Część serwerowa systemu automatycznej ewaluacji zadań algorytmicznych

(The server part of the system of algorithmic tasks automatic evaluation)

Kamil Breczko

Praca inżynierska

Promotor: dr Wiktor Zychla

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

7 lutego 2019

Streszczenie

Praca przedstawia część serwerową systemu automatycznej ewaluacji zadań algorytmicznych. Podstawowe wymagania stawiane przed aplikacją to: zarządzanie grupami zajęciowymi, system uwierzytelnienia i autoryzacji, system powiadomień, automatyzacja wykonania kodu źródłowego oraz jego ocena. Głównym powodem do powstania projektu jest niewielka ilość aplikacji automatyzujących wyżej wymienione procesy.

The paper presents the server part of the system of algorithmic tasks automatic evaluation. The basic requirements for the application are: groups management, authentication and authorization system, notification system, automation of source code execution and its evaluation. The main reason for the project is a small number of applications that automate the above-mentioned processes.

Spis treści

Rozdział 1.

Wprowadzenie

Niniejszy rozdział przedstawia genezę powstania problemu i opis środowiska, w jakim problem został zidentyfikowany. Na koniec zostanie porównane kilka istniejących rozwiązań oraz zostaną omówione najważniejsze cechy jakimi się charakteryzują.

1.1. Geneza powstania problemu

Rozwój informatyki wraz z postępującą cyfryzacją spowodował wzrost zainteresowania programowaniem komputerów. Pod pojęciem programowanie komputerów rozumiana jest ogromna liczba procesów do wytwarzania kodu źródłowego. Tymi procesami są: projektowanie, tworzenie, testowanie oraz przyszłe utrzymywanie napisanego już kodu. Osoba, która zajmuje się wytwarzaniem oprogramowania nazywana jest programistą.

Wraz z rozwojem technologii powstały nowe maszyny i urządzenia, które sterowane są oprogramowaniem zaprojektowanym przez programistów. Tym samym rozwinięto języki programowania ułatwiając wytwarzanie czystego i zrozumiałego kodu dla innych osób. Obecnie istnieje wiele języków programowania, które umożliwiają nam rozwiązać różne problemy. Ze względu na cel i przeznaczenie programów, programiści podzielili się na trzy główne grupy. Pierwszą grupą są programiści aplikacji, którzy zajmują się wytwarzaniem programów komputerowych realizujące określone zadania. Drugą grupą są programiści wytwarzający oprogramowanie działające w środowisku WWW. Ostatnią grupą są programiści systemowi, którzy wytwarzają oprogramowania do celów zarządzania pracą sprzętu komputerowego.

„Profesjonalista zna kilka języków programowania i w razie potrzeby uczy się nowych. Nie ma »jedynego najlepszego języka« dla wszystkich ludzi i dziedzin." Bjarne Stroustrup. Osoba w zawodzie programista musi się nieustannie uczyć i rozwijać.

1.1.1. Kursy

Oprócz zajęć przeprowadzanych w szkołach czy na studiach organizowane są również różne kursy i warsztaty. Zajęcia są prowadzone na różnym stopniu trudności i tematyki. Można wyróżnić dwa główne rodzaje zajęć. Pierwszy rodzaj zajęć zajmuje się nauką języków programowania oraz nowych technologii. Drugi rodzaj zajęć polega na poznawanie struktur danych oraz rozwiązywaniu problemów algorytmicznych. Częścią wspólną, która łączy wszystkie zajęcia jest podział członków grupy na: uczestnika, który uczy się oraz osobę prowadzącą, która przekazuje wiedzę. Dodatkowo wyróżnia się osobę sprawdzającą rozwiązania zadań programistycznych.

1.1.2. Zawody

Inną formą rozwoju umiejętności programowania są zawody oraz maratony programistyczne organizowane dla różnych grup wiekowych oraz osobom z różnym poziomem doświadczenia. W Polsce przeważnie organizowane przez Ministerstwo Edukacji dla młodych ludzi zaczynających przygodę z programowaniem, czyli uczniów szkół średnich i studentów. Coraz większą popularność mają także spotkania programistyczne, zwane coding dojo. Coding dojo jest spotkaniem, na którym grupa ludzi rozwiązuje zadania programistyczne. Głównym jego celem jest wymiana doświadczenia i szlifowanie przydatnych na co dzień umiejętności [1].

1.1.3. Nauka we własnym zakresie

Jednak najczęściej używaną formą rozwoju jest nauka we własnym zakresie. Źródłem informacji mogą być poradniki, serwisy czy też książki, gdzie programiści rozwiązują opisane zadania oraz problemy.

1.2. Ewaluacja zadań programistycznych

W poprzedniej podrozdziale zostały wymienione różne rodzaje zajęć, które składały się na kursy, zawody programistyczne czy nauce we własnym zakresie. Przeważnie wszystkie wymienione formy zajęć sprowadzają się do napisania kodu źródłowego, który rozwiązuje problem lub zadanie programistyczne. W celu sprawdzenia poprawności rozwiązania, kod źródłowy dostarczony przez użytkownika jest uruchamiany. Sprawdzeniem rozwiązania zajmuje się specjalnie wydzielona osoba lub sam prowadzący. W rzeczywistości sprawdzanie takich rozwiązań jest bardzo kłopotliwe, mając na uwadze bezpieczeństwo i sprawiedliwość w ocenie. Dostarczony kod może nie rozwiązywać problemu, a przeciwnie, zaatakować urządzenie na którym jest wykonywany kod.

Głównym problemem sprawdzania poprawności rozwiązań jest uruchomienie programu w imieniu użytkownika. Osoba sprawdzająca nie może zaufać kodowi źródłowemu, ponieważ nie ma pewności że kod dostarczony przez użytkownika rozwiązuje tylko dany problem. Istnieje obawa, że program może próbować zniszczyć lub przejąć kontrolę nad urządzeniem na którym jest uruchomiony.

Jednym z rozwiązań tego problemu jest statyczna analiza kodu przed jego wykonaniem oraz odrzucenie tych rozwiązań, które posiadają niedozwolone instrukcje. Rozwiązanie jest trudne do wdrożenia, ponieważ odrzucenie wszystkich rozwiązań, które powodują niepożądane zdarzenia jest niemożliwe. Tym bardziej, gdy zadanie algorytmiczne jest dostępne do rozwiązania w kilku językach programowania. W takim przypadku wsparcie wielu języków może okazać się uciążliwe i podatne na obejścia dla osoby atakującej.

Jedynym rozsądnym wyjściem jest odizolowanie uruchomionego programu od reszty zasobów i procesów. Uruchomiony program nie powinien mieć dostępu do plików, portów oraz sieci w celu ochrony wrażliwych danych na maszynie, w której jest uruchomiony. W tak przygotowanym środowisku jedyną podatnością na atak pozostaje wydajność naszego systemu. W przypadku automatyzacji tych procesów, należy posiadać kontrolę nad uruchomionymi zadaniami. Uruchomiony szkodliwy kod bez nadzoru może za bardzo obciążyć system, tym samym źle wpłynąć na inne rozwiązania. Dodatkowym zabezpieczeniem jest więc ograniczenie czasu wykonywania kodu [2].

1.3. Przegląd i porównanie istniejących rozwiązań

Osoby sprawdzające na wiele sposobów radzą sobie problemami wymienionymi w poprzedniej podrozdziale. Jedną z popularnych metod jest wykonanie kodu na komputerze osoby oddającej rozwiązanie zadania lub specjalnym wcześniej przygotowanym i skonfigurowanym środowisku. Te rozwiązanie ma swoje zalety jak i wady. Zaletą takiej techniki jest to, że osoba sprawdzająca nie jest narażona na atak. W prawidłowo przygotowanym środowisku dostarczony program może wykonać się bez zakłóceń przez inne procesy, tym samym nie zaburzając wyniku programu oraz szybkości wykonania. Główną wadą jest ilość zasobów oraz czasu poświęconego na przygotowanie i skonfigurowanie środowisk. W celu poprawnego skonfigurowania środowiska należy zwrócić uwagę na zainstalowanie odpowiednich narzędzi. Jeśli zadanie programistyczne jest dostępne do rozwiązania w kilku językach programowania to system powinien być do tego przygotowany wspierając dany język oraz popularne technologie w jakich dany problem można rozwiązać.

Rozwiązaniem, który uwzględnia wszystkie wymienione założenia jest oprogramowanie Docker. Narzędzie pozwalające na umieszczenie programu oraz potrzebnych technologii w przenośnym wirtualnym kontenerze, który można uruchomić na każdym urządzeniu z systemem Linux. W przypadku systemów nie wspierających

narzędzia Docker, symulowany jest system Linux w którym uruchamiane są kontenery. W celu zautomatyzowania powtarzających się procesów można utworzyć nowy obraz systemu, który będzie zawierał paczkę języków programowania oraz wspieranych narzędzi. Tym samym oszczędzi czas na ponowne przygotowanie kontenerów. Najważniejszą zaletą takiego rozwiązania jest możliwość utworzenia kontenera w razie potrzeby i zniszczenia po zakończeniu pracy. W przypadku gdy zostanie uruchomiony złośliwy kod to szkody pozostaną tylko w kontenerze, który następnie zostanie zniszczony [3].

Dodatkowym atutem może być możliwość uruchomienia skryptu, który zautomatyzuje sprawdzanie rozwiązań dostarczając dane wejściowe i porównując wynik programu z danymi wyjściowymi. W ramach zajęć, oprócz samego wykonania i sprawdzenia zadania, prowadzący powinien zidentyfikować i zweryfikować osobę oddającą rozwiązanie, w celu wykrycia oszustw. Ważną i istotną rzeczą jest także szybki dostęp do rozwiązań dostarczonych przez pozostałych członków w celu wykrycia popełnionego plagiatu.

W tym przypadku samo oprogramowanie Docker nie wystarczy. Istnieje wiele czynności, które są powtarzane i możliwe do zautomatyzowania tak aby nie zajmowało zbyt wiele czasu. Do tego celu powstały serwisy umożliwiające definiowanie zadań i pozwalające na automatyczne sprawdzanie rozwiązań. Obecnie na rynku jest niewielka ilość aplikacji czy serwisów wspomagających ewaluacje zadań algorytmicznych. Większość twórców dostępnych aplikacji nie dzieli się pomysłem na rozwiązanie podobnych problemów. Wysyłając rozwiązanie do takich serwisów należy pamiętać, że nie mamy pewności jak są przechowywane nasze dane, a tym bardziej gdzie jest wykonywany nasz kod źródłowy. Organizatorzy zawodów czy prowadzący zajęcia nie mogą pozwolić na wyciek takich danych. Troska o bezpieczeństwo naszego kodu oraz polityka serwisów udostępniających podobne rozwiązania to główne powody zaprojektowania własnego systemu automatycznej ewaluacji zadań algorytmicznych.

