Studio projektowe

Projekt kompilatora online

Autorzy:
Piotr Karaś
Patrycja Kopacz

Prowadzący
dr hab. Konrad Kułakowski

1. Diagram Gantta

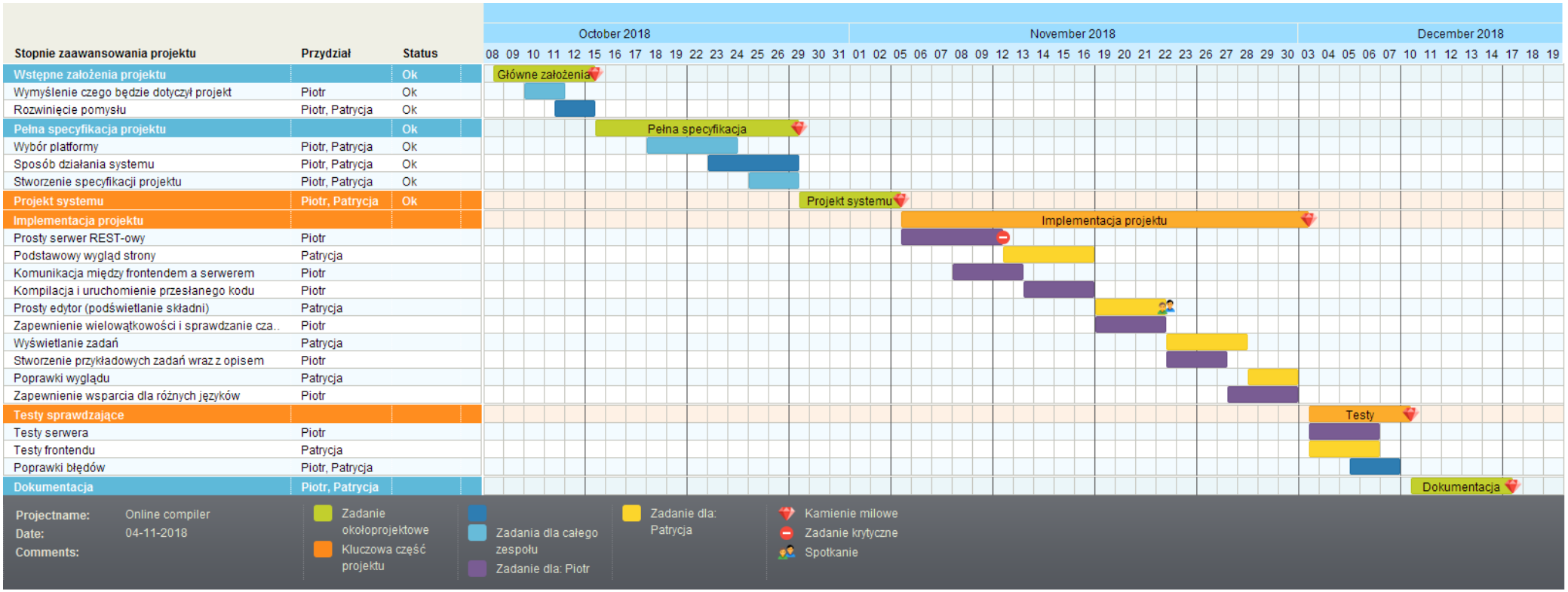


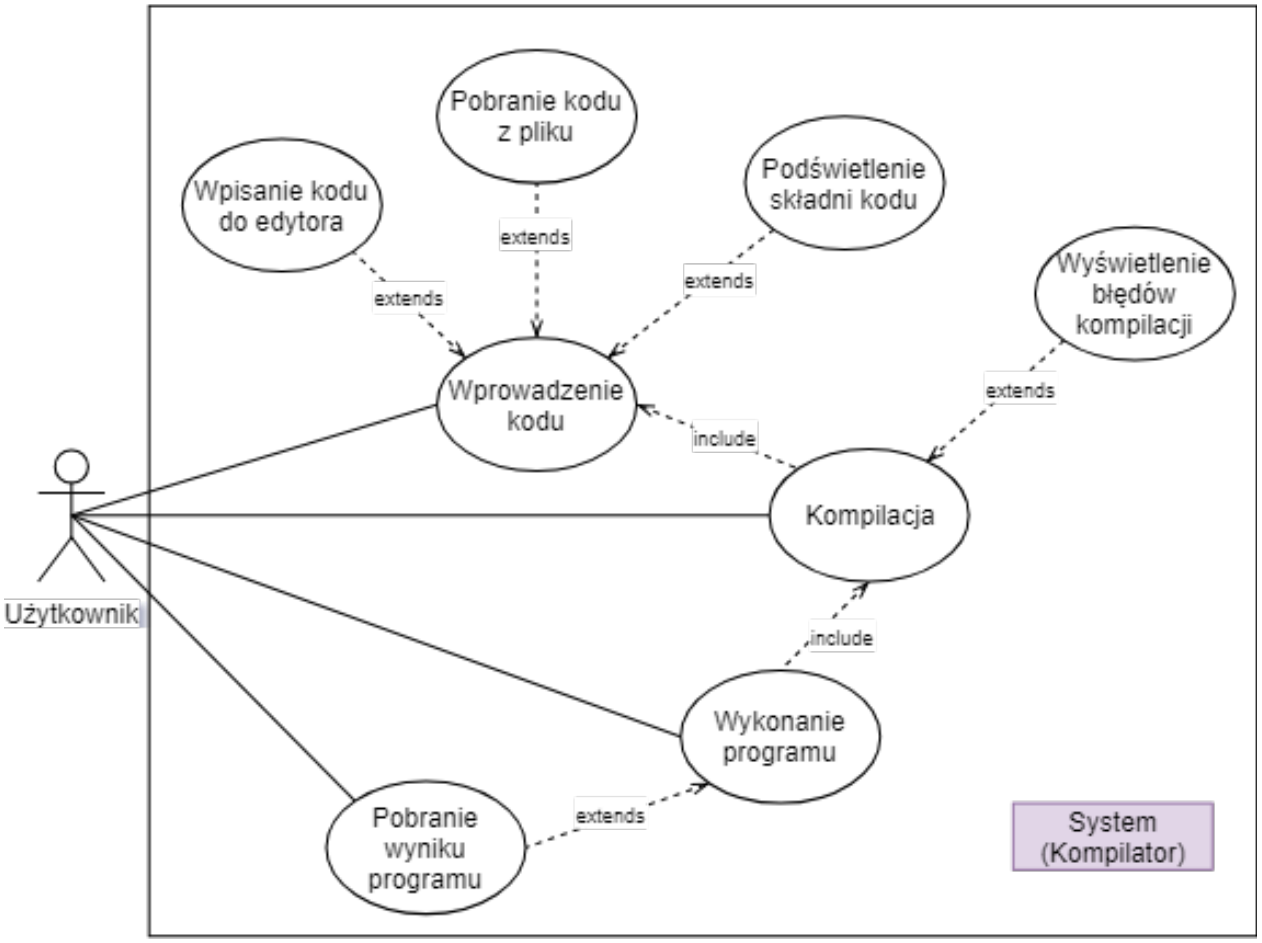
Diagram Gantta ukazuje przepływ pracy nad projektem od jego początkowych faz, aż po ukończenie systemu. Ramy czasowe projektu zostały ustalone jako przedział pomiędzy początkiem października, a 17 grudnia, zatem około dwóch miesięcy.
Zgodnie z harmonogramem wcześniej ustalonym, został stworzony diagram Gantta:
➔ 15.10.2018 – stworzenie wstępnych założeń projektu
➔ 29.10.2018 – pełna specyfikacja projektu
➔ 05.11.2018 – zaimplementowany serwer wraz ze wstępnym widokiem strony
➔ 03.12.2018 – działający kompilator umożliwiający skompilowanie na już w pełni działającej stronie internetowej
➔ 10.12.2018 – wykonanie testów sprawdzających
➔ 17.12.2018 – gotowy projekt wraz z dokumentacją

