Algenic – serwis z konkursami algorytmicznymi

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

Kacper Tonia Sławomir Kalandyk Mateusz Ruciński

1 Wymagania projektowe

1.1 Dramatis personae

Użytkownik:

- Wysyła rozwiązania zadań konkursowych i jest za nie odpowiednio punktowany
- Po zakończeniu konkursu, każdy użytkownik może zobaczyć ranking
- Sformatowanie standardowego wyjścia programu leży po stronie użytkownika wysyłającego rozwiązanie, np. jeśli akceptowane rozwiązanie jest postaci „`x y`", gdzie `x` i `y` to liczby, to użytkownik powinien zadbać, aby wyjście jego programu przyjmowało taką formę
- Rozwiązanie zadania powinno być wysyłane przez użytkownika jako plik z kodem

Organizator konkursów:

- Można założyć, że jest tylko jeden organizator konkursów
- Tylko organizator konkursów może tworzyć nowe konkursy i dodawać do nich zadania
- Dodaje odpowiednio sformatowane przypadki testowe do zadań, względem których jest porównywane standardowe wyjście programu użytkownika
- Ustala ilość punktów otrzymanych przez użytkownika za rozwiązanie (w pełni lub częściowe) zadania

Serwer/administrator:

- Zleca kompilację kodu użytkownika, dokonuje oceny i zwraca odpowiedni feedback użytkownikowi
- Przechowuje standardowe wyjście błędu i loguje je, w razie potrzeby wyświetla użytkownikowi
- Przechowuje historie przesłanych rozwiązań
- Dba o podstawowe kwestie bezpieczeństwa – nie przechowuje haseł jako plaintext, uniemożliwia SQL injection

Pozostałe wymagania:

- System antyplagiatowy nie jest konieczny
- Kwestia prezentacji projektu - czy na serwerze zewnętrznym (dostęp przez internet), czy na własnej maszynie, wybór jest dowolny

2 Przypadki użycia

2.1 Dodanie rozwiązania zadania

Aktorzy: Użytkownik
Zakres: Serwis z konkursami algorytmicznymi Algenic
Cel (story): Dodanie rozwiązania do zadania
Warunki początkowe: Użytkownik jest uczestnikiem aktywnego konkursu, konkurs posiada przynajmniej jedno (nierozwiązane przez użytkownika) zadanie
Warunek końcowy dla powodzenia: Zadanie zostaje rozwiązane, użytkownik zostaje powiadomiony o poprawnym wykonaniu zadania
Warunek końcowy dla niepowodzenia: Zadanie nie zostaje rozwiązane, użytkownik zostaje powiadomiony o niepowodzeniu oraz rodzaju błędu
Zdarzenie wyzwalające (trigger): Użytkownik wybiera zadanie z konkursu, którego jest uczestnikiem i wybiera funkcję dodania rozwiązania zadania
Scenariusz główny:

1. Użytkownik wybiera konkurs
2. Użytkownik wybiera zadanie
3. Serwis wyświetla treść zadania i udostępnia możliwość wrzucenia rozwiązania w postaci pliku z rozwiązaniem
4. Użytkownik wrzuca plik z rozwiązaniem zadania
5. Serwis wielokrotnie kompiluje plik z rozwiązaniem z różnymi danymi wejściowymi
6. Serwis porównuje dane wyjściowe kompilacji z poprawnymi rozwiązaniami zadania
7. Serwis zapisuje plik oraz wynik kompilacji (sukces/treść błędu) w historii
8. Serwis zwraca wiadomość o poprawności wykonania zadania. Koniec przypadku użycia.

Rozszerzenia scenariusza głównego:

5a. Kompilacja nie powiodła się

5a1. Serwis zwraca wiadomość o niepowodzeniu kompilacji wraz z treścią błędu. Koniec przypadku użycia

6a. Dane wyjściowe kompilacji nie stanowią poprawnego rozwiązania zadania

6a1. Serwis zwraca wiadomość o niepoprawnym rozwiązaniu zadania. Koniec przypadku użycia.