Rozdział 2.

Część dla użytkownika

Niniejszy rozdział przedstawia wymagania stawiane części serwerowej systemowi automatycznej ewaluacji zadań algorytmicznych. Omawiane są kolejno: cel i główne założenia projektu, wymagania funkcjonalne i niefunkcjonalne użytkownika oraz systemowe. Na końcu zostały przedstawione podstawowe przypadki użycia aplikacji.

2.1. Cel i główne założenia projektu

Celem niniejszej pracy jest opracowanie systemu automatycznej ewaluacji zadań algorytmicznych wraz z zautomatyzowaniem większości czynności jakie wykonywane są podczas organizacji kursów, warsztatów a także zawodów programistycznych. Proces organizowania spotkań wspomagających naukę programowania składa się z części: zarządzanie grupami uczestników, listami zadań w tym również testami oraz wysłanymi rozwiązaniami. Głównym powodem stworzenia systemu ewaluacji zadań algorytmicznych jest niewielka ilość aplikacji, które dają możliwość przeprowadzenia ewaluacji zadań w sposób bezpieczny i dostępny dla dowolnego użytkownika.

Zakres pracy obejmuje następujące zagadnienia:

- zaprojektowanie aplikacji sieciowej z interfejsem do łatwej komunikacji i zarządzaniem systemem;
- zaprojektowanie struktury bazy danych;
- działanie systemu oparte o chmurę;
- zabezpieczenie systemu przed atakami;
- uwierzytelnianie w oparciu o standard OAuth2;
- działanie systemu w oparciu o aplikację bezstanową;

- uruchamianie niezaufanego kodu w odrębnym środowisku;
- wsparcie dla wielu języków programowania;
- powiadomienia o zmianach w systemie;

2.2. Funkcjonalność projektu

Podrozdział przedstawia funkcjonalność projektu, która została podzielona na mniejsze części składowe: uwierzytelnianie i autoryzacja, zarządzanie grupami zajęciowymi, system powiadomień użytkowników oraz logi aplikacji. Ponadto zostaną omówione uprawnienia oraz role użytkowników w obrębie systemu oraz grupy.

2.2.1. Uwierzytelnianie i autoryzacja

Uwierzytelnianie opiera się na protokole autoryzacyjnym OAuth2 z użyciem którego użytkownik bezproblemowo zaloguje się do systemu za pomocą innego dostawcy np. Google lub serwera z konkretną grupą docelową (usos, github). Informacje, które aplikacja ściąga od dostawcy są to: e-mail, imię, nazwisko oraz link do zdjęcia profilowego. Jeśli serwer autoryzacji nie dostarczy link do zdjęcia profilowego to zostanie użyte domyślne zdjęcie z serwera. W celu uzyskania bezstanowości serwera oraz zapewnienia bezpiecznej i szybkiej autoryzacji skorzystano z standardu Json Web Token, za pomocą którego tokeny są generowane i przechowywane w sposób bezpieczny po stronie klienta. Aplikacja definiuje trzy rodzaje tokenów:

- token dostępu, który pozwala na dostęp do zasobów chronionych;
- token odświeżania, który pozwala na odnowienie tokenu dostępu;
- token tymczasowy, który pozwala na wygenerowanie nowych tokenów dostępu i odświeżania;

2.2.2. Uprawnienia i role użytkownika w systemie

W celu określenia dostępu do zasobów na serwerze zostały stworzone uprawnienia. Uprawnienia są przydzielone do ról użytkownika. Każda rola charakteryzuje się innym zbiorem uprawnień. Podczas pierwszego logowania w aplikacji, osoba domyślnie otrzymuje rolę *użytkownik* systemu. Możliwe jest posiadanie tylko jednej roli w obrębie serwisu. W Tablicy 2.1 zostały zaprezentowane wszystkie dostępne role w odniesieniu do systemu.

Tablica 2.1: Role użytkowników w serwisie

Identyfikator	**Nazwa roli**
0	administrator (ang. administrator)
1	twórca (ang. creator)
2	użytkownik (ang. user)

Uprawnienia zostały podzielone ze względu na zasięg. Wyróżnia się uprawnienia w zasięgu grupy oraz aplikacji.

Tablica 2.2: Uprawnienia użytkowników

Id	**Opis**	**Zasięg**
0	może wyświetlać grupy (ang. can view group)	aplikacja
1	może tworzyć grupę (ang. can create group)	aplikacja
2	może usuwać grupę (ang. can delete group)	grupa
3	może edytować grupę (ang. can edit group)	grupa
10	może wyświetlać zadania (ang. can view task)	grupa
11	może tworzyć zadania (ang. can create task)	grupa
12	może edytować zadania (ang. can edit task)	grupa
13	może usuwać zadania (ang. can delete task)	grupa
14	może aktywować zadania (ang. can activate task)	grupa
15	może dezaktywować zadania (ang. can deactivate task)	grupa
20	może wyświetlać członków (ang. can view member)	grupa
21	może zapraszać użytkowników (ang. can invite member)	grupa
22	może usuwać członka (ang. can delete member)	grupa
30	może wyświetlać wiadomości (ang. can view news)	grupa
31	może tworzyć wiadomości (ang. can create news)	grupa
32	może usuwać wiadomości (ang. can delete news)	grupa
33	może edytować wiadomości (ang. can edit news)	grupa
40	może wyświetlać statystyki (ang. can view statistics)	grupa
50	może wysyłać rozwiązania (ang. can send solution)	grupa
60	może wyświetlać testy (ang. can view test)	grupa
61	może tworzyć testy (ang. can create test)	grupa
62	może edytować testy (ang. can edit test)	grupa

Administrator Osoba, która posiada role *administrator* ma pełne uprawnienia do aplikacji oraz do wszystkich utworzonych grup.

Twórca Uprawnienia dla osoby, która posiada role *twórca*:

- może wyświetlać grupy;
- może tworzyć grupę;

Użytkownik Uprawnienia dla osoby, która posiada role *użytkownik*:

- może wyświetlać grupy;

2.2.3. Zarządzanie grupami zajęciowymi

Poniżej została dokładnie omówiona funkcjonalność zarządzania grupami zajęciowymi. W skład funkcjonalności zarządzania grupami zajęciowymi wchodzą kolejno: grupa i uczestnicy, uprawniania oraz role w grupie, zadania, języki programowania, rozwiązania, testy oraz wyniki testów.

Grupa i uczestnicy

Grupa zajęciowa składa się z użytkowników systemu, którzy w obrębie grupy zwani są uczestnikami. Tylko osoba z rolą *twórcy* jest w stanie utworzyć grupę. Po utworzeniu grupy taka osoba staje się właścicielem, która posiada pełne uprawnienia do grupy. Rola *właściciela* przysługuje użytkownikowi, któremu założył grupę. Właściciel grupy jest tylko jeden i nie można go zmienić. W Tablicy 2.3 zostały zaprezentowane wszystkie dostępne role w zasięgu grupy.

Tablica 2.3: Role uczestników w grupie

Identyfikator	**Nazwa roli**
0	właściciel (ang. owner)
1	administrator (ang. administrator)
2	użytkownik (ang. user)

Uprawnienia a rola w grupie

Uczestnik, który posiada role *administrator* grupy, podobnie jak *właściciel*, ma pełne uprawnienia do grupy. Dodatkowo użytkownik oraz administrator grupy ma możliwość wypisania się z grupy. Osoby posiadające rolę *administratora grupy* lub *właściciela*, dalej będą zwani osobami, które zarządzają grupą.

Uprawnienia dla osoby, która posiada role *użytkownika* grupy:

- może wysyłać rozwiązania;
- może wyświetlać zadania;

- może wyświetlać wiadomości;

Zadanie

Zadanie jest to problem programistyczny zdefiniowany w konkretnej grupie przez uczestnika grupy.

Zdefiniować zadanie może tylko osoba zarządzająca daną grupą lub *administrator* serwisu. W każdym rodzaju zadania obowiązkowe jest określenie daty zakończenia zadania. Data zakończenia zadania musi być datą przyszłą od momentu definiowania zadania. Tym samym zadanie może przyjmować jeden ze stanów: aktywne lub nieaktywne. Zadanie jest aktywne w przypadku gdy data zakończenia zadania jest przyszła. Zadanie jest nieaktywne, jeśli data zakończenia jest przeszła lub osoba z uprawnieniami edycji zadania osobiście zamknęła zadanie. W celu ponownej aktywacji, zadanie powinno posiadać co najmniej jeden test.

Możliwe jest wybranie języka programowania, w którym uczestnicy będą wysyłali rozwiązywania. W przypadku edycji zadania nie jest możliwe usunięcie języka programowania, w którym istnieje już rozwiązanie.

Istnieje wiele schematów oraz metryk sprawdzających poprawność rozwiązania danego zadania. W serwisie wyróżnia się trzy główne typy zadań, w których rozwiązanie zadania sprawdza się różnymi sposobami.

Zadanie poprawnościowe W zadaniu poprawnościowym uczestnik musi napisać kod źródłowy, który przy odpowiednich wprowadzonych danych wejściowych zwróci wynik rozwiązujący dany problem. System wymaga, aby przy wprowadzonych danych wejściowych istniała tylko jedna poprawna odpowiedź. W takim przypadku sprawdzenie poprawności rozwiązania polega na porównaniu identycznościowym oczekiwanego wyniku z wynikiem zwróconym przez program.

Zadanie wydajnościowe Zadania wydajnościowe polegają na sprawdzeniu jakości napisanego kodu. Oprócz poprawności, brany jest pod uwagę czas wykonania programu. Na podstawie tych wartości jest określane czy rozwiązanie spełniło dany test i rozwiązuje zdefiniowany problem.

Zadanie typu code golf Zadanie typu code golf jest zadaniem rekreacyjnym. Głównym celem jest napisanie możliwie jak najkrótszego kodu źródłowego, który rozwiązuje zdefiniowany problem.

Tablica 2.4: Typy zadań

Identyfikator	Rodzaj zadania
0	poprawnościowe (ang. performance)
1	wydajnościowe (ang. validity)
2	code golf (ang. code golf)

Język programowania

System aktualnie obsługuje kilka języków programowania. Poniżej w tabeli zostały zaprezentowane wszystkie dostępne języki programowania.

Tablica 2.5: Dostępne języki programowania

Identyfikator	Nazwa języka programowania
0	python
1	javascript
2	c++
3	java
4	bash
5	c

Rozwiązanie

Rozwiązaniem nazywany jest kod źródłowy wysłany przez uczestnika grupy. Kod źródłowy musi być napisany w dostępnym języku programowania. Domyślnym algorytmem do sprawdzenia poprawności napisanego kodu jest porównanie identycznościowe. Porównanie identycznościowe polega na porównaniu danych wyjściowych programu z danymi oczekiwanymi, które zostały zdefiniowane przez użytkownika tworzącego test.