Sama implementacja projektu została podzielona na kilka głównych zadań takich jak:

- Stworzenie prostego serwera REST-owego
- Podstawowy wygląd strony
- Komunikacja między frontendem a serwerem
- Kompilacja i uruchomienie przesłanego kodu
- Prosty edytor (podświetlanie składni)
- Zapewnienie wielowątkowości i sprawdzanie czasu
- Wyświetlanie zadań
- Stworzenie przykładowych zadań wraz z opisem
- Poprawki wyglądu
- Zapewnienie wsparcia dla różnych języków

Poszczególne zadania są przydzielone pojedynczym osobom biorącym udział w projekcie lub też obydwojgu, jeżeli powinno ono być zależne od decyzji wszystkich członków zespołu.

2. Diagram przypadków użycia



➔ Aktorzy

1. Użytkownik, czyli osoba korzystająca z kompilatora online. W założeniu system jest przeznaczony dla osób uczących się, które wcześniej nie zetknęły się z danym językiem programowania.
2. Kompilator, czyli nasz szeroko pojęty system – warstwa webowa, jak i serwer.

➔ Przypadki użycia

1. Wprowadzenie kodu – ogólnie rozumiane wprowadzenie kodu – od napisania własnego, do pobrania z istniejących szablonów.
2. Wpisanie kodu do edytora – użytkownikowi zostanie udostępniony edytor, dzięki któremu będzie miał możliwość napisania swojego programu w łatwy sposób.
3. Pobranie kodu z pliku – użytkownik będzie miał możliwość pobrać przykładowe programy, tak aby mógł się uczyć na przykładach. Będzie też możliwość pobrania „szablonu” programu, do którego użytkownik będzie musiał coś dopisać, aby program działał. Będzie on miał przedstawioną listę zadań – kawałki kodu, które powinien wpisać w odpowiednie miejsce, aby cały program zwrócił żądany rezultat.

4. Podświetlenie składni kodu – oraz inne narzędzia wspomagające pisanie kodu, a będące zaimplementowane przez kompilator. Poza podświetlaniem składni może to być zmiana tematu, auto-wcięcia, podpowiedzi gdzie użytkownik mógłby wprowadzić zmiany, aby kod był czytelniejszy.
5. Kompilacja – po wprowadzeniu kodu użytkownik może go skompilować – zatem wysłać do kompilatora swój kod, aby ten go skompilował.
6. Wyświetlenie błędów kompilacji – po kompilacji użytkownikowi zostanie pokazany status z jakim zakończył się ten proces – jeśli coś poszło nie tak to zostanie podświetlona linijka, która spowodowała błąd.
7. Wykonanie programu – jeśli kompilacja przebiegła zgodnie z planem program można wykonać – użytkownik powinien zobaczyć rezultat uruchomienia swojego kodu, a także móc wejść z nim w interakcję poprzez zapewnienie swojego wejścia.
8. Pobranie wyniku programu – użytkownik może pobrać wynik programu jak i sam program – napisany przez siebie kod.

