



---

# POLITECHNIKA POZNAŃSKA

---

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI  
Instytut Informatyki

Praca dyplomowa inżynierska

## **APLIKACJA INTERNETOWA SŁUŻĄCA DO GENEROWANIA PLANÓW LEKCJI DLA SZKÓŁ PODSTAWOWYCH ORAZ ŚREDNICH**

Mateusz Biernacki, 140681

Dominik Boła, 136524

Maciej Gorał, 132228

Grzegorz Piątkowski, 135868

Promotor

dr inż. Izabela Janicka-Lipska

POZNAŃ 2022



# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>1</b>
<b>2</b>	<b>Podstawy teoretyczne</b>	<b>3</b>
2.1	Problem optymalizacyjny . . . . .	3
2.2	Problem NP-trudny . . . . .	3
2.3	Heurystyka . . . . .	3
2.4	Algorytm ewolucyjny . . . . .	3
<b>3</b>	<b>Analiza i porównanie możliwych rozwiązań</b>	<b>5</b>
3.1	Analiza problemu . . . . .	5
3.2	Aktualnie dostępne rozwiązania . . . . .	6
3.2.1	aSc TimeTables . . . . .	6
3.2.2	Prime Timetable . . . . .	6
3.2.3	SuperSaas . . . . .	6
3.3	Możliwe podejścia . . . . .	6
3.4	Wymagania funkcjonalne i нефункционалне . . . . .	7
3.5	Przypadki użycia . . . . .	8
<b>4</b>	<b>Metodyka pracy oraz przygotowanie infrastruktury informatycznej</b>	<b>13</b>
4.1	Wstęp . . . . .	13
4.2	Pojęcia . . . . .	13
4.2.1	DevOps . . . . .	13
4.2.2	Continuous Integration and Continuous Deployment . . . . .	14
4.2.3	Kontrola wersji . . . . .	14
4.3	Narzędzia i technologie . . . . .	14
4.3.1	Amazon Web Services . . . . .	14
4.3.2	Git . . . . .	14
4.3.3	GitHub . . . . .	14
4.3.4	Docker . . . . .	14
4.3.5	CircleCI . . . . .	15
4.3.6	Secure Shell . . . . .	15
4.3.7	tmux . . . . .	15
4.3.8	shell . . . . .	16
4.3.9	Black . . . . .	16
4.3.10	Pylint . . . . .	16
4.4	Przygotowanie infrastruktury informatycznej . . . . .	16
4.4.1	Przygotowanie strony serwerowej przy pomocy platformy AWS . . . . .	16
4.4.2	Przygotowanie CI/CD Pipeline przy pomocy platformy CircleCI . . . . .	16

4.4.3	Zastosowanie DevOps lifecycle od strony GitHub . . . . .	23
<b>5</b>	<b>Projekt i implementacja aplikacji internetowej w technologii Vue.js</b>	<b>25</b>
5.1	Narzędzia i technologie . . . . .	25
5.1.1	Node.js . . . . .	25
5.1.2	Vue.js . . . . .	25
5.1.3	Vuex . . . . .	25
5.1.4	Jest . . . . .	26
5.1.5	Json Web Tokens . . . . .	26
5.1.6	Postman . . . . .	26
5.1.7	Visual Studio Code . . . . .	27
5.1.8	Axios . . . . .	27
5.1.9	Bootstrap . . . . .	27
5.2	Połączenie z backendem . . . . .	27
<b>6</b>	<b>Projekt i implementacja strony serwerowej opartej na architekturze REST w technologii Django oraz bazy danych MySQL</b>	<b>29</b>
6.1	Narzędzia i technologie . . . . .	29
6.1.1	Django . . . . .	29
6.1.2	MySQL . . . . .	30
6.1.3	MySQL Workbench . . . . .	30
6.2	Implementacja serwera REST API . . . . .	30
6.2.1	REST . . . . .	30
6.2.2	Modele . . . . .	31
6.2.3	Adresy internetowe . . . . .	32
6.2.4	Widoki . . . . .	32
6.3	Główne funkcjonalności . . . . .	33
6.3.1	Rejestracja i logowanie . . . . .	33
6.3.2	Uzupełnianie danych potrzebnych do generacji planu . . . . .	34
6.3.3	Generacja oraz zwracanie planów . . . . .	34
6.4	Baza danych MySQL . . . . .	34
<b>7</b>	<b>Projekt i implementacja algorytmu generującego plan lekcji</b>	<b>37</b>
7.1	Wstęp . . . . .	37
7.2	Przyjęte założenia . . . . .	38
7.3	Narzędzia . . . . .	38
7.4	Zastosowanie algorytmu ewolucyjnego . . . . .	38
7.5	Przygotowanie danych wejściowych . . . . .	38
7.6	Ułożenie poprawnych planów zajęć . . . . .	39
7.7	Funkcja oceny . . . . .	39
7.8	Ewolucja planów zajęć . . . . .	39
7.9	Wielowątkowość . . . . .	41
7.10	Parametry . . . . .	42
<b>8</b>	<b>Testowanie</b>	<b>43</b>
8.1	Frontend . . . . .	43
8.2	Backend . . . . .	44
8.2.1	Testowanie manualne . . . . .	44

8.2.2	Testy automatyczne . . . . .	44
8.3	Algorytm . . . . .	45
8.3.1	Testy manualne . . . . .	45
8.3.2	Testy automatyczne . . . . .	45
8.3.3	Testy jakościowe i wydajnościowe . . . . .	46
<b>9</b>	<b>Instrukcja użytkowania</b>	<b>51</b>
9.1	Strona główna . . . . .	51
9.2	Widok szkoły . . . . .	51
9.3	Rejestracja . . . . .	52
9.4	Logowanie . . . . .	52
9.5	Ankieta . . . . .	53
9.6	Dodawanie przedmiotów . . . . .	53
9.7	Dodawanie nauczycieli . . . . .	54
9.8	Dodawanie sal lekcyjnych . . . . .	54
9.9	Dodawanie klas . . . . .	55
9.10	Edycja danych . . . . .	55
<b>10</b>	<b>Zakończenie</b>	<b>56</b>
10.1	Podsumowanie pracy . . . . .	56
10.2	Możliwości rozwoju aplikacji . . . . .	56
	<b>Literatura</b>	<b>58</b>
<b>A</b>	<b>Załączniki</b>	<b>60</b>



# Rozdział 1

## Wstęp

W dzisiejszych czasach trudno znaleźć osobę która nigdy nie korzystała z Internetu. Większość naszego społeczeństwa używa go codziennie. Wykorzystywany jest prawie w każdej dziedzinie życia, służyć może np. do komunikacji w czasie rzeczywistym, dokonywania szybkich płatności, zakupów internetowych, nauki czy też pracy. Przykłady można wymieniać bez końca, jednak trzeba zwrócić szczególną uwagę na narzędzia, dzięki którym możemy korzystać z sieci w tak szerokim zakresie. Jednym z najpopularniejszych wykorzystywanych oraz dynamicznie rozwijanych rozwiązań są aplikacje webowe. Jedną z ich głównych zalet jest możliwość korzystania z nich niezależnie od używanego urządzenia, o ile ma ono dostęp do Internetu oraz posiada przeglądarkę internetową. W przeciwieństwie do aplikacji desktopowych, wszelkie aktualizacje są dokonywane przez administratora, co w kontekście rozwoju oprogramowania jest bardzo wygodne zarówno dla programisty jak i użytkownika. Są to jedne z wielu zalet, które z pewnością przyczyniły się do szybkiego rozwoju aplikacji internetowych oraz związanych z nimi technologii.

Tematem podjętym w pracy jest aplikacja służąca do generowania planów zajęć. Zaprojektowanie tego typu rozwiązania, daje możliwość nauki rozmaitych technologii informatycznych, powszechnie wykorzystywanych w praktyce. Główną motywacją do podjęcia takiego tematu stanowią wady obecnie stosowanego przez większość szkół manualnego tworzenia planów zajęć. Ręczne tworzenie planu jest czasochłonne i wymaga dużego nakładu pracy. Dla osób odpowiedzialnych za ich przygotowanie (dalej zwanych planistami) jest to zadanie monotonne, a także przytłaczające. Planiści, nawet ci z dużym doświadczeniem, nie są zdolni do utworzenia planu, który optymalnie wykorzystywałby godziny uczniów, nauczycieli, a także dostępność sal lekcyjnych. Skutkuje to znaczną liczbą niewykorzystanego czasu w środku dnia lekcyjnego.

Celem pracy jest zaprojektowanie aplikacji, dzięki której po podaniu niezbędnych danych, możliwe byłoby automatyczne wygenerowanie planu zajęć dla szkoły. Aplikacja ma umożliwić planiście dodawanie danych o przedmiotach, nauczycielach, salach i klasach. Na podstawie podanych danych planista ma mieć możliwość generacji rozkładu zajęć dla wszystkich klas w szkole. Aplikacja ma być przeznaczona dla szkół podstawowych oraz średnich. Ograniczenie to wynika z założenia niepodzielności klasy. W przypadku uczelni wyższych niejednorodny podział na grupy znacząco zwiększa poziom skomplikowania rozwiązywanego problemu.

Projekt można podzielić na cztery główne części: konfigurację infrastruktury informatycznej, implementację back-end, implementację front-end oraz implementację algorytmu.

Praca ma następującą strukturę. Rozdział drugi poświęcony jest podstawom teoretycznym. Rozdział trzeci zawiera analizę problemu i dostępnych rozwiązań. Rozdział czwarty to opis metodyki pracy oraz infrastruktury informatycznej. Rozdział piąty omawia część frontendową aplikacji. Rozdział szósty charakteryzuje backend aplikacji. Rozdział siódmy wyjaśnia działanie algorytmu

generacji planu. Rozdział ósmy to opis testów. Rozdział dziewiąty jest instrukcją użytkowania. Rozdział dziesiąty stanowią wnioski.

Implementacja aplikacji została wykonana przez cztery osoby. Mateusz Biernacki zaimplementował algorytm ewolucyjny generujący plan lekcji i jest autorem rozdziału siódmego oraz dziesiątego. Dominik Boła zaimplementował aplikację serwerową opartą na architekturze REST i jest autorem rozdziału pierwszego oraz szóstego. Maciej Goral zaimplementował aplikację internetową w technologii Vue.js i jest autorem rozdziału trzeciego, piątego oraz dziewiątego. Grzegorz Piątkowski zaimplementował algorytm ewolucyjny generujący plan lekcji, przygotował infrastrukturę informatyczną, jest autorem rozdziału drugiego oraz czwartego. Pozostałe rozdziały zostały wykonane wspólnie.



## Rozdział 2

# Podstawy teoretyczne

### 2.1 Problem optymalizacyjny

Problem optymalizacyjny [23] jest to problem obliczeniowy, który polega na znalezieniu maksymalnej/minimalnej wartości pewnego parametru. Wartość takiego parametru zazwyczaj opisywana jest funkcją, dzięki której wartość parametru zależy od przeszukiwanych danych wejściowych. Jeśli poszukiwana jest jak najmniejsza wartość parametru, mówimy o problemie minimalizacyjnym i odpowiednio w przypadku poszukiwania największej wartości parametru, mówimy o problemie maksymalizacyjnym.

### 2.2 Problem NP-trudny

Problem NP-trudny [21] jest problemem obliczeniowym, dla którego nie jest możliwym znalezienie rozwiązania w czasie wielomianowym przy wykorzystaniu niedeterministycznej maszyny Turinga, a sprawdzenie znalezionej rozwiązania jest co najmniej tak trudne jak każdego innego problemu z grupy NP. Problem optymalizacyjny jest jednym z problemów należących do grupy NP-trudnych.

### 2.3 Heurystyka

Heurystyka [14] jest techniką rozwiązywania problemów w przypadku, gdy znalezienie dokładnego rozwiązania jest zbyt kosztowne. Metoda heurystyczna oferuje zmniejszenie kosztów rozwiązania problemu, jednak ceną takiego podejścia jest spadek dokładności rozwiązania czy nawet jego poprawności. Przy wykorzystaniu metody heurystycznej otrzymanie optymalnego rozwiązania możliwe jest tylko w szczególnych przypadkach. Tego typu podejście wykorzystuje się również, w przypadku, gdy algorytm dokładny umożliwiające znalezienie rozwiązania optymalnego nie jest znany, w celu zawężenia pola badań.

### 2.4 Algorytm ewolucyjny

Algorytm ewolucyjny [15] jest heurystycznym podejściem do rozwiązywania problemów, które nie mogą zostać rozwiązane w czasie wielomianowym, takie jak grupa problemów NP-trudnych, czy po prostu w celu zmniejszenia kosztów znalezienia rozwiązania problemu. Algorytmy ewolucyjne stosowane samodzielnie używane są zazwyczaj do rozwiązywania problemów optymalizacyjnych. Zastosowanie i działanie algorytmu ewolucyjnego jest bardzo proste do zrozumienia ze względu na

to, że mamy do czynienia na co dzień z podobnym zjawiskiem w naturze czyli z selekcją naturalną. Przebieg działania algorytmu ewolucyjnego składa się z 4 głównych kroków.

1. **Inicjalizacja** – W celu rozpoczęcia działania algorytmu, potrzebna jest pierwsza grupa rozwiązań (dalej nazywana populacją). Populacja zawierać będzie założoną liczbę możliwych rozwiązań (dalej nazywaną osobnikami). Zazwyczaj podczas inicjalizacji osobniki tworzone są w sposób losowy. Takie podejście jest wręcz zalecane, ponieważ umożliwia to przebadanie dużej różnorodności osobników, dzięki czemu finałowe rozwiązanie będzie lepsze.
2. **Selekcja** – Gdy pierwotna populacja jest gotowa, jej osobniki trzeba poddać ocenie. Funkcja oceny powinna składać się ze ściśle opisanych warunków opisujących środowisko, do którego osobniki muszą się przystosować. Im dokładniej środowisko zostanie opisane w funkcji oceny, tym lepsze będzie finałne rozwiązanie. Gdy funkcja jest poprawnie przygotowana, każdy z osobników musi zostać poddany ocenie, po której otrzymuje parametr oceny. Dzięki temu można wyróżnić rozwiązania lepsze od reszty. Z populacji zostaje wybrana założona liczba osobników o najwyższym parametrze oceny. Reszta osobników zostaje zabita.
3. **Ewolucja** – Ewolucja składa się z dwóch kroków: krzyżowania oraz mutacji.
  - a) Krzyżowanie – po otrzymaniu wybranych osobników z selekcji, użyte są one do stworzenia nowego pokolenia dla algorytmu, stając się osobnikami-rodzicami. Wykorzystując charakterystyki osobników-rodziców, utworzona zostaje populacja osobników-dzieci poprzez wymieszanie ze sobą charakterystyk osobników-rodziców. Po utworzeniu nowego pokolenia osobników-dzieci, osobniki-rodzice zostają zabite.
  - b) Mutacja – jest to prawdopodobnie najważniejszy krok ewolucji. Bez niego cała populacja bardzo szybko utknęłaby w miejscu, nie oferując żadnego sensownego rozwiązania. W tym kroku charakterystyka każdego osobnika-dziecka z nowego pokolenia poddana jest małym losowym zmianom w celu zróżnicowania ich od osobników-rodziców. Na końcu tego kroku osobniki-dzieci stają się nowym pokoleniem osobników w populacji, która może ponownie zostać poddana selekcji.
4. **Finalizacja** – Ostatecznie działanie algorytmu musi dobiec końca. W tym kroku z populacji zostaje wybrany osobnik z najwyższym parametrem oceny i zwrócony jako rozwiązanie. Są dwie możliwości, w których zakończenie działania algorytmu może zostać wywołane. Gdy osiągnie on maksymalny czas działania (np. założona maksymalna liczba pokoleń) lub gdy osiągnięty zostanie poszukiwany pułap parametru oceny.

## Rozdział 3

# Analiza i porównanie możliwych rozwiązań

### 3.1 Analiza problemu

Podstawowym problemem w automatycznym tworzeniu planu zajęć jest dobór warunków wykorzystywanych przy generacji. Warunki te można podzielić na niezbędne do utworzenia poprawnego planu oraz warunki dodatkowe, których spełnienie zwiększa użyteczność planu z punktu widzenia planisty.

Wśród warunków niezbędnych należy wyróżnić warunek braku konfliktów. Konflikt ma miejsce, gdy występuje jedna z następujących sytuacji:

- w jednej godzinie lekcyjnej, jednej klasie został przyporządkowany więcej niż jeden przedmiot,
- w jednej godzinie lekcyjnej, jednemu nauczycielowi została przyporządkowana więcej niż jedna klasa,
- w jednej godzinie lekcyjnej, jednej sali została przyporządkowana więcej niż jedna klasa.

W przypadku szkół podstawowych oraz średnich do warunków niezbędnych należy również zaliczyć brak niewykorzystanych godzin w środku dnia lekcyjnego uczniów. Dodatkowo niektóre zajęcia, takie jak wychowanie fizyczne, mogą być przeprowadzone tylko w specjalnie przeznaczonych do tego salach.

Warunki dodatkowe mogą różnić się w zależności od czynników, które należy wziąć pod uwagę przy pod uwagę przy generacji plany wynikających ze specyfikacji szkoły oraz wymagań personelu dydaktycznego. Do tych czynników można zaliczyć:

- ograniczenia dostępności nauczycieli, wynikające z pracy w innych placówkach oświatowych lub innych powodów,
- ograniczenia wynikające z odległości między salami,
- obecność zajęć nieobowiązkowych, które muszą w danym dniu lekcyjnym być skrajnie pierwsze lub ostatnie,
- konieczność minimalizacji niewykorzystanych godzin w środku dnia lekcyjnego dla uczniów,
- konieczność grupowania zajęć w przypadku kilku godzin lekcyjnych tego samego przedmiotu jednego dnia – w takim przypadku zajęcia te powinny następować bezpośrednio po sobie oraz w tej samej sali,

- konieczność równomiernego rozłożenie przedmiotów w trakcie tygodnia lekcyjnego.

## 3.2 Aktualnie dostępne rozwiązania

### 3.2.1 aSc TimeTables

aSc TimeTables [2] to aplikacja desktopowa wspomagająca przygotowywanie planów zajęć. Narzędzie umożliwia generowanie planów na podstawie zdefiniowanych wymagań, wprowadzenie do nich ręcznych poprawek oraz wyszukiwanie konfliktów we wprowadzonych zmianach. aSc TimeTables jest najbardziej rozbudowanym rozwiązaniem tego typu dostępnym na rynku, pozwalającym na tworzenie planów zajęć dla szkół i uczelni. Do dodatkowych funkcji programu należy możliwość importu danych z pliku, zdolność mapowania szkoły oraz udostępnienia planów uczniom i nauczycielom za pomocą aplikacji mobilnej. Z wszechstronnością i bogactwem funkcji wiąże się wysoki poziom umiejętności potrzebny do poprawnego wykorzystania aplikacji. Do pozostałych wad programu należy brak regularnych aktualizacji, podatność na błędy w generacji planu, wysoka cena oraz dostępność ograniczona do systemu Windows.

### 3.2.2 Prime Timetable

Prime Timetables [25] to aplikacja internetowa przeznaczona dla organizacji edukacyjnych umożliwiająca zarówno ręczne jak i automatyczne układanie planów lekcji. Prime Timetables pozwala na wspólne tworzenie planów przez kilku użytkowników oraz udostępnianie gotowych planów dla uczniów i nauczycieli posiadających konta w serwisie. Aplikacja posiada rozbudowany zestaw narzędzi umożliwiających określanie ograniczeń związanych z automatyczną generacją planu. Główną wadą rozwiązania jest wysoka opłata miesięczna, której wysokość dodatkowo zależy od liczby nauczycieli w szkole.

### 3.2.3 SuperSaas

SuperSaas [35] to program do zarządzania szkołami i innymi instytucjami, którego głównym atutem jest wbudowany system rezerwacji. Przy pomocy konta WordPress użytkownicy aplikacji mogą umawiać terminy wizyt, a także dokonywać za nie płatności. SuperSaas cechuje niska cena oraz dostępność z poziomu przeglądarki. Duża część funkcjonalności aplikacji nie jest przeznaczona dla szkół. Pomimo możliwości wspomagania ręcznego układania planów zajęć, program nie pozwala na automatyczną ich generację, ani nawet wykrywanie konfliktów.

## 3.3 Możliwe podejścia

Możliwe rozwiązania można podzielić w zależności od kilku aspektów. Pierwszym z nich jest wybór rodzaju aplikacji – desktopowej, mobilnej lub internetowej. Ze względu na fakt, że korzystanie z aplikacji wymagać ma wprowadzania dużej ilości danych, można założyć, że z punktu widzenia użytkownika najkorzystniejsze będzie użycie w tym celu fizycznej klawiatury. Powoduje to odrzucenie wyboru aplikacji mobilnej. Zaletami wyboru aplikacji desktopowej jest możliwość korzystania z niej bez dostępu do Internetu oraz bezpieczeństwo związane z lokalnym przechowywaniem danych. Pomimo tych korzyści rozwiązanie to nie oferuje zalet związanych z wyborem aplikacji internetowej – dostępu z dowolnego urządzenia wyposażonego w kompatybilną przeglądarkę, braku wymagań systemowych związanych z obliczeniami i przechowywaniem danych oraz

braku konieczności aktualizowania aplikacji przez użytkownika. Te czynniki jednoznacznie przesądają o wykorzystaniu w projekcie aplikacji internetowej.

