# Wielowarstwowy system rekrutacji dla szkół z interfejsem webowym i aplikacją mobilną - technika TDD

Andrzej Westfalewicz, Filip Zyskowski

7 listopada 2019

# 1 Wstęp

Technika TDD (Test-Driven Developement) jest techniką tworzenia oprogramowania, w której główną ideą jest w pierwszej kolejności pisanie testów do nieistniejącej funkcjonalności, a dopiero potem napisanie kodu implementującego tę funkcjonalność. Ten dokument będzie przedstawiał jak w naszym projekcie wygląda to podejście od strony implementacji kodu oraz uruchomienia środowiska testowego.

# 2 Projekt testów

Dzięki stworzonej już architekturze, w łatwy sposób możemy implementować kolejne funkcjonalności w technice TDD. Przejrzysta struktura i proste ich wywoływanie pozwala sprawnie i szybko przejść z pisania testów do samej implementacji.

## 2.1 Struktura testów

Projekt będzie testowany tylko pod względem logiki serwera.
Testy logiki znajdują się w folderze *src/RecruitMe.Logic.Tests*. Struktura folderów odpowiada strukturze folderów projektu logiki (*RecruitMe.Logic*), co znaczy tyle, że dla dowolnego pliku testu w projekcie testów, istnieje klasa testowana w tym samym folderze co plik testu, ale w projekcie logiki.

Testy zostały napisane z użyciem frameworka nUnit. Dla klas napisanych z pomocą tego narzędzia charakterystyczne są dwa atrybuty metod: *Setup* - wskazująca metodę, która uruchamia się przed wykonaniem każdego testu, oraz *Test* - oznaczającą pojedyńczy test.
W pojedyńczym pliku testu znajdzie się jedna metoda *Setup* oraz co najmniej jedna metoda *Test*.


Figure 1: Metoda *Setup()* uruchamiająca się przed każdym testem

W metodzie *Setup* będziemy zawsze tworzyć atrapę (*mock*) bazy danych oraz używanej przez testowaną klasę tabeli/tabel. Tak stworzona atrapa pozwala w łatwy i powtarzalny sposób testować nasze metody, nie martwiąc się, że ewentualny błąd w metodzie zależy w jakiś sposób od bazy danych.


Figure 2: Przykładowy test

W metodach oznaczonej atrybutem *[Test]* znajduje się logika pojedyńczego testu. Wizualnie składa się z dwóch części: części przygotowania obiektów pod konkretny test oraz części działania i sprawdzenia w jednym.
Pierwsza część zawiera deklarację i inicjalizację obiektów potrzebnych do testu konkretnej funkcjonalności. Tutaj używane są klasy bez wewnętrznej logiki lub przetestowane już w innych testach.
Druga część posiada wywołanie funkcjonalności, którą chcemy przetestować. Wywołanie funkcjonalności jest częścią asynchronicznej metody anonimowej. Sprawdzenie funkcjonalności sprowadza się do wyboru odpowiedniej metody z klasy *Assert* i przekazania do niej wyżej wymienionej metody anonimowej. Gdy chcemy sprawdzić, czy obiekt zwracany przez naszą testowaną funkcjonalność jest równy innemu, korzystamy wtedy z metody *Assert.That*.

## 2.2 Uruchomienie środowiska testowego

Używając środowiska programistycznego *Visual Studio* możemy łatwo sprawdzić czy napisane testy są napisane poprawnie oraz czy kod logiki jest zgodny z tymi testami. Otwierając cały projekt w wyżej wymienionym programie, możemy kliknąć prawym klawiszem myszy na projekt testów (*RecruitMe.Logic.Tests*) i kliknąć *Run tests*.
Wyświetli się nam wtedy okienko z listą wszystkich dostępnych testów. Jednocześnie zostanie automatycznie uruchomione środowisko testowe, na którym będą przeprowadzane testy - będzie to widoczne po tym, że przy każdym teście w liście pojawi się kręcące się kółko. Sprawdzanie zakończy się, gdy przy każdym elemencie na liście pojawi się zielone lub czerwone kółko. Pojawienie się czerwonego kółka przy dowolnym z testów sprawia, że testy nie przechodzą poprawnie i wymagana jest interwencja w kodzie projektu - albo w samej logice testów, albo projektu.

Opcjonalnie, możemy również przeprowadzić testy uruchamiając w terminalu polecenie *dotnet test RecruitMe.sln* (będąc w głównym folderze rozwiązania). Wymagane jest wtedy jednak przywrócenie i zbudowanie całego projektu odpowiednio poleceniami *dotnet restore* oraz *dotnet build*. Polecenia *build* i *test* uruchamiamy w konfiguracji *Release*.

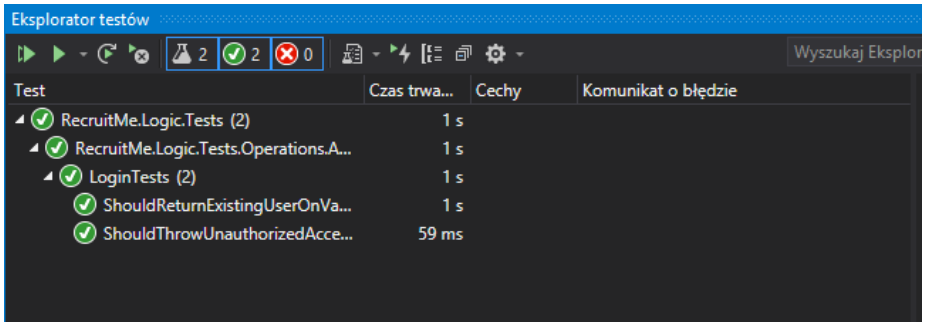


Figure 3: Efekty uruchomienia testów

# 3 Historia dokumentu

| Autor | Data | Wersja | Wprowadzone zmiany |
|---|---|---|---|
| Filip Zyskowski | 06.12.2019 | v0.1 | Pierwsza wersja dokumentu |