3. Poglądowy diagram sekwencji programu

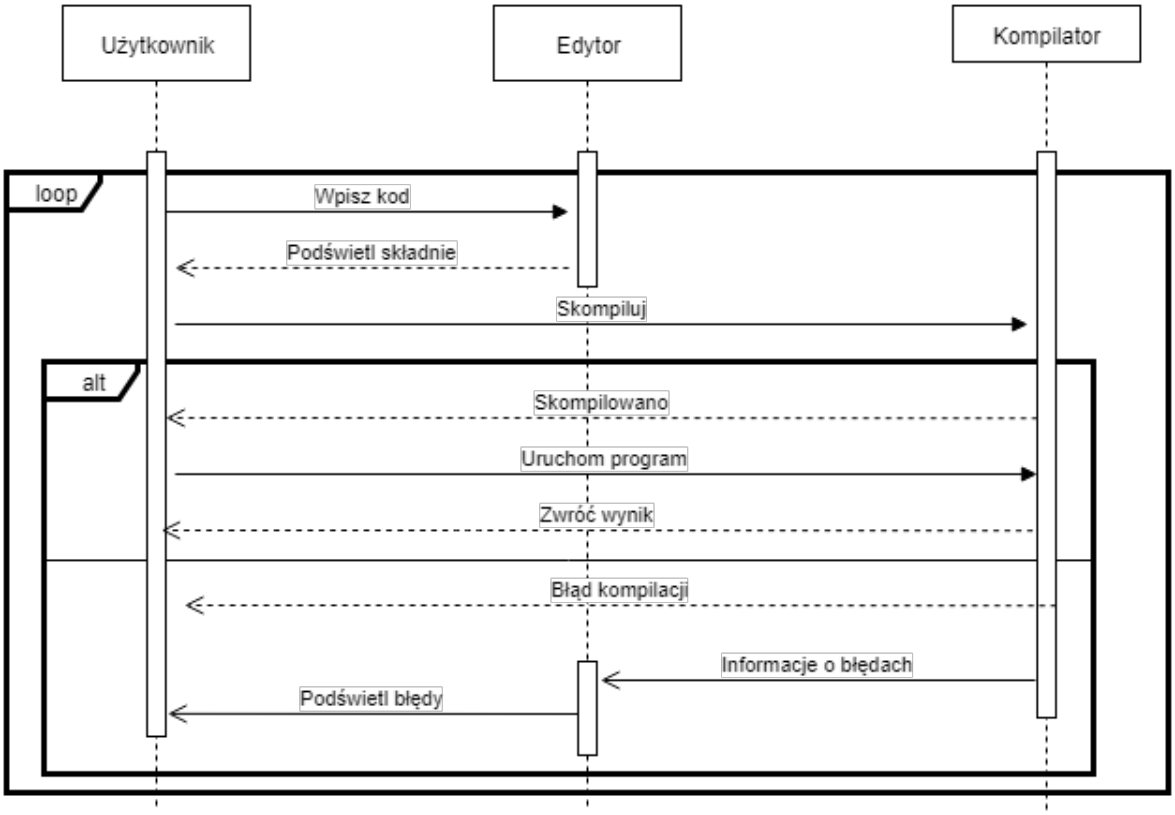