Drugim aspektem, który należy rozpatrzyć jest poziom złożoności aplikacji, związany głównie z wyborem możliwych do wprowadzenia przez planistę warunków wykorzystywanych przy generacji planu. Większa liczba warunków może wiązać się z lepszą jakością planu, ale także dłuższym czasem jego generacji. Należy również rozpatrzyć stosunek nakładu pracy przeznaczonej na implementację warunku w stosunku do potencjalnego zysku dla użytkowników. Każdy dodatkowy warunek spowodowałby zmniejszenie przejrzystości interfejsu graficznego, szczególnie dla użytkowników, dla których byłby on zbędny. Biorąc pod uwagę te czynniki w projekcie wykorzystane zostaną jedynie te warunki, których obecność jest niezbędna do poprawnej generacji planu.

Ostatnim ważnym do rozpatrzenia aspektem jest wybór użytkowników, którzy będą posiadać konta w systemie. Pierwszym z nich jest planista – osoba odpowiedzialna za układanie planów. W rozwiązaniu, w których planiści nie posiadają kont, wprowadzone informacje nie byłyby przechowywane w bazie w danych, co oznacza ich utratę przypadku zakończenia sesji. Konieczność posiadania przez planistę konta eliminuje ten problem i dodatkowo utrudnia ataki na stronę. Pozostali potencjalni użytkownicy to nauczyciele oraz uczniowie. W projekcie zakładamy możliwość podania przez nauczyciela swojej dyspozycyjności. Ze względu na konieczność podania przez planistów adresu e-mail każdego dodanego nauczyciela, jest to możliwe poprzez unikatowy link wysłany w wiadomości, bez obowiązku zakładania konta. Dla uczniów, ze względu na brak bezpośredniego wpływu na plan zajęć, również nie zakłada się możliwości stworzenia konta.

### 3.4 Wymagania funkcjonalne i нефункционалне

Wymagania funkcjonalne:

- możliwość rejestracji w aplikacji,
- możliwość logowania w aplikacji,
- możliwość dodania danych przedmiotów, nauczycieli, sal i klas,
- możliwość edycji danych przedmiotów, nauczycieli, sal i klas,
- możliwość usunięcia danych przedmiotów, nauczycieli, sal i klas,
- możliwość generacji planu na podstawie podanych danych,
- możliwość wyświetlania wygenerowanych planów z podziałem na plany dla klas, nauczycieli i sale,
- możliwość przesłania do nauczycieli ankiet dyspozycyjności,
- możliwość wypełnienia ankiety dyspozycyjności przez nauczyciela.

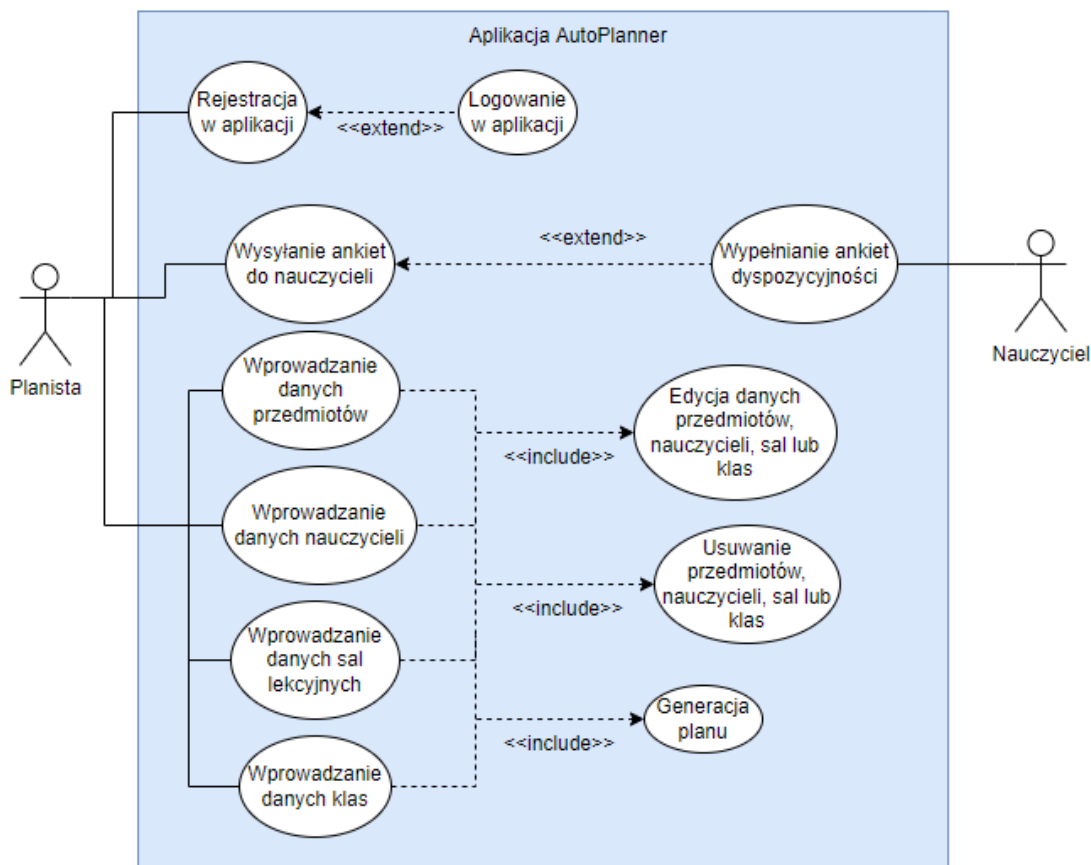
Wymagania нефункционалне:

- responsywność na urządzeniach mobilnych,
- uwierzytelnianie oparte o tokeny JWT,
- możliwość przerywania dodawania danych bez utraty postępu,
- kompatybilność z przeglądarkami Chrome, Firefox, Opera oraz Edge,

- formularze dodawania danych proste i intuicyjne w obsłudze,
- szyfrowanie danych w bazie danych.

### 3.5 Przypadki użycia

Na rysunku 3.1 przedstawiono diagram przypadków użycia.



RYСУNEK 3.1: Diagram przypadków użycia

**Przypadek użycia:** Rejestracja w aplikacji

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje adres e-mail.
2. Aplikacja nie zgłasza problemu ze składnią adresu e-mail.
3. Planista wpisuje nazwę użytkownika oraz hasło.
4. Planista zatwierdza wprowadzone dane.
5. Aplikacja akceptuje wprowadzone dane.
6. Aplikacja tworzy nowe konto użytkownika.

**Rozszerzenia:**

- 2.A Aplikacja zgłasza problem ze składnią adresu e-mail.
  - 2.A.1 Planista poprawia składnię adresu e-mail.
- 5.A Aplikacja zgłasza problem dotyczący wymagań hasła.
  - 5.A.1 Planista wpisuje hasło, które spełnia wymagania.
- 5.B Aplikacja zgłasza, że na wpisany adres e-mail jest już założone konto.
  - 5.B.1 Planista wpisuje nowy adres e-mail.

**Przypadek użycia:** Logowanie w aplikacji

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje adres e-mail.
2. Planista wpisuje hasło.
3. Planista zatwierdza wprowadzone dane.
4. Aplikacja akceptuje wprowadzone dane.
5. Aplikacja przechodzi do widoku użytkownika zalogowanego.

**Rozszerzenia:**

- 4.A Aplikacja zgłasza, że wprowadzone hasło jest nieprawidłowe.
  - 4.A.1 Planista ponownie wpisuje hasło.
- 4.B Aplikacja zgłasza, że użytkownik o podanym adresie e-mail nie istnieje.
  - 4.B.1 Planista ponownie wpisuje adres e-mail.

**Przypadek użycia:** Wprowadzanie danych przedmiotów

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje nazwę przedmiotu.
2. Planista zatwierdza wprowadzone dane.
3. Aplikacja akceptuje wprowadzone dane.
4. Aplikacja dodaje przedmiot do listy z lewej strony ekranu.

**Rozszerzenia:**

- 3.A Aplikacja zgłasza, że przedmiot o danej nazwie został już wcześniej wprowadzony.
  - 3.A.1 Planista wpisuje nową nazwę przedmiotu.
- 3.B Aplikacja zgłasza, że nazwa przedmiotu zawiera niedozwolone znaki.
  - 3.B.1 Planista wpisuje nową nazwę przedmiotu.

**Przypadek użycia:** Wprowadzanie danych nauczycieli

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje imię i nazwisko nauczyciela.
2. Planista wpisuje adres e-mail nauczyciela.
3. Aplikacja nie zgłasza problemu ze składnią adresu e-mail.
4. Planista wybiera z listy wprowadzony przez nauczyciela przedmiot.
5. Aplikacja akceptuje wprowadzone dane.
6. Aplikacja dodaje nauczyciela do listy z lewej strony ekranu.

**Rozszerzenia:**

- 3.A Aplikacja zgłasza problem ze składnią adresu e-mail.
  - 3.A.1 Planista poprawia składnię adresu e-mail.
- 4.A Planista dodaje kolejne przedmioty prowadzone przez nauczyciela.
- 5.A Aplikacja zgłasza, że nauczyciel o danym adresie e-mail został już wcześniej wprowadzony.
  - 5.A.1 Planista wpisuje nowy adres e-mail.
- 5.B Aplikacja zgłasza, że imię lub nazwisko nauczyciela zawiera niedozwolone znaki.
  - 5.B.1 Planista wpisuje nowe imię i nazwisko nauczyciela.

**Przypadek użycia:** Wprowadzanie danych sal lekcyjnych

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje nazwę sali.
2. Planista nie dodaje preferowanych przedmiotów.
3. Planista zatwierdza wprowadzone dane.
4. Aplikacja akceptuje wprowadzone dane.
5. Aplikacja dodaje salę do listy z lewej strony ekranu.

**Rozszerzenia:**

- 2.A Planista dodaje jeden lub więcej preferowany przedmiot.
- 4.A Aplikacja zgłasza, że sala o danej nazwie została już wcześniej wprowadzona.
  - 4.A.1 Planista wpisuje nową nazwę sali.
- 4.B Aplikacja zgłasza, że nazwa sali zawiera niedozwolone znaki.
  - 4.B.1 Planista wpisuje nową nazwę sali.



**Przypadek użycia:** Wprowadzanie danych klas

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wpisuje nazwę klasy.
2. Planista wybiera z listy kolejne przedmioty.
3. Planista wpisuje tygodniowe liczby godzin dla każdego przedmiotu.
4. Planista nie wybiera prowadzącego dany przedmiot.
5. Planista zatwierdza wprowadzone dane.
6. Aplikacja akceptuje wprowadzone dane.
7. Aplikacja dodaje klasę do listy z lewej strony ekranu.

**Rozszerzenia:**

- 4.A Planista wybiera z listy prowadzącego przedmiot.
- 6.A Aplikacja zgłasza, że klasa o danej nazwie została już wcześniej wprowadzona.
  - 6.A.1 Planista wpisuje nową nazwę klasy.
- 6.B Aplikacja zgłasza, że nazwa klasy zawiera niedozwolone znaki.
  - 6.B.1 Planista wpisuje nową nazwę klasy.
- 6.C Aplikacja zgłasza, że wprowadzona liczba godzin ma nieprawidłowy format.
  - 6.C.1 Planista wpisuje nową liczbę godzin.

**Przypadek użycia:** Edycja danych przedmiotów, nauczycieli, sal lub klas

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wybiera przedmiot, nauczyciela, salę lub klasę, których dane chce edytować.
2. Aplikacja przechodzi do ekranu edycji z aktualnymi danymi wybranego przedmiotu, nauczyciela, sali lub klasy.
3. Planista edytuje dane.
4. Planista zatwierdza edycję danych.
5. Aplikacja akceptuje edycję danych.
6. Aplikacja powraca do ekranu dodawania przedmiotu, nauczyciela, sali lub klasy.

**Rozszerzenia:**

- 5.A Aplikacja zgłasza, że nowe dane są nieprawidłowe, w ten sam sposób jak miało to miejsce w przypadku ich dodawania.
  - 5.A.1 Planista poprawia wpisane dane.

**Przypadek użycia:** Usuwanie przedmiotów, nauczycieli, sal lub klas

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wybiera przedmiot, nauczyciela, salę lub klasę, których dane chce edytować.
2. Aplikacja przechodzi do ekranu edycji z aktualnymi danymi wybranego przedmiotu, nauczyciela, sali lub klasy.
3. Planista wybiera opcję usunąć przedmiot, nauczyciela, salę lub klasę.
4. Aplikacja usuwa przedmiot, nauczyciela, salę lub klasę z listy z lewej strony ekranu.

**Przypadek użycia:** Wysyłanie ankiet do nauczycieli

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wybiera opcję 'wyślij ankiety dyspozycyjności' na ekranie dodawania nauczycieli.
2. Aplikacja wysyła wiadomości e-mail z ankietami dyspozycyjności na adresy e-mail wszystkich do tej pory dodanych nauczycieli.
3. Aplikacja wyświetla informację, że ankiety zostały wysłane.

**Przypadek użycia:** Wypełnienie ankiety dyspozycyjności

**Aktor główny:** Nauczyciel

**Scenariusz główny:**

1. Nauczyciel poprzez link w wiadomości e-mail przechodzi do ekranu wypełniania ankiety.
2. Nauczyciel zaznacza w siatce godzin swoją dyspozycyjność.
3. Nauczyciel zatwierdza wprowadzone dane.
4. Aplikacja zapisuje dane w celu wykorzystania przy generacji planu.

**Przypadek użycia:** Generacja planu

**Aktor główny:** Planista

**Scenariusz główny:**

1. Planista wybiera opcję 'generuj plan'.
2. Aplikacja rozpoczyna generację planu i przechodzi do ekranu oczekiwania.
3. Aplikacja kończy generację planu i przechodzi do ekranu szkoły.
4. Planista przegląda wygenerowane plany z podziałem na plany klas, nauczycieli i sal lekcyjnych.

## Rozdział 4

# Metodyka pracy oraz przygotowanie infrastruktury informatycznej

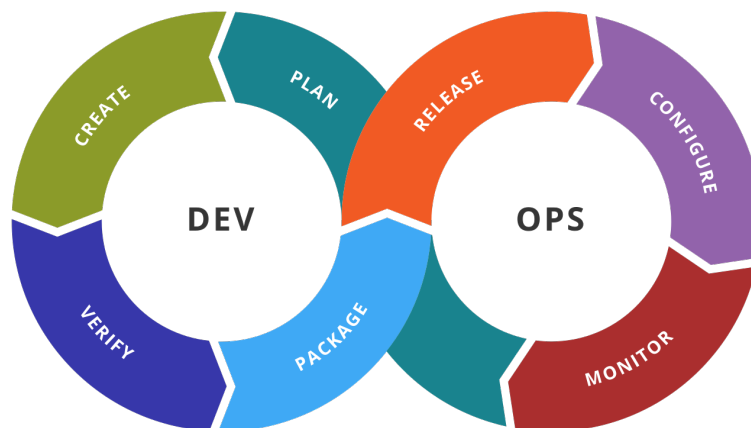
### 4.1 Wstęp

Projekt powstawał w metodyce DevOps. Takie podejście pozwoliło na szybsze dostarczenie finalnego produktu. Wysoki poziom kooperacji wynikający z metodyki DevOps pozwolił na zmniejszenie kosztów dostarczenia produktu oraz znaczne zwiększenie jego spójności. Potrzebna jest jednak mocno rozwinięta infrastruktura informatyczna służąca podtrzymaniu DevOps lifecycle.

### 4.2 Pojęcia

#### 4.2.1 DevOps

DevOps [9] to zestaw praktyk, narzędzi i filozofii kulturowej, które automatyzują i integrują procesy pomiędzy zespołami programistów oraz IT. Kładzie nacisk na wzmocnienie pozycji zespołu, komunikację i współpracę między zespołami oraz automatyzację technologii. Ruch DevOps rozpoczął się około 2007 roku, kiedy społeczności programistów i operatorów IT wyraziły zaniepokojenie tradycyjnym modelem rozwoju oprogramowania, w którym programiści piszący kod pracowali oddzielnie od operatorów, którzy wdrażali i wspierali kod. Termin DevOps, będący połączeniem słów *development* i *operations*, odzwierciedla proces integracji tych dyscyplin w jeden, ciągły proces (zob. rysunek 4.1).



RYСУNEK 4.1: Schemat DevOps lifecycle [26]

### 4.2.2 Continuous Integration and Continuous Deployment

*Continuous Integration and Continuous Deployment* (CI/CD) [6] to metoda częstego dostarczania aplikacji do klientów poprzez wprowadzenie automatyzacji do etapów tworzenia aplikacji. Główne pojęcia przypisane do CI/CD to ciągła integracja, ciągłe dostarczanie i ciągłe wdrażanie. CI/CD jest rozwiązaniem problemów, jakie integracja nowego kodu może powodować dla zespołów programistycznych i operacyjnych. W szczególności, CI/CD wprowadza ciągłą automatyzację i ciągłe monitorowanie w całym cyklu życia aplikacji, od fazy integracji i testowania po dostarczanie i wdrażanie. Łącznie, te połączone praktyki są często określane jako CI/CD i są wspierane przez zespoły programistów i operatorów pracujących razem z podejściem DevOps lub SRE (*site reliability engineering*).

### 4.2.3 Kontrola wersji

Kontrola wersji [38], znana również jako kontrola źródła, jest praktyką śledzenia i zarządzania zmianami w kodzie oprogramowania. Systemy kontroli wersji to narzędzia programowe, które pomagają zespołom programistów zarządzać zmianami w kodzie źródłowym w czasie.

## 4.3 Narzędzia i technologie

### 4.3.1 Amazon Web Services

*Amazon Web Services* (AWS) [1] jest spółką zależną firmy Amazon, dostarczającą platformy chmury obliczeniowej na żądanie oraz interfejsy API osobom prywatnym, firmom i rządowi na zasadzie *pay-as-you-go*. Te usługi internetowe w chmurze obliczeniowej zapewniają różnorodne podstawowe abstrakcyjne elementy infrastruktury technicznej oraz narzędzia i bloki do obliczeń rozproszonych. Jedną z tych usług jest *Amazon Elastic Compute Cloud* (EC2), która pozwala użytkownikom mieć do dyspozycji wirtualny klaster komputerów, dostępny przez cały czas, przez Internet. Wirtualne komputery AWS emulują większość atrybutów prawdziwego komputera, w tym sprzętowe jednostki centralne (CPU) i procesory graficzne (GPU) do przetwarzania danych, pamięć lokalną/RAM, pamięć masową HDD/SSD, wybór systemów operacyjnych, sieci oraz wstępnie załadowane oprogramowanie użytkowe, takie jak serwery internetowe, bazy danych i zarządzanie relacjami z klientami (CRM).

### 4.3.2 Git

Git [12] to darmowe narzędzie open-source służące do kontroli wersji, zaprojektowane do obsługi wszystkiego, od małych do bardzo dużych projektów z dużą prędkością i wydajnością.

### 4.3.3 GitHub

GitHub [13] jest dostawcą hostingu internetowego dla rozwoju oprogramowania i kontroli wersji przy użyciu Git. Oferuje on funkcje rozproszonej kontroli wersji i zarządzania kodem źródłowym (SCM) Git, a także własne funkcje.

### 4.3.4 Docker

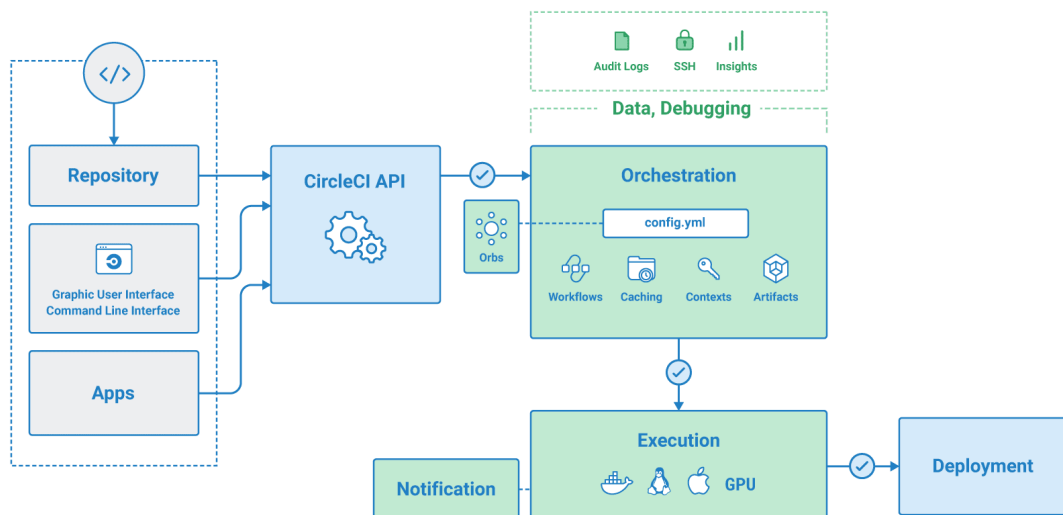
Docker [11] to platforma konteneryzacji typu open source. Umożliwia ona programistom pakowanie aplikacji w kontenery – ustandaryzowane komponenty wykonywalne łączące kod źródłowy aplikacji z bibliotekami systemu operacyjnego (OS) i zależnościami wymaganymi do uruchomienia

tego kodu w dowolnym środowisku. Kontenery upraszczają dostarczanie aplikacji rozproszonych i stają się coraz bardziej popularne w miarę jak organizacje przechodzą na rozwój cloud-native i hybrydowe środowiska wielochmurowe.