W celu zapewnienia bezpieczeństwa serwera przed atakami, które mogą spowolnić działanie aplikacji, została wprowadzona blokada na wysyłanie rozwiązań. Użytkownik może wysłać kod źródłowy w odstępie nie mniejszym niż jedna minuta od ostatniego poprawnego wysłanego rozwiązania. Ostatnim rozwiązaniem może być kod źródłowy wysłany w innym zadaniu lub grupie. Po wysłaniu, rozwiązanie może przyjąć jeden z statusów określonych w Tablicy 2.6.

Tablica 2.6: Statusy rozwiązań

Identyfikator	**Status rozwiązania**
0	nowy (ang. new)
1	w oczekiwaniu (ang. pending)
2	kompletny (ang. completed)
3	anulowany (ang. canceled)

Testy

Testy są to zestawy punktowanych prób, którym poddaje się rozwiązanie zadania w celu sprawdzenia poprawności rozwiązania. Osoba posiadająca uprawnienia do tworzenia i edycji testów może zdefiniować nieograniczoną liczbę testów w zadaniu. Testy można podzielić na trzy typy: *publiczny*, *prywatny* oraz *ukryty*. Pierwszy typ testu zwany *publiczny*, widoczny jest dla wszystkich uczestników. Uczestnik może zobaczyć wynik, dane wejściowe oraz dane wyjściowe testu. Drugi typ testu zwany *prywatny*, widoczny jest dla każdego uczestnika oraz umożliwia podgląd tylko wyniku testu. Ostatni typ testu zwany *ukryty*, widoczny jest tylko dla osób zarządzających grupą. Test ukryty staje się automatycznie widoczny dla uczestników, kiedy data wygaśnięcia jest przeszła.

Testy można dodawać i usuwać pod warunkiem, że zadanie jest nieaktywne oraz nie istnieje rozwiązanie.

Tablica 2.7: Typy testów

Identyfikator	**Typ testu**
0	publiczny (ang. public)
1	prywatny (ang. private)
2	ukryty (ang. hidden)

2.2.4. Wynik testu

Wynikiem testu jest czas rozwiązania zadania oraz jeden z komunikatów umieszczony w Tablicy 2.8. W przypadku zadania typu code golf, zwracana jest także liczba znaków. W celu zabezpieczenia maszyny na której zostanie uruchomiony kod źródłowy z danym testem zostało nałożone ograniczenie na czas wykonania kodu źródłowego. Maksymalny czas pracy uruchomionego programu wynosi jedna minuta.

Tablica 2.8: Stany wyników testów

Identyfikator	**Stan testu**
0	przekroczony limit czasu (ang. time limit exceeded)
1	błąd wykonania (ang. runtime error)
2	zła odpowiedź (ang. wrong answer)
3	ok (ang. ok)
4	przekroczony limit kodu źródłowego (ang. code source limit exceeded)

2.2.5. System powiadomień

Jeśli zostanie wprowadzona zmiana na koncie użytkownika to dany użytkownik zostanie o tym poinformowany poprzez wiadomość e-mail. Czynnością wprowadzającą zmianę na koncie użytkownika są:

- dodanie do grupy;
- usunięcie z grupy;
- zmiana roli w obrębie systemu oraz grupy;

W przypadku gdy zaproszona osoba poprzez e-mail nie istnieje w systemie to zostanie wysłana wiadomość z prośbą o dołączenie do serwisu.

2.2.6. Logi

Aplikacja umożliwia użytkownikowi wykonanie czterech głównych operacji na zasobach: usuwanie, dodawanie, edycja oraz podgląd. Operacjami wprowadzającymi zmiany w serwisie są: usuwanie, dodawanie oraz edycja. Jeśli użytkownik skorzysta z podanych operacji, to wykonana czynność zostanie zarejestrowana w serwisie wraz z informacją o użytkowniku oraz datą wykonania.

2.3. Przypadki użycia

Niniejszy podrozdział przedstawia najważniejsze przypadki użycia pozwalające na określenie, w jaki sposób użytkownik będzie mógł korzystać z systemu.

Przypadek użycia 2.3..1.
Cel: Logowanie do serwisu
Aktor: Osoba nie zalogowana w serwisie
Przebieg:

1. Użytkownik kieruje żądanie do serwera zasobów wybierając w ścieżce adresowej serwer autoryzujący.
2. Serwer zasobów przekierowuje użytkownika do formularza logowania na serwerze autoryzacyjnym.
3. Użytkownik uwierzytelnia się w serwisie autoryzacyjnym.
4. Użytkownik zgadza się na wysyłanie danych o sobie do serwera zasobów przez serwer autoryzujący.
5. Serwer zasobów otrzymuje odpowiedź i przekierowuje użytkownika na stronę części klienckiej.

Przypadek użycia 2.3..2.
Cel: Utworzenie grupy
Aktor: Uczestnik zarządzający grupą lub *administratora* serwisu
Przebieg:

1. Użytkownik wypełnia formularz do tworzenia grupy
2. Użytkownik wysyła żądanie do serwisu
3. Serwer zwraca model reprezentujący stworzoną grupę

Przypadek użycia 2.3..3.
Cel: Utworzenie zadania
Aktor: Uczestnik zarządzający grupą lub *administrator* serwisu
Warunki początkowe:

1. Istnieje grupa do której zostanie przypisane zadanie

Przebieg:

1. Użytkownik wypełnia model do tworzenia zadania
   (a) Sprecyzowanie języków programowania
   (b) Ustawienie terminu oddania zadania
2. Użytkownik wysyła żądanie do serwisu
3. Serwer zwraca model reprezentujący stworzone zadanie

Przypadek użycia 2.3..4.
Cel: Dodanie testu do zadania
Aktor: Uczestnik zarządzający grupą lub *administrator* serwisu
Warunki początkowe:

1. Istnieją zasoby określone w ścieżce adresu

2. Nie istnieje rozwiązanie w danym zadaniu

Przebieg:

1. Użytkownik wypełnia model do tworzenia testów
   (a) Możliwość ustawienia czasu na wykonanie zadania
   (b) Możliwość ustawienia maksymalnej ilości znaków dla kodu źródłowego
2. Użytkownik wysyła żądanie do serwisu
3. Serwer zwraca model reprezentujący stworzony test

Rozszerzenia:

1. Zadanie miało status aktywny
   (a) Użytkownik dezaktywuje zadanie
   (b) Ponowne wysłanie żądanie o utworzenie testu

Przypadek użycia 2.3..5.
Cel: Dodanie użytkownika do grupy
Aktor: Uczestnik zarządzający grupą lub *administrator* serwisu
Przebieg:

1. Użytkownik wypełnia formularz do zapisywania użytkowników do grupy
2. Aplikacja wysyła powiadomienie e-mail na adres w formularzu o dołączeniu do grupy
3. Serwer zwraca model reprezentujący zapisanego uczestnika w grupie

Rozszerzenia:

1. Użytkownik o podanym e-mailu istnieje w grupie
   (a) Aplikacja zwraca komunikat o niepowodzeniu

Przypadek użycia 2.3..6.
Cel: Wysłanie rozwiązania
Aktor: Uczestnik grupy
Przebieg:

1. Użytkownik wypełnia formularz do wysyłania rozwiązania
2. Aplikacja zapisuje rozwiązanie i uruchamia testy
3. Serwer zwraca model reprezentujący zapisane rozwiązanie

Rozszerzenia:

1. Od ostatniego wysłanego rozwiązania nie minęła minuta
   (a) Aplikacja zwraca komunikat o niepowodzeniu

Przypadek użycia 2.3..7.
Cel: Podgląd wyników wysłanego rozwiązania
Aktor: Uczestnik grupy
Przebieg:

1. Użytkownik wysyła żądanie o model reprezentujący rozwiązanie
2. Serwer zwraca model wraz z ukończonymi wynikami testów oraz statusem rozwiązania

Rozszerzenia:

1. Uczestnik chce mieć wynik konkretnego testu
   (a) Użytkownik wysyła żądanie o model reprezentujący wynik konkretnego testu
   (b) Serwer zwraca wynik testu lub komunikat oznaczający, że test jest w trakcie wykonywania

Przypadek użycia 2.3..8.
Cel: Zmiana roli użytkownika w serwisie
Aktor: Administrator serwisu
Przebieg:

1. Użytkownik wypełnia formularz do zmiany roli w serwisie
2. Aplikacja wysyła powiadomienie e-mail o zmianie roli na adres danego użytkownika
3. Serwer zwraca model reprezentujący użytkownika w serwisie

Rozdział 3.

Część dla programisty

Niniejszy rozdział przedstawia implementację części serwerowej systemu automatycznej ewaluacji zadań algorytmicznych. Omówione są kolejno: modele danych, moduły aplikacji oraz technologie i narzędzia, które zostały wykorzystane do osiągnięcia celu. Na koniec rozdziału umieszczono przegląd interfejsu programistycznego aplikacji oraz opis wdrożenia oprogramowania.

3.1. Model danych systemu

Wszystkie modele do odbierania oraz wysyłania danych zaprojektowane są zgodnie ze wzorcem budowniczy (ang. builder). Celem wzorca jest rozdzielenie procesu tworzenia obiektu na kilka mniejszych etapów. Taki sposób tworzenia zapewnia brak zmiany stanu stworzonego obiektu [4].

Modele są reprezentowane w postaci JavaScript Object Notation (JSON). Jest to lekki i prosty format wymiany danych niezależny od konkretnego języka programowania. Typ MIME dla formatu JSON to *application/json*. Dane są reprezentowane jako tablica asocjacyjna, gdzie klucze i wartości są otoczone cudzysłowami. W przypadku gdy wartością jest liczba lub stała np. false, true, null to nie jest konieczny cudzysłów. Podany format pozwala na zagnieżdżenie obiektów oraz tablic [5].

W niniejszym podrozdziale zostały przedstawione oraz omówione modele użyte w celu reprezentowania danych systemu. Omówione zostaną kolejno modele danych:

- uwierzytelnianie i autoryzacja użytkownika;
- zarządzanie grupami zajęciowymi;
- komunikacja z narzędziem do wykonywania kodu źródłowego;

Na koniec zostanie przedstawiony model przechowywany w bazie danych.