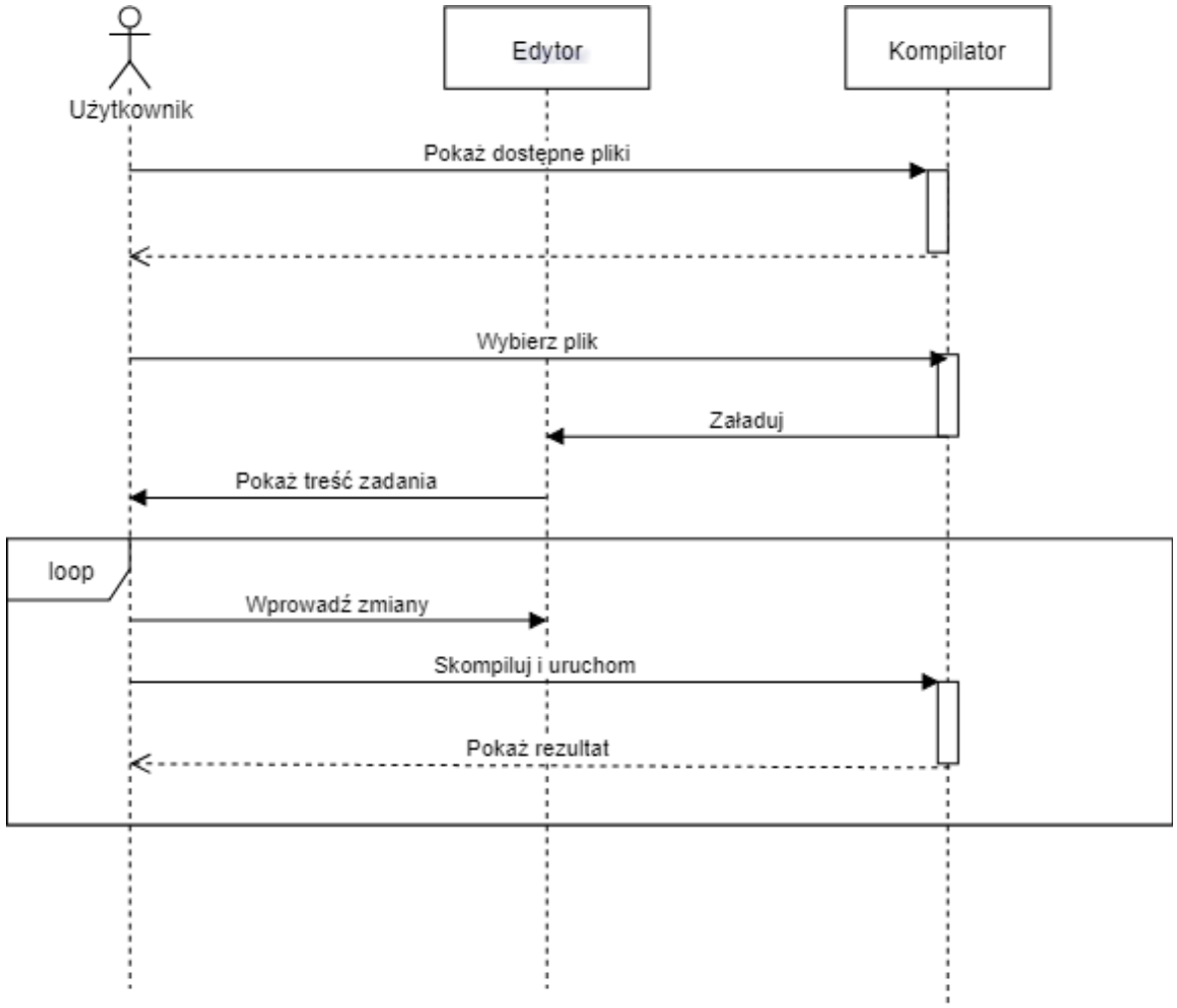