#### 4.3.5 CircleCI

CircleCI [7] jest platformą obsługującą *Continuous Integration and Continuous Delivery* (CI/CD), która pomaga zespołom programistycznym szybko i pewnie wypuszczać kod poprzez możliwość tworzenia *pipeline* (brak odpowiednika w języku polskim) automatyzujących proces budowania, testowania i wdrażania. Pozwala to zespołom szybko się rozwijać, łatwo skalować i budować spójne produkty (zob. rysunek 4.2).

- *Pipeline* jest jednostką pracy najwyższego poziomu, obejmującą cały plik „circleci/config.yml” projektu. *Pipeline* zawiera przepływy pracy (*workflow*), które koordynują zadania. Mają one ustalony, liniowy cykl życia i są powiązane z konkretnym aktorem. *Pipeline* uruchamia się po wprowadzeniu zmiany do projektu, który zawiera plik konfiguracyjny CircleCI, a także może być zaplanowany, uruchamiany ręcznie za pomocą aplikacji CircleCI lub interfejsu API.



RYСУNEK 4.2: Diagram przedstawiający działanie CircleCI [8]

#### 4.3.6 Secure Shell

*Secure Shell* (SSH) [33] to protokół zdalnej administracji, który pozwala użytkownikom kontrolować i modyfikować zdalne serwery przez Internet. Usługa ta została stworzona jako bezpieczny zamiennik dla nieszyfrowanego protokołu Telnet i wykorzystuje techniki kryptograficzne, aby zapewnić szyfrowaną komunikację ze zdalnym serwerem. Zapewnia mechanizm uwierzytelniania zdalnego użytkownika, przesyłania danych wejściowych od klienta do hosta i przekazywania danych wyjściowych z powrotem do klienta.

#### 4.3.7 tmux

tmux [36] jest multiplekserem terminali. Umożliwia tworzenie, dostęp i sterowanie wieloma terminalami z jednego ekranu. tmux może zostać odłączony od ekranu i kontynuować pracę w tle,

a następnie ponownie dołączony.

#### 4.3.8 shell

*Shell* [32] to termin UNIX dla interaktywnego interfejsu pośredniczącego między użytkownikiem a systemem operacyjnym. *Shell* jest warstwą programowania, która rozumie i wykonuje polecenia wprowadzane przez użytkownika. W niektórych systemach, powłoka nazywana jest interpreterem poleceń. *Shell* zwykle implikuje interfejs ze składnią poleceń.

#### 4.3.9 Black

Black [4] jest formaterem kodu w języku Python. Wykorzystanie gotowego rozwiązania oraz zastosowanie go dla wszystkich plików „.py” standaryzuje kod co znacznie ułatwia pracę nad większym projektem.

#### 4.3.10 Pylint

Pylint [28] jest narzędziem do statycznej analizy kodu w języku Python. Pylint szuka błędów programistycznych, pomaga egzekwować standardy kodowania i oferuje proste sugestie refaktoryzacji.

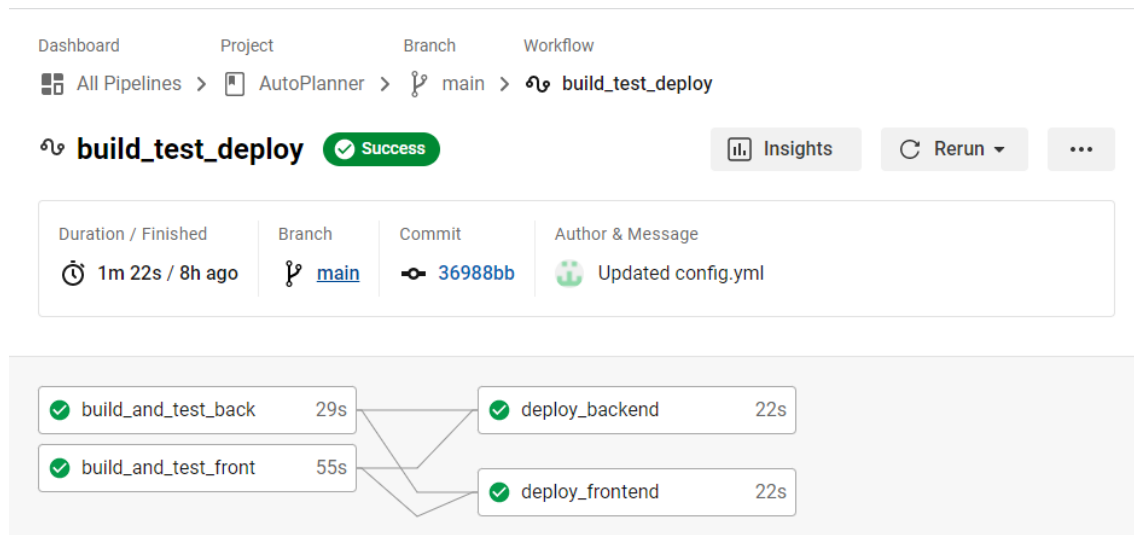
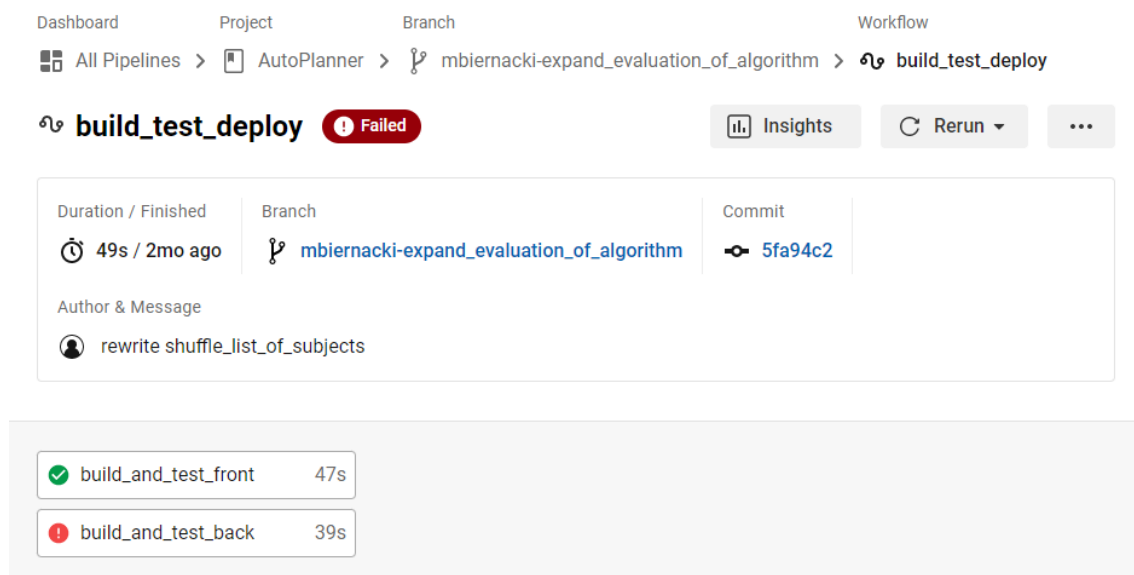
### 4.4 Przygotowanie infrastruktury informatycznej

#### 4.4.1 Przygotowanie strony serwerowej przy pomocy platformy AWS

Do przygotowania strony serwerowej aplikacji wykorzystano jedną z usług platformy *Amazon Web Services* jaką jest *Amazon Elastic Compute Cloud* (EC2). Do przygotowania serwera strony frontendowej oraz backendowej przygotowano dwie niezależne instancje usługi EC2. Obie z przygotowanych instancji są typu *t2.micro* zawierającego się w ramach pakietu *AWS Free Tier*. Dzięki wyborowi tego typu instancji udało się zachować zerowy wkład pieniężny na potrzeby pracy inżynierskiej. Zasoby możliwe do wykorzystania w ramach typu *t2.micro* nie są wystarczające na przypadek skomercjalizowania projektu jednak w zupełności wystarczają na potrzeby przygotowania projektu w ramach pracy inżynierskiej. System operacyjny wykorzystany na obu instancjach EC2 to *Linux Ubuntu 20.04.3 LTS* w wersji serwerowej. Zastosowanie takiej wersji systemu zmniejsza zapotrzebowanie systemu na zasoby obliczeniowe przez ograniczenie procesów systemowych (np. brak GUI) dzięki czemu większa ilość zasobów może zostać udostępniona na potrzeby zadań serwerowych. W celu zwiększenia bezpieczeństwa komunikacji, reguły połączeń przychodzących (*Inbound Rules*) oraz reguły połączeń wychodzących (*Outbound Rules*) zostały skonfigurowane tak, aby jak najbardziej uszczegółwić typy możliwych połączeń z instancjami.

#### 4.4.2 Przygotowanie CI/CD Pipeline przy pomocy platformy CircleCI

Do przygotowania *CI/CD pipeline* wykorzystano platformę CircleCI. Cały proces przygotowano w taki sposób, aby *pipeline* uruchamiany został w momencie wprowadzenia zmian w repozytorium. Zadanie (*job*) budowania (*build*) i testowania (*test*) uruchamiane jest dla każdych zmian wprowadzonych w zdalnym repozytorium. Zadanie wdrażania (*deployment*) uruchamiane jest tylko i wyłącznie, jeśli zmiany zostały wprowadzone w ramach gałęzi Git (*branch*) *main* oraz jeśli zadanie budowania i testowania zakończyło się bez błędów (zob. rysunek 4.3 oraz rysunek 4.4).

RYSUNEK 4.3: Pipeline uruchomiony dla zmian wprowadzonych w gałęzi Git *main*.RYSUNEK 4.4: Pipeline, w którym wystąpiły błędy. Uruchomiony dla zmian wprowadzonych w gałęzi Git innej niż *main*.

Każdy przepływ pracy składa się z zadań, na które składają się jeszcze mniejsze kroki, podczas których wywoływane mogą być polecenia interfejsu *shell*, wykonywane w kontenerach Docker stworzonych na potrzeby wykonania danego zadania. Platforma CircleCI umożliwia tworzenie własnych obrazów Docker jak i zarówno dostarcza szereg gotowych rozwiązań przygotowanych w celu ułatwienia i przyspieszenia tworzenia *pipeline* na potrzeby małych projektów. Na potrzeby projektu inżynierskiego utworzono przepływ pracy *build\_test\_deploy* składający się z 4 zadań *build\_and\_test\_back*, *build\_and\_test\_front*, *deploy\_frontend* oraz *deploy\_backend* (zob. listing 4.1).

- *build\_and\_test\_back* – podczas tego kroku budowany jest kontener Docker na bazie obrazu z przygotowanym Pythonem w wersji 3.9. Na początku tego zadania w obrębie kontenera wykonywane jest przełączenie (*checkout*) na gałąź Git, której zmiany wywołały start *pipeline*. Następnie wykonywany jest krok instalujący wymagania projektowe (*requirements*) z pliku „requirements.txt”. Kolejnym krokiem jest sformatowanie plików „.py” przy pomocy gotowego rozwiązania jakim jest narzędzie Black. Następny krok to sprawdzenie kodu przy pomocy Pylint. Wykorzystanie tego narzędzia zwiększa jakość publikowanego kodu oraz go standaryzuje w przypadku rozwijania oprogramowania przez wiele osób. Dwa ostatnie kroki odpowiedzialne są za uruchomienie testów jednostkowych backendu oraz algorytmu (zob. listing 4.2 oraz rysunek 4.5).
- *build\_and\_test\_front* – w tym zadaniu na początku budowany jest kontener Docker na bazie obrazu z przygotowanym Node.js w wersji 17.2.0. Następnie wykonywane jest przełączenie do gałęzi Git, której zmiany wywołały start *pipeline*. Kolejne dwa kroki odpowiedzialne są za przygotowanie środowiska tj. zaktualizowanie wersji Node.js do *latest* oraz zainstalowanie dependencji projektu. W tak przygotowanym środowisku ostatnim krokiem jest uruchomienie testów jednostkowych z przygotowanego pliku (zob. listing 4.3 oraz rysunek 4.6).
- *deploy\_frontend* – w tym zadaniu głównym narzędziem jest *aws-cli* (*AWS command line interface*). Pozwala on na obsługę instancji EC2 z poziomu interfejsu *shell*. Pierwszym krokiem zadania jest przełączenie do gałęzi Git, której zmiany wywołały wystartowanie *pipeline*. W następnym kroku ustawiany jest *access\_key\_id* oraz *secret\_access\_key* potrzebny do połączenia SSH. Dzięki wykorzystaniu zmiennych środowiskowych platformy CircleCI wartości te pozostają ukryte dla osób nie mających dostępu do projektu z poziomu platformy. Ostatnim, jednak bardzo rozbudowanym krokiem, jest połączenie z użyciem protokołu SSH i wykonanie operacji wdrożenia z poziomu instancji serwera. Po podłączeniu się do maszyny zdalnej wykonywane są takie czynności jak pobranie (*pull*) najnowszych zmian ze zdalnego repozytorium, reinstalacja dependencji oraz restart serwera frontendowego. Po stronie serwera wykorzystywane jest narzędzie *tmux*. Dzięki otwarciu sesji w tle, możliwe jest wykonywanie innych czynności z równoległym działającym serwerem oraz utrzymanie sesji serwera po zakończeniu sesji *SSH* (zob. listing 4.4 oraz rysunek 4.7).
- *deploy\_backend* – podobnie jak w zadaniu *deploy\_frontend* głównym narzędziem jest *aws-cli*. Pierwszym krokiem zadania jest przełączenie do gałęzi Git, której zmiany wywołały wystartowanie *pipeline*. W następnym kroku ustawiany jest *access\_key\_id* oraz *secret\_access\_key* potrzebny do połączenia SSH. Dzięki wykorzystaniu zmiennych środowiskowych platformy CircleCI wartości te pozostają ukryte dla osób nie mających dostępu do projektu z poziomu platformy. Ostatnim krokiem jest połączenie z użyciem protokołu SSH i wykonanie operacji wdrożenia z poziomu instancji serwera. Po rozpoczęciu sesji zdalnej z instancją serwera, wykonywane są takie czynności jak pobranie najnowszych zmian ze zdalnego repozytorium, reinstalacja wymagań projektowych z pliku „requirements.txt” oraz restart serwera backendowego. Po stronie serwera wykorzystywane jest narzędzie *tmux* (zob. listing 4.5 oraz rysunek 4.8).



```

1 workflows:
2   build_test_deploy:
3     jobs:
4       - build_and_test_back
5       - build_and_test_front
6       - deploy_frontend:
7         requires:
8           - build_and_test_back
9           - build_and_test_front
10        filters:
11          branches:
12            only:
13              - main
14       - deploy_backend:
15         requires:
16           - build_and_test_back
17           - build_and_test_front
18        filters:
19          branches:
20            only:
21              - main

```

LISTING 4.1: Część skryptu config.yml odpowiedzialna za określenie przepływów pracy

```

1 build_and_test_back:
2   docker:
3     - image: cimg/python:3.9
4   steps:
5     - checkout
6     - run:
7       name: Install requirements
8       command:
9         sudo apt-get update;
10        pip install --upgrade pip && pip install -r requirements.txt && pip
11        install pylint
12     - run:
13       name: Format .py files
14       command:
15        pip install black && black $(git ls-files '*.py')
16     - run:
17       name: pylint
18       command:
19        pylint $(git ls-files '*.py')
20     - run:
21       name: unit_test_django
22       command:
23        cd Backend && python manage.py test backend_api
24     - run:
25       name: unit_test_algorithm
26       command:
27        cd Algorithm && pytest

```

LISTING 4.2: Część skryptu config.yml odpowiadająca za wykonanie zadania *build\_and\_test\_back*

Dashboard Project Branch Workflow Job

All Pipelines > AutoPlanner > main > build\_test\_deploy > build\_and\_test\_back (1595)

**build\_and\_test\_back** Success Rerun ...

Duration / Finished	Queued	Executor / Resource Class	Branch	Commit
🕒 27s / 2m ago	⌚ 0s	🚢 Docker / Medium <sup>ⓘ</sup>	🔗 main	🔑 80479cf, 02713fe, 31efe3f

Author & Message

👤 Merge pull request #40 from grzesiupia/gpiatkow-test\_chapter\_algorithm

**STEPS** TESTS TIMING BETA ARTIFACTS RESOURCES NEW

- ▶ ✓ Spin up environment 1s 🔗 ⬇
- ▶ ✓ Preparing environment variables 0s 🔗 ⬇
- ▶ ✓ Checkout code 3s 🔗 ⬇
- ▶ ✓ Install requirements 17s 🔗 ⬇
- ▶ ✓ Format .py files 2s 🔗 ⬇
- ▶ ✓ pylint 1s 🔗 ⬇
- ▶ ✓ unit\_test\_django 0s 🔗 ⬇
- ▶ ✓ unit\_test\_algorithm 1s 🔗 ⬇

RYSUNEK 4.5: Przykładowy przebieg zadania *build\_and\_test\_back*. Uruchomiony dla zmian wprowadzonych w gałęzi Git *main*.

```

1 build_and_test_front:
2   docker:
3     - image: cimg/node:17.2.0
4   steps:
5     - checkout
6     - run:
7       name: Update node.js
8       command:
9         npm install node.js@latest
10    - run:
11      name: Install dependencies
12      command:
13        npm install ./Frontend
14    - run:
15      name: unit_tests
16      command:
17        cd Frontend &&
18        npm run test:unit

```

LISTING 4.3: Część skryptu *config.yml* odpowiadająca za wykonanie zadania *build\_and\_test\_front*

The screenshot displays the AWS CodePipeline console interface. At the top, the navigation bar shows the path: Dashboard > Project > Branch > Workflow > Job. The selected job is 'build\_and\_test\_front' (1489), which is in a 'Success' state, indicated by a green checkmark. Below the job name, a summary box provides details: Duration / Finished (1m 1s / 28m ago), Queued (0s), Executor / Resource Class (Docker / Medium), Branch (main), and Commit (24b4291, 1067d0c, 9211fb0). The author and message are listed as 'Merge pull request #34 from grzesiupia/dbola\_backend\_chapter'. Below this, a tabbed interface shows 'STEPS' as the active view. The steps listed are: 'Spin up environment' (3s), 'Preparing environment variables' (0s), 'Checkout code' (3s), 'Update node.js' (0s), 'Install dependencies' (49s), and 'unit\_tests' (4s). Each step has a green checkmark indicating success and icons for viewing logs or downloading artifacts.

RYСУNEK 4.6: Przykładowy przebieg zadania *build\_and\_test\_front*. Uruchomiony dla zmian wprowadzonych w gałęzi Git *main*.

```

1 deploy_frontend:
2   executor: aws-cli/default
3   steps:
4     - checkout
5     - aws-cli/setup:
6       aws-access-key-id: AWS_ACCESS_KEY_ID_FRONT
7       aws-secret-access-key: AWS_SECRET_ACCESS_KEY_FRONT
8     - run:
9       name: deploy_frontend
10      command: |
11        sudo apt-get update
12        # SSH to the server to deploy and Perform steps to deploy
13        ssh -o StrictHostKeyChecking=no $EC2_USERNAME@$EC2_PUBLIC_DNS_FRONT 'cd
14          GroupProject;
15          git pull --rebase;
16          cd Frontend;
17          npm install;
18          tmux kill-session -t FRONTEND_SERVER
19          sleep 5;
20          tmux new-session -d -s "FRONTEND_SERVER";
21          tmux send-keys -t FRONTEND_SERVER "npm run serve" C-m;
          exit'
```

LISTING 4.4: Część skryptu config.yml odpowiadająca za wykonanie zadania *deploy\_frontend*

Dashboard Project Branch Workflow Job

All Pipelines > AutoPlanner > main > build\_test\_deploy > **deploy\_frontend (1479)**

**deploy\_frontend** Success Rerun ...

Duration / Finished	Queued	Executor / Resource Class	Branch	Commit
🕒 21s / 8h ago	⌚ 0s	🚢 Docker / Medium <sup>?</sup> ⓘ	🔗 main	🔑 36988bb

Author & Message

👤 Updated config.yml

**STEPS** TESTS TIMING BETA ARTIFACTS ● RESOURCES ● NEW

▶ <span>✓</span> Spin up environment	2s	🔗 ⬇
▶ <span>✓</span> Preparing environment variables	0s	🔗 ⬇
▶ <span>✓</span> Checkout code	3s	🔗 ⬇
▶ <span>✓</span> Install AWS CLI - latest	2s	🔗 ⬇
▶ <span>✓</span> Configure AWS Access Key ID	2s	🔗 ⬇
▶ <span>✓</span> Update node.js	0s	🔗 ⬇
▶ <span>✓</span> deploy_frontend	10s	🔗 ⬇

RYSUNEK 4.7: Przykładowy przebieg zadania *deploy\_frontend*. Uruchomiony dla zmian wprowadzonych w gałęzi Git *main*.

```

1 deploy_backend:
2   executor: aws-cli/default
3   steps:
4     - checkout
5     - aws-cli/setup:
6       aws-access-key-id: AWS_ACCESS_KEY_ID_BACK
7       aws-secret-access-key: AWS_SECRET_ACCESS_KEY_BACK
8     - run:
9       name: deploy_back
10      command: |
11        sudo apt-get update
12        # SSH to the server to deploy and Perform steps to deploy
13        ssh -o StrictHostKeyChecking=no $EC2_USERNAME@$EC2_PUBLIC_DNS_BACK 'cd
14          GroupProject;
15          git pull --rebase;
16          python -m pip install -r requirements.txt;
17          cd Backend;
18          tmux kill-session -t BACKEND_SERVER
19          tmux new-session -d -s "BACKEND_SERVER";
20          tmux send-keys -t BACKEND_SERVER "python3 manage.py runserver
          0.0.0.0:8000" C-m;
          exit'
```

LISTING 4.5: Część skryptu config.yml odpowiadająca za wykonanie zadania *deploy\_backend*

Dashboard Project Branch Workflow Job

All Pipelines > AutoPlanner > main > build\_test\_deploy > **deploy\_backend (1491)**

**deploy\_backend** Success Rerun ...

Duration / Finished	Queued	Executor / Resource Class	Branch	Commit
20s / 34m ago	0s	Docker / Medium	main	24b4291, 1067d0c, 9211fb0

Author & Message

Merge pull request #34 from grzeslupia/dbola\_backend\_chapter

STEPS TESTS TIMING BETA ARTIFACTS RESOURCES NEW

- Spin up environment 1s
- Preparing environment variables 0s
- Checkout code 3s
- Install AWS CLI - latest 2s
- Configure AWS Access Key ID 2s
- deploy\_back 11s

RYSUNEK 4.8: Przykładowy przebieg zadania *deploy\_backend*. Uruchomiony dla zmian wprowadzonych w gałęzi Git *main*.