3.1.1. Uwierzytelnianie i autoryzacja

Do autoryzacji użytkownika został zaprojektowany model, w którym polami są kolejno token dostępu oraz token odświeżania. Model jest zwracany w postaci JSON, w przypadku, gdy część kliencka zażąda odnowienie tokena dostępu lub wygenerowania nowych tokenów za pomocą tokenu tymczasowego.

```
{
  "accessToken": "...",
  "refreshToken": "..."
}
```

Listing 3..1: Model służący do autoryzacji w systemie

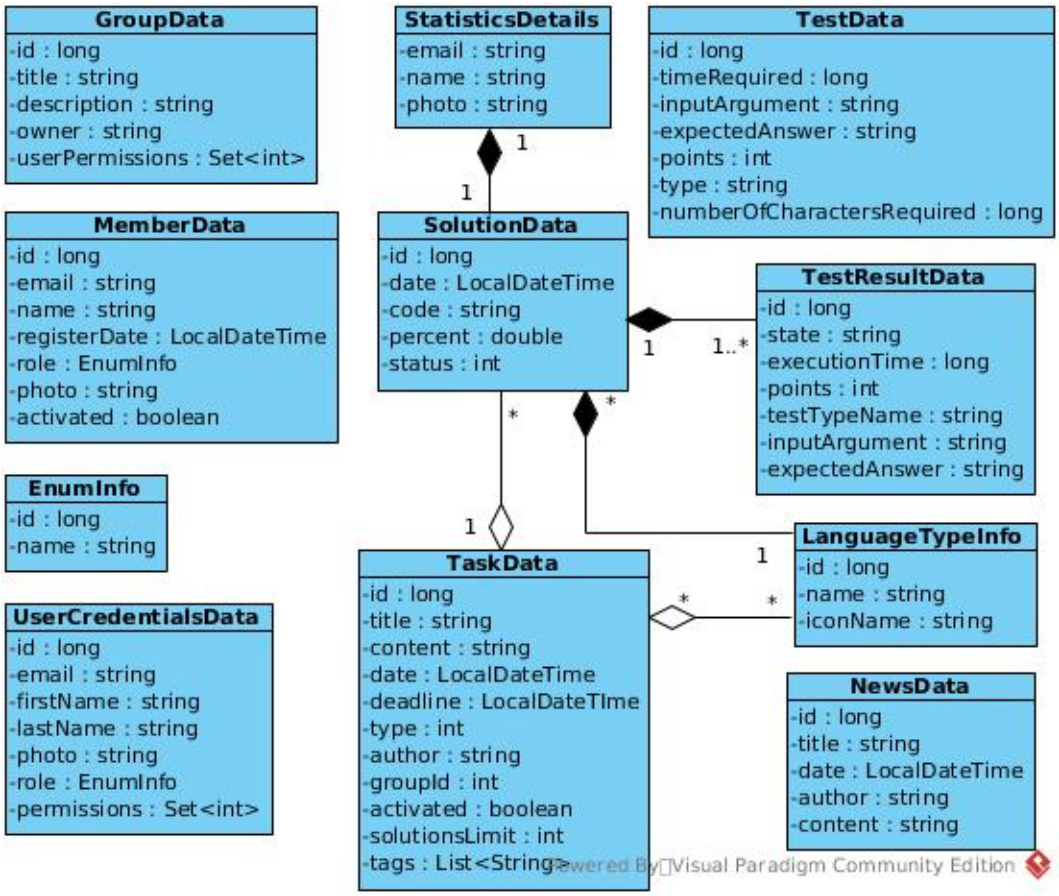
3.1.2. Zarządzanie grupami zajęciowymi

W celu komunikacji między częścią serwerową a częścią kliencką zostały zaprojektowane modele danych. Poniżej zostaną omówione modele do reprezentowania: użytkownika, grupy, testów, ogłoszeń publikowanych w grupie, wyników testów, szczegółowych statystyk, języków programowania oraz członków grupy.

Model reprezentujący dane zwracane

Modele reprezentuje dane systemu poza aplikacją:

- dane użytkownika;
- dane grupy;
- dane ogłoszenia w grupie;
- dane członka grupy;
- dane testu;
- dane wyniku testu;
- dane rozwiązania;
- dane zadania;
- szczegóły statystyk;
- informacje o języku programowania;



Rysunek 3.1: Diagram UML modeli danych

Model reprezentujący *informacje o języku programowania* przechowuje klasę ikonki (z biblioteki devicon [6]).

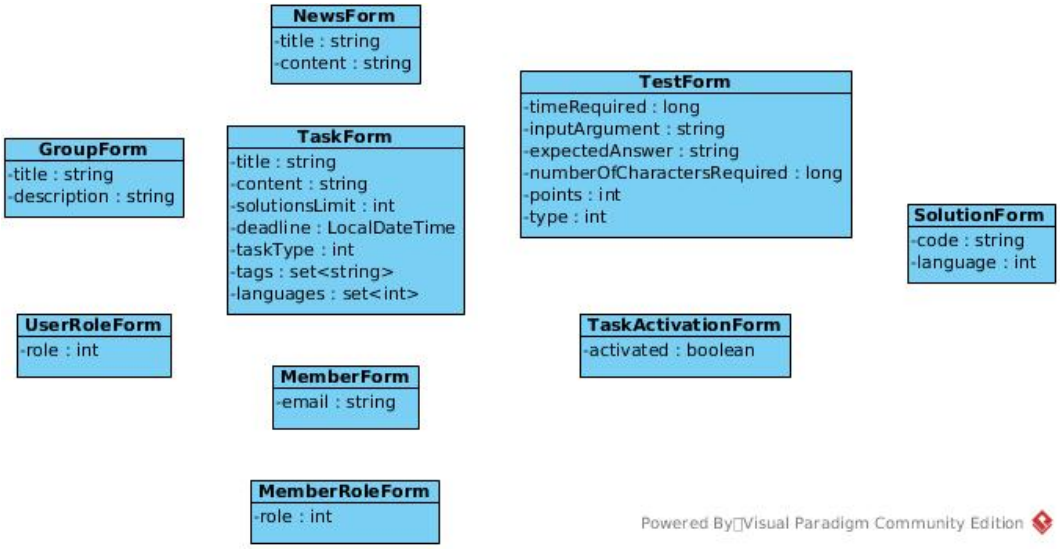
W przypadku zdjęcia w modelu *dane użytkownika* oraz *dane uczestnika grupy* wysyłany jest link dostarczony z serwera autoryzującego. Jeśli użytkownik nie posiada zdjęcia, to wysyłany jest link do domyślnej fotografii znajdującej się na serwerze na którym uruchomiona jest aplikacja.

Pole *nazwa* w *szczegóły statystyk* oraz *dane członka grupy* składa się z imienia i nazwiska. Jeśli użytkownik nie posiada imienia i nazwiska w aplikacji, to wartość pola jest pusta. Podobną wartość przyjmuje pole *owner* w modelu *dane grupy*, ale, jeśli użytkownik nie posiada imienia i nazwiska, to jest zwracany adres e-mail.

Model reprezentujący formularz

Modele służące do reprezentowania formularzy wysyłanych przy żądaniu utworzenia nowego zasobu:

- formularz grupy;
- formularz zapisu ogłoszenia w grupie;
- formularz zapisu użytkownika do grupy;
- formularz zadania;
- formularz rozwiązania;
- formularz testu;

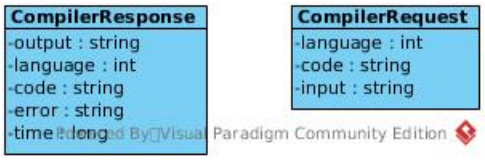


Rysunek 3.2: Diagram UML formularzy

Poza wymaganym czasem wykonania zadania oraz liczbą znaków w *formularzu testu* oraz limitem rozwiązań w *formularzu zadania* żadne pole ani nie może być puste, ani zawierać tylko białych znaków. Jeśli zadanie jest typu wydajnościowego to pole z wymaganym czasem wykonania zadania powinno przyjąć wartość dodatnią. Podobnie jak pole z wymaganą liczbą znaków w zadaniu typu *code golf*.

3.1.3. Komunikacja z narzędziem do wykonywania kodu

Aplikacja w celu ewaluacji zadania algorytmicznego wysyła żądanie do osobnej aplikacji, który wykona kod źródłowy i zwróci wynik. W celu komunikacji zostały zaprojektowane struktury danych dla wysyłania żądań oraz odsyłanej odpowiedzi zwrotnej.



Rysunek 3.3: Diagram UML modeli do komunikacji z compilebox

Model dla żądania wykonania kodu (*CompilerRequest*) zawiera kolejno: identyfikator języka programowania, kod źródłowy oraz treść wejścia dla programu. W ramach wykonania kodu źródłowego zwracany jest model (*CompilerResponse*) zawierający kolejno: wynik programu, język programowania, kod źródłowy, błąd wykonania (opcjonalny) oraz czas wykonania zadania w milisekundach.

3.1.4. Baza danych

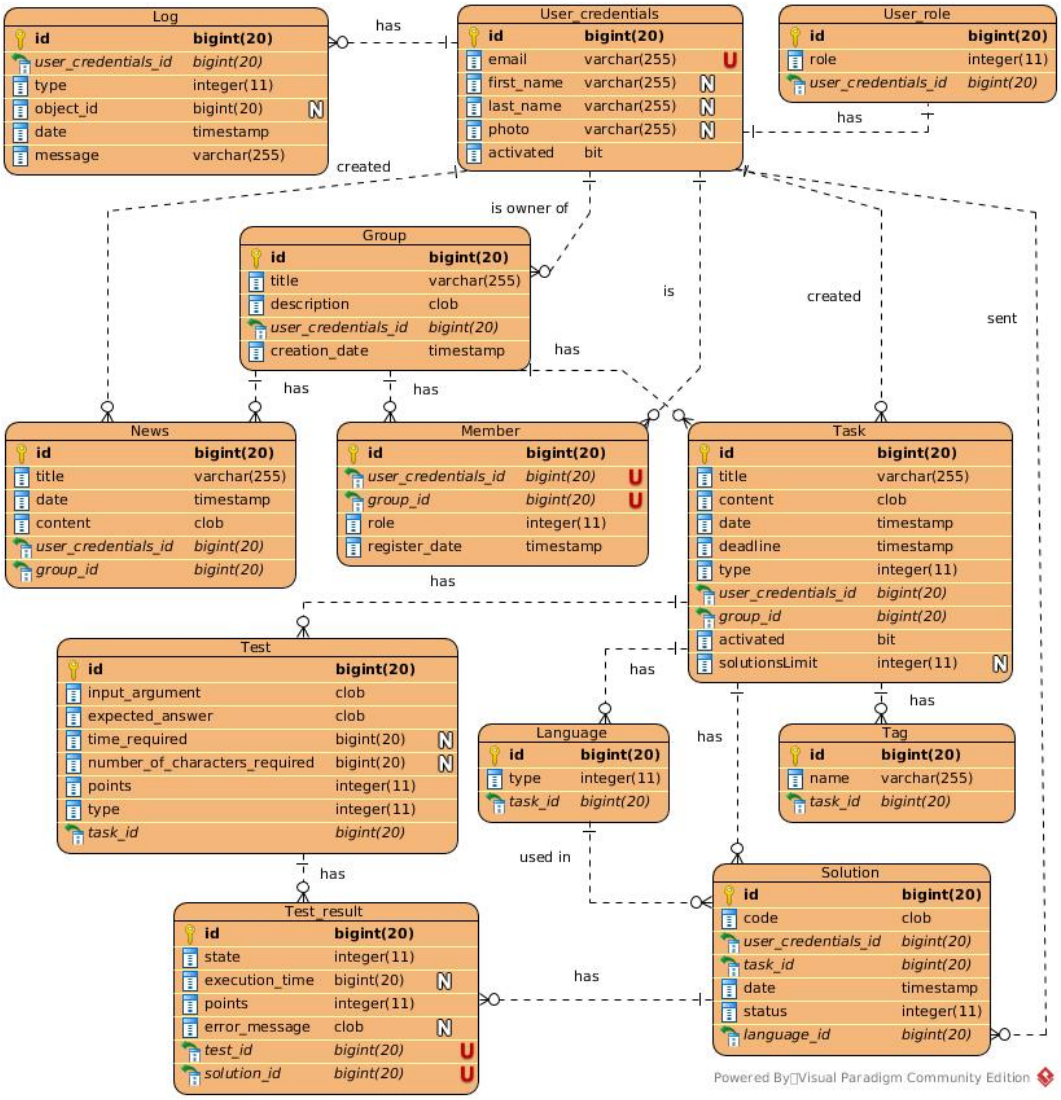
Część bazodanowa zajmuje się przechowywaniem istotnych danych systemu:

- zarejestrowanych użytkowników i ich ról w systemie;
- stanu grupy zajęciowej;
- zadań algorytmicznych;
- testów przydzielonych do zadań;
- rozwiązania zadania algorytmicznego oraz wynik testu;

Z punktu widzenia uczestników grup, dostęp do większości encji mają wszyscy użytkownicy oprócz encji *Test* oraz *Log*. Dane z tabeli *Test* są dostępne tylko dla użytkowników z rolą *administrator* systemu lub osób zarządzających grupą. Wszystkie encje są identyfikowane w systemie za pomocą pola *id*. W celu uniknięcia konfliktu nazw tabel z zastrzeżonymi nazwami w bazie danych, każda tabela posiada przedrostek *„app_”*.

3.2. Moduły aplikacji

Ze względu na funkcjonalność systemu, projekt został podzielony na moduły. Omawiane są kolejno moduły: uwierzytelnianie i autoryzacja, ewaluacja zadań, zarządzanie grupami zajęciowymi oraz system powiadomień.



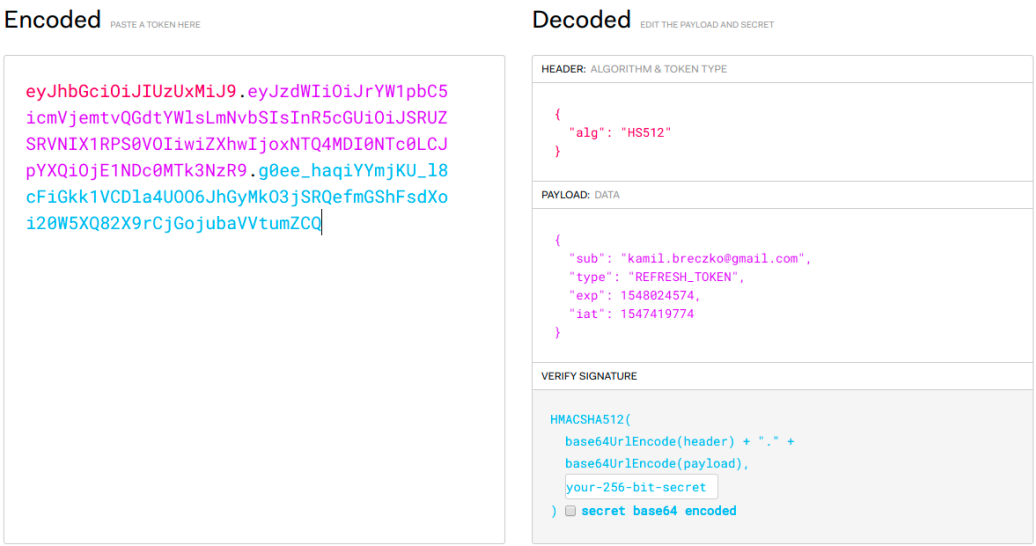
Rysunek 3.4: Diagram ERD bazy danych

3.2.1. Uwierzytelnianie i autoryzacja

W celu poprawy bezpieczeństwa serwisu, aplikacja korzysta z standardu OAuth2 oraz JSON Web Token (JWT).

Standard OAuth2 został użyty do uwierzytelnienia użytkownika w serwisie. Serwerem autoryzującym jest platforma Google, na której została zarejestrowana aplikacja. Za pomocą OAuth2 odpowiedzialność za przechowywanie i bezpieczeństwo haseł przechodzi na serwer autoryzujący. Tym samym użytkownik nie jest zmuszony do zapamiętywania kolejnego hasła [7].

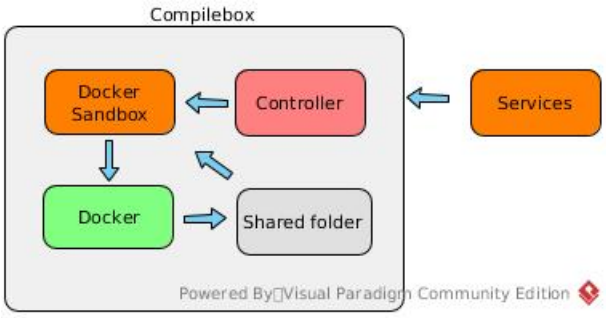
Do autoryzacji użytkownika skorzystano z tokenów i standardu JSON Web Token (JWT). JWT to rodzaj tokenu przechowywany po stronie klienta. Token jest generowany i szyfrowany za pomocą klucza po stronie serwera. Tylko serwer posiadający klucz może zweryfikować jego poprawność. Token w standardzie JWT składa się z trzech części, zakodowanych za pomocą kodowania base64: nagłówek, ładunek i podpis [8].



Rysunek 3.5: Wygenerowany token wraz z podglądem na zakodowane dane. Zrzut ekranu ze strony jwt.io

3.2.2. Ewaluacja zadań

Moduł ewaluacji zadań opiera się na narzędziu compilebox [9], który jest odrębnym serwerem zaprojektowanym w środowisku uruchomieniowym Nodejs. Kod źródłowy został poddany refaktoryzacji oraz dopasowany pod potrzeby projektu [10].



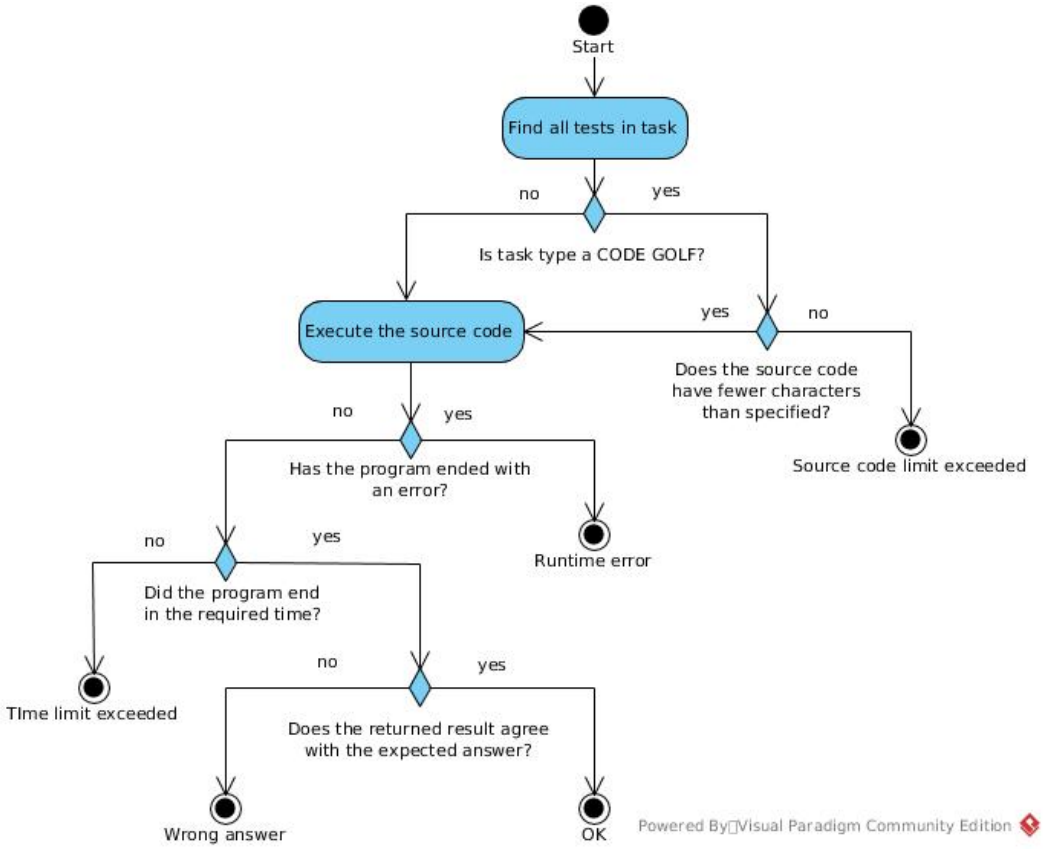
Rysunek 3.6: Schemat działania systemu ewaluacji zadań

Po wysłaniu kodu źródłowego przez użytkownika oraz zapisaniu rozwiązania w bazie danych aplikacja przygotowuje wszystkie testy, które należy poddać dane rozwiązanie. Jeśli jest to rozwiązanie dla zadania typu *code golf*, to sprawdzany jest warunek o spełnienie ograniczenia na ilość znaków w kodzie źródłowym. W przypadku gdy kod źródłowy jest za duży to wynik testu kończy się komunikatem *„Przekroczono limit kodu źródłowego”*, który następnie jest zapisywany w bazie. W przeciwnym przypadku, gdy kod źródłowy spełnia wymagania lub zadanie nie jest typu *code golf*, to wysyłane jest żądanie o wykonanie kodu źródłowego do compilebox. Proces oceny rozwiązanego zadania jest wykonywany asynchronicznie, w sposób nie blokujący. W tym czasie do użytkownika wysyłany jest model reprezentujący wysłane rozwiązanie.