2.2 Utwórz konkurs

Aktorzy: Organizator konkursów
Zakres: Serwis z konkursami algorytmicznymi Algenic
Cel (story): Organizator konkursów chce utworzyć nowy konkurs
Warunki początkowe: Organizator konkursów jest zalogowany
Warunek końcowy dla powodzenia: Konkurs został utworzony
Warunek końcowy dla niepowodzenia: Konkurs nie został utworzony
Zdarzenie wyzwalające (trigger): Organizator konkursów wybiera opcję utworzenia nowego konkursu
Scenariusz główny:

1. Organizator konkursów wybiera funkcję utworzenia nowego konkursu
2. Serwis wyświetla formularz utworzenia nowego konkursu
3. Organizator konkursu wprowadza nazwę konkursu
4. Zostaje utworzony nowy konkurs w stanie „nierozpoczęty". Koniec przypadku użycia.

2.3 Dodaj zadanie do istniejącego konkursu

Aktorzy: Organizator konkursów
Zakres: Serwis z konkursami algorytmicznymi Algenic
Cel (story): Organizator konkursów chce dodać zadanie do konkursu
Warunki początkowe: Organizator konkursów ma możliwość dokonywania zmian w konkursie w stanie "nierozpoczęty"
Warunek końcowy dla powodzenia: Zadanie zostało dodane do konkursu
Warunek końcowy dla niepowodzenia: Zadanie nie zostało dodane do konkursu, zostaje wyświetlony komunikat o niepowodzeniu
Zdarzenie wyzwalające (trigger): Organizator konkursów wybiera konkurs w stanie "nierozpoczęty" i wybiera opcję dodania zadania do konkursu **Scenariusz główny:**

1. Organizator konkursów wybiera konkurs w stanie "nierozpoczęty"
2. Organizator konkursów wybiera funkcję dodania zadania do konkursu
3. Serwis wyświetla formularz do dodania nowego zadania
4. Organizator konkursów wypełnia formularz
5. Nowe zadanie zostaje dodane do konkursu. Koniec przypadku użycia.

Rozszerzenia scenariusza głównego:

4a. Formularz został wypełniony nieprawidłowo

4a1. Zostaje wyświetlony komunikat o niepowodzeniu. Koniec przypadku użycia.

2.4 Rejestracja

Aktorzy: Użytkownik
Zakres: Serwis z konkursami algorytmicznymi Algenic
Cel (story): Użytkownik chce się zarejestrować
Warunki początkowe: Użytkownik nie jest zarejestrowany
Warunek końcowy dla powodzenia: Użytkownik zostaje zarejestrowany
Warunek końcowy dla niepowodzenia: Użytkownik nie zostaje zarejestrowany, zostaje wyświetlony komunikat o niepowodzeniu
Zdarzenie wyzwalające (trigger): Użytkownik wybiera funkcję rejestracji
Scenariusz główny:

1. Serwis wyświetla formularz rejestracji
2. Użytkownik wprowadza email oraz hasło (email służy jako login)
3. Następuje weryfikacja wprowadzonych danych po stronie serwisu
4. Użytkownik zostaje zarejestrowany. Koniec przypadku użycia.

Rozszerzenia scenariusza głównego:

3a. Weryfikacja zakończona niepowodzeniem (email już istnieje w bazie, lub hasło nie spełnia warunków)

3a1. Zostaje wyświetlony komunikat o niepowodzeniu. Koniec przypadku użycia.

2.5 Logowanie

Aktorzy: Użytkownik
Zakres: Serwis z konkursami algorytmicznymi Algenic
Cel (story): Użytkownik chce się zalogować
Warunki początkowe: Użytkownik jest w stanie niezalogowanym
Warunek końcowy dla powodzenia: Użytkownik zmienia stan na zalogowany
Warunek końcowy dla niepowodzenia: Użytkownik nie zmienia stanu na zalogowany, zostaje wyświetlony komunikat o błędzie
Zdarzenie wyzwalające (trigger): Użytkownik wybiera funkcję zalogowania
Scenariusz główny:

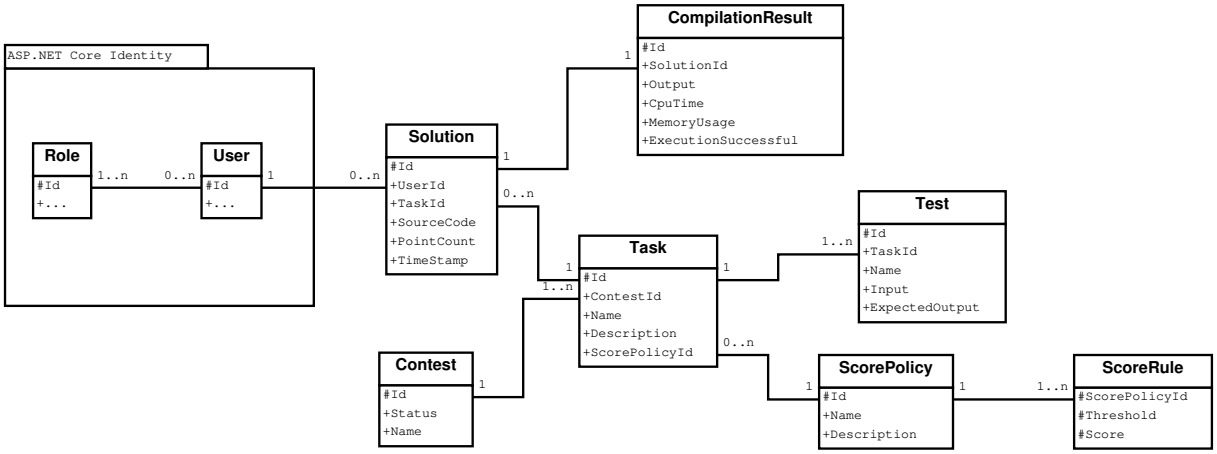
1. Serwis wyświetla formularz do logowania
2. Użytkownik wprowadza login oraz hasło
3. Następuje autentykacja danych po stronie serwisu
4. Stan użytkownika zostaje zmieniony na zalogowany. Koniec przypadku użycia.

Rozszerzenia scenariusza głównego:

3a. Autentykacja nie powiodła się

3a1. Serwis wyświetla komunikat o błędzie. Koniec przypadku użycia.

3 Baza danych



Rysunek 1: Diagram bazy danych

4 Technologia i narzędzia

Język programowania: C#
Framework: ASP.NET Core 2.2
System kontroli wersji: Git
Platforma: GitHub
Serwis do testów automatycznych: Travis CI
Zdalna kompilacja: JDoodle

Projekt został oparty o framework ASP.NET Core ze względu na jego popularność, wieloplatformowość i fakt bycia oprogramowiem typu open source. Zrezygnowaliśmy z sięgnięcia po najnowszą wersję frameworka, 3.0, na rzecz bardziej dojrzałej i potencjalnie stabilniejszej wersji 2.2.

Rozważyliśmy wykorzystanie usługi Azure, co pozwoliłoby na skupienie większości aspektów pracy nad programem w jednym miejscu (począwszy od przechowywania plików projektu, poprzez zarządzanie bazą danych projektu, skończywszy na automatycznym testowaniu zmian i wdrażaniu aktualnej wersji aplikacji w chmurze Azure). Stosunkowo długie czasy testowania i wdrażania w chmurze, a także względnie niewielka złożoność naszego projektu sprawiły, że zdecydowaliśmy się na uruchamianie i testowanie aplikacji lokalnie, wraz z lokalną bazą danych w technologii SQL Server.

Ostatecznie wybraliśmy platformę GitHub m.in. ze względu na wbudowane narzędzia do zarządzania projektem, takie jak tablica zadań, które ułatwią systematyzację zadań związanym z etapem implementacji i przydzielanie owych zadań członkom zespołu. Jako element praktyki *Continuous Integration*, zmiany dodawane do głównego repozytorium są automatycznie testowane na platformie **Travis CI**.

Na późniejszym etapie prac nad projektem możemy ponownie rozważyć wykorzystanie Azure, chociażby w celu uruchomienia wspólnej, trwałej bazy danych w chmurze.