#### 4.4.3 Zastosowanie DevOps lifecycle od strony GitHub

W celu zwiększenia jakości dostarczanych rozwiązań zastosowano następujący przebieg wdrażania. Członek zespołu, chcąc wprowadzić nową funkcjonalność lub rozwinąć już istniejącą, zaciąga najnowszą wersję projektu ze zdalnego repozytorium. Gdy lokalne repozytorium jest zaktualizowane, deweloper tworzy nową gałąź Git, w której będzie wprowadzał zmiany. W momencie gdy autor chce podzielić się swoją pracą z innymi członkami zespołu, publikuje (*push*) swoje zmiany do zdalnego repozytorium oraz tworzy *Pull Request* (PR). W takim PR można prześledzić wszystkie zmiany jakie zostały po kolei wprowadzone przez autora. Każda wprowadzona zmiana pojawiająca się w zdalnym repozytorium wywołuje uruchomienie *pipeline* na platformie CircleCI. Wynik działania zadań z uruchomionego przepływu pracy *pipeline* również można sprawdzić z poziomu PR. W przypadku gdy zmiany wprowadzone w ramach PR przechodzą wszystkie testy oraz nie wywołują żadnych konfliktów, autor prosi o sprawdzenie i zatwierdzenie zmian przez innego członka zespołu. Gdy współpracownik zatwierdzi zmiany, gałąź Git stworzona przez autora na potrzeby zmian może zostać scalona (*merge*) z gałęzią Git *main*. Po wprowadzeniu zmian do gałęzi Git *main* następuje ostatni etap tj. uruchomienie *pipeline* z poziomu tej gałęzi a co za tym idzie, ponowne zbudowanie i przetestowanie aplikacji oraz ostatecznie wdrożenie (zob. rysunek 4.9).

## Expand Chapter 4 - DevOps #35

**Merged** MaciejGoral merged 3 commits into `main` from `gpiatkow-devops_chapter` 2 days ago

Conversation 2 Commits 3 Checks 1 Files changed 12

**grzesiupia** commented 2 days ago

This PR expands Chapter 4 - DevOps. No other chapter should be impacted by this PR.

**grzesiupia** force-pushed the `gpiatkow-devops_chapter` branch from `a1be191` to `e872669` 2 days ago

**gpiatkow** added 3 commits 2 days ago

- expand devops chapter 34757de
- update gitignore to ignore toc and pdf files 81e38af
- update gitignore ✓ 52c1375

**grzesiupia** force-pushed the `gpiatkow-devops_chapter` branch from `23da70a` to `52c1375` 2 days ago

**grzesiupia** requested review from **katan666** and **MaciejGoral** 2 days ago

**grzesiupia** added the **LaTeX** label 2 days ago

**MaciejGoral** merged commit `5867e78` into `main` 2 days ago  
1 check passed [View details](#) [Revert](#)

---

**MaciejGoral** reviewed 2 days ago [View changes](#)

**MaciejGoral** left a comment • edited

Seems fine

**katan666** reviewed 2 days ago [View changes](#)

**katan666** left a comment

Looks good, ready to merge

**grzesiupia** deleted the `gpiatkow-devops_chapter` branch 2 days ago [Restore branch](#)

RYSUNEK 4.9: Przykładowy przebieg wdrażania zmian przy pomocy *Pull Request*

## Rozdział 5

# Projekt i implementacja aplikacji internetowej w technologii Vue.js

### 5.1 Narzędzia i technologie

#### 5.1.1 Node.js

Node.js [20] jest środowiskiem uruchomieniowym umożliwiającym używanie języka JavaScript poza przeglądarką. Środowisko to charakteryzuje asynchroniczność oraz sterowanie zdarzeniami. Asynchroniczność umożliwia wykonywanie wielu czynności w tym samym czasie bez względu na jednowątkowość wynikającą z ograniczenia języka JavaScript. Sterowanie zdarzeniami jest rozwiązaniem typowym dla interfejsów graficznych. Zapewnia ono elastyczność oraz możliwość tworzenia bardziej interaktywnych elementów GUI. Ponadto Node.js udostępnia menadżera pakietów środowiska Node (NPM – Node Package Manager) dającego możliwość zarządzania zainstalowanymi funkcjonalnościami w prosty i przejrzysty sposób.

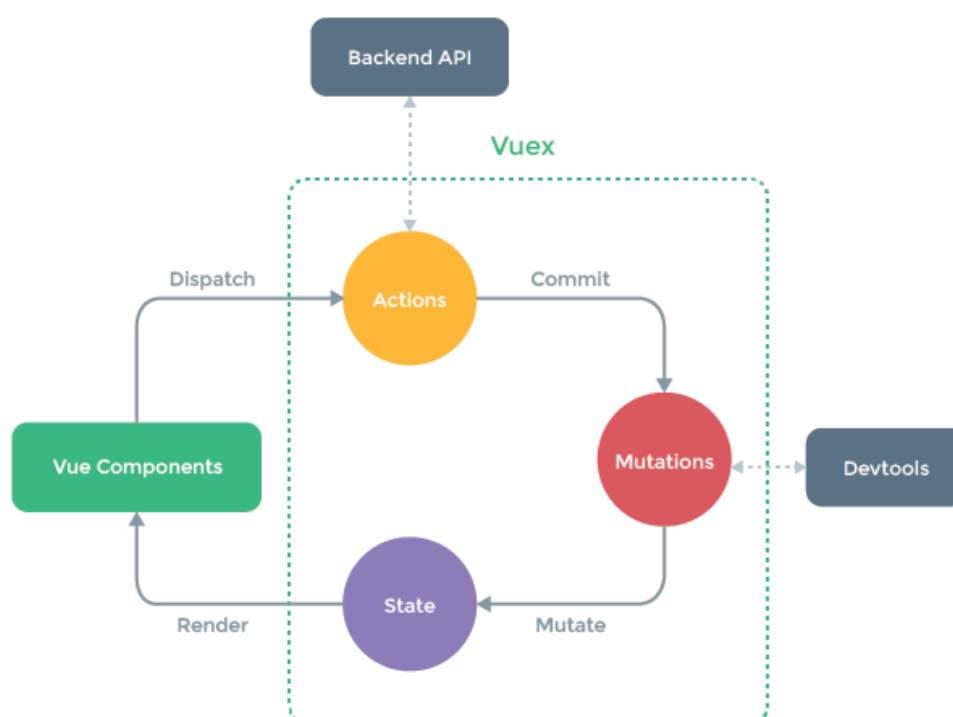
#### 5.1.2 Vue.js

Vue.js [40] to platforma programistyczna języka JavaScript służąca do budowania interfejsów użytkownika. W stosunku do dwóch najpopularniejszych alternatyw – platform programistycznych React oraz Angular – wyróżnia się prostotą, szybkością działania oraz niewielkim rozmiarem. Platforma programistyczna Vue.js została zaprojektowana tak, aby zapewnić jak największą elastyczność. Przy jej użyciu możliwe jest tworzenie nie tylko prostych komponentów, ale i aplikacji typu *single-page-application* oraz *multi-page-application*.

Cechą charakterystyczną Vue.js jest wykorzystanie szablonów jako sposobu na powiązanie języka znaczników HTML z warstwą logiki JavaScript. Powiązanie to umożliwia wykorzystywanie w prosty sposób instrukcji warunkowych oraz pętli do wyświetlania zawartości aplikacji.

#### 5.1.3 Vuex

Vuex [41] to biblioteka oferująca scentralizowany magazyn danych dostępny dla wszystkich komponentów w aplikacji. Stan danych w magazynie Vuex jest zmieniany poprzez mutacje wykonywane w reakcji na działanie dyspozytora (zob. rysunek 5.1). Takie podejście sprawia, że dane z części backendowej aplikacji mogą zostać pobrane tylko raz, a później będą one dostępne bezpośrednio w części frontendowej za pośrednictwem magazynu.



RYSUNEK 5.1: Schemat przepływu danych w Vuex [31]

#### 5.1.4 Jest

W projekcie wykorzystano testową platformę programistyczną Jest [16] będącą częścią Vue Test Utils. Vue Test Utils to zestaw funkcjonalności upraszczających testowanie komponentów Vue.js. Zestaw ten zapewnia metody umożliwiające symulowanie działań użytkownika w aplikacji oraz przechwytywanie i porównywanie rezultatów tych interakcji z oczekiwanymi. Jest cechuje brak konieczności konfiguracji, izolacja testów oraz szybkość i bezpieczeństwo działania.

#### 5.1.5 Json Web Tokens

Json Web Token [17] jest otwartym standardem przesyłania zabezpieczonych danych. Dane w formacie Json są podpisywane cyfrowo, co umożliwia weryfikację uprawnień. W aplikacjach internetowych JWT stosowane są głównie do autoryzacji użytkowników oraz zapewnienia bezpieczeństwa przesyłania informacji pomiędzy frontendem a backendem. Niewielki rozmiar tokenu sprawia, iż możliwe jest przesyłanie go w treści zapytania HTTP lub nawet w jego nagłówku. Ta cecha sprawia również, że token może być przechowywany w pamięci przeglądarki, eliminując konieczność ponownego uwierzytelniania po rozpoczęciu nowej sesji.

#### 5.1.6 Postman

Postman [24] jest zestawem narzędzi do testowania API (*Application Programming Interface*). Zapewnia on możliwość wysyłania zapytań HTTP dowolnego typu oraz podgląd odpowiedzi i kodów błędów, jeśli takie wystąpiły. Główną zaletą Postmana jest możliwość tworzenia kolekcji



zapytań, które ułatwiają organizację pracy podczas planowania połączeń pomiędzy częścią frontendową i backendową aplikacji. Dodatkowo narzędzie pozwala na współdzielenie kolekcji z zaproszonymi użytkownikami, co znacząco upraszcza proces testowania manualnego. Poza testowaniem manualnym Postman umożliwia tworzenie automatycznych testów przy pomocy języka JavaScript. Dzięki generatorowi losowych danych możliwa jest symulacja działań nawet kilku tysięcy różnych użytkowników w systemie.

### 5.1.7 Visual Studio Code

Visual Studio Code [39] jest edytorem kodu, którego głównymi zaletami jest wsparcie dla debugowania, inteligentnego uzupełniania kodu, refaktoryzacji oraz kontroli wersji. Dużą korzyścią płynącą z korzystania z programu Visual Studio Code jest dostęp do rozszerzeń, usprawniających pracę z kodem w dowolnym języku programowania. Rozszerzenia zapewniają również wsparcie dla platform programistycznych, w tym Vue.js, najbardziej istotnego dla tej części projektu. Mały rozmiar oraz wysoka wydajność znacznie przyspieszają korzystanie z aplikacji i sprzyjają intensywnej iteracji rozwiązań.

### 5.1.8 Axios

Axios [3] jest biblioteką języka JavaScript służącą do wykonywania zapytań HTTP z poziomu Node.js lub przeglądarki. W aplikacjach internetowych wykorzystywany jest do uzyskiwania danych z części backendowej aplikacji. Axios bazuje na obietnicach (promise), co pozwala na obsługiwanie akcji asynchronicznie. Biblioteka może być użyta poprzez zwykły Javascript lub platformę programistyczną taką jak Vue.js. W porównaniu z innymi bibliotekami służącymi do wykonywania zapytań HTTP Axios oferuje wsparcie dla starszych przeglądarek, możliwość ustawienia ograniczenia czasowego dla zapytań, ochronę przed CSRF (*Cross-Site Request Forgery*), a także automatyczną transformację danych JSON.

### 5.1.9 Bootstrap

Bootstrap [5] jest platformą programistyczną CSS (*Cascading Style Sheets*) upraszczającą projektowanie interfejsu graficznego aplikacji internetowych. Bootstrap pomaga zapewnić responsywność stron, a więc poprawne ich wyświetlanie na urządzeniach mobilnych. Przed pojawieniem się tego rozwiązania często występowała konieczność przygotowywania oddzielnych stylów dla ekranów o różnych rozdzielczościach. Dzięki zastosowaniu platformy programistycznej elementy strony internetowej zostają przeskalowane i przemieszczone tak, aby pomieścić się na ekranie niezależnie od jego wielkości i proporcji. Dodatkowo Bootstrap pozwala na zastosowanie zaawansowanych komponentów takich jak paski nawigacji, wskaźniki postępu czy miniatury.

## 5.2 Połączenie z backendem

Połączenie z częścią backendową aplikacji jest realizowane przy pomocy biblioteki Axios. W tym celu wykorzystywane są dwa typy zapytań HTTP. Pierwszym z nich jest zapytanie typu GET. Przykładowym zastosowaniem tego zapytania jest odbieranie danych nauczycieli (zob. listing 5.1). Ze względu na fakt, że zapytanie typu GET nie posiada ciała wiadomości, token uwierzytelniający JWT przesyłany jest w nagłówku. Dane zwrotne otrzymane z backendu zapisywane są w magazynie Vuex. Są tam również przechowywane informacje o powodzeniu operacji i komunikaty o błędzie, jeśli taki wystąpił.

```
1 export const fetchTeachers = ({
2   commit
3 }, object) => {
4   axios.get("api/get/all/teachers",{ headers: { 'x-access-token': `${object.token}
5     }' })
6     .then((response) => {
7       console.log(response)
8       commit("SET_TEACHERS", response.data)
9       commit("GET_TEACHERS_SUCCESS", true)
10    })
11    .catch(function (error) {
12      commit("GET_TEACHERS_SUCCESS", false)
13      commit("GET_TEACHERS_ERROR", error.response.data.message)
14    });
15 }
```

LISTING 5.1: Funkcja wykonująca zapytanie typu GET

Drugim rodzajem zapytania HTTP wykorzystywanym do wymiany danych z backendem jest POST. W przeciwieństwie do komunikatu typu GET, komunikat typu POST posiada ciało wiadomości. Pozwala to na przesłanie tokenu uwierzytelniającego w treści komunikatu. Przykładowym zastosowaniem zapytania tego typu jest przesyłanie danych nauczyciela (zob. listing 5.2). Poza tokenem uwierzytelniającym, w treści komunikatu przesyłane są informacje, które mają zostać zapisane w bazie danych. W odpowiedzi na komunikat zostaje przesłana informacja o powodzeniu operacji, a w przypadku niepowodzenia także komunikat o błędzie. Te dane są zapisywane w magazynie Vuex, co umożliwia ich wykorzystanie do wyświetlania komunikatów o błędzie użytkownikowi.

```
1 export const sendTeacher = ({
2   commit
3 }, object) => {
4   console.log(object.token, object.name, object.email, object.list_of_subjects)
5   axios
6     .post("api/add/teacher", {
7       token: object.token,
8       name: object.name,
9       email: object.email,
10      list_of_subjects: object.list_of_subjects
11    })
12     .then(function (response) {
13       console.log(response);
14       commit("ADD_TEACHER_SUCCESS", true)
15       router.go();
16    })
17     .catch(function (error) {
18       commit("ADD_TEACHER_SUCCESS", false)
19       commit("ADD_TEACHER_ERROR", error.response.data.message)
20    })
21  });
22 }
```

LISTING 5.2: Funkcja wykonująca zapytanie typu POST

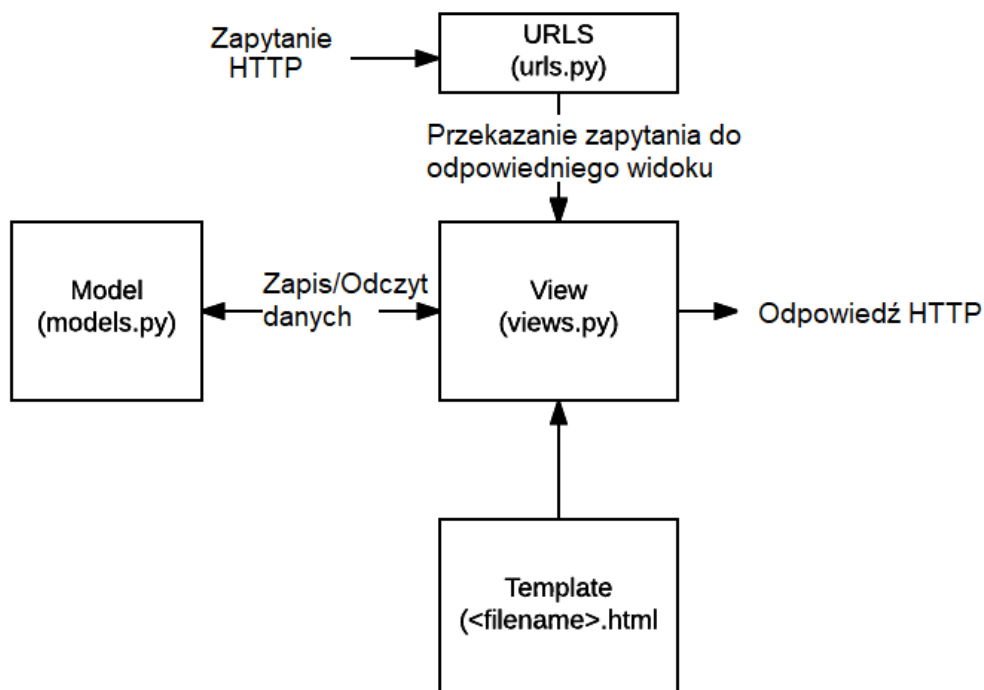
## Rozdział 6

# Projekt i implementacja strony serwerowej opartej na architekturze REST w technologii Django oraz bazy danych MySQL

### 6.1 Narzędzia i technologie

#### 6.1.1 Django

Django [10] jest darmową, wysokopoziomową platformą programistyczną przeznaczoną do tworzenia aplikacji internetowych. Dostarcza wiele narzędzi ułatwiających szybką oraz prostą implementację. Oparta jest na wzorcu architektonicznym model-template-view (zob. rysunek 6.1).



RYSUNEK 6.1: Schemat przepływu danych w Django [30]

Napisana jest w języku Python. Jednymi z najważniejszych cech Django są:

- łatwy i bezpieczny dostęp do bazy danych,
- duża skalowalność oraz wydajność,
- wbudowane zabezpieczenia przed popularnymi atakami,
- rozbudowana dokumentacja.

### 6.1.2 MySQL

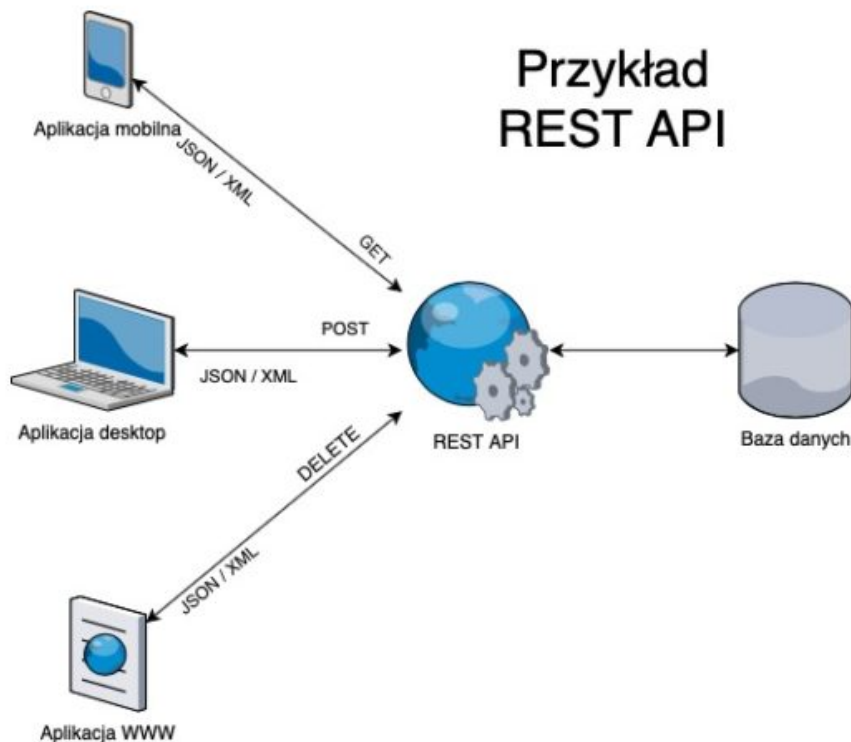
MySQL [18] jest systemem służącym do zarządzania relacyjnymi bazami danych. Model relacyjny zapewnia łatwość w projektowaniu oraz implementacji. Udostępniony jest na licencji wolnego oprogramowania i dostępny jest dla wszystkich popularnych systemów operacyjnych. Bazy danych oparte na tym systemie są w stanie obsługiwać olbrzymie liczby zapytań w bardzo krótkim czasie.