Diagram przedstawia podstawową funkcjonalność naszego projektu i to w jaki sposób jest ona dostarczana użytkownikowi. Proces zachodzący podczas korzystania z naszej aplikacji jest następujący – początkowo użytkownik wprowadza swój kod lub go pobiera, korzysta przy tym z naszego edytora (podświetlanie składni itp). Po zakończeniu tej części następuje próba kompilacji – kod wysyłany jest do serwera. Następnie mogą się zdarzyć dwie

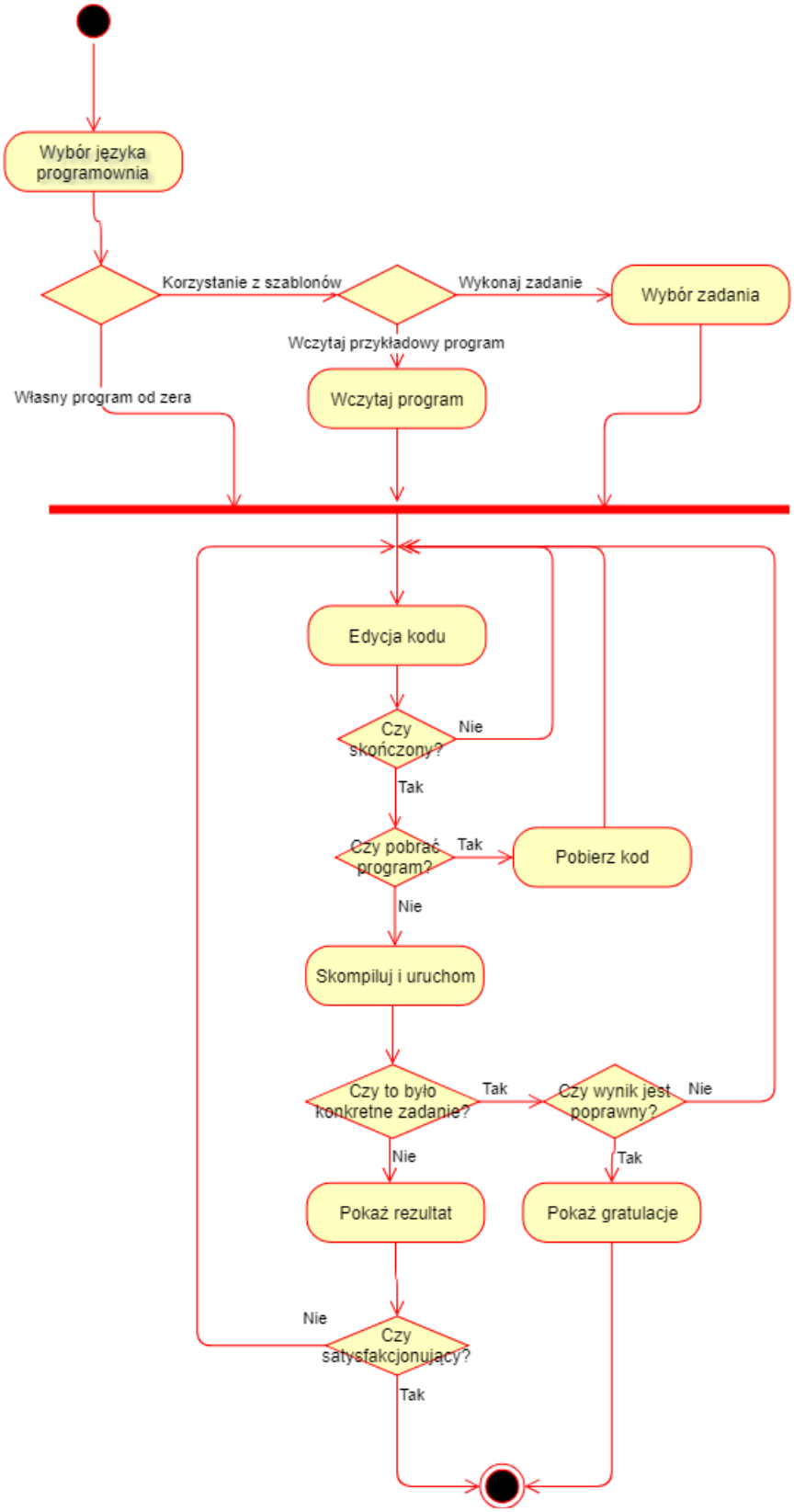
różne rzeczy: program jest napisany błędnie i proces kompilacji zakończy się błędami – w takim wypadku edytor otrzyma informację w której linijce nastąpił błąd i ją podkreśli lub też wszystko przebiegło poprawnie – wtedy użytkownik otrzyma informację o sukcesie. Po takiej informacji może on uruchomić swój program i zobaczyć jego rezultat.

4. Diagram sekwencji dla wykonywania konkretnego zadania.



Przy wykonywaniu konkretnego zadania użytkownik wpierw wybiera, które z nich chce wykonać. W tym celu pobiera z serwera listę dostępnych zadań (razem z ich stopniem trudności) i dokonuje wyboru. Po wyselekcjonowaniu serwer przesyła dane o zadaniu do edytora – ten wyświetla już napisany kod i pokazuje użytkownikowi, gdzie ten powinien dopisać własną część. Oprócz tego w oddzielnym oknie powinna zostać wyświetlona treść zadania z ewentualnymi podpowiedziami jak je wykonać.
Po wprowadzeniu zmian przez użytkownika ten zatwierdza je, przesyłając na serwer swój kod, który ten stara się skompilować i uruchomić. Po tym użytkownikowi zostanie pokazany rezultat – czy kompilacja przebiegła pomyślnie, jeśli tak to jaki jest wynik napisanego programu. Jeśli wynik zgadza się z przyjętymi założeniami zostaną wyświetlone gratulacje – w przeciwnym wypadku zachęta do dalszych prób.