Komunikacja polega na wysyłaniu żądania HTTP z odpowiednio zaprojektowanym modelem zawierającym potrzebne dane do wykonania kodu źródłowego. Dane są wysyłane za pomocą formatu JSON. Narzędzie otrzymuje żądanie z kodem źródłowym, identyfikatorem języka programowania obsługiwanym w compilebox oraz dane wejściowe do programu. Po otrzymaniu żądania tworzony jest kontener Docker z przygotowanego obrazu systemu. A następnie uruchamiany jest kod źródłowy z przekazanymi danymi wejściowymi. W zależności od typu języka programowania jest on na początku kompilowany albo od razu interpretowany. Po zakończeniu działania programu dane wyjściowe oraz strumień błędów są zapisywane do odpowiednich plików, które następnie są zwracane w odpowiedzi na żądanie. Na koniec kontener jest zatrzymywany i usuwany.

Po otrzymaniu odpowiedzi od compilebox, aplikacja sprawdza poprawność rozwiązania. Na początku sprawdzany jest strumień błędów. Jeśli istnieje zawartość to zwracany jest komunikat *„błąd wykonania”*. Następnie jeśli test posiada wymagany czas na wykonanie kodu, to porównywany jest czas wykonania kodu z wymaganym. W przypadku przekroczenia czasu lub program działał dłużej niż minutę to zwracany jest komunikat *„przekroczono limit czasu”*. Na koniec porównywany jest wynik rozwiązania z oczekiwanym. Po poprawnym rozwiązaniu zadania jest zwracany komunikat *„ok”* z pełną liczbą punktów. W każdym przypadku wynik jest zapisywany

w bazie danych.



Rysunek 3.7: Schemat oceny rozwiązań

Wprowadzone zmiany w compilebox

W celu poprawnego działania i komunikacji między aplikacjami wprowadzono następujące zmiany w compilebox:

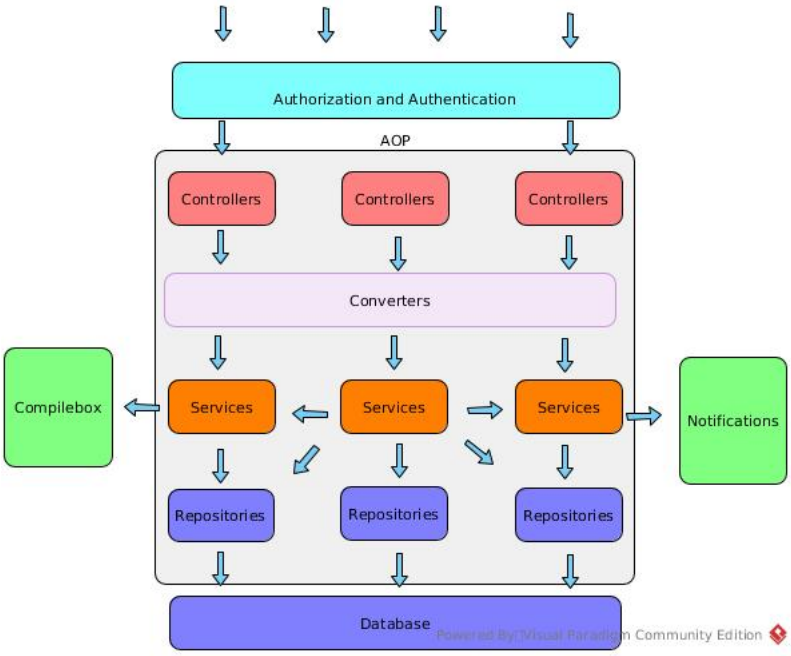
- utworzono nowy obraz kontenera zawierający:
  - wsparcie dla języków programowania: C, C++, Java 11, Python 2, JavaScript (Nodejs v8), Bash;
  - obraz systemu Ubuntu w najnowszej wersji (aktualnie Ubuntu 18.04 LTS);
- pomiar wykonywania programu w milisekundach;
- usunięcie zbędnego kodu np. dostęp poprzez stronę www;
- rozdzielenie wynik pomiaru czasu od zwracanych danych;
- nałożenie limitu czasu na uruchomiony kontener;

- nałożenie limitu na liczbę uruchomionych kontenerów w jednym czasie;
- usunięcie zależności (express-brute) do ograniczenia liczby połączeń z serwerem;
- uruchamianie kontenera bez dostępu do sieci;

Pomimo nałożenia limitu na liczbę uruchamianych kontenerów, narzędzie compilebox może nie poradzić sobie w momencie, kiedy wielu użytkowników wysyła rozwiązania. Aktualnie narzędzie zbiera żądania od aplikacji i dopiero po wykonaniu jednego zadania przechodzi do drugiego. Podana metoda może powodować zerwanie połączeń. Aplikacja poprawnie obsługuje podany przypadek, wtedy takie rozwiązania posiadają status *anulowany*.

3.2.3. Zarządzanie grupami zajęciowymi

Moduł do zarządzania grupami jest zaimplementowany według wzorca kontroler-serwis-repozytorium.



Rysunek 3.8: Schemat działania systemu do zarządzania grupami

Kontrolery zawierają wszystkie dostępne punkty końcowe pozwalające na zarządzanie grupami. Mając do dyspozycji konkretne punkty końcowe, w kontrolerach wykonywane są następujące czynności: sprawdzenie poprawności ścieżki, autoryzacja użytkownika, logowanie zdarzeń, zabezpieczenie przed zbyt częstym wysyłaniem rozwiązań. Wykonywane są za pomocą aspektów. Główną rolą kontrolerów jest zlecenie serwisom wykonania określonej akcji.

```
@RequestMapping(value = "/{groupId}/news/{newsId}", method = RequestMethod.DELETE)
@PreAuthorize("hasRole('ROLE_ADMIN') || @authorisationService.isGroupAdmin(#groupId, principal)")
@ResponseStatus(HttpStatus.NO_CONTENT)
@NewsPath
public void deleteNews(@PathVariable("groupId") long groupId, @PathVariable("newsId") long newsId) {
    newsService.deleteNews(newsId);
}
```

Rysunek 3.9: Aspekt uruchamiany przed wykonaniem metod

W serwisach znajduje się cała logika biznesowa. Działają w oparciu o repozytoria. Metody serwisowe wykonują określone akcje zlecone przez kontrolery, operując na encjach oraz modelach, które zostaną zwrócone klientowi. Na metodach serwisowych zakładane są transakcje logiczne, które pozwalają na wyeliminowaniu niepożądanych zjawisk spowodowane operacjami na bazie danych. Poprzez transakcje definiowane są poziomy izolacji oraz propagacji.

Repozytoria służą do komunikacji z bazą danych. Zajmują się pobieraniem, zapisywaniem oraz aktualizacją danych. W celu usunięcia powtarzającego się kodu, skorzystano z biblioteki spring data. Biblioteka umożliwia definiowanie własnych zapytań SQL oraz generowanie zapytań na podstawie nazw metod.

3.2.4. System powiadomień

Aplikacja obsługuje system powiadomień e-mail. Powiadomienia działają w oparciu o technologię spring-boot-starter-mail. Proces wysyłania powiadomień wygląda następująco:

1. Osoba uprawniona wprowadza zmianę na koncie użytkownika.
2. Poinformowanie odpowiedniego serwisu do wysyłania e-maili o zdarzeniu.
3. Wygenerowanie szablonu e-mail za pomocą silnika szablonów thymeleaf.
4. Zlecenie asynchronicznie wysłania wiadomości e-mail za pomocą protokołu SMTP. W polu nadawcy wiadomości umieszczany jest stały adres *sprawdzarka.ii@gmail.com* założony na potrzeby projektu.

W celu ujednolicenia szablonów wykorzystano silnik szablonów thymeleaf. System powiadomień obsługuje dwa języki: polski i angielski. Domyślnym językiem jest angielski. W przypadku gdy w żądaniu został określony język polski to wiadomość zostanie wysłana w języku polskim.

Rysunek 3.10: Powiadomienie o dołączeniu do grupy

3.3. Wykorzystane narzędzia i technologie

W niniejszym podrozdziale zostały przedstawione narzędzia i technologie od których zależy aplikacja. Głównymi czynnikami wpływającymi na wybór technologii i narzędzi są: ogólnodostępność, popularność oraz licencja oprogramowania. Dzięki tak dobranym technologią projekt będzie łatwy w rozwijaniu oraz utrzymywaniu.

Projekt został napisany głównie w języku obiektowym Java 11. Język Java cechuje się przenośnością i dużym wsparciem twórców środowisk programowania i narzędzi. Program napisany w języku Java można uruchomić na każdym systemie oraz urządzeniu, na którym istnieje implementacja maszyny wirtualnej Java (JVM). Dzięki popularności języka programowania powstaje wiele bibliotek i narzędzi do wytwarzania efektywnego i czystego kodu. W projekcie został także użyty język HTML5 do projektowania szablonów e-mail. Poniżej znajduje się lista narzędzi i technologii od których zależy projekt.

- Relacyjna baza danych MySql z silnikiem InnoDb - służy do przechowywania istotnych dla aplikacji danych. Głównym powodem wybrania tego typu bazy jest relacyjność przechowywanych modeli oraz stała struktura danych. MySql połączona z silnikiem InnoDb daje więcej możliwości. InnoDb obsługuje transakcje, zakładanie blokad na poziomie wierszy oraz klucze obce. Silnik jest zgodny z ACID(atomowość, spójność, izolacja, trwałość), który zapewnia, że żadne dane nie zostaną utracone lub błędnie przesłane. Baza danych zapewnia bardzo dobre prędkości odczytu i zapisu dla scenariuszy OLTP(przetwarzanie transakcyjne). Do podstawowych cech systemów OLTP należy natychmiastowy

dostęp do aktualnych informacji oraz wykonywanie dużej liczby prostych zapytań pochodzących od wielu użytkowników [11] [12].

- Java Persistence API - popularny standard mapowania obiektowo-relacyjnego dla języka Java. Wykorzystany w celu komunikacji między serwerem a bazą danych. Standard określa mechanizmy, które pozwalają na zarządzanie wierszami w bazie danych za pomocą obiektów w Javie. Najbardziej popularną implementacją standardu JPA jest biblioteka Hibernate, która została wykorzystana w projekcie. Poza translacją danych pomiędzy bazą danych a obiektami, Hibernate umożliwia tworzenie oraz aktualizację tabel a także zwiększenie wydajności operacji na bazie danych. Hibernate może zapamiętywać otrzymywane dane w obrębie transakcji minimalizując ilość zapytań do bazy [13].

- Spring Framework - szkielet tworzenia aplikacji na platformie Java. Składa się z wielu projektów. Został stworzony w oparciu o wiele sprawdzonych i przetestowanych rozwiązań. Większość projektów opiera się na wzorcu proxy zakładanego na poziomie kodu Javy lub bajtowego. Działa na poziomie kompilacji. Dodatkowe zależności: Spring Boot, Spring Data, Spring AOP, Spring Security, Spring Web.

- Logback - biblioteka do logowania w aplikacji.