### 6.1.3 MySQL Workbench

MySQL Workbench [19] to narzędzie do projektowania, tworzenia oraz zarządzania bazami danych MySQL. Posiada bardzo przejrzysty i intuicyjny interfejs, przez co cieszy się dużą popularnością. Wiele podstawowych czynności takich jak np. tworzenie i edycja tabel, można wykonać bez znajomości zapytań SQL, gdyż są one generowane automatycznie.

## 6.2 Implementacja serwera REST API

### 6.2.1 REST



RYSUNEK 6.2: Przykładowy schemat działania REST API [27]

REST [29] jest stylem architektonicznym wprowadzającym pewien standard komunikacyjny dla internetowych systemów informatycznych (zob. rysunek 6.2). Jego najważniejszymi zaletami są szybkość oraz uniwersalność. Interfejsy programistyczne spełniające założenia REST mogą komunikować się z dowolnym urządzeniem sieciowym, pod warunkiem wysyłania przez nie zapytań w odpowiednim formacie. Jednymi z głównych zasad tego stylu są:

- zastosowanie modelu klient-serwer,
- bezstanowość,
- wykorzystywanie pamięci cache przeglądarki w celu zapamiętywania odpowiedzi,
- interfejs programistyczny jednolity dla każdej aplikacji klienckiej.

Zapytania zazwyczaj wysyłane są przy pomocy protokołu HTTP. Zarówno do zapytań, jak i do odpowiedzi, najczęściej wykorzystywane są metody GET oraz POST. Informacje przesyłane między klientem a serwerem muszą być precyzyjne, zwykle występują one w formacie JSON.

### 6.2.2 Modele

Jednym z najważniejszych mechanizmów zaimplementowanych w Django są modele. Są one swego rodzaju mapowaniem klas języka Python na tabele bazy danych. Takie rozwiązanie zapewnia bardzo szybki i wygodny dostęp do przechowywanych informacji. Klasy te zdefiniowane są w pliku „models.py”. Wykorzystując wbudowane funkcje wykorzystywanej platformy programistycznej, można zarówno wygenerować tabele bazy danych na podstawie modeli, jak i wygenerować modele na podstawie gotowych tabel.

W projekcie znajduje się następujących siedem modeli:

- Classrooms,
- Lessons,
- Planners,
- Polls,
- Subjects,
- Teachers,
- Timetables.

Każdy odpowiada jednej tabeli z bazy danych, pola w każdej z klas są analogiczne do kolumn w odpowiednich tabelach. Zawierają informacje takie jak np. typ danych, maksymalna długość oraz klucz główny. Implementacja przykładowej klasy przedstawiona na listingu 6.1

```
1 class Teachers(models.Model):
2     '''This class represents table that hold information about teachers'''
3     teacheremail = models.CharField(db_column='teacherEmail', primary_key=True,
4                                     max_length=255)
5     teachername = models.CharField(db_column='teacherName', max_length=255)
6     teachsubject = models.CharField(db_column='teachSubject', max_length=255)
7     planneremail = models.CharField(db_column='plannerEmail', max_length=255)
8
9     class Meta:
10         managed = False
```

```
10 db_table = 'Teachers'
```

LISTING 6.1: Implementacja klasy Teachers

### 6.2.3 Adresy internetowe

Kolejnym elementem implementacji są adresy internetowe. W pliku „urls.py” określone zostały wykorzystywane w projekcie adresy URL, oraz funkcje, które mają zostać wywołane w przypadku otrzymania od klienta zapytania na dany adres. Implementacja adresów przedstawiona została na listingu 6.2

```
1  urlpatterns = [  
2      path('admin/', admin.site.urls),  
3      path('api/auth/signup/', views.add_user),  
4      path('api/auth/signin/', views.get_user),  
5      path('api/add/subject', views.add_subject),  
6      path('api/add/teacher', views.add_teacher),  
7      path('api/add/classroom', views.add_classroom),  
8      path('api/add/class', views.add_class),  
9      path('api/get/all/subjects', views.get_subjects),  
10     path('api/get/all/teachers', views.get_teachers),  
11     path('api/get/all/classrooms', views.get_classrooms),  
12     path('api/get/all/classes', views.get_classes),  
13     path('api/sendEmail', views.send_email),  
14     path('api/polls/<int:pollNumber>', views.add_poll_data),  
15     path('api/generatePlan', views.generate_plan),  
16     path('api/delete/subject', views.del_subject),  
17     path('api/delete/teacher', views.del_teacher),  
18     path('api/delete/classroom', views.del_classroom),  
19     path('api/delete/class', views.del_class),  
20     path('api/edit/subject', views.edit_subject),  
21     path('api/edit/teacher', views.edit_teacher),  
22     path('api/edit/classroom', views.edit_classroom),  
23     path('api/edit/class', views.edit_class),  
24     path('api/get/plans/class', views.get_classes_timetable),  
25     path('api/get/plans/classroom', views.get_classrooms_timetable),  
26     path('api/get/plans/teacher', views.get_teachers_timetable)  
27 ]
```

LISTING 6.2: Implementacja adresów URL

W przypadku, jeśli zapytanie przyjdzie na niezadeklarowany adres, zwrócony zostanie komunikat HTTP z kodem 404 (Page Not Found).

### 6.2.4 Widoki

Widoki to nic innego, jak funkcje języka Python wykonywane w momencie odbioru zapytania przez serwer. Kiedy klient wyśle zapytanie na dany adres, Django sprawdza, czy jest on zadeklarowany we wspomnianym wcześniej pliku „urls.py”, jeśli tak, to wywoływana jest odpowiednia funkcja z pliku „views.py” wraz z jej parametrem, którym jest samo zapytanie. Dzięki takiej parametryzacji, widok ma łatwy dostęp do otrzymanych informacji, sprawdza on czy są one prawidłowe, oraz wykonuje na nich odpowiednie działania. W tym projekcie, w większości przypadków wykonywane są operacje zapisu i odczytu z bazy danych. Do komunikacji wykorzystywane są metody GET oraz POST z protokołu HTTP. Implementacje przykładowych widoków obsługujących zapytania dla danych metod HTTP przedstawione zostały na listingach 6.3 oraz 6.4.

```

1  @csrf_exempt
2  def get_subjects(request):
3  ''' A function that returns a list of subjects for a given user '''
4  if request.method == 'GET':
5      payload = request.headers.get('x-access-token')
6      try:
7          user_data = jwt.decode(payload, None, None)
8          array = Subjects.objects.filter(planneremail = user_data['email'])
9          subjects_list = []
10         for i in array:
11             subjects_list.append({'subject_name': i.subjectname})
12         response=json.dumps(subjects_list)
13         return HttpResponse(response, content_type='text/json')
14     except Exception as exc:
15         response = json.dumps({'message': str(exc)})
16         return HttpResponse(response, content_type='text/json')

```

LISTING 6.3: Widok obsługujący zapytanie typu GET

```

1  @csrf_exempt
2  def add_subject(request):
3  '''Function that takes new subject request and returns response with the
4  appropriate message,
5  depending on whether the addition of a subject was successful '''
6  if request.method == 'POST':
7      payload = json.loads(request.body)
8      name = payload['subject_name']
9      token = payload['token']
10     try:
11         user_data = jwt.decode(token, None, None)
12         subject = Subjects(subjectname = name, planneremail = user_data["email"])
13         subject.save(force_insert = True)
14         response=json.dumps({'message': 'Pomyślnie dodano lekcje'})
15         return HttpResponse(response, content_type='text/json')
16     except Exception as exc:
17         response = json.dumps({'message': str(exc)})
18         return HttpResponse(response, content_type='text/json')

```

LISTING 6.4: Widok obsługujący zapytanie typu POST

## 6.3 Główne funkcjonalności

### 6.3.1 Rejestracja i logowanie

Rejestracja oraz logowanie zrealizowane są za pomocą dwóch adresów, co za tym idzie również dwóch widoków, obsługujących zapytania z metodą POST. Do zarejestrowania nowego użytkownika wymagane jest podanie danych takich informacji jak adres e-mail, login oraz hasło. Funkcja obsługująca rejestrację zapisuje te dane do odpowiedniej tabeli, pod warunkiem że dany użytkownik nie istnieje już w bazie danych, a następnie zwraca odpowiedni komunikat. W przypadku logowania wymagane jest podanie adresu e-mail oraz hasła podanych przy rejestracji. Jeśli podane dane są poprawne, generowany oraz zwracany klientowi jest token (JWT). Zakodowane w nim są wszystkie dane podane przy rejestracji, co pozwala na identyfikację użytkownika przy dalszej komunikacji. Generacja JWT na listingu 6.5.

```

1  user = Planners.objects.get(planneremail=get_email, password=get_password)
2

```

```
3
4 access_token_payload = {
5     'email': user.planneremail,
6     'login': user.login,
7     'password': user.password
8 }
9
10 token = jwt.encode(access_token_payload, settings.SECRET_KEY, algorithm='HS256').
    decode('utf-8')
```

LISTING 6.5: Widok obsługujący zapytanie typu POST

### 6.3.2 Uzupełnianie danych potrzebnych do generacji planu

Dane które są wymagane do generacji planu lekcji, można podzielić na 4 grupy:

- nauczyciele,
- klasy,
- sale lekcyjne,
- przedmioty.

W projekcie zaimplementowane są widoki, pozwalające dodawać, edytować, usuwać oraz zwracać informacje o każdej z tych grup. W większości przypadków są to proste operacje zapisu i odczytu z bazy danych, jednak w przypadku dodawania informacji o nauczycielach istnieje dodatkowa funkcjonalność, pozwalająca na wysłanie do nich wiadomości e-mail z adresem do indywidualnej ankiety, w której można określić preferowane godziny pracy. Przy pomyślnej realizacji zapytania o wysłanie wiadomości e-mail, w bazie danych tworzone są wiersze, do których zapisane będą dane z ankiet. Są one identyfikowane po unikatowym numerze, przekazywanym jako parametr w adresie URL. Dzięki takiemu rozwiązaniu, generowany jest indywidualny link do ankiety dla każdego nauczyciela.

### 6.3.3 Generacja oraz zwracanie planów

Generacja oraz zwracanie planów zrealizowane są za pomocą czterech widoków. Funkcja generacji planu ma za zadanie odczytać wszystkie potrzebne informacje i w odpowiednim formacie przekazać je do generatora. Ze względu na to, że klient nie powinien oczekiwać na odpowiedź oraz że czas generacji planu może być stosunkowo długi, widok ten w przypadku pomyślnego przygotowania danych wejściowych, zwraca informację o rozpoczęciu generacji planu, nie czekając na zakończenie procesu. Dzięki temu, proces generacji nie zaburza połączenia klient-serwer. Kiedy generator skończy pracę, plany lekcji zapisywane są w bazie danych w trzech wersjach, z perspektywy: klas, nauczycieli oraz sal lekcyjnych. Każdy z tych planów, jest zwracany za pomocą pozostałych trzech widoków.

## 6.4 Baza danych MySQL

Baza danych składa się z siedmiu tabel. Większość wykorzystywanych relacji to „jeden do wielu”. Tabela przechowująca dane użytkowników (Planistów) nazywa się „Planners”. Zawiera ona następujące kolumny:

- plannerEmail – adres email planisty (klucz główny),



- login – nazwa planisty,
- password – hasło planisty.

Kolejne tabele zawierają informacje potrzebne algorytmowi do wygenerowania planu. Pierwszą z nich jest tabela „Subjects”. Zawiera ona listę przedmiotów, dodanych przez danego planistę. Jest to pierwsza z relacji, która zawiera złożony klucz główny, tzn. taki, który składa się z więcej niż jednej kolumny. Dane przechowywane są w następujący sposób:

- subjectName – nazwa przedmiotu (klucz główny),
- plannerEmail – adres email planisty (klucz główny oraz klucz obcy).

Podążając za przyjętą w projekcie kolejnością uzupełniania danych, następną tabelą jest „Teachers”, przechowująca informacje o nauczycielach. Występują w niej następujące kolumny:

- teacherEmail – adres email nauczyciela (klucz główny),
- plannerEmail – adres email planisty (klucz główny oraz klucz obcy),
- teacherName – imię oraz nazwisko nauczyciela,
- teachSubject – przedmioty, które prowadzi dany nauczyciel.

Rekordy kolejnej tabeli zawierają informacje o dostępnych salach lekcyjnych. Nazywa się ona „Classrooms”. Podobnie jak dwie poprzednie, ta również zawiera klucz główny złożony z dwóch kolumn. Występują w niej następujące dane:

- classroomId – identyfikator sali lekcyjnej (klucz główny),
- plannerEmail – adres email planisty (klucz główny oraz klucz obcy),
- preferredSubject – przedmioty, dla których dedykowana jest dana sala.

Ostatnią tabelą zawierającą dane wymagane do wygenerowania planu lekcji jest tabela „Lessons”. Zawiera ona informacje o klasach. Jako jedyna tabela, posiada klucz główny składający się aż z trzech kolumn. Przechowywane w niej są następujące dane:

- className – nazwa klasy (klucz główny),
- subjectName – nazwa przedmiotu (klucz główny oraz klucz obcy),
- plannerEmail – adres email planisty (klucz główny oraz klucz obcy),
- lessonCount – liczba godzin danego przedmiotu w tygodniu,
- teacherEmail – adres email nauczyciela (klucz obcy).

Poza powyższymi tabelami, przechowywanymi informacjami koniecznymi do wygenerowania planu, jest jeszcze taka, która zawiera informacje na temat preferowanych godzin pracy danych nauczycieli. Dane te, są opcjonalne. Tabela nosi nazwę „Polls”, a jej kolumny wyglądają następująco:

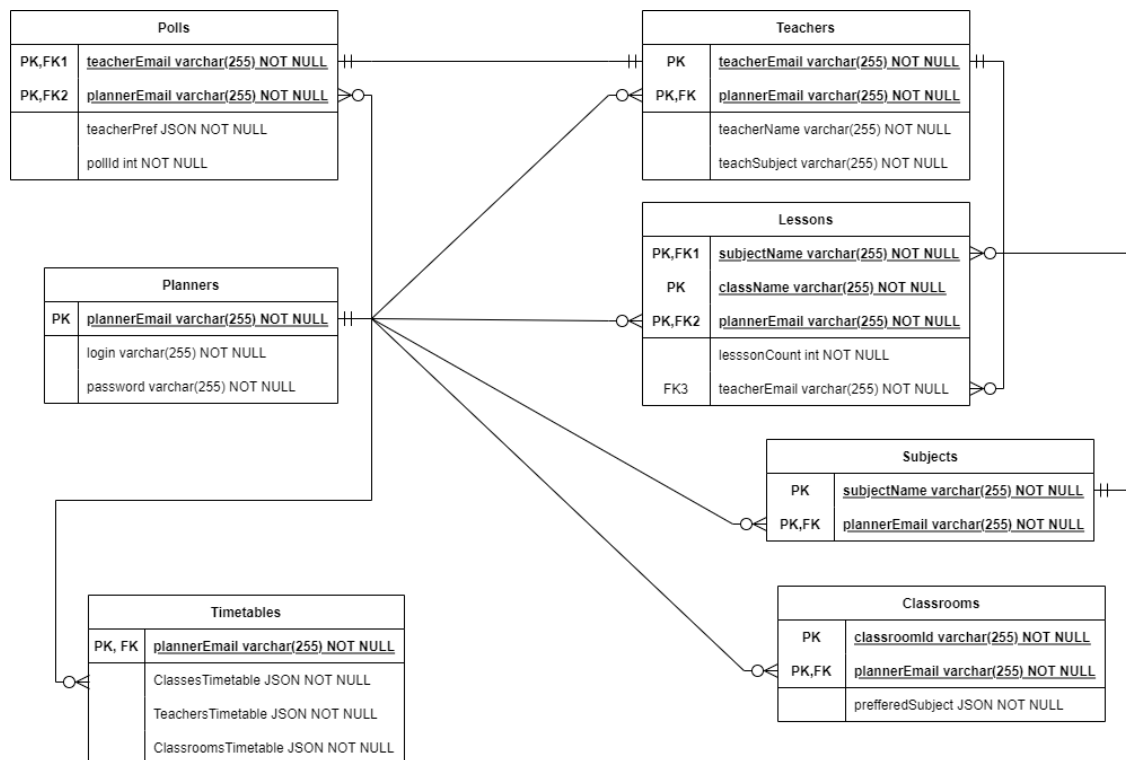
- plannerEmail – adres email planisty (klucz główny oraz klucz obcy),
- teacherEmail – adres email nauczyciela (klucz główny oraz klucz obcy),
- teacherPref – dane dotyczące preferowanych godzin pracy nauczyciela,

- pollId – numer ankiety, wykorzystywany do generacji odpowiedniego adresu URL.

Ostatnia tabela w bazie danych nazywa się „Timetables”. Przechowuje ona wygenerowane plany lekcji. Jej struktura wygląda następująco:

- plannerEmail – adres email planisty (klucz główny oraz klucz obcy),
- ClassesTimetable – plan lekcji z punktu widzenia klas,
- TeachersTimetable – plan lekcji z punktu widzenia nauczyciela,
- ClassroomsTimetable – plan lekcji z punktu widzenia sal lekcyjnych.

Diagram związków encji opisanej bazy danych, znajduje się na rysunku 6.3.



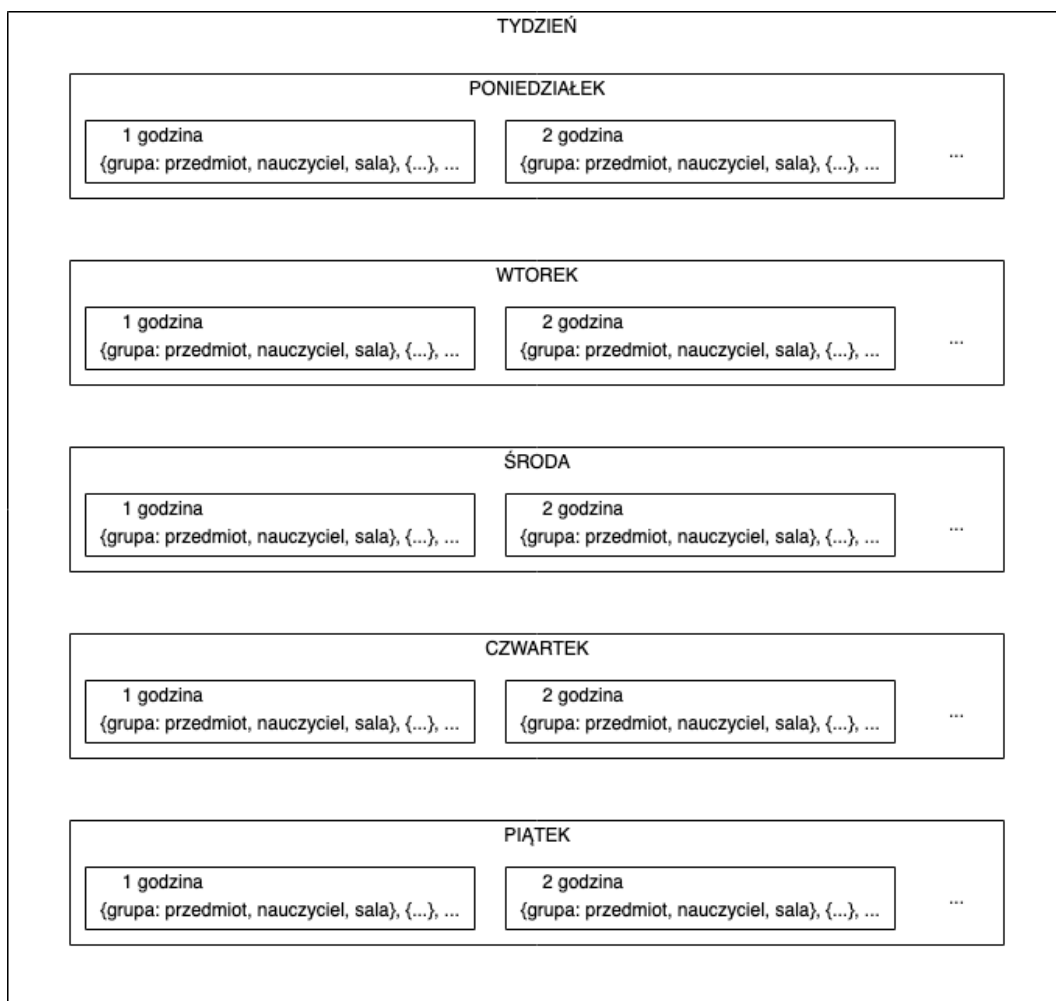
RYСУNEK 6.3: Diagram związków encji

## Rozdział 7

# Projekt i implementacja algorytmu generującego plan lekcji

### 7.1 Wstęp

Problem ułożenia najlepszego planu zajęć jest problemem NP-zupełnym. W projekcie zostało zaimplementowane podejście ewolucyjne. Na rysunku 7.1 została zobrazowana struktura generowanego planu zajęć.



RYSUNEK 7.1: Struktura planu zajęć

Zagadnienie implementacji algorytmu można podzielić na cztery główne części:

- przygotowanie danych wejściowych,
- ułożenie planu zajęć,
- ocena planu zajęć,
- ewolucja planu zajęć.

