration Init`, gdzie w tym wypadku `Init` to nazwa migracji. Domyślnie możliwość tworzenia migracji powinna być włączona (jeśli nie, to włączymy je komendą `Enable-Migrations`). Na koniec wymuszamy aktualizacje bazy danych komendą `Update-Database`. W przypadku zmiany struktury bazy danych, zawsze trzeba w pierwszej kolejności utworzyć nową migrację oraz następnie zaktualizować bazę danych.

Operacje na bazie danych

Ostatnim krokiem przed rozpoczęciem wykonywania operacji na bazie danych jest utworzenie obiektu klasy kontekstowej bazy danych wewnątrz klasy głównej naszego programu. Rekordy do bazy danych można dodawać przy użyciu metody `Add` na odpowiedniej tabeli. Proszę zwrócić uwagę, że przy dodawaniu nie nadajemy numeru `Id`, będzie on uzupełniony przez `Entity Framework` automatycznie z użyciem autoinkrementacji. UWAGA! Po operacjach, które mają wpływ na dane w bazie danych (dodanie lub usunięcie rekordu) należy zapisać aktualny stan bazy danych za pomocą metody `(SaveChanges())`. Możemy rekordy z tabeli zwrócić w formie listy używając metody `ToList`, a następnie podpiąć jako źródło do kontrolki typu `listBox`, aby wyświetlić pełną listę studentów.

```
public partial class FormStudent : Form
{
    private University university;
    public FormStudent()
    {
```

```
        InitializeComponent();
        university = new University();
    }
    private void buttonAdd_Click(object sender, EventArgs e)
    {
        university.Students.Add(new Student() { Name = textBoxName.Text, Average =
    float.Parse(textBoxAverage.Text) });
        university.SaveChanges();
        listBoxStudents.DataSource = university.Students.ToList<Student>();
    }
}
```

Usunięcie wszystkich rekordów z tabeli można uzyskać z wykorzystaniem poniższej komendy.

```
uni.Students.RemoveRange(uni.Students);
uni.SaveChanges();
```

Operacje na tabelach czy kolekcjach najwygodniej wykonywać z wykorzystaniem kwerend `LINQ`. W łatwy sposób można odfiltrować studentów poniżej średniej lub posortować ich po niej. W celu usunięcia wpisu, musimy najpierw go wyszukać, a dopiero następnie usunąć. Po usunięciu należy pamiętać o zapisaniu stanu bazy danych.

```
private void buttonGetTop_Click(object sender, EventArgs e)
{
    listBoxStudents.DataSource = university.Students.Where(s => s.Average > 4.5).
    ToList<Student>();
}

private void buttonSort_Click(object sender, EventArgs e)
{
    listBoxStudents.DataSource = university.Students.OrderBy(s => s.Average).Reverse
    ().ToList<Student>();
}
private void buttonRemove_Click(object sender, EventArgs e)
{
    var student = university.Students.First(s => s.Id == int.Parse(textBoxId.Text));
    university.Students.Remove(student);
    university.SaveChanges();
    listBoxStudents.DataSource = university.Students.ToList<Student>();
}
```

3.3 Zadanie 3

W tym zadaniu należy skupić się na kilku aspektach dotyczących interfejsu graficznego. Jego technologia jest dowolna, choć zalecany jest `Windows Forms`. Osoby pracujące na systemach innych niż `Windows` mogą poszukać technologii umożliwiającej uruchomienie GUI na ich systemie operacyjnym. Ogólnie zadanie pozostawia dużą dowolność co do finalnej formy, gdyż jego założeniem jest dopasowanie do uprzednio wybranego API oraz samodzielna eksploracja dostępnych możliwości.

Po pierwsze należy do programu dodać dodatkowe kontrolki, które poprawią atrakcyjność oraz funkcjonalność `GUI`. Jeśli wybrane `API` pozwala wybór wartości parametrów z ograniczonej puli lub wprost są typu logicznego (true/false) to warto rozważyć dodanie kontrolek `CheckBox` lub `RadioButton`, ewentualnie przy większej liczbie opcji `ComboBox`. Tak jak przy zadaniu z poprzednich laboratoriów należy zadbać o odpowiednie walidacje.

Drugim bardzo ważnym aspektem jest wizualizacja surowych lub przetworzonych danych (w zależności jakiego typu dane pobieramy z `API`). Dane można wizualizować na dowolny wybrany sposób. Jednak poza samą wizualizacją, równie ważne jest przetworzenie danych i policzenie różnych statystyk w miarę możliwości.