- Lombok - biblioteka do automatycznego generowanie powtarzanego kodu. Głównym celem stworzenia lombok jest poprawa jakości wytwarzanego kodu w języku Java. Biblioteka udostępnia zbiór adnotacji na poziomie metod i klas, które pozwalają na generowanie kodu nie wnoszącego żadnej funkcjonalności biznesowej. Kod jest dopisywany do aplikacji na poziomie kompilacji [14].

- Thymeleaf - silnik szablonów Java, który potrafi generować dokumenty w XML, XHTML oraz HTML5. Głównym powodem skorzystania z narzędzia jest dbałość o jakość kodu i łatwe utrzymywanie go w przyszłości. W serwisie został użyty do tworzenia szablonów e-mail wysyłanych do użytkowników serwisu [15].

- Swagger2 -narzędzie ułatwiające projektowanie, budowanie, dokumentowanie oraz testowanie usług REST API. Swagger2 został wzbogacony przez swagger-ui, czyli graficzny interfejs użytkownika [16].

- Compilebox - narzędzie oparte na oprogramowaniu Docker oraz środowisku uruchomieniowego Nodejs. Służy do uruchamiania niezaufanego kodu w izolowanym środowisku utworzonym w kontenerze Docker. Komunikacja z narzędziem odbywa się za pomocą REST-api. Compilebox został rozwinięty i dopasowany pod potrzeby serwisu. Wprowadzone zmiany zostały omówione w sekcji *Ewaluacja zadań.*

- JSON Web Token for Java - w skrócie JJWT to biblioteka Java implementująca specyfikacje JSON Web Token (JWT), JSON Web Signature (JWS),

JSON Web Encryption (JWE), JSON Web Key (JWK) oraz JSON Web Algorithms (JWA). Dostarczona biblioteka zapewnia bezpieczną i szybką autoryzację użytkownika w systemie za pomocą tokenów [17].

- Docker - narzędzie pozwalające na umieszczenie programu w lekkim, przenośnym i izolowanym kontenerze działającym na jądrze systemu operacyjnego. Docker został użyty do izolowania uruchomionych programów wysłanych przez użytkowników. Dodatkowo jest alternatywną wersją wdrożenia systemu na chmurę oraz serwery lokalne. Używając narzędzia Docker, użytkownik jest zwolniony od instalowania wszystkich zależności oraz narzędzi potrzebnych do uruchomienia systemu.

- Terraform - narzędzie służące do zarządzania infrastrukturą. Pozwala na zbudowanie i skonfigurowanie środowiska w którym uruchomione zostaną aplikacje. Wspierany jest przez wiele firm takich jak AWS, Azure czy Google Cloud. W aplikacji został użyty do zautomatyzowania tworzenia serwerów, sieci oraz bazy danych w chmurze Google Cloud.

- GitHub - serwis internetowy bazujący na systemie kontroli wersji git. Wspiera projektowanie aplikacji posiadając między innymi rozbudowany system zadań, który posłużył do dokumentowania funkcjonalności aplikacji.

- Maven - narzędzie automatyzujące budowę aplikacji. Pozwala na łatwe zarządzanie zależnościami zwalniając programistę od ręcznego pobierania i konfigurowania bibliotek.

3.4. Interfejs systemu

W celu komunikacji z aplikacją stworzono REST-owe API. *„REST (ang. Representational State Transfer) jest wzorcem narzucającym dobre praktyki tworzenia architektury aplikacji."* Aplikacja udostępnia adresy URL, które identyfikują jednoznacznie zasoby serwisu. Klient w celu komunikacji powinien wysłać odpowiednią metodą HTTP żądanie w celu wykonania określonej czynności [18]. Protokół HTTP umożliwia wykonanie następujących operacji:

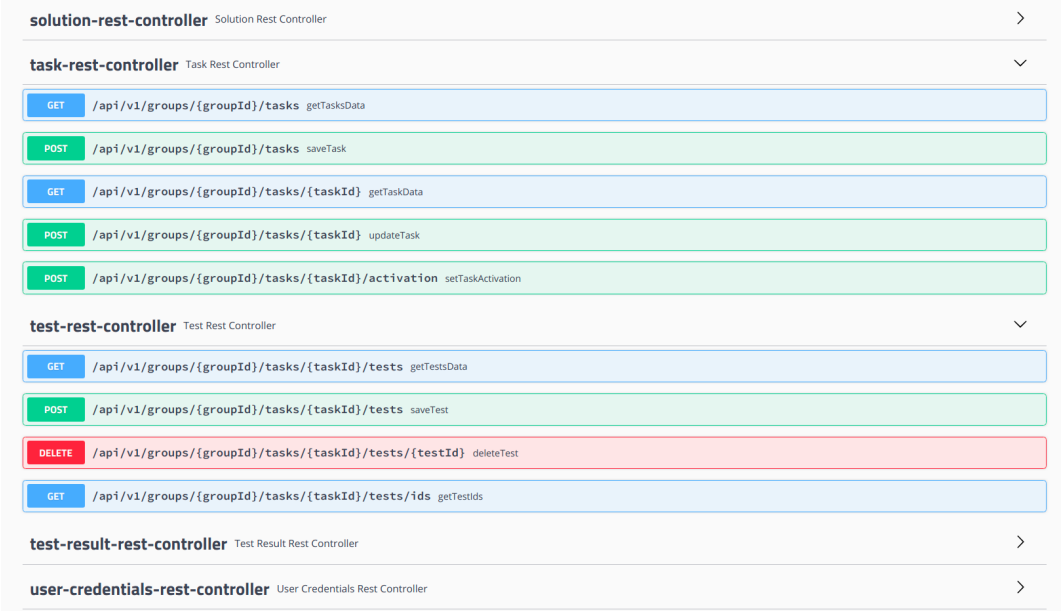
- GET - pobieranie zasobu
- POST - tworzenie zasobu
- PUT - aktualizacja zasobu
- DELETE - usunięcie zasobu

Aby utworzyć nowy zasób lub zaktualizować istniejący, klient powinien wysłać model w formacie JSON. Podobna sytuacja ma miejsce w przypadku pobierania

zasobu. Zwracane modele zasobów są reprezentowane w formacie JSON. W przypadku usuwania zasobów, punkt końcowy ani nie zwraca, ani nie przyjmuje żadnego modelu. Wszystkie dostępy do zasobów są poprzedzone prefiksem */api/v{id}* w celu wersjonowania API. Do uzyskania dostępu do zasobów serwisu wymagana jest autoryzacja użytkownika. Autoryzacja użytkownika opiera się na tokenie dostępu wysłanym w nagłówku HTTP.

Aby uzyskać token dostępu, najpierw należy uwierzytelnić się w aplikacji za pomocą serwera autoryzującego Google. Użytkownik powinien wysłać żądanie pod adres */oauth2/authorization/google*. Po autoryzacji w serwisie Google oraz potwierdzeniu zgody na dostęp do przetwarzania informacji przez aplikację, użytkownik zostaje przekierowany do części klienckiej pod adres */login_process/{temporary_token}*, gdzie w adresie URL znajduje się token tymczasowy. Za pomocą tak uzyskanego tokenu tymczasowego można poprosić o parę tokenów: token odświeżający i token dostępu.

Pełny spis punktów końcowych generowany jest pod adresem */swagger-ui.html*. Strona jest dostępna po uruchomieniu aplikacji.



Rysunek 3.11: Fragment wygenerowanego widoku na punkty końcowe aplikacji

3.5. Opis wdrożenia

Poniżej zostały opisane wymagania oraz sposoby wdrożenia aplikacji. Omówione zostały sposoby wdrożenia aplikacji na serwery lokalne jak i na chmurę [19].

3.5.1. Wdrożenie na serwer lokalny

Istnieją dwa sposoby uruchomienia aplikacji. Pierwszy sposób polega na uruchomieniu aplikacji za pomocą Docker. Drugi wymaga zainstalowania wszystkich narzędzi i wymaganych programów.

Wdrożenie za pomocą Docker

Pierwszy sposób uruchomienia aplikacji polega na kontenerach Docker. Kontenery umożliwiają łatwą przenośność oraz wdrożenie aplikacji.

Tablica 3.1: Wymagania do uruchomienia aplikacji przy użyciu narzędzia Docker

Docker 18.06.1-ce
Java 11.0.1
Apache Maven 3.5.2

Baza danych W celu uruchomienia bazy danych został użyty gotowy obraz z zainstalowanym serwerem Mysql. Aby poprawnie funkcjonowała baza danych, należy przekazać jako argument nazwę oraz hasło dla super użytkownika.

```
#!/bin/bash
mkdir -p /usr/local/etc/sprawdzarka/mysql

sudo docker run -d -p 3306:3306
    -v /usr/local/etc/sprawdzarka/mysql:/var/lib/mysql
    -e MYSQL_ROOT_PASSWORD=root
    -e MYSQL_DATABASE=app_db
    --name mysql
    mysql:latest
```

Listing 3..2: Uruchomienie bazy danych przy użyciu Docker

Aplikacja Na początku należy przejść do folderu, gdzie znajduje się aplikacja a następnie zbudować ją za pomocą Maven. Po poprawnym zbudowaniu aplikacji należy utworzyć obraz Docker oraz uruchomić go z pełnym dostępem do sieci.

```
#!/bin/bash
mvn install
sudo docker build -t backend .
sudo docker run -d --network host --name backend backend
```

Listing 3..3: Uruchomienie aplikacji przy użyciu Docker

Compilebox Utworzyć ścieżkę */usr/local/etc/sprawdzarka/tmp* w celu przechowywania tymczasowych plików utworzonych przez compilebox. Następnie przejść do folderu, gdzie znajduje się narzędzie compilebox i zbudować obrazy Docker. Pierwszy z aplikacją i drugi z środowiskiem, w którym zostanie uruchomiony niezaufany kod. Po utworzeniu obrazu, uruchomić kontener mapując port kontenera 8080 na port 8080. Przy uruchomieniu kontenera, należy udostępnić utworzoną ścieżkę oraz proces do tworzenia nowych kontenerów.

```
#!/bin/bash
mkdir -p /usr/local/etc/sprawdzarka/tmp
sudo docker build -t compilebox .
sudo docker build -t virtual_machine ./Setup/

sudo docker run -d -p 8080:8080
    -v /var/run/docker.sock:/var/run/docker.sock
    -v /usr/local/etc/sprawdzarka/tmp:/usr/local/src/API/temp
    -e TMP_PATH="/usr/local/src/sprawdzarka/tmp"
    --name compilebox
    compilebox
```

Listing 3..4: Uruchomienie compilebox przy użyciu Docker

Wdrożenie bez użycia Docker

Drugi sposób przedstawia uruchomienie systemu bez narzędzia Docker. Polega na skonfigurowaniu i uruchomieniu wszystkich aplikacji i narzędzi.