## 7.2 Przyjęte założenia

Przyjęte zostało, że:

- grupa jest najmniejszą, niepodzielną jednostką szkolną (oznacza to, że w jednej grupie nie mogą znajdować się żadne inne podgrupy, dla których konieczne byłoby oddzielne ułożenie planu),
- sale nie mają określonego limitu uczniów,
- grupy nie mają określonej liczby uczniów.

## 7.3 Narzędzia

Algorytm został zaprogramowany w języku *Python*. Pozwoliło to na szybkie wdrażanie zmian do kodu źródłowego algorytmu. Dużą zaletą wspomnianego języka jest jego prostota i czytelność, co w przypadku tak złożonego zagadnienia, jak algorytm generujący plany zajęć jest niezwykle ważne. *Python* automatycznie zarządza pamięcią, Programista nie musi zwracać uwagi na zagadnienia związane np. z uwalnianiem pamięci po nieużywanych zasobach, co powoduje, że *Python* jest językiem bezpieczniejszym niż np. *C++*.

## 7.4 Zastosowanie algorytmu ewolucyjnego

Algorytm wykorzystuje podejście ewolucyjne, z wykluczeniem mechanizmu krzyżowania. Z połączenia dwóch różnych planów zajęć trudne jest utworzenie trzeciego. Dlatego też, algorytm w celu ewolucji planów zajęć, korzysta wyłącznie z mechanizmu mutacji. Oznacza to, że każdy nowowygenerowany plan zajęć posiada wyłącznie jednego rodzica.

## 7.5 Przygotowanie danych wejściowych

Przygotowanie danych wejściowych jest kluczowe w działaniu algorytmu. Na podstawie danych otrzymanych od back-end, zostaje ułożona lista czteroelementowych krotek, gdzie pierwszym elementem jest nazwa klasy, drugim elementem jest nazwa przedmiotu, trzecim elementem jest nazwa nauczyciela, a czwartym elementem jest nazwa sali. Wartość elementu sali jest na początku wartością pustą null, a wartość elementu nauczyciela, jest wartością pustą null, wtedy i tylko wtedy kiedy nie został wskazany nauczyciel dla konkretnej klasy. W takiej liście znajdują się wszystkie jednostki lekcyjne występujące w całej szkole. Przykładowo jeżeli pewna klasa IIC ma mieć 5 matematyk w tygodniu z nauczycielem Jan Kowalski, to do listy zostanie dodane pięć krotek postaci (IIC, matematyka, Jan Kowalski, null). Struktura wspomnianej listy znajduje się na rysunku 7.2.

NAZWA GRUPY	NAZWA PRZEDMIOTU	NAZWA NAUCZYCIELA	NAZWA SALI
IIC	matematyka	Jan Kowalski	null
IIC	matematyka	Jan Kowalski	null
IIC	matematyka	Jan Kowalski	null
IIC	język polski	Andrzej Nowak	null
IIC	język polski	Andrzej Nowak	null
IA	język polski	null	null

RYSUNEK 7.2: Struktura listy krotek

## 7.6 Ułożenie poprawnych planów zajęć

Głównym założeniem układania planu zajęć jest to, że na podstawie tej samej listy krotek, zawsze zostanie wygenerowany konkretny plan zajęć. Układanie planu zajęć przebiega następująco:

1. z listy krotek zostaje zabrana pierwsza z brzegu krotka,
2. wybrana krotka zostaje przydzielona do pierwszej możliwej konkretnej godziny w
3. konkretnym dniu (czyli do takiej jednostki godzinowej, której są spełnione
4. wprowadzone przez użytkownika założenia oraz jest wolna sala),
5. poprzednie czynności zostają powtarzane tak długo, aż cała lista zostanie przeiterowana.

Jeżeli po zakończeniu działania procesu układania planu zajęć lista krotek nie będzie pusta, to ułożony plan zajęć jest niekompletny i niepoprawny. Proces układania planu zajęć został przedstawiony na rysunku 7.3.

## 7.7 Funkcja oceny

Funkcja oceny przyznaje planowi zajęć pewną liczbę punktów. Ocena może mieć wartość z zakresu od minus nieskończoności do 0, gdzie im wartość większa, tym lepsza ocena. Wpływ na końcową ocenę mają następujące elementy:

1. liczba tzw. okienek występujących w planie zajęć każdego nauczyciela,
2. liczba tzw. okienek występujących w planie zajęć każdej klasy,
3. liczba trudnych przedmiotów w ciągu jednego dnia w planie zajęć każdej klasy,
4. liczba sytuacji, w których klasa ma rozdzielone zajęcia tego samego przedmiotu innym przedmiotem (np. 1. godzina lekcyjna – matematyka, 2. godzina lekcyjna – biologia, 3. godzina lekcyjna – matematyka),
5. liczba sytuacji, w których zajęcia wychowania fizycznego nie występują na początku lub na końcu planu zajęć dnia.

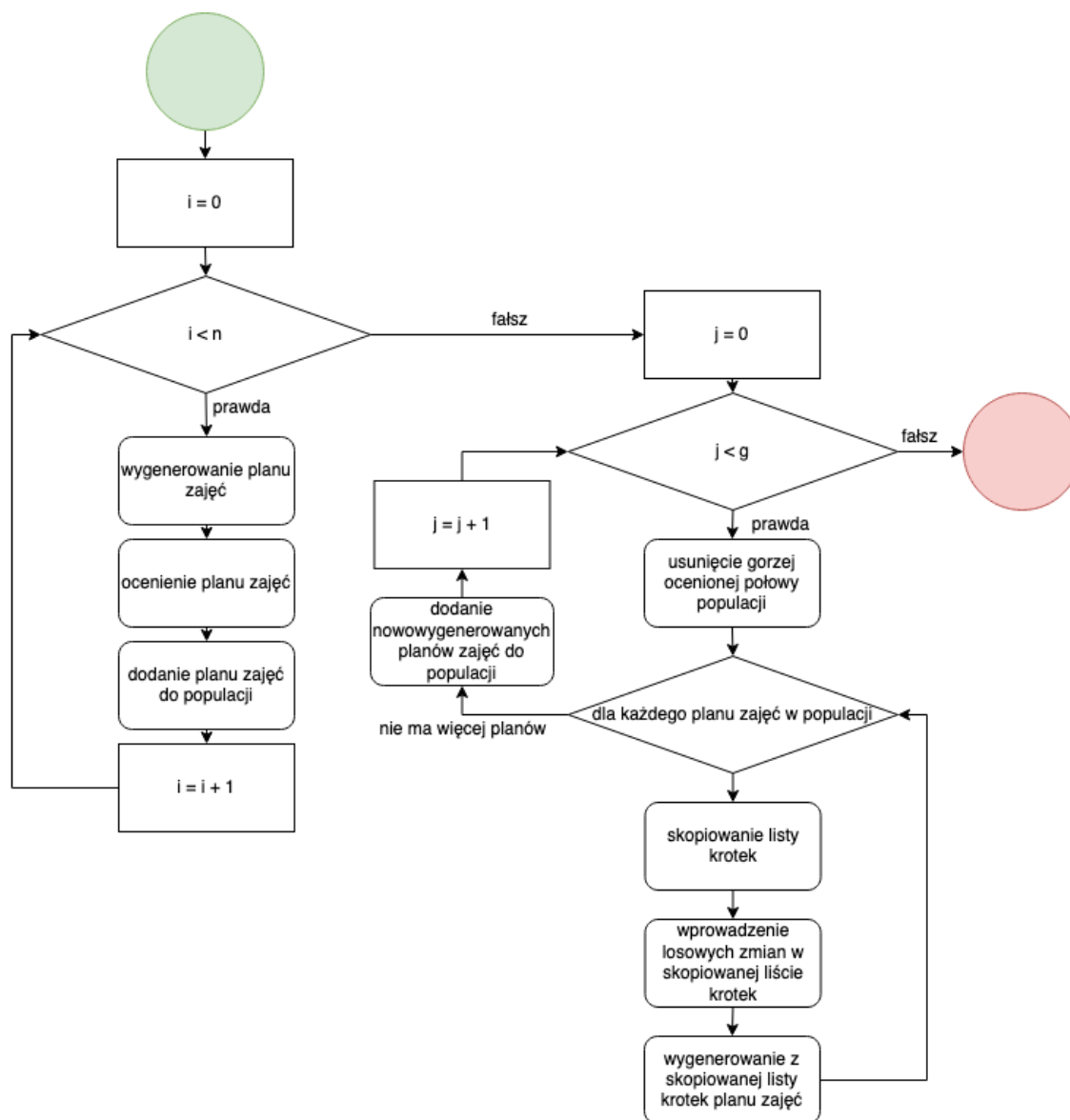
## 7.8 Ewolucja planów zajęć

Część ewolucyjna algorytmu wykorzystuje wszystkie poprzednie części. Na początku algorytm generuje pewną populację planów zajęć, która będzie nazywana dalej pierwszą generacją. Zawartość listy krotek jest identyczna dla każdej jednostki z populacji, ale kolejność krotek w liście jest



RYSUNEK 7.3: Proces układania planu zajęć

pseudolosowo zmieniona. Dzięki takiemu rozwiązaniu, każda jednostka w populacji może wygenerować zupełnie inny plan zajęć. Za pomocą funkcji oceny, do każdego z planów zostaje przydzielona pewna liczba punktów.



RYSUNEK 7.4: Algorytm – diagram

Połowa najgorzej ocenionych planów zostaje zabita. Z każdego planu zajęć, któremu udało się przeżyć, generowany jest kolejny plan. Plan, z którego został wygenerowany nowy plan, będzie nazywany w dalszej części pracy rodzicem, natomiast nowo wygenerowany plan będzie nazywany potomkiem. Lista krotek potomka, będzie zawierała tę samą kolejność co rodzic, ale losowo wybrane rekordy zamienia się w sposób pseudolosowy pozycjami w liście, taka zamiana będzie nazywana dalej mutacją. Nowo wygenerowane jednostki zostają ocenione oraz dodane do populacji, w ten sposób powstaje druga generacja planów zajęć.

Czynność z poprzedniego akapitu powtarza się jeszcze  $g$ -razy, gdzie  $g$  to liczba wszystkich generacji. Działanie całego algorytmu zostało przedstawione na rysunku 7.4.

## 7.9 Wielowątkowość

Podczas implementacji algorytmu była rozważana oraz testowana możliwość zrównoleglenia części kodu. Mowa konkretnie o pętli iterującej się przez każdy plan w populacji, w celu wygenerowania nowego, zmutowanego. Nie spowodowało to jednak znaczącego przyspieszenia działania

algorytmu. Biorąc pod uwagę jeszcze fakt, że algorytm ostatecznie działa na maszynie jednowątkowej, pomysł implementacji rozwiązania wielowątkowego został porzucony.

## 7.10 Parametry

Algorytm przewiduje parametryzację trzech zmiennych:

- rozmiar populacji  $p$ ,
- liczbę wszystkich generacji utworzonych podczas działania algorytmu  $g$ ,
- liczbę mutacji wprowadzonych do nowotworzonego planu zajęć  $m$ .

Największy wpływ na końcową ocenę końcową ma parametr  $g$  oraz  $m$ . Zakładając, że parametry  $p$  i  $m$  są stałe, to od wartości parametru  $g$  zależy liniowo czas wykonywania algorytmu oraz logarytmicznie ocena końcowa. Im większy parametr  $g$ , tym większa będzie końcowa ocena, ale im większa wartość  $g$ , tym mniejszy jest wzrost oceny końcowej. Ostatecznie parametry zostały ustawione następująco:

- $p = 10$ ,
- $g = 1500$ ,
- $m = 3$ .



## Rozdział 8

# Testowanie

### 8.1 Frontend

Testy w części frondendowej aplikacji wykonywane są przy pomocy platformy programistycznej Jest. W projekcie można wyróżnić podział na dwa typy testów. Pierwszym typem są testy spójności magazynu Vuex. Przykładowym testem tego typu jest test przypisanie tokenu JWT (zob. listing 8.1). Celem testu jest sprawdzenie, czy funkcje odpowiadające za zapisywanie i odczytywanie danych zostały poprawnie zdefiniowane.

```
1 describe('mutations', () => {
2   test('setToken', () => {
3     const token = "exampletoken"
4     const state = {
5       token: ""
6     }
7     store.commit('SET_TOKEN', { token })
8     expect(store.getters.getToken.token).toBe(token)
9   })
}
```

LISTING 8.1: Test spójności magazynu Vuex

Drugim typem testów są testy interfejsu użytkownika. Przykładem takiego testu jest test przycisku, który odpowiada za zwiększanie liczby prowadzonych przez nauczyciela przedmiotów (zob. listing 8.2). Celem testu jest sprawdzenie, czy akcja wykonana przez użytkownika, w tym przypadku symulowana, skutkuje odpowiednią zmianą zmiennych w kodzie aplikacji.

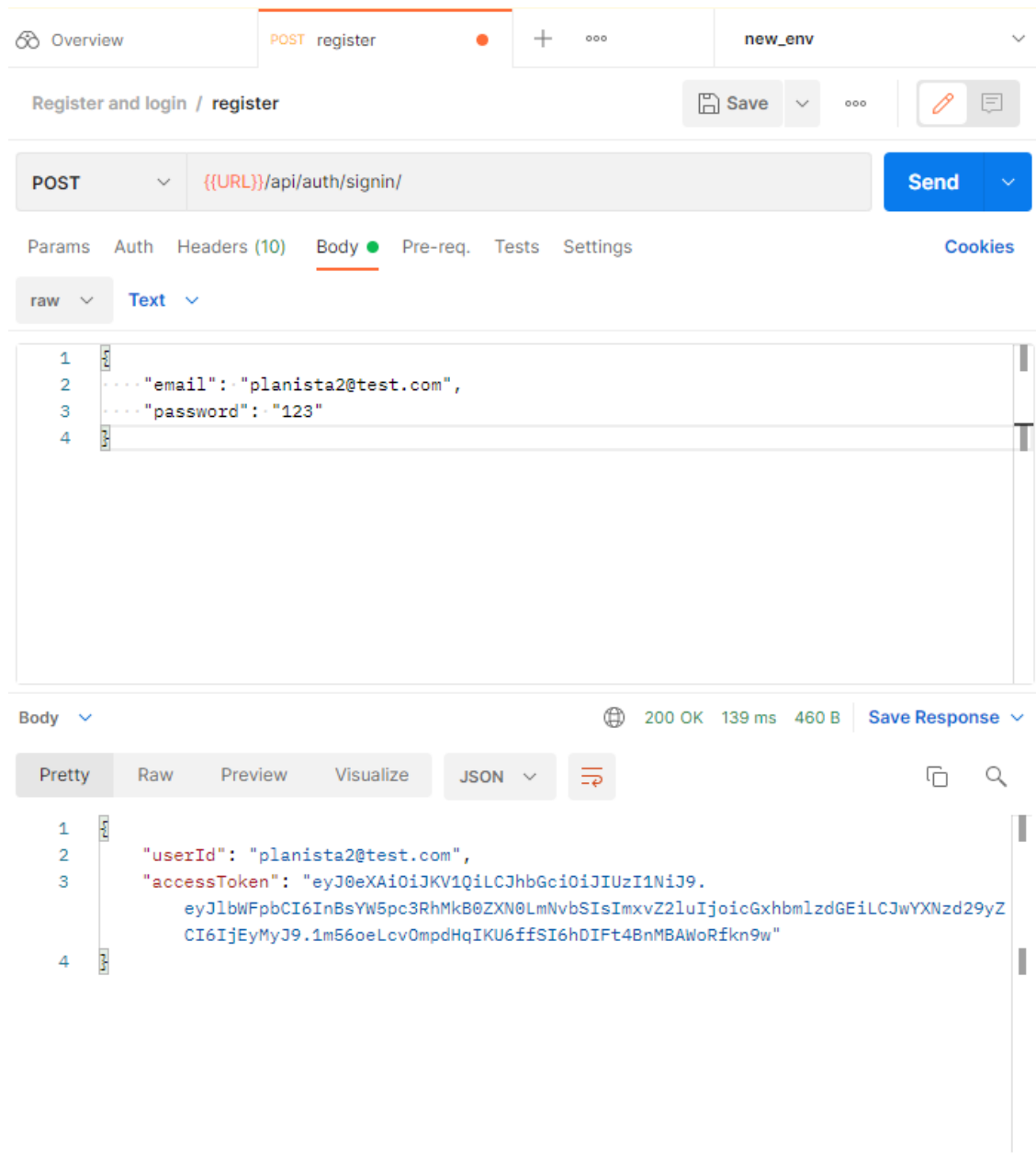
```
1 describe('userInput', () => {
2   test('addSubject', async () => {
3     const wrapper = mount(Step2)
4     const button = wrapper.find('addSubjectButton')
5
6     expect(subjectNumber).toBe(0)
7     await button.trigger('click')
8     expect(subjectNumber).toBe(1)
9   })
})
```

LISTING 8.2: Test interfejsu użytkownika

## 8.2 Backend

### 8.2.1 Testowanie manualne

Część backendowa, testowana była w dużym stopniu manualnie. Głównym narzędziem była aplikacja Postman. Przykładowe zapytanie wraz z odpowiedzią pokazano na rysunku 8.1.



RYСУNEK 8.1: Przykładowe zapytanie oraz odpowiedź w Postman

Na zaawansowanych etapach implementacji, testy wykonywane były również z pomocą aplikacji frontendowej, dzięki czemu można była na bieżąco weryfikować integrację.

### 8.2.2 Testy automatyczne

Testami automatycznymi objęte zostały widoki oraz adresy URL. Do testowania widoków, wykorzystywane są testy integracyjne, natomiast do adresów testy jednostkowe. W obu przypadkach wykorzystywane są gotowe rozwiązania zaimplementowane w Django. Testy jednostkowe,

sprawdzające poprawność implementacji adresów internetowych, sprawdzają czy podczas wysłania zapytania na dany adres, wywoływana jest odpowiednia funkcja. Przykład implementacji takiego testu na listingu 8.3.

```
1 class TestUrls(SimpleTestCase):
2
3     def test_add_user(self):
4         url = reverse('user')
5         self.assertEqual(resolve(url).func, add_user)
```

LISTING 8.3: Implementacja przykładowego testu jednostkowego

Natomiast testy integracyjne, odpowiadające za widoki, mają za zadanie sprawdzić czy widok zwrócił odpowiednią odpowiedź HTTP. Przykład takiego testu na listingu 8.4.

```
1 class TestViews(TestCase):
2
3     def test_add_user_POST(self):
4
5         email = 'test1@test2.com'
6         client = Client()
7         url = reverse('user')
8
9         Planners.objects.create(
10             planneremail = email,
11             login = 'test',
12             password = 'test'
13         )
14
15         response = client.post(url, {
16             'Sukces': 'Pomyślnie dodano użytkownika'
17         })
18
19         self.assertEqual(response.status_code, 200)
```

LISTING 8.4: Implementacja przykładowego testu integracyjnego

## 8.3 Algorytm

### 8.3.1 Testy manualne

Testy manualne algorytmu przeprowadzane były przy pomocy przykładowych danych pierwotnie przygotowanych przez grupę inżynierską oraz ostatecznie uzyskanych z przykładowej istniejącej szkoły średniej. Dzięki wykorzystaniu danych z prawdziwej szkoły powstała możliwość porównania planów generowanych przez algorytm z planem stworzonym przez wyszkolonego planistę w szkole średniej. Dzięki temu możliwe było zwrócenie uwagi na niedoskonałości planu wygenerowanego względem tego stworzonego przez planistę oraz wyeliminowanie ich. Dzięki takim testom manualnym plany generowane przez algorytm udało się doprowadzić do stanu co najmniej bliskiego w stosunku do planu stworzonego przez wyszkolonego planistę.

### 8.3.2 Testy automatyczne

Automatyczne testy jednostkowe algorytmu odbywają się przy pomocy biblioteki *pytest*. Dzięki tej bibliotece testy napisane w języku Python można wywoływać przy pomocy jednej komendy. Przykładowym testem jednostkowym jest sprawdzenie czy dane uzyskane z backendu zostały prawidłowo przetworzone przez algorytm w celu ułatwienia wykonywania na nich różnych operacji.