5. Diagram czynności.

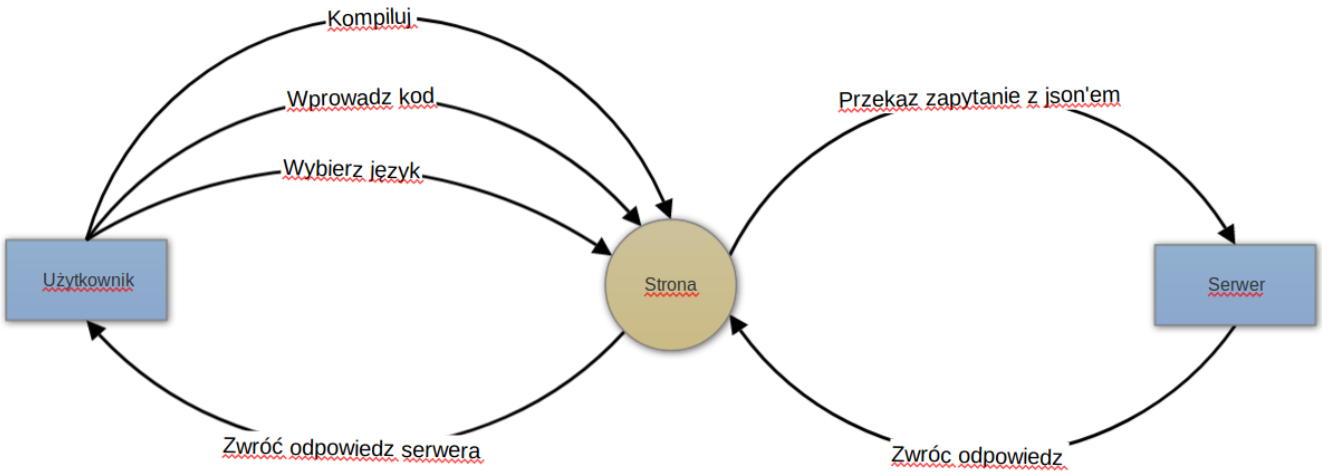


Na tym diagramie została przedstawiona sekwencja kroków, które są wykonywane przez modelowany system. Nie jest to jednak szczegółowy diagram, a bardziej uogólniony – pokazujący kolejne kroki w naszym programie. Przykładowo przy

edycji kodu można by pokazać szereg zabiegów, które ją wspomagają m.in. podświetlanie składni, możliwość wyboru tematu (theme), przy powrocie z kompilacji podświetlanie linijki w której wystąpił błąd, wyłączenie edycji linii, których nie powinno się edytować itp.
Na samym początku użytkownik dokonuje wyboru języka programowania, po niej natomiast ma trzy opcje: pisze własny kod, wybiera jedno z dostępnych zadań (trzeba dopisać jedną metodę/kawałek kodu spełniający jakąś specyfikacje) lub też pobiera jeden z dostępnych programów i może go wtedy dowolnie edytować. W przypadku zadania użytkownik może edytować tylko kawałek kodu – reszta jest niedostępna (wyłączona jest jego edycja). Po tym wyborze użytkownik edytuje program i kiedy jest on już gotowy może go pobrać na swój komputer lub skompilować i uruchomić go na serwerze. Po zwróceniu wyniku użytkownikowi zostaje wyświetlony wynik i w zależności od rezultatu użytkownik może wrócić do edycji. Natomiast jeśli wykonywał on konkretne zadanie to serwer sprawdza otrzymany wynik programu z tym jaki powinien wyjść i po porównaniu pokazuje czy rozwiązanie zaproponowane przez użytkownika jest poprawne.