Tablica 3.2: Wymagania do uruchomienia aplikacji

Docker v18.06.1-ce
Java v11.0.1
Apache Maven v3.5.2
MySql v5.7.24 (nazwa użytkownika: root, hasło: root)
Npm v6.4.1
Nodejs v10.12.0

Baza danych Po instalacji Mysql należy zalogować się do konsoli za pomocą użytkownika o nazwie *„root”*. Następnie stworzyć bazę danych przechowującą istotne informację dla aplikacji.

```
CREATE DATABASE `app_db`
    DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Listing 3..5: Tworzenie bazy danych

Compilebox Do poprawnego działania narzędzia, należy utworzyć obraz *virtual_machine*. Tak stworzony obraz posłuży do tworzenia środowiska, w którym będzie uruchamiany niezaufany kod. Następnie nadać pełne prawa do skryptów uruchamianych w kontenerach oraz zainstalować wszystkie zależności, a na koniec uruchomić aplikację compilebox.

```
#!/bin/bash
sudo docker build -t virtual_machine ./Setup/
sudo chmod 777 ./Payload/script.sh
sudo chmod 777 ./Payload/javaRunner.sh
npm install .
node run
```

Listing 3..6: Pobieranie zależności i uruchomienie narzędzia compilebox

Aplikacja Przejść do folderu, w którym znajduje się projekt. Następnie zbudować i uruchomić aplikację za pomocą Maven.

```
#!/bin/bash
mvn install
mvn spring-boot:run
```

Listing 3..7: Uruchomienie aplikacji

3.5.2. Wdrożenie na chmurę

Jeśli aplikacja zostanie wdrożona na chmurę to należy wcześniej skonfigurować chmurę, pobrać adresy IP odpowiednich instancji i wpisać do pliku o nazwie *„application-cloud.properties”*. W przypadku bazy danych nie jest konieczny Docker, ponieważ istnieje specjalna usługa *sql* oferowana przez Google Cloud. Budowanie obrazu dla aplikacji oraz budowanie i uruchomienie obrazu dla compilebox wygląda identycznie jak dla wersji lokalnej. W celu uruchomienia aplikacji należy przypisać do zmiennej środowiskowej CONFIG wartość *„cloud”*.

```
#!/bin/bash
sudo docker run -d --network host -e CONFIG=cloud --name backend backend
```

Listing 3..8: Uruchomienie aplikacji w chmurze przy użyciu Docker

Aby przenieść obrazy Docker do odpowiednich instancji można skorzystać z repozytorium na Dockerhub.

Rozdział 4.

Podsumowanie i wnioski

Niniejszy rozdział stanowi podsumowanie zaprojektowanej aplikacji. Omówione są spełnione wymagania oraz możliwość rozwoju aplikacji w przyszłości.

4.1. Efekt końcowy pracy

Rezultatem wielomiesięcznej pracy nad projektem jest część serwerowa systemu automatycznej ewaluacji zadań algorytmicznych spełniający wymagania oraz funkcjonalność opisaną w rozdziale 2 (część dla użytkownika). Projekt inżynierski został zrealizowany ze względu na małą ilość aplikacji pozwalających na ewaluację zadań w bezpiecznym środowisku oraz planowanie i organizowanie zajęć programistycznych. Aplikacja umożliwia w pełni zarządzanie i kontrolę nad całym systemem, bez konieczności martwienia się o poufne dane oraz bezpieczeństwo urządzeń. Część serwerowa jest zgodna i wspiera część kliencką zrealizowaną w pracy inżynierskiej pod tytułem *„część kliencka systemu automatycznej ewaluacji zadań algorytmicznych”* wykonaną przez Mateusza Patera. Z powodzeniem udało się wdrożyć aplikację na chmurę oraz na lokalny serwer [20].

Zaprojektowane serwery są bezstanowe, czyli nie posiadają żadnego stanu oraz nie przechowują żadnych informacji o zalogowanym użytkowniku w pamięci operacyjnej urządzenia. Tak zbudowane serwery umożliwiają łatwe skalowanie aplikacji w przyszłości. Możliwe jest bezpieczne zwiększanie liczby instancji aplikacji nie martwiąc się o przechowywanie stanu. Skalowanie może być zależne od ruchu na serwerze lub obciążenia urządzenia.

W tworzeniu aplikacji zastosowano narzędzia i technologie, które są ogólnodostępne i popularne wśród programistów. Pomogły nie tylko w zrealizowaniu wymagań oraz postawionych celów, a także w napisaniu czystego i czytelnego kodu w celu utrzymania oraz możliwego rozwoju aplikacji.

4.2. Możliwość rozwoju aplikacji

System został opracowany i zaprojektowany, aby móc łatwo i przyjemnie rozszerzać możliwości aplikacji. Na przyszłość zostały zaplanowane do realizacji następujące funkcjonalności:

- nowy rodzaj zadania polegający na sprawdzeniu poprawności, pomiaru czasy wykonania kodu źródłowego oraz zarezerwowanej pamięci;
- wprowadzenie paginacji i wyszukiwania podczas zwracania listy danych;
- wprowadzenie ograniczenia na liczbę tworzonych grup dla użytkownika z rolą *twórca*;
- możliwość zmiany, dodania lub usunięcia testu podczas aktywnego zadania;
- wprowadzenie możliwości edycji grupy oraz testów;
- ponowne wykonanie rozwiązania w przypadku błędu po stronie serwera;
- wprowadzenie czatu pod konkretnymi zadaniami;
- wprowadzenie bazy zadań dostępnych dla wszystkich lub pewniej grupy użytkowników serwisu;
- możliwość definiowania zadań z możliwością wielu poprawnych odpowiedzi;
- sprawdzenie rozwiązań na podstawie zdefiniowanej bazy poprawnych odpowiedzi lub napisanego algorytmu przez autora zadania;
- zmiana komunikacji między aplikacją a compilebox;

W przypadku wdrożenia systemu na chmurę dostępne są usprawnienia:

- wprowadzenie loadbalancer do równoważenia obciążenia serwerów aplikacji;
- wprowadzenie narzędzia do ciągłej integracji np. Jenkins;

Komunikacja między aplikacją a narzędziem compilebox może polegać na kolejce. Przykładową implementacją kolejki w Google Cloud jest *pub/sub*. Usługa obsługuje wysyłanie zadań do kolejki i powiadomienia subskrybentów o istniejących zadaniach w kolejce. Jeśli instancji z aplikacją compilebox będzie więcej niż jedna, to można skonfigurować usługę w taki sposób, aby tylko jedna instancja otrzymywała konkretne zadanie. W przypadku niepowodzenia, wycofanie zadania i ponowne wprowadzenie do kolejki. Innym możliwym rozwiązaniem, mniej zależnym od chmury, jest użycie własnej kolejki. W tym celu można skorzystać z gotowych kontenerów Docker z oprogramowaniem *RabbitMQ* lub *Apache Kafka*. Jednym z rozwiązań mającym na celu zrównoważenie obciążenia urządzeń jest użycie oprogramowania Kubernetes, który automatycznie instaluje, skaluje i zarządza kontenerami w obrębie kilku instancji chmurowych.

Spis tablic

Spis rysunków

Listings

Bibliografia

[1] Marcin Zajączkowski & Warszawa-DP Study Group, *Czym jest Coding Dojo i Code Kata?*, url: https://solidsoft.wordpress.com/2010/12/09/czym-jest-coding-dojo-i-code-kata/ (term. wiz. 04. 01. 2019)

[2] Asad Memon, *How we used Docker to compile and run untrusted code*, url: https://blog.remoteinterview.io/how-we-used-docker-to-compile-and-run-untrusted-code-2fafbffe2ad5 (term. wiz. 04.01.2019)

[3] Docker Inc., *Why docker*, url: https://www.docker.com/why-docker (term. wiz. 04.01.2019)

[4] Agnieszka Pieszczek , *Wzorce projektowe: Builder – zbudujmy to lepiej*, url: https://codecouple.pl/2018/01/12/4-wzorce-projektowe-builder-zbudujmy-to-lepiej/ (term. wiz. 04.01.2019)

[5] D. Crockford , *The application/json Media Type for JavaScript Object Notation (JSON)*, url: https://www.ietf.org/rfc/rfc4627.txt (term. wiz. 04.01.2019)

[6] Julien Monty, *Devicon*, url: http://konpa.github.io/devicon/ (term. wiz. 25.01.2019)

[7] Aaron Parecki , *OAuth 2.0*, url: https://oauth.net/2/ (term. wiz. 12.01.2019)

[8] Marek Zając, *Json Web Token*, url: https://zajacmarek.com/2017/05/json-web-token/ (term. wiz. 13.01.2019)

[9] Asad Memon, *compilebox*, url: https://github.com/remoteinterview/compilebox (term. wiz. 26.01.2019)

[10] Kamil Breczko, *sprawdzarka-compilebox*, url: https://github.com/kbreczko/sprawdzarka-backend (term. wiz. 01.02.2019)

[11] Jakuba Wróblewskiego, *Czym różnią się hurtownie danych od zwykłych baz danych? Typowe zastosowania.*, url: http://edu.pjwstk.edu.pl/wyklady/hur/scb/wyklad1/w1.htm (term. wiz. 13.01.2019)

[12] 2ndQuadrant, *PostgreSQL vs MySQL*, url: https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/ (term. wiz. 13.01.2019)

[13] Marcin Pietraszek, *Pogodynka - JPA i Spring Data*, url: http://www.samouczekprogramisty.pl/pogodynka-jpa-i-spring-data/ (term. wiz. 14.01.2019)

[14] Tomasz Woliński, *Lombok – jak pozbyć się getterów i setterów z kodu*, url: https://stormit.pl/lombok/ (term. wiz. 13.01.2019)

[15] Thymeleaf Team, *Tutorial: Using Thymeleaf*, url: https://stormit.pl/lombok/ (term. wiz. 12.01.2019)

[16] Dilip Krishnan, *Springfox Reference Documentation*, url: https://springfox.github.io/springfox/docs/current/#introduction (term. wiz. 12.01.2019)

[17] Lindsay Brunner , *A Beginner's Guide to JWTs in Java*, url: https://stormpath.com/blog/beginners-guide-jwts-in-java (term. wiz. 12.01.2019)

[18] Patryk Jar, *REST – ciekawszy sposób na komunikację client-server*, url: http://yarpo.pl/2012/07/29/rest-ciekawszy-sposob-na-komunikacje-client-server/ (term. wiz. 12.01.2019)

[19] Kamil Breczko, *sprawdzarka-config*, url: https://github.com/kbreczko/sprawdzarka-config (term. wiz. 01.02.2019)

[20] Kamil Breczko, *sprawdzarka-backend*, url: https://github.com/kbreczko/sprawdzarka-backend (term. wiz. 01.02.2019)