Poniższy test sprawdza, czy ilość danych przygotowanych dla algorytmu zgadza się z ilością otrzymanych danych (zob. listing 8.5).

```

1  def test_data_structures(groups_data=GROUP_OLD, teachers_data=TEACHERS_OLD,
2      classrooms_data=CLASSES_OLD):
3      school = School(groups_data, teachers_data, classrooms_data)
4      assert len(school.groups) == len(groups_data), "Groups copied incorrectly to
      dict of class Group"
5      assert len(school.classrooms) == len(classrooms_data), "Classrooms copied
      incorrectly to dict of class Classroom"
6      assert len(school.teachers) == len(teachers_data), "Teachers copied incorrectly
      to dict of class Teacher"

```

LISTING 8.5: Implementacja przykładowego testu jednostkowego sprawdzającego spójność danych

Kolejnym przykładem automatycznego testu jednostkowego jest sprawdzenie samego procesu ewolucji. Ocena gotowego najlepszego planu z pierwszego pokolenia jest zapisywana. Następnie przeprowadzony zostaje proces ewolucji. Ocena najlepszego planu uzyskanego z ostatniego pokolenia jest zapisywana. Podczas testu sprawdzane jest czy ocena uzyskana przez plan po procesie ewolucji jest lepsza, niż planu przed rozpoczęciem ewolucji (zob. listing 8.6).

```

1  def test_evolution(groups_data=GROUP_OLD, teachers_data=TEACHERS_OLD,
2      classrooms_data=CLASSES_OLD):
3      population_size = 10
4      num_of_generations = 100
5      num_of_mutations = 20
6
7      population = Population(groups_data=groups_data, teachers_data=teachers_data,
8          classrooms_data=classrooms_data)
9      population.new_population(number_of_instances=population_size)
10     before_evolution = population.get_best_specimen().evaluation
11     population.evolute(num_of_generations, num_of_mutations)
12     after_evolution = population.get_best_specimen().evaluation
13
14     assert before_evolution < after_evolution, "Before evaluation is not smaller
15         than after evaluation"
16     print("Evaluation before evolution is smaller than after evolution")

```

LISTING 8.6: Implementacja przykładowego testu jednostkowego sprawdzającego proces ewolucji

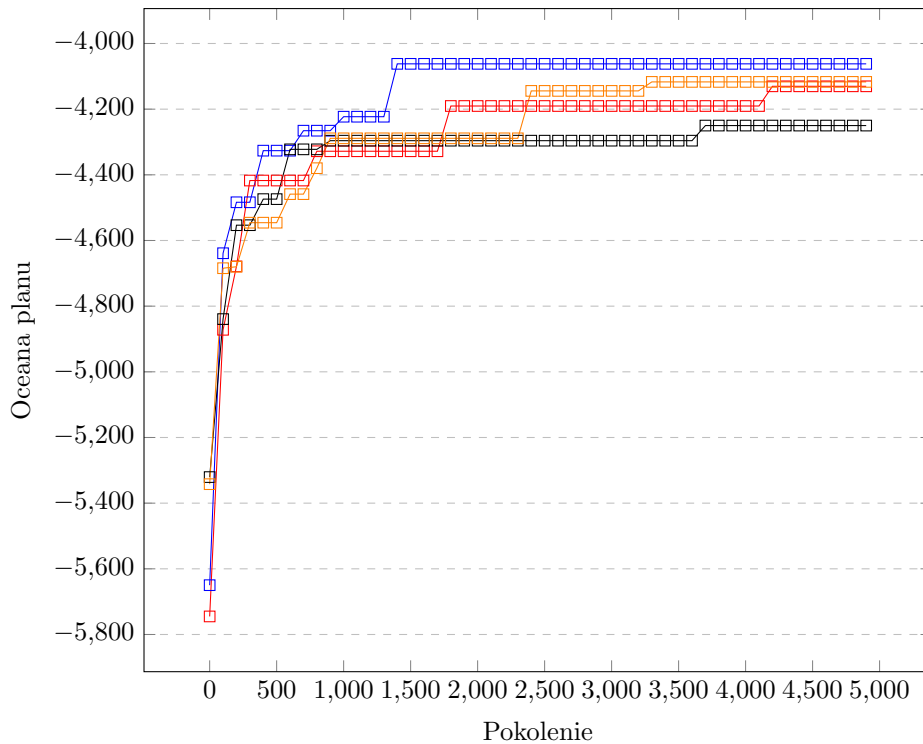
### 8.3.3 Testy jakościowe i wydajnościowe

Wykonano testy jakościowe oraz wydajnościowe dla przykładowego zestawu danych. Dane pochodziły z prawdziwej szkoły średniej dzięki czemu możliwe było odniesienie wyników do warunków rzeczywistych. Po przetworzeniu danych otrzymane została lista krotek o długości 602. Przeprowadzony zostały po 4 testy dla różnych ustawień algorytmu.

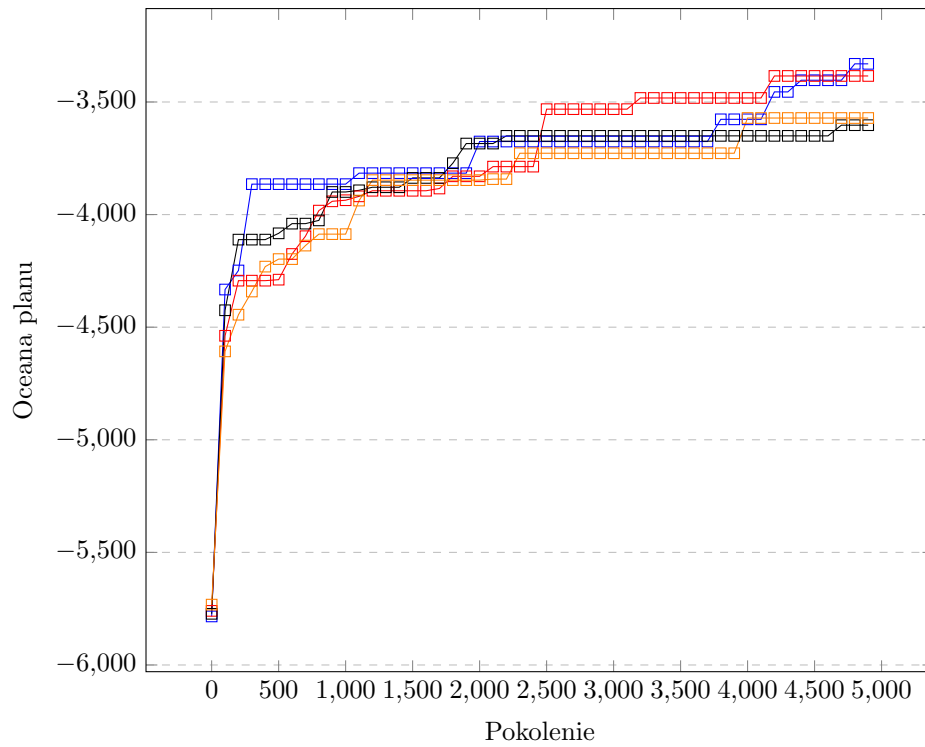
- Pierwszy zestaw testów został przeprowadzony dla ustawień algorytmu w postaci: rozmiar populacji = 10, liczba pokoleń = 5000, liczba mutacji = 3% liczby krotek w liście (zaokrąglone w dół). Zauważyć można bardzo niską powtarzalność otrzymanych wyników (zob. rysunek 8.2). Wynika to z dość dużej liczby wprowadzanych losowych zmian w jednym czasie. Przez taki zabieg plan zajęć mocno ztraca element ewolucji.

- Drugi zestaw testów przeprowadzony został dla następujących ustawień algorytmu: rozmiar populacji = 10, liczba pokoleń = 5000, liczba mutacji = 2% liczby krotek w liście (zaokrąglone w dół). Zauważyć można delikatny wzrost powtarzalności wyników idący w parze ze wzrostem otrzymanej finalnej oceny planu zajęć (zob. rysunek 8.3).
- Trzeci zestaw testów wykonany został dla ustawień algorytmu w postaci: rozmiar populacji = 10, liczba pokoleń = 5000, liczba mutacji = 1% liczby krotek w liście (zaokrąglone w dół). Dla tych ustawień zauważyć można delikatnie zwiększoną powtarzalność uzyskiwanych wyników wraz z minimalnym wzrostem uzyskanej oceny końcowej względem liczby mutacji wynoszącej 2% (zob. rysunek 8.4).
- Czwarty zestaw testów został wykonany dla następujących parametrów przyjętych przez algorytm: rozmiar populacji = 10, liczba pokoleń = 5000, liczba mutacji = 0.5% liczby krotek w liście (zaokrąglone w dół). Dla takiego zestawu ustawień można zauważyć zarówno ogromny wzrost powtarzalności uzyskiwanych wyników jak i bardzo wysoki wynik końcowy ocenianych planów zajęć (zob. rysunek 8.5).

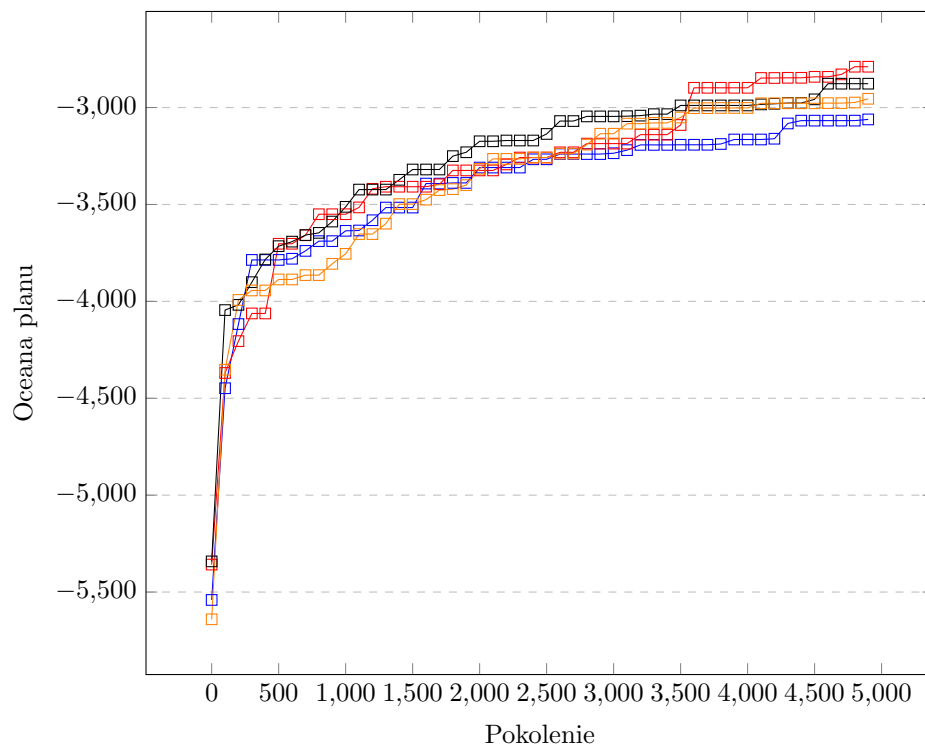
Niestety liczba mutacji w postaci 0.5% liczby krotek dla tego zestawu danych mocno zbliżyła się do minimalnej wartości jaką może przyjąć algorytm dla tego parametru (2) dlatego w tym miejscu testowanie z tak przyjętymi parametrami zostało zakończone. Każdy pojedynczy test z przedstawionych zestawów został wykonany zarówno na komputerze osobistym o specyfikacji (Intel Core i7 4790k, 16GB RAM DDR3-1866MHz) jak i instancji AWS EC2 t2.micro (1 core, 1GB RAM). Dla obu platform czas przebiegu testu (wygenerowania planu o otrzymanej ocenie końcowej) był mocno powtarzalny jednak znacznie się różniący pomiędzy platformami. Komputerowi osobistemu wykonanie jednego testu zajęło ok. 4 minut. Instancji t2.micro wykonanie jednego testu zajęło ok. 30 minut.



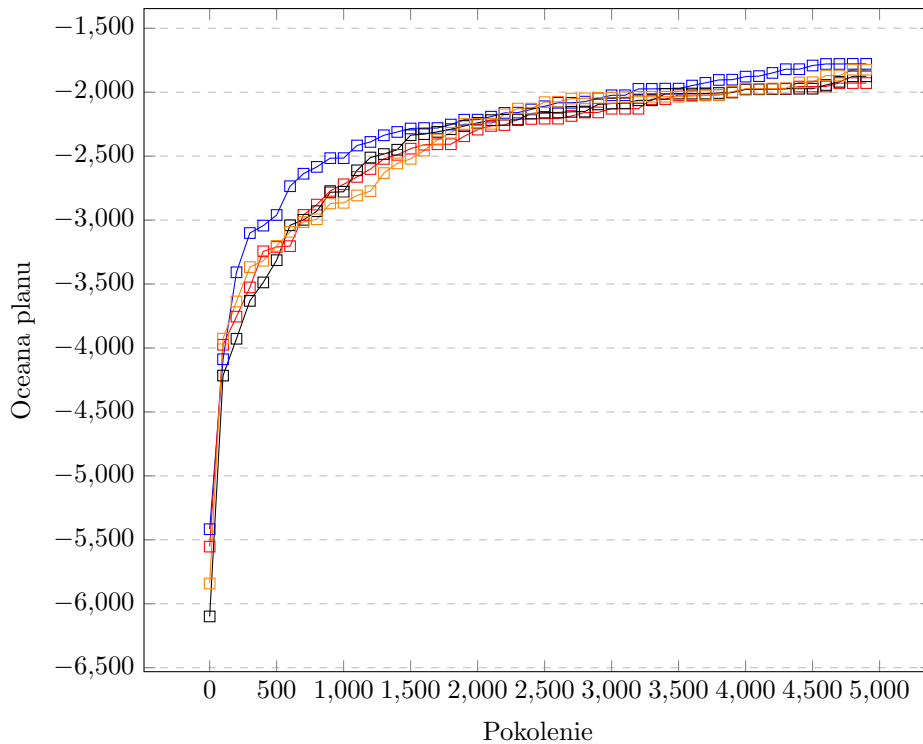
RYSUNEK 8.2: Zmiana oceny planu podczas ewolucji badana co 100 pokoleń.  
Rozmiar populacji = 10, Liczba pokoleń = 5000, Liczba mutacji = 3% krotek  
(więcej=lepiej)



RYSUNEK 8.3: Zmiana oceny planu podczas ewolucji badana co 100 pokoleń.  
Rozmiar populacji = 10, Liczba pokoleń = 5000, Liczba mutacji = 2% krotek  
(więcej=lepiej)

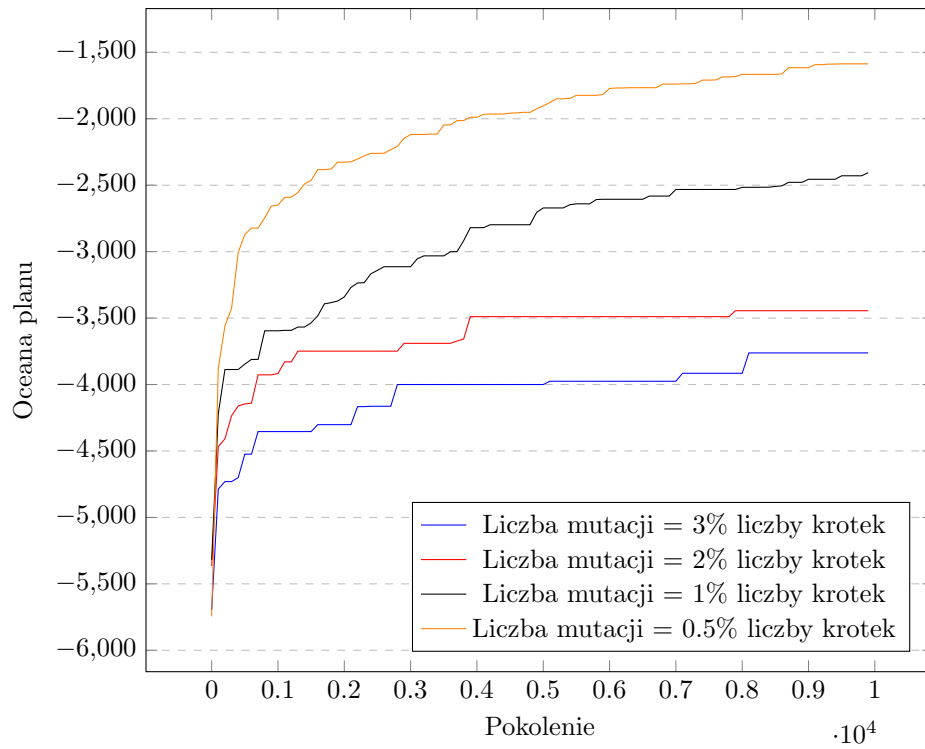


RYSUNEK 8.4: Zmiana oceny planu podczas ewolucji badana co 100 pokoleń.  
Rozmiar populacji = 10, Liczba pokoleń = 5000, Liczba mutacji = 1% krotek  
(więcej=lepiej)



RYSUNEK 8.5: Zmiana oceny planu podczas ewolucji badana co 100 pokoleń.  
 Rozmiar populacji = 10, Liczba pokoleń = 5000, Liczba mutacji = 0.5% krotek  
 (więcej=lepiej)

Ostatni zestaw testów został przeprowadzony po jednym teście z każdego z powyższych zestawów ze zmianą liczby pokoleń na 10000. Takie ustawienie parametrów pozwoliło na znalezienie trudnych do przejścia progów dla każdego ustawienia. Testy zostały zestawione w ramach jednego wykresu w celu ułatwienia porównania otrzymanych wyników (zob. rysunek 8.6). Zauważyć można, że dla wartości 3% oraz 2% bardzo szybko napotkany zostaje próg nie do przekroczenia przez zbyt dużą losowość. Przez taki próg marnowane są ogromne zasoby obliczeniowe generujące koszty, bez idącego za nimi oczekiwanego wzrostu jakości uzyskanego planu. Z wykresu bardzo szybko wywnioskować można, że najlepszym ustawieniem jest liczba mutacji = 0.5%. Przez pełen okres wykonywania testu dla tego parametru jakość otrzymywanego planu zwiększała się. Zauważyć można, że optymalnym ustawieniem parametrów dla zaimplementowanego algorytmu jest: rozmiar populacji = 10, liczba pokoleń = 1500, liczba mutacji = 0.5% liczby krotek w liście (zaokrąglone w dół). Z rysunku przeczytać można, że po przekroczeniu ok.1500 pokolenia dla liczby mutacji = 0.5% wykres znacznie się spłaszcza, co oznacza, że wykorzystując dużą ilość zasobów do wygenerowania kolejnych 8500 pokoleń nie jest otrzymywany już oczekiwany wzrost jakości.



RYSUNEK 8.6: Zmiana oceny planu podczas ewolucji badana co 100 pokoleń.  
Rozmiar populacji = 10, Liczba pokoleń = 5000  
(więcej=lepiej)

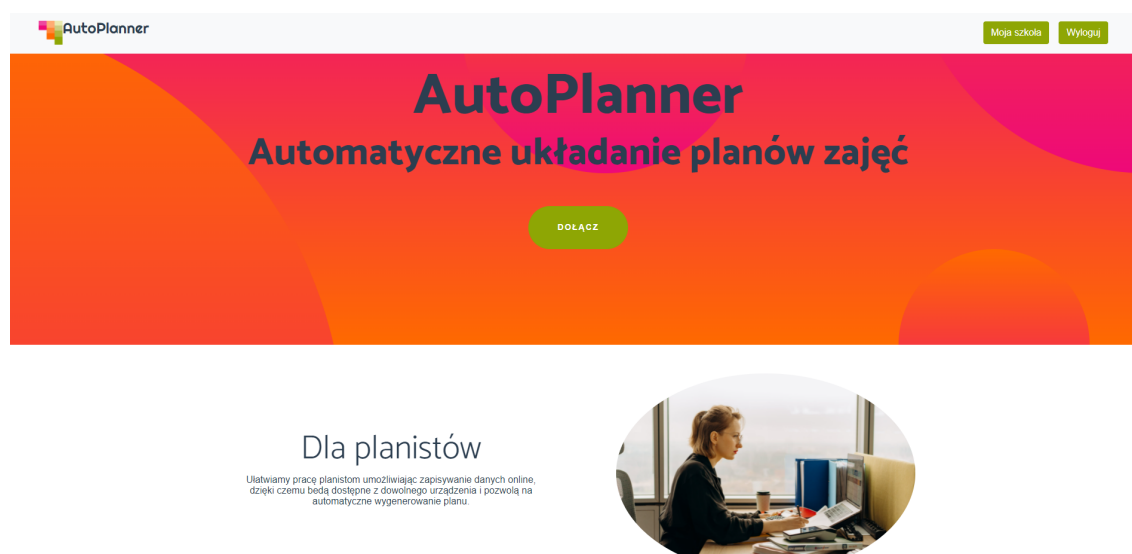


## Rozdział 9

# Instrukcja użytkowania

### 9.1 Strona główna

Strona główna (zob. rysunek 9.1) aplikacji została stworzona na bazie szablonu Bootstrap o nazwie One Page Wonder [22]. Zawiera ona krótki opis aplikacji, a także korzyści płynących z wykorzystania jej dla planistów, nauczycieli oraz uczniów. Pasek menu znajdujący się zawsze na górze strony jest stałym elementem aplikacji pojawiającym się w każdym z widoków. Pozwala on na przejście do widoków logowania i rejestracji, a w przypadku gdy użytkownik jest już zalogowany na wylogowanie lub przejście do widoku szkoły.



RYSUNEK 9.1: Aplikacja internetowa – Strona główna

### 9.2 Widok szkoły

Widok szkoły (zob. rysunek 9.2) pozwala na przejście do dodawania danych potrzebnych do wygenerowania planu, a przypadku gdy plan został już wygenerowany jest również miejscem, w którym jest on wyświetlany. Rozkład zajęć jest możliwy do wyświetlenia na trzy sposoby – z podziałem na klasy, nauczycieli lub sale lekcyjne.

AutoPlanner

Moja szkoła Wyloguj

Rozpocznij dodawanie danych

Plany dla klas Plany dla nauczycieli Plany dla sali

	Poniedziałek	Wtorek	Środa	Czwartek	Piątek
1A					
1	matematyka	chemia	chemia	wf	chemia
2	matematyka	biologia	j. polski	wf	j. polski
3	matematyka	religia	wf	wf	j. polski
4	j. polski	religia	biologia	biologia	fizyka
5	fizyka	matematyka	biologia		
6		matematyka	matematyka		

	Poniedziałek	Wtorek	Środa	Czwartek	Piątek
1B					
1	j. polski	j. polski	wf	j. polski	matematyka
2	j. polski	fizyka	wf	matematyka	religia
3	j. polski	matematyka	j. polski	matematyka	chemia
4	matematyka	matematyka	fizyka	religia	biologia
5	matematyka	biologia	fizyka	fizyka	
6					

RYSUNEK 9.2: Aplikacja internetowa – Widok szkoły

### 9.3 Rejestracja

Widok rejestracji (zob. rysunek 9.3) umożliwia utworzenie konta w serwisie. Od użytkownika wymaga się podania adresu e-mail, nazwy użytkownika oraz hasła. Adres e-mail musi być unikatowy. Wynika to z konieczności weryfikacji konta poprzez wiadomość wysłaną przy pomocy serwera SMTP. Rozwiązanie to ma na celu zapobieganie atakom na stronę poprzez masowe tworzenie nowych kont.

