

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka techniczna (ITE)**
Specjalność: **Inżynieria systemów informatycznych (INS)**

PRACA DYPLOMOWA
INŻYNIERSKA

**Aplikacja klient-serwer do obliczeń
i prezentacji rozwiązań**

**Client-server application for
calculation and presentation
of solutions**

Mateusz Urbańczyk

Opiekun pracy
dr inż. Tomasz Kapłon

Słowa kluczowe: Obliczenia, Matematyka, Aplikacja webowa, Klient-serwer, Django, React

WROCŁAW, 2023

Streszczenie

Prawdopodobnie każdy człowiek miał w swoim życiu taką sytuację, że nie umiał rozwiązać jakiegoś problemu matematycznego. Najczęściej zdarzało się to wtedy, gdy chodził do szkoły. Szukał wtedy rozwiązania w wielu różnych miejscach. Jednym z nich były strony w internecie, które oferują możliwość wykonywania obliczeń oraz ich nauki poprzez analizę rozwiązania krok po kroku. Takich stron naprawdę jest wiele. Niestety większość z nich jest płatnych, a wizualizacja wyniku w postaci rozwiązania krok po kroku nie jest zbyt zrozumiała dla użytkownika. Dlatego też w swojej pracy dyplomowej autor zdecydował się na rozwiązanie problemu tych użytkowników poprzez stworzenie aplikacji webowej o architekturze klient-serwer. W czasie etapu projektowania zdecydowano, że powstały serwis będzie umożliwiał wykonywanie między innymi następujących obliczeń: rozwiązywanie równania liniowego oraz kwadratowego, obliczanie średniej arytmetycznej oraz ważonej liczb, sortowanie bąbelkowe oraz szybkie (quicksort). Implementację części backendowej wykonano w języku Python z wykorzystaniem frameworka Django, a części frontendowej w języku TypeScript z wykorzystaniem biblioteki React. W celu przetestowania oprogramowania wykonano testy jednostkowe, integracyjne oraz funkcjonalne. Powstała aplikacja została wdrożona w serwisie Microsoft Azure i jest dostępna w internecie. W pracy opisane są także perspektywy dalszego rozwoju aplikacji, w dodatku A instrukcja jej wdrożenia oraz w dodatku B zawartość dołączonej płyty CD.

Słowa kluczowe: Obliczenia, Matematyka, Aplikacja webowa, Klient-serwer, Django, React

Abstract

Probably every person had a situation in their life when they could not solve a mathematical problem. In most cases it happened when they were at school. Then they were looking for a solution in many different ways. One of them were websites, that offered the possibility of performing calculations and learning them by analyzing the solution step-by-step. There are many such online sites. Unfortunately, most of them are paid, and the visualization of the result in a form of a step-by-step solution is not very understandable for the user. Therefore, in his diploma thesis, the author decided to solve the problem of these users by creating a web application with a client-server architecture. At the early stage of designing process, author decided that the service would enable user to perform inter alia the following calculations: solving linear and quadratic equations, calculating the arithmetic mean and weighted numbers, bubble sorting, and quicksort. The backend part was implemented in Python using the Django framework, and the frontend part in TypeScript using the React library. In order to test the software, few additional test concerning functionality, unit and integration were performed. The resulting application has been deployed in Microsoft Azure and is available on the Internet. The work also describes the prospects for further development of the application, in Appendix A the instructions for its deployment, and in Appendix B the contents of the attached CD.