W celu kompilowania kodów przysyłanych przez użytkowników wykorzystamy serwis **JDoodle**, oferujący REST-owe API do zdalnej kompilacji. Kompilowanie niezaufanego kodu na serwerze niosłoby poważne zagrożenie, o ile nie podjęlibyśmy specjalnych środków bezpieczeństwa. Użycie istniejącej usługi jak JDoodle redukuje potencjalne problemy z bezpieczeństwem.

W darmowym wariancie JDoodle pozwala na przeprowadzenie ok. 200 kompilacji dziennie. Może być to zbyt mała wartość podczas intensywnego testowania, jednak prostota użycia i bogata ilość dostępnych języków programowania przekonuje nas ostatecznie do owego rozwiązania.

5 Testy

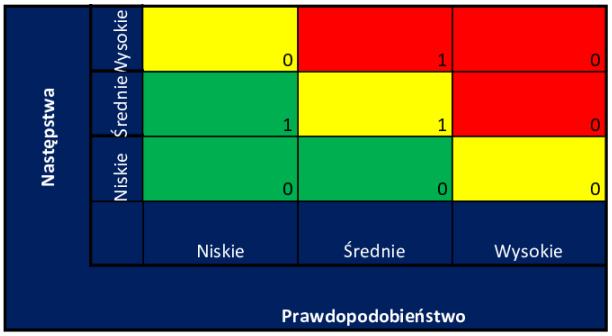
Testy wchodzące w skład projektu możemy podzielić na testy jednostkowe i funkcjonalne. Wybranym frameworkiem testowym jest **xunit**. Do testów funkcjonalnych dodatkowo wykorzystujemy **Selenium**

WebDriver – framework automatyzujący operacje na przeglądarkach internetowych.

Celem testów jednostkowych jest weryfikacja działania możliwie małych, odizolowanych od siebie elementów kodu. Mija się z celem jednak testowanie np. każdej metody z osobna, gdyż wymuszałoby to uczynienie wszystkich metod publicznymi. Zamiast tego proponujemy, aby klasy upubliczniały absolutne minimum funkcjonalności. Ów publiczny interfejs będziemy poddawać testom jednostkowym.

Użytkownik przeważnie nie jest jednak zainteresowany działaniem pojedynczych elementów składowych aplikacji, stąd pomysł wprowadzenia testów funkcjonalnych, które testują znaczną część aplikacji z punktu widzenia użytkownika. W naszym przypadku testy funkcjonalne opierają się na realizacji szeregu zautomatyzowanych operacji w przeglądarce internetowej i sprawdzeniu, czy odpowiedź aplikacji spełnia nasze oczekiwania.

6 Analiza ryzyka



Rysunek 2: Macierz ryzyka

Macierz ryzyka tworzymy w celu określenia poziomu ryzyka związanego z produkcją i działaniem produktu w celu obniżenia negatywnego wpływu ryzyka na funkcjonowanie danego podmiotu i podejmowanie odpowiednich działań służących przeciwdziałaniu i ograniczaniu ryzyka. Po stworzeniu listy ryzyk i oszacowaniu ryzyka dla każdego elementu z listy można wywnioskować, że najwyższe ryzyko jest związane z błędem ludzkim (programisty). Minimalizować to ryzyko możemy, testując oprogramowanie przed wdrożeniem oraz dbać o obecność backupów na wypadek problemów z aktualizacją.

	Ryzyko	Prawdopodobieństwo	Następstwa	Poziom ryzyka	Sposób zapobiegania	skala	wartość
1	wadliwy kod przsłany przez użytkownika	3	0	niskie	korzystanie z JDoodle	0	brak
2	błędy oprogramowania	6	9	wysokie	praca lokalnie, testy	3	niskie
3	problem z serwerem	6	6	średnie	korzystanie z Microsoft Azure	6	średnie
4	podłączenie sie cudzego złośliwego oprogramowania	3	6	niskie	korzystanie z Microsoft Azure	9	wysokie

Rysunek 3: Lista ryzyk