AutoPlanner

Logowanie Rejestracja

### Zarejestruj się

Email

Nazwa użytkownika

Hasło

Zarejestruj

RYSUNEK 9.3: Aplikacja internetowa – Widok rejestracji

### 9.4 Logowanie

AutoPlanner

Logowanie Rejestracja

### Witaj ponownie!

Email

Hasło

Zaloguj

[Zapomniałeś hasła?](#) [Nie masz konta? Zarejestruj się!](#)

RYSUNEK 9.4: Aplikacja internetowa – Widok logowania

Widok logowania (zob. rysunek 9.4) pozwala na dostęp do konta i zapisanych na nim danych z dowolnego urządzenia. Do uwierzytelnienia użytkownika wykorzystywany jest adres e-mail oraz

hasło podane w procesie rejestracji. Powodzenie procesu logowania powoduje otrzymanie przez aplikację tokenu JWT, zapisywanego w pamięci przeglądarki. W przypadku utraty hasła użytkownik posiada możliwość odzyskania go po podaniu adresu e-mail powiązanego z istniejącym kontem.

## 9.5 Ankieta

Widok ankiety (zob. rysunek 9.5) pozwala nauczycielowi na podanie preferencji godzinowych pracy. W przeciwieństwie do pozostałych widoków jest on dostępny jedynie poprzez bezpośredni link wysyłany w wiadomości e-mail. Takie rozwiązanie sprawia, że jedynym użytkownikiem, od którego wymagane jest posiadanie konta jest planista. Nauczyciele jako użytkownicy bez konta są identyfikowani dzięki unikalności adresu URL.

#	Poniedziałek	Wtorek	Środa	Czwartek	Piątek
1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Wyślij

RYСУNEK 9.5: Aplikacja internetowa – Widok ankiet dla nauczycieli

## 9.6 Dodawanie przedmiotów

Krok 1 - Dodaj przedmioty

biologia  
chemia  
fizyka  
j. polski  
matematyka  
religia  
wf

Nazwa przedmiotu

Dodaj  
Przejdź dalej  
Poprzedni krok

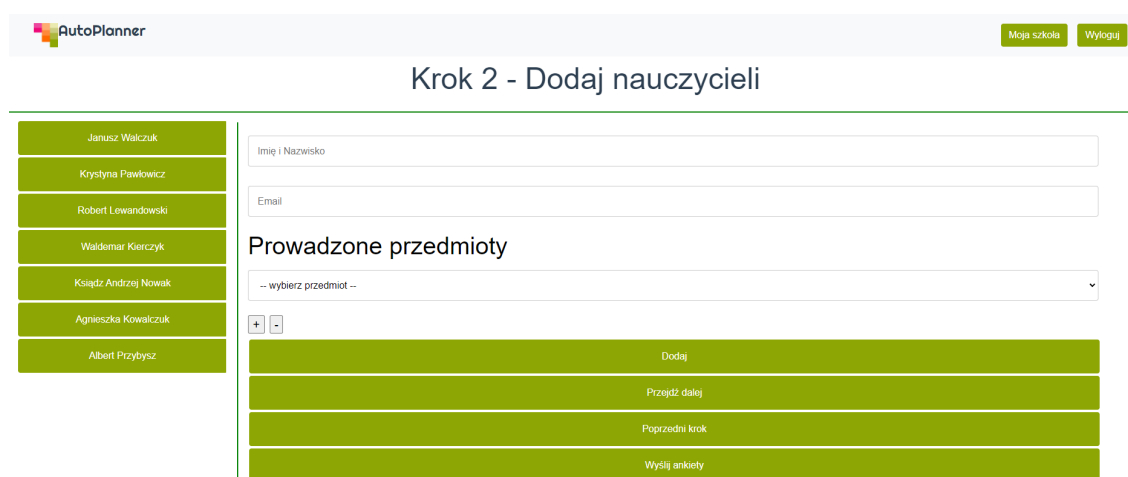
RYСУNEK 9.6: Aplikacja internetowa – Widok dodawania przedmiotów

Widok dodawania przedmiotów (zob. rysunek 9.6) stanowi pierwszy krok w procesie podawania danych koniecznych do wygenerowania planu zajęć. W celu dodawania przedmiotu należy podać jedynie jego nazwę. Powiązania z nauczycielami, salami lekcyjnymi i klasami będą mogły być wprowadzone w kolejnych krokach. Podanie nazw przedmiotów na początku procesu dodawania danych pozwala na to, aby w późniejszych etapach mogły być one wybierane z listy rozwijanej. Zapobiega to konieczności wielokrotnego ręcznego wprowadzania tych samych informacji i ułatwia tworzenie powiązań w bazie danych. W lewej części ekranu znajduje się lista już wprowadzo-

nych przedmiotów. Wybranie z nich jednego powoduje przejście do ekranu edycji. Analogiczne rozwiązanie zostało zastosowane we wszystkich kolejnych ekranach dodawania danych.

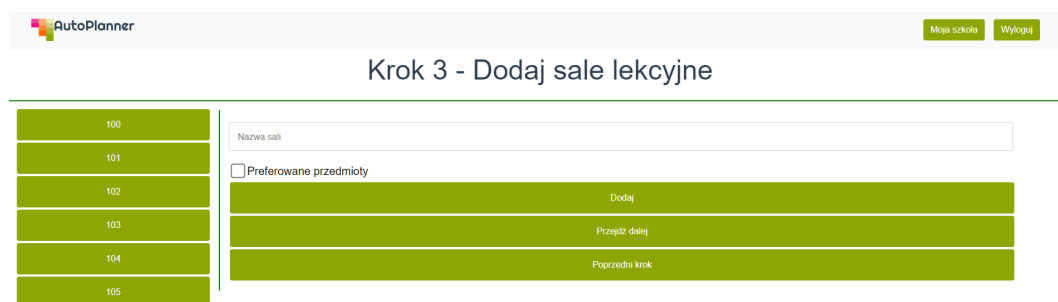
## 9.7 Dodawanie nauczycieli

W widoku dodawania nauczycieli (zob. rysunek 9.7) planista ma możliwość wprowadzenia danych personelu dydaktycznego oraz jego powiązań z przedmiotami. Każdy nauczyciel musi posiadać imię i nazwisko, unikalny w skali szkoły adres e-mail oraz przynajmniej jeden prowadzony przedmiot. Ekran umożliwia dodanie kolejnych prowadzonych przedmiotów w przypadku, gdy nauczyciel prowadzi więcej niż jeden. Planista ma również możliwość wysłania ankiet dyspozycyjności ma adresy e-mail do tej pory wprowadzonych nauczycieli.



RYСУNEK 9.7: Aplikacja internetowa – Widok dodawania nauczycieli

## 9.8 Dodawanie sal lekcyjnych



RYСУNEK 9.8: Aplikacja internetowa – Widok dodawania sali lekcyjnych

Dodanie informacji o salach lekcyjnych (zob. rysunek 9.8) stanowi trzeci krok dodawania danych. Sala lekcyjna musi posiadać nazwę, a opcjonalnie także listę przedmiotów, które mogą być w niej prowadzone. W przypadku, gdy nie zostanie wybrany żaden preferowany przedmiot, sala zostaje uznana za salę zwykłą, co oznacza, że będzie mógł być w niej prowadzony dowolny przedmiot.

## 9.9 Dodawanie klas

Ostatnim etapem w procesie dodawania niezbędnych danych jest wprowadzenie parametrów klas (zob. rysunek 9.9). Każda klasa musi posiadać nazwę oraz listę przedmiotów, które mają się pojawić w jej planie zajęć. Każdy element listy musi zawierać nazwę przedmiotu, liczbę godzin lekcyjnych w tygodniu przeznaczonych na przedmiot oraz opcjonalnie prowadzącego przedmiot. Brak wyboru nauczyciela umożliwia przypisanie zajęć dowolnemu prowadzącemu dany przedmiot.

The screenshot shows the 'Krok 4 - Dodaj klasy' interface. On the left is a sidebar with buttons for class selection: 1A, 1B, 1C, and 1D. The main content area has a header 'Krok 4 - Dodaj klasy'. Below it is a form with a 'Nazwa klasy' input field. Underneath is a section titled 'Lista przedmiotów' containing a dropdown menu with the text '-- wybierz przedmiot --'. To the right of the dropdown are two input fields: 'Liczba godzin tygodniowo' and 'Liczba godzin', followed by a checkbox and another dropdown menu with the text '-- wybierz prowadzącego --'. Below these fields are three buttons: '+', '-', and a large green 'Dodaj' button. At the bottom are two more green buttons: 'Przejdź dalej' and 'Poprzedni krok'.

RYSUNEK 9.9: Aplikacja internetowa – Widok dodawania klas

## 9.10 Edycja danych

The screenshot shows the 'Edycja danych nauczyciela' interface. On the left is a sidebar with buttons for teacher selection: Janusz Walczuk, Krystyna Pawłowicz, Robert Lewandowski, Waldemar Kierczyk, Ksiądz Andrzej Nowak, Agnieszka Kowalczyk, and Albert Przybysz. The main content area has a header 'Edycja danych nauczyciela'. Below it are two input fields: the first contains 'Waldemar Kierczyk' and the second contains 'w.kierczyk@wp.pl'. Below these is a section titled 'Prowadzone przedmioty' with a dropdown menu showing 'chemia'. Below the dropdown are three buttons: '+', '-', and a large green 'Zapisz zmiany' button. At the bottom are two more green buttons: 'Wróć bez zapisywania' and 'Usuń nauczyciela'.

RYSUNEK 9.10: Aplikacja internetowa – Widok edycji danych nauczyciela

Dla czterech powyższych ekranów istnieją odpowiadające im ekrany edycji danych. Ze względu na ich analogiczną budowę ich struktura zostanie omówiona na bazie widoku edycji danych nauczyciela (zob. rysunek 9.10). Przejście do tego ekranu umożliwiają przyciski znajdujące się po prawej stronie widoku dodawania nauczycieli. Przyciski te są wciąż obecne w widoku edycji i pozwalają na przechodzenie pomiędzy danymi poszczególnych osób bez zapisywania wprowadzonych zmian. W chwili przejścia do ekranu edycji pola z danymi zostają uzupełnione pierwotnie wprowadzonymi informacjami o nauczycielu. Takie rozwiązanie ma celu zapobieganie konieczności ponownego wprowadzania wszystkich danych, w przypadku gdy tylko niektóre z nich wymagają zmian. Przyciski u dołu pozwalają na zapisanie wprowadzonych zmian, powrót do ekranu dodawania nauczycieli lub całkowite usunięcie nauczyciela z bazy danych.

## Rozdział 10

# Zakończenie

### 10.1 Podsumowanie pracy

Celem pracy inżynierskiej była implementacja aplikacji internetowej służącej do generacji planów zajęć dla szkół podstawowych oraz średnich. Motywacją do podjęcia takiego tematu, był fakt, że układanie planów zajęć jest zadaniem trudnym [34] [37]. W dużych szkołach plan może być układany nawet miesiącami. Zautomatyzowanie procesu układania planu zajęć może zaoszczędzić pracy osobom, które za to odpowiadają. Dodatkowo pojawia się potrzeba dobrego gospodarowania czasem uczniów oraz nauczycieli, czego ręcznie układane plany zajęć nie zapewniają.

Szczególne uwagi zostały zwrócone na to, by aplikacja wyświetlała swoją zawartość w przeglądarce w sposób czytelny oraz estetyczny. Jest to jedyna część aplikacji, z którą potencjalny klient miałby bezpośrednią styczność. Oznacza to, że wygląd zawartości wpływałby w znacznym stopniu na odbiór aplikacji przez użytkownika.

Niezwykle ważne było założenie, że projekt ma powstawać w metodyce *DevOps*. Czas włożony na wdrożenie praktyk *DevOps* zaowocował dużą oszczędnością czasu. Fakt, że programowane rozwiązania były wdrażane automatycznie na projektowe maszyny, spowodował znaczne przyspieszenie postępu prac. Pozwoliło to również na ograniczenie zagrożeń związanych z jednoczesną edycją kodu źródłowego przez więcej niż jedną osobę.

Kluczową częścią projektu była implementacja algorytmu układającego plan zajęć. Problem ten był na tyle złożony, że sam wybór sposobu podejścia nie był trywialną częścią. Ostatecznie zdecydowano się na podejście ewolucyjne. Algorytm pozwala na ułożenie dobrych planów zajęć w szybkim czasie (w porównaniu z ręcznym układaniem planów).

Działanie strony serwerowej zostało oparte na darmowej platformie programistycznej *Django*. Wybór ten zapewnił bezpieczne oraz wydajne działanie całej aplikacji. Część serwerowa obsługuje wszystkie żądania klienta, sprawia to, że jej poprawne działanie było ważną częścią projektu.

Wszystkie wymagania wspomniane w podrozdziale 3.4 udało się spełnić.

### 10.2 Możliwości rozwoju aplikacji

Do aplikacji Autoplaner może zostać w przeszłości dodana funkcjonalność generowania planów zajęć dostosowanych do warunków panujących na uczelniach. Wymagałoby to dodania możliwości swobodnego dzielenia klas na grupy oraz grup na jeszcze mniejsze podgrupy. Przykładowo kierunek studiów można podzielić na cztery grupy dziekańskie, a każdą grupę dziekańską na dwie grupy laboratoryjne. Należałoby również przyjąć sytuację, w której jedna osoba może należeć do kilku grup jednocześnie, gdzie pierwsza grupa może nie być podgrupą drugiej grupy, np. do grupy dziekańskiej i do grupy językowej.

Zaimplementowana może również zostać możliwość określenia rozmiaru grup oraz sal, tzn. liczba osób przynależna do konkretnej grupy oraz liczba miejsc siedzących dostępnych w konkretnej sali. Algorytm musiałby wtedy sprawdzać, czy dana sala ma wystarczającą liczbę miejsc, aby przeprowadzić zajęcia dla konkretnej grupy.

Ciekawym rozwinięciem aplikacji może być również automatyczne tworzenie wirtualnych maszyn, na których działałby wyłącznie algorytm. Każde żądanie utworzenia planu zajęć powodowałoby automatyczne utworzenie maszyny wirtualnej, na której będą wykonywane odpowiednie obliczenia. W przypadku tworzenia kilku planów jednocześnie wyeliminowałoby to problem dzielenia zasobów przez szkoły. W takim modelu za utrzymanie konkretnej maszyny odpowiadałaby szkoła, a po wykonaniu odpowiednich obliczeń maszyna wirtualna zostałaby usunięta. Wymagałoby to głębokiej integracji z konkretnym dostawcą usług internetowych takich jak np. *Amazon Web Services*.

Następną możliwą funkcjonalnością jest pokazywanie czasu potrzebnego do wygenerowania konkretnego planu zajęć oraz określania, czy przy generacji planu bardziej klientowi będzie zależeć na czasie wygenerowania planu, czy na jego dobrej jakości. W połączeniu z funkcjonalnością z poprzedniego akapitu klient mógłby nawet określić parametry generowanej wirtualnej maszyny.

W celu komercjalizacji projektu konieczna byłaby integracja z wybranym operatorem płatności internetowych. Klient mógłby wtedy płacić od razu z poziomu aplikacji. Cena usługi generacji planu zajęć byłaby uzależniona od czasu potrzebnego na wygenerowanie planu zajęć. Należałoby wtedy uwzględnić to, że w przypadku jednoczesnej pracy kilku instancji algorytmu, działanie algorytmu byłoby dłuższe. Wspomniany problem nie występowałby w przypadku zaimplementowania funkcjonalności automatycznej generacji wirtualnych maszyn.

# Literatura

- [1] Amazon web services. [on-line] <https://aws.amazon.com/>. (29.12.2021).
- [2] aSc TimeTables. [on-line] <https://www.asctimetables.com/#!/home>. (29.12.2021).
- [3] Axios. [on-line] <https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html>. (29.12.2021).
- [4] Black. [on-line] <https://github.com/psf/black>. (17.01.2022).
- [5] Bootstrap. [on-line] <https://getbootstrap.com/docs/5.1/getting-started/introduction/>. (29.12.2021).
- [6] Ci/cd. [on-line] <https://www.redhat.com/en/topics/devops/what-is-ci-cd>. (29.12.2021).
- [7] Circleci. [on-line] <https://circleci.com/>. (29.12.2021).
- [8] Circleci diagram. [on-line] <https://circleci.com/docs/2.0/about-circleci/>. (29.12.2021).
- [9] Devops. [on-line] <https://www.atlassian.com/devops>. (29.12.2021).
- [10] Django. [on-line] <https://www.djangoproject.com/start/overview/>. (16.01.2022).
- [11] Docker. [on-line] <https://www.docker.com/>. (17.01.2022).
- [12] Git. [on-line] <https://git-scm.com/>. (29.12.2021).
- [13] Github. [on-line] <https://github.com/>. (29.12.2021).
- [14] Heuristic. [on-line] <https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/solving-hard-problems/a/using-heuristics>. (29.12.2021).
- [15] Heuristic. [on-line] <https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac>. (29.12.2021).
- [16] Jest. [on-line] <https://vue-test-utils.vuejs.org/guides/>. (29.12.2021).
- [17] Json web tokens. [on-line] <https://jwt.io/introduction>. (29.12.2021).
- [18] Mysql. [on-line] <https://dev.mysql.com/doc/>. (17.01.2022).
- [19] Mysql workbench. [on-line] <https://www.mysql.com/products/workbench/>. (17.01.2022).
- [20] Node.js. [on-line] <https://nodejs.org/en/about/>. (29.12.2021).
- [21] Np-hard problem. [on-line] <https://xlinux.nist.gov/dads/HTML/nphard.html>. (29.12.2021).
- [22] One page wonder. [on-line] <https://startbootstrap.com/theme/one-page-wonder>. (29.12.2021).
- [23] Optimization problem. [on-line] <https://xlinux.nist.gov/dads/HTML/optimization.html>. (29.12.2021).
- [24] Postman. [on-line] <https://www.postman.com/product/what-is-postman/>. (29.12.2021).
- [25] Prime Timetable. [on-line] <https://primetimetable.com/help/#overview&q>. (29.12.2021).



- [26] Przykładowy schemat devops lifecycle. [on-line]  
<https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg>. (17.01.2022).
- [27] Przykładowy schemat działania rest api. [on-line]  
<https://zaprogramujzycie.pl/wp-content/uploads/2020/10/REST-API-547x480.jpg>.  
 (17.01.2022).
- [28] Pylint. [on-line] <https://pypi.org/project/pylint/>. (17.01.2022).
- [29] Representational state transfer. [on-line]  
[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm). (17.01.2022).
- [30] Schemat przepływu danych django. [on-line] <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction/basic-django.png>. (16.01.2022).
- [31] Schemat przepływu danych w vuex. [on-line] <https://vuex.vuejs.org/vuex.png>. (29.12.2021).
- [32] shell. [on-line] [https://en.wikipedia.org/wiki/Shell\\_\(computing\)](https://en.wikipedia.org/wiki/Shell_(computing)). (17.01.2022).
- [33] Ssh. [on-line] <https://www.ucl.ac.uk/isd/what-ssh-and-how-do-i-use-it>. (17.01.2022).
- [34] Straszny plan lekcji. [on-line]  
<https://chetkowski.blog.polityka.pl/2018/01/08/straszny-plan-lekcji/>. (25.01.2022).
- [35] SuperSaaS. [on-line] [https://www.supersaas.com/info/doc/getting\\_started](https://www.supersaas.com/info/doc/getting_started). (29.12.2021).
- [36] tmux. [on-line] <https://linuxize.com/post/getting-started-with-tmux/>. (17.01.2022).
- [37] Ułożenie planu lekcji to karkołomne zadanie. [on-line] <https://gazetalubuska.pl/ulozenie-planu-lekcji-to-karkolomne-zadanie-gdy-wprowadzilismy-wszystkie-parametry-komputer-odmowil-ar/c1-14394313>. (25.01.2022).
- [38] Version control. [on-line] [https://httpd.apache.org/ABOUT\\_APACHE.html](https://httpd.apache.org/ABOUT_APACHE.html). (29.12.2021).
- [39] Visual studio code. [on-line] <https://code.visualstudio.com/docs>. (29.12.2021).
- [40] Vue.js. [on-line] <https://vuejs.org/v2/guide/>. (29.12.2021).
- [41] Vuex. [on-line] <https://vuex.vuejs.org/>. (29.12.2021).

## Dodatek A

### Załączniki

1. Teksty źródłowe aplikacji Autoplanner.
2. Instrukcja uruchomienia aplikacji – readme.txt.



© 2022 Mateusz Biernacki, Dominik Boła, Maciej Goral, Grzegorz Piątkowski

Instytut Informatyki, Wydział Informatyki i Telekomunikacji  
Politechnika Poznańska

Skład przy użyciu systemu  $\text{\LaTeX}$  na platformie Overleaf.