Keywords: Calculations, Maths, Web application, Client-server, Django, React

Spis treści

1. Wstęp	6
1.1. Wprowadzenie	6
1.2. Cel pracy	6
2. Projektowanie	7
2.1. Przedmiot pracy	7
2.2. Przegląd istniejących rozwiązań	7
2.3. Model klient-serwer	9
2.4. Analiza wymagań funkcjonalnych i нефункциональных aplikacji	10
2.4.1. Wymagania funkcjonalne	10
2.4.2. Wymagania нефункциональные	11
2.5. Modelowanie UML	11
2.5.1. Diagram przypadków użycia	11
2.5.2. Opisy przypadków użycia	12
3. Implementacja	14
3.1. Frontend	14
3.1.1. Główne fragmenty kodu	14
3.1.2. Wygląd stron	16
3.2. Backend	25
3.2.1. Główne fragmenty kodu	26
3.3. Opis pozostałych użytych technologii	27
3.3.1. Visual Studio Code	27
3.3.2. GitHub	28
4. Wdrożenie	29
4.1. Wdrażanie aplikacji w chmurze	29
4.2. Docker	30
5. Testowanie	31
5.1. Sposób testowania aplikacji	31
5.2. Testy jednostkowe	32
5.3. Testy integracyjne	33
5.4. Testy funkcjonalne	34
6. Podsumowanie	36
6.1. Podsumowanie pracy	36
6.2. Perspektywy dalszego rozwoju aplikacji	36
Literatura	38
A. Instrukcja wdrożeniowa	40
B. Opis załączonej płyty CD/DVD	42

Spis rysunków

2.1. Wygląd strony głównej serwisu wolframalpha.com [21]	8
2.2. Wygląd aplikacji Microsoft Math Solver [11]	9
2.3. Schemat architektury klient-serwer [6]	9
2.4. Diagram przypadków użycia aplikacji	11
3.1. Strona główna	17
3.2. Strona z równaniem liniowym	18
3.3. Strona z równaniem kwadratowym	19
3.4. Strona z kalkulatorem	20
3.5. Strona ze średnią arytmetyczną	21
3.6. Strona ze średnią ważoną	21
3.7. Strona z sortowaniem bąbelkowym	22
3.8. Strona z sortowaniem szybkim	23
3.9. Strona z obliczaniem odległości pomiędzy punktami	24
3.10. Strona z obliczaniem właściwości trójkąta	25
3.11. Stworzone gałęzie na GitHubie	28
3.12. Przykładowe commity na GitHubie	28
4.1. Widok działającej usługi ContainerRegistry	29
4.2. Widok działającej usługi AppService	30
5.1. Zrzut ekranu z aplikacji Postman	31
5.2. Wynik testów jednostkowych	33
5.3. Wynik testów integracyjnych	34
5.4. Wynik testów funkcjonalnych	35
A.1. Właściwości uruchomionej usługi AppService	40

Spis listingów

3.1.	Główny komponent aplikacji frontendowej	14
3.2.	Przykładowe hook'i	15
3.3.	Wysłanie zapytania przy pomocy biblioteki Axios	16
3.4.	Lista endpointów serwera	26
3.5.	Walidacja początkowa argumentów	26
3.6.	Funkcja obliczająca odległość między punktami	26
4.1.	Dockerfile	30
5.1.	Test jednostkowy funkcji obliczającej odległość między punktami	32
5.2.	Test integracyjny obliczania odległości między punktami poprzez wysłanie żądania do serwera	33
5.3.	Test funkcjonalny przy użyciu narzędzia Selenium podstrony służącej do obliczania odległości pomiędzy punktami	34

