Algenic – serwis z konkursami algorytmicznymi
Dokumentacja techniczna

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Elektrotechniki, Automatyki,
Informatyki i Inżynierii Biomedycznej

Kacper Tonia Sławomir Kalandyk Mateusz Ruciński

1 Struktura projektu

Algenic składa się z następujących projektów:

- *Algenic*: pliki aplikacji sieciowej (kod HTML, CSS itd.), logika aplikacji.
- *Algenic.Commons*: podstawowe interfejsy, narzędzia niebędące związane ze specyfiką aplikacji.
- *Algenic.Compilation*: stanowi warstwę abstrakcji w procesie kompilacji kodu. Zawiera przede wszystkim klasy pozwalające na komunikację z API serwisu JDoodle.
- *Algenic.UnitTests*: testy jednostkowe.
- *Algenic.FunctionalTests*: testy Selenium.

1.1 Algenic.Algenic

Znajduje się tu plik Startup.cs, w którym jest kod wykonujący się przy starcie programu, gdzie duża jego część odpowiada za konfigurację, m.in umożliwienie na wstrzykiwanie podanych serwisów, ustanowienie połączenia z bazą danych itp.

1.1.1 Algenic.Algenic.Areas & Algenic.Algenic.Pages

W = Areas i Pages znajduje się kod odpowiedzialny za warstwę graficzną aplikacji w postaci plików *.cshtml, oraz ukryta za nimi logika aplikacji poprzez przypisany do każdego pliku *.cshtml plik *.cshtml.cs, który udostępnia widokowi dane poprzez data binding.

1.1.2 Algenic.Algenic.Commands

Zawierają się w nim polecenia wraz z odpowiadającymi im handlerami. Służą one do wykonywania operacji wnoszących zmiany w bazie danych.

Każdy handler implementuje interfejs ICommandHandler i metodę HandleAsync.

Każde polecenie (command) zawiera dane posiadające pewne znaczenie dla wykonania konkretnej operacji. Ponadto, polecenia mogą być tworzone tylko poprzez statyczną metodę Create.

1.1.3 Algenic.Algenic.Queries

= Queries zawiera klasy przypominające te z Commands, jednak służą one tylko do pobierania danych z bazy.

Każdy handler implementuje interfejs IQueryHandler i metodę HandleAsync.

Każde zapytanie (query) zawiera dane (np. identyfikatory) mające na celu umożliwienie pobrania tylko konkretnych danych.

Do każdego zapytania jest również przypisana klasa kończąca się słowem Result, udostępniająca dane pobrane z bazy.

1.1.4 Algenic.Algenic.Data

W Data przechowywane są klasy związane z dostępem do bazy danych. Każda klasa w Data/Models odpowiada konkretnej tabeli w bazie, oraz ma przypisany plik konfiguracyjny z Data/Configurations ustanawiający relacje pomiędzy nią i innymi klasami.

Na podstawie klas w Data/Models i ich konfiguracji, generowana jest baza danych przy pomocy migracji Entity Framework Core.

Ponadto znajduje się tam ApplicationDbContext, klasa reprezentująca całą bazę danych.

1.1.5 Algenic.Algenic.ViewModels

W ViewModels znajdują się klasy udostępniające wybrane dane do data binding dla widoku aplikacji. Jest to rozwiązanie, które pozwala na odseparowanie widoku od bezpośredniego połączenia z obiektami reprezentującymi dane w bazie, oraz wydzielenie z widoku (plików *.cshtml) logiki aplikacji.

1.1.6 Algenic.Algenic.Routing

Znajduje się tam tylko jedna klasa, DefaultRedirections. Udostępnia ona stałe strony, na które użytkownik może zostać przekierowany np. w wypadku braku praw dostępu do elementów aplikacji.

1.1.7 Algenic.Algenic.Configuration

Configuration posiada klasy wygenerowane przez ASP .NET Core związane z konfiguracją aplikacji.

1.2 Algenic.Compilation

Znajduje się tu m.in klasa JDoodleCompiler (implementująca IRemoteCompiler) służąca do wysłania zapytania do serwisu JDoodle i obsłużenie odpowiedzi na to zapytanie.

1.2.1 Algenic.Compilation.Outputs

W Outputs są klasy reprezentujące potencjalne dane zwracane przez REST-owe API JDoodle w odpowiedzi na wysyłane zapytanie HTTP.

1.2.2 Algenic.Compilation.Utilities

Utilities posiada m.in klasę CompilationRequest odpowiedzialną za budowanie zapytania do serwisu JDoodle.

1.3 Algenic.Commons

Znajdują się tu interfejsy ICommandHandler oraz IQueryHandler, implementowane odpowiednio przez handlery poleceń oraz zapytań do bazy danych.