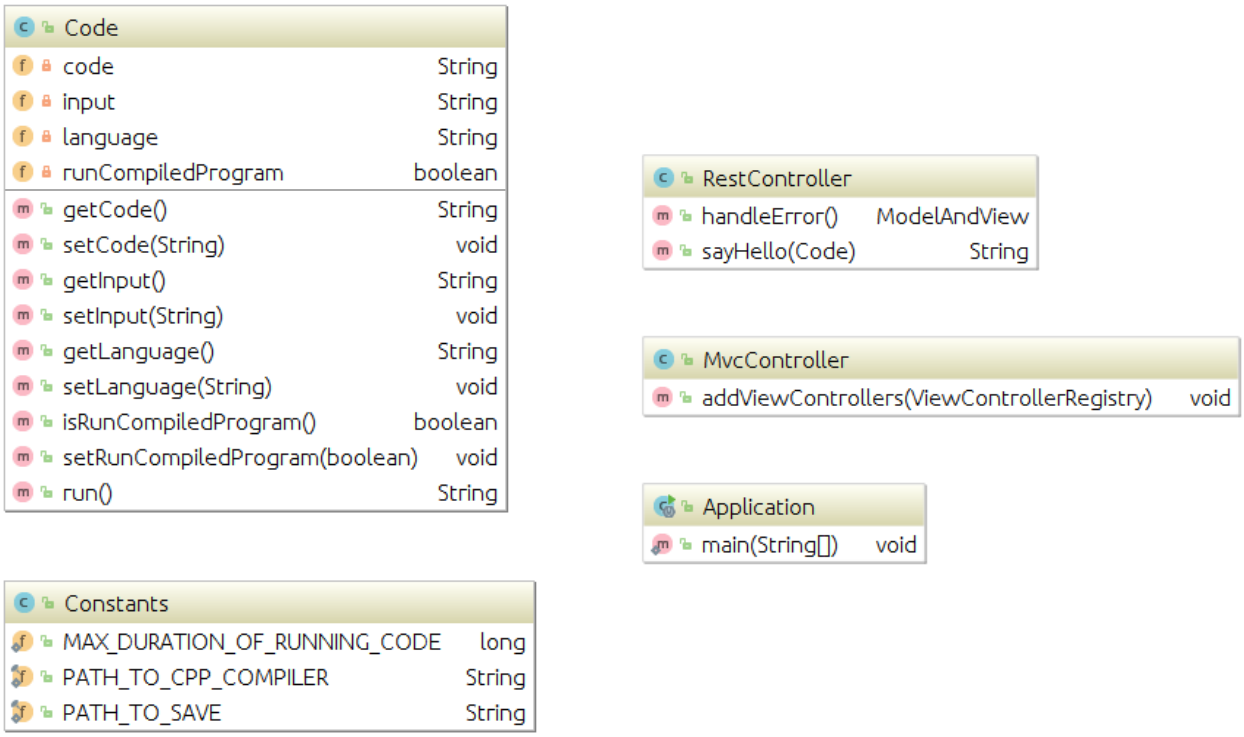
6. Diagram kontekstowy

Diagram kontekstowy przedstawia w jaki sposób dane przepływają w systemie oraz opisuje procesy przetwarzające dane. Jak widać w diagramie, użytkownik poprzez stronę internetową, komunikuje się z serwerem, który przetwarza zapytania wysłane za jego pośrednictwem.



7. Diagramy klas

Kod kompilatora składa się z dwóch pakietów, a każdy z nich ma kilka klas, ich atrybuty oraz metody pokazane są niżej.



8. Przesyłany JSON – Kompilacja kodu

URL: http://localhost:8080/api/compile/code

Metoda: POST

W JSON'ie przysyłany jest kod wpisany przez użytkownika, input programu, język programowania oraz flaga *runCompiledProgram*.

Przykład w kodzie JavaScript, gdzie *code* i *input* są zmiennymi pobranymi z pól tekstowych, w których użytkownik wpisuje swoje dane:

```
$.ajax({
  url: "http://localhost:8080/api/compile/code",
  datatype: 'json',
  type: "post",
  contentType: "application/json",
  data: JSON.stringify({
    code: code,
    input: input,
    language: "Cpp",
    runCompiledProgram : false
  })
}).then(function (data, status, jqxhr) {
  $("#compileResponse").val('Status success, Server response: \n' +
data);
});
```

Odpowiedź:

W przypadku poprawnego zapytania oraz kodu, serwer zwróci tekst „*Compilation succeeded*!” oraz odpowiedz programu.

Zwracane błędy:

W przypadku błędu kompilacji lub błędnego wysłanego JSON’a, serwer zwróci odpowiedz „*Compilation failed*!” wraz ze stosownymi błędami.