Rozdział 1

Wstęp

1.1. Wprowadzenie

Nauka matematyki nie dla wszystkich ludzi jest prosta. Niektórzy bardzo szybko są w stanie nauczyć się rozwiązywania zarówno prostych, jak i skomplikowanych problemów z tej dziedziny. Zazwyczaj wystarczy, że spojrzą na przykładowe rozwiązanie, przemyślą je przez chwilę i pozwala im to na wykonanie analogicznych zadań. Inni natomiast na naukę muszą poświęcić znacznie więcej czasu, ale często mimo tego nie są w stanie całkowicie zrozumieć rozwiązania. Wraz ze wzrostem trudności problemów, większość z nich się poddaje lub szuka pomocy u innych. Może to być nauczyciel w szkole, korepetitor, koledzy i koleżanki lub ktoś inny. Jeżeli te sposoby zawodzą, to szukają rozwiązania w internecie. Wchodzą na różne fora matematyczne i opisują swój problem. Wadą tego sposobu jest to, że odpowiedzi piszą nie zawsze osoby, które posiadają wystarczającą wiedzę w danym temacie, co powoduje, że często nie są one prawidłowe. Dlatego też niektórzy wybierają strony, które pozwalają im na natychmiastowe uzyskanie wyniku obliczeń z takich dziedzin, jak algebra, geometria, statystyka i innych. Portale te pozwalają na bardzo szybkie uzyskanie rozwiązania oraz wskazówki w postaci analizy krok po kroku przebiegu obliczeń, aż do otrzymania wyniku. Niestety zazwyczaj na tych stronach darmowe jest tylko uzyskanie samego wyniku obliczeń w postaci liczbowej. Za wyświetlenie wyniku w inny sposób oraz rozwiązania krok po kroku trzeba zapłacić. Kolejną wadą takich serwisów internetowych jest to, że posiadają one bardzo dużo różnych opcji wykonywania obliczeń i mniej doświadczeni użytkownicy mogą się zniechęcić, nie umiając znaleźć tych, które ich interesują. Dlatego też autor zdecydował się na to, aby w swojej pracy dyplomowej wykonać aplikację webową, która ułatwi życie zarówno osobom uczącym się matematyki, ale również tym, którzy ją znają doskonale, jednak potrzebują wykonać obliczenia, których nie są w stanie wykonać w pamięci albo rozwiązywanie ich jest zbyt czasochłonne lub po prostu osoby te nie chcą wkładać wysiłku i wolą od razu uzyskać rozwiązanie. Sam autor jest osobą, której taka aplikacja by się mogła przydać, gdyby istniała w przeszłości.

1.2. Cel pracy

Celem pracy dyplomowej jest wykonanie aplikacji webowej o architekturze klient-serwer, która będzie alternatywą dla obecnie istniejących rozwiązań i umożliwi użytkownikom korzystającym z niej wykonywanie podstawowych działań matematycznych oraz ich naukę poprzez analizę wskazówek zawierających rozwiązanie krok po kroku. Aplikacja powinna umożliwiać bezpłatny dostęp do funkcji, które na innych stronach są płatne. Korzystanie z niej powinno być możliwe z urządzeń desktopowych i mobilnych.

Rozdział 2

Projektowanie

2.1. Przedmiot pracy

Przedmiotem pracy jest projekt, implementacja oraz wdrożenie aplikacji webowej służącej do wykonywania obliczeń użytkowników oraz umożliwienia im nauki ich. Aplikacja ma pozwolić na wykonywanie podstawowych działań matematycznych, takich jak:

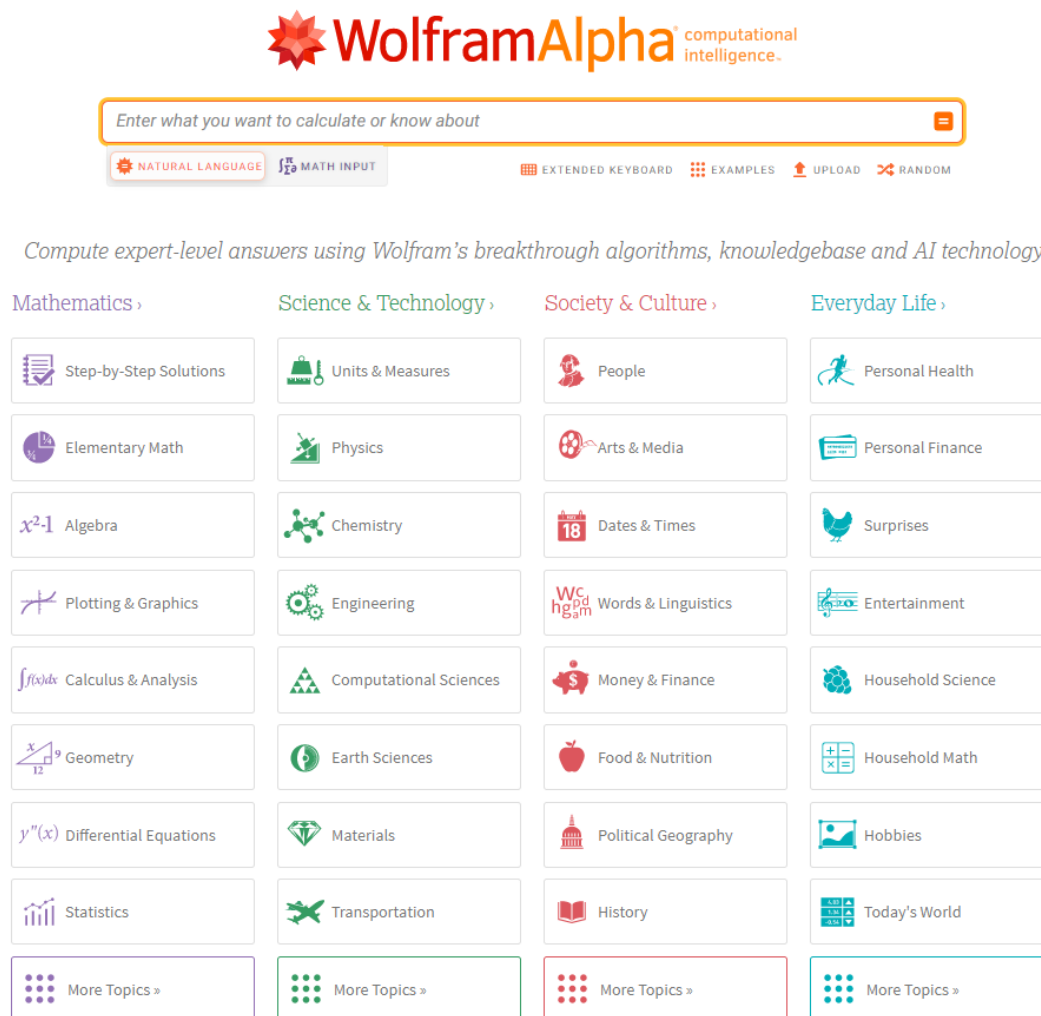
- rozwiązywanie równania liniowego oraz kwadratowego,
- obliczanie średniej arytmetycznej oraz ważonej liczb,
- sortowanie bąbelkowe oraz szybkie (quicksort) liczb,
- obliczanie odległości pomiędzy punktami na płaszczyźnie,
- obliczanie długości boków oraz wartości funkcji trygonometrycznych kątów ostrych trójkąta prostokątnego,
- prostego kalkulatora.

Logowanie do serwisu nie jest potrzebne ze względu na brak przechowywania jakichkolwiek danych użytkowników oraz posiadanie tych samych uprawnień przez nich. Zgodnie z tytułową architekturą aplikacji, objaśnioną w podrozdziale 2.3, wszystkie obliczenia poza tymi wymaganymi dla działania kalkulatora będą wykonywane na serwerze. Komunikacja z serwerem ma odbywać się przy pomocy protokołu HTTPS. Wyniki obliczeń powinny być przejrzyste dla użytkownika, wyświetlane w postaci liczbowej, wykresu oraz rozwiązania krok po kroku. Sposoby wyświetlania wyników mają być dopasowane do wybranego działania. Zakłada się, że serwis będzie składał się ze strony głównej, na której użytkownik ma możliwość wybrania interesującego go działania oraz podstrony z obliczeniami, innej dla każdego działania, w której użytkownik może wpisać jego parametry i zobaczyć wynik. Aplikacja zostanie wdrożona na Microsoft Azure, przy użyciu usługi App Service, która jest darmowa dla osób posiadających licencję studencką.

2.2. Przegląd istniejących rozwiązań

Obecnie istnieje bardzo dużo aplikacji dostępnych w internecie, które pozwalają na wykonywanie obliczeń z różnych dziedzin nauki, nie tylko matematyki, ale także fizyki, chemii i wielu innych, które wymagają obliczeń matematycznych. Większość z nich to profesjonalne strony, które oferują wykonanie obliczeń zarówno bardzo prostych, jak i złożonych. Wizualizacja wyniku na tych stronach odbywa się na wiele sposobów. Użytkownik może odczytać wynik obliczeń w postaci liczbowej oraz graficznej, np. w postaci wykresu, schematu. Serwisy te oferują także możliwość wyświetlenia rozwiązania krok po kroku. Pomocne jest to dla osób, które mają problem ze zrozumieniem danego działania lub po prostu chcą się go nauczyć. Jednym z najbar-

dziej popularnych serwisów umożliwiających wykonywanie obliczeń, który warto opisać, jest serwis wolframalpha.com. Na rysunku 2.1 zaprezentowano wygląd strony głównej portalu.



Rys. 2.1: Wygląd strony głównej serwisu wolframalpha.com [21]

WolframAlpha jest aplikacją webową, istniejącą od maja 2009 r. Jest to strona o bardzo dużych możliwościach. Oferuje ona wykonanie obliczeń z dziedziny algebry, geometrii, statystyki, fizyki, chemii, dat, przeliczania walut, różnych obliczeń inżynierskich, czy nawet liczenia kalorii spożywanych posiłków. Oferuje szereg możliwości wizualizacji wyniku, w zależności od wybranych obliczeń. Niestety większość z funkcji oferowanych przez tę stronę jest płatnych. Wykupienie konta ProPremium, które umożliwia dostęp do wszystkich funkcji nie jest aż tak drogie, bo kosztuje 7,75 funta miesięcznie, ale zniechęca niektórych użytkowników do korzystania z tej strony.

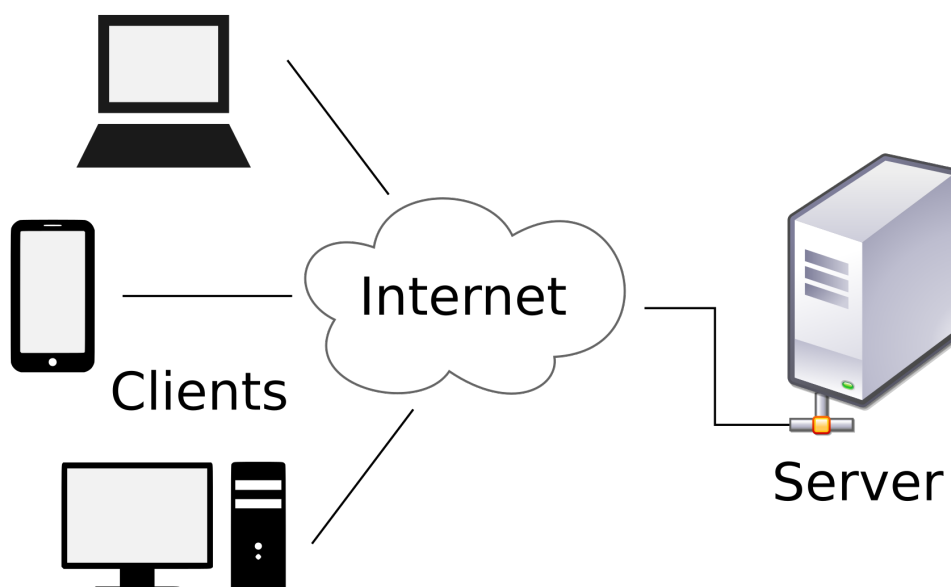
Kolejną aplikacją oferującą możliwość wykonywania obliczeń, którą warto przytoczyć, jest aplikacja mobilna Microsoft Math Solver [10], która pozwala na wykonywanie prostych obliczeń z dziedzin takich jak: algebra, trygonometria, rachunek różniczkowy. Zaletą aplikacji jest możliwość skanowania przy pomocy aparatu działań do wykonania, napisanych na przykład na kartce papieru oraz ich automatyczne wykonywanie. Niestety aplikacja oferuje bardzo ograniczoną listę możliwych obliczeń. Rysunek 2.2 zawiera wygląd aplikacji.



Rys. 2.2: Wygląd aplikacji Microsoft Math Solver [11]

2.3. Model klient-serwer

Klient-serwer (ang. *client-server*, *client-server model*) jest architekturą systemu komputerowego, a zwłaszcza oprogramowania, która umożliwia podział zadań (ról) [7]. Klient to program lub proces, który oprócz wykonywania własnych operacji, łączy się z innym programem lub procesem, zwanym serwerem i poprzez kanały komunikacyjne zleca mu wykonanie określonych działań, w szczególności dostarczenia żądanych danych lub wyników przetwarzania danych. Serwer natomiast jest programem lub procesem, który na zlecenie klientów świadczy określone usługi - obsługuje ich żądania i zwraca im wyniki przetwarzania, np. wykonuje obliczenia lub dostarcza im dane [13]. Na rysunku 2.3 przedstawiono schemat przykładowej architektury.