Klasa statyczna Fail udostępnia kilka prostych metod do sprawdzania podstawowych warunków, takich jak sprawdzenie, czy obiekt jest nullem, czy string jest pusty, lub sprawdzenie warunku podanego w argumencie metody. W wypadku nie spełnienia warunków, wyrzucany jest wyjątek wraz z odpowiednią wiadomością, która może być zmodyfikowana w argumencie metody.

2 Struktura strony internetowej

- Strona główna: zalogowany użytkownik może tutaj obejrzeć wyniki konkursów, w których brał udział.
- */ScorePolicies*: służy tworzeniu nowych polityk oceniania. Dostęp tylko dla egzaminatorów.
- */Contests*: umożliwia przeglądanie, dołączanie do istniejących konkursów. Egzaminatorzy mogą dodatkowo tworzyć nowe konkursy bądź edytować istniejące.
- */Admin*: panel administratora
- */Tasks/X*: wyświetlanie zadania o id X
- */Results/X*: wyświetlanie rezultatów konkursu o id X

3 Połączenie z bazą danych

W celu maksymalnego odseparowania warstwy prezentacji od logiki domenowej, wykorzystaliśmy tzw. handlery, które przede wszystkim "opakowują" operacje na bazie danych.

Command handler wprowadza pewne zmiany w bazie danych.

Query handler pobiera dane z bazy danych, nie modyfikując jej zawartości.

Wprowadzenie handlerów nie tylko redukuje ilość kodu w plikach bezpośrednio związanych z renderowaniem widoków, ale też ułatwia testowanie. Funkcjonalność pojedynczego handlera jest względnie łatwa w testowaniu.

4 Interfejsy

4.1 Komunikacja z klientem

Klient komunikuje się z aplikacją poprzez protokół HTTP, głównie za pomocą zapytań typu GET i POST.

4.2 Komunikacja z bazą danych

Sama komunikacja z bazą danych została zrealizowana za pomocą mapowania obiektowo-relacyjnego przy użyciu Entity Framework Core. Wszelkie polecenia służące do wprowadzania zmian w bazie, a także pobieranie danych z bazy odbywa się za pośrednictwem obiektów modelu, na podstawie którego została wygenerowana baza danych przy pomocy migracji.

5 Autoryzacja i uwierzytelnianie

Za autoryzację i uwierzytelnianie odpowiada framework ASP .NET Core, który jest rdzeniem całego projektu. Pozwala on na utworzenie ról użytkowników i przypisywanie ich. Po uprzednim odpowiednim skonfigurowaniu modelu bazy danych, tworzy on relacje pomiędzy tabelami utworzonymi na bazie modelu, a utworzonymi przez siebie tabelami odpowiedzialnymi za wszelkie działania związane z użytkownikami.

6 Zdalna kompilacja

W celu kompilowania plików z kodem źródłowym przysyłanych przez użytkowników został użyty serwis JDoodle. Oferuje on REST-owe API do zdalnej kompilacji. Ogranicza to potencjalne zagrożenia związane z kompilacją niezaufanego kodu na serwerze.

7 Testy automatyczne

Testy automatyczne w projekcie dzielą się na 2 grupy: jednostkowe i funkcjonalne.

7.1 Testy jednostkowe

Aby ułatwić testowanie, operacje na bazie danych zostały wydzielone i zawarte w tzw. handlerach. QueryHandler ma za zadanie pobrać dane z bazy i zwrócić wynik, natomaist CommandHandler dodaje nowe dane bądź modyfikuje istniejące wpisy. Dla każdego testu tworzona jest w pamięci operacyjnej (in-memory) tymczasowa baza danych, zatem testy nie wpływają ani na siebie, ani na oryginalną bazę.

7.2 Testy funkcjonalne

Testy funkcjonalne wykorzystują 3 konta użytkowników, tworzonych domyślnie, gdy aplikacja jest uruchamiana w trybie developerskim.

Obecne testy funkcjonalne badają następujące funkcjonalności programu:

1. Dostęp do panelu administratora
   - jako zwykły użytkownik (brak dostępu)
   - jako egzaminator (brak dostępu)
   - jako administrator (dostęp przyznany)
2. Dodawanie konkursów
   - egzaminator tworzy nowy konkurs
   - dane nowego konkursu powinny pojawić się w tabeli konkursów
3. Uprawnienia do dodawania konkursów
   - jako egzaminator (pojawia się odpowiedni formularz)
   - jako zwykły użytkownik (brak formularza na stronie)
4. Zmiana właściciela konkursu przy odebraniu uprawnień egzaminatora obecnemu właścicielowi konkursu