Rys. 2.3: Schemat architektury klient-serwer [6]

Architektura klient-serwer ma za zadanie umożliwić wielu użytkownikom jednoczesny dostęp do informacji, bez względu na ich lokalizację w sieci. Komunikacja pomiędzy klientami

i serwerem umieszczonymi na różnych komputerach odbywa się najczęściej przy pomocy protokołu TCP/IP. W większości przypadków komunikację rozpoczyna klient, nawiązując połączenie z serwerem. Klient następnie wysyła żądanie do serwera w określonym formacie i czeka na jego odpowiedź. Serwer cały czas oczekuje na zapytania od klientów. W momencie otrzymania żądania przetwarza je i wysyła odpowiedź. Komunikacja pomiędzy stronami w modelu OSI przebiega w warstwie aplikacji [6].

Zalety architektury klient-serwer

- Na serwerze przechowywane są wszystkie informacje, co umożliwia lepsze zabezpieczenie danych. O tym, kto ma prawo do ich odczytywania oraz zmiany może decydować serwer.
- Wiele rozwiniętych technologii wspomaga działanie, użyteczność i bezpieczeństwo tego typu rozwiązania.
- Dzięki centralnemu zarządzaniu bezpieczeństwem sieci oparte na serwerach są o wiele bezpieczniejsze od peer-to-peer.
- Zwiększenie wydajności pracujących w niej komputerów poprzez zdjęcie z nich ciężaru przetwarzania żądań innych klientów. Co więcej, żądania są przetwarzane przez serwer, który najczęściej ma większą moc obliczeniową niż zwykły komputer.
- Usunięcie problemów ze znalezieniem zasobów.

Wady architektury klient-serwer

- Duża liczba klientów jednocześnie próbujących otrzymać dane z jednego serwera powoduje różnego typu problemy związane z przepustowością łącza oraz technicznymi możliwościami przetworzenia ich żądań.
- W czasie, gdy serwer nie działa, dostęp do danych jest całkowicie niemożliwy.
- Do uruchomienia jednostki będącej serwerem z możliwością obsługi dużej liczby klientów potrzebne jest specjalne oprogramowanie oraz sprzęt komputerowy, które nie występują w większości komputerów domowych [26].

Najczęściej spotykanymi podstawowymi serwerami, które działają w oparciu o architekturę klient-serwer są: serwer WWW, serwer poczty elektronicznej, serwer plików oraz serwer aplikacji. Na ogół z usług jednego serwera korzysta wielu klientów jednocześnie. Jeden klient może w tym samym czasie korzystać z usług wielu serwerów. Większość dziś spotykanych systemów zarządzania bazą danych działa w oparciu o architekturę klient-serwer [6].

2.4. Analiza wymagań funkcjonalnych i нефункциональных aplikacji

2.4.1. Wymagania funkcjonalne

- a) Aplikacja powinna umożliwiać wykonywanie obliczeń matematycznych takich jak:
- rozwiązywanie równania liniowego,
 - rozwiązywanie równania kwadratowego,
 - obliczanie odległości pomiędzy punktami na płaszczyźnie,
 - obliczanie długości trzeciego boku trójkąta prostokątnego oraz wartości funkcji trygonometrycznych (sinus, cosinus, tangens, cotangens) jego kątów ostrych na podstawie znanych długości dwóch boków,
 - obliczanie średniej arytmetycznej oraz ważonej liczb,
 - wykonywanie dodawania, mnożenia, dzielenia oraz potęgowania liczb (kalkulator),
 - sortowanie liczb (bąbelkowe, quicksort).

- b) Wynik obliczeń powinien być wyświetlany w postaci liczbowej oraz na wykresie (w zależności od wybranego typu działania).
- c) Powinna istnieć możliwość wyświetlenia rozwiązania krok po kroku (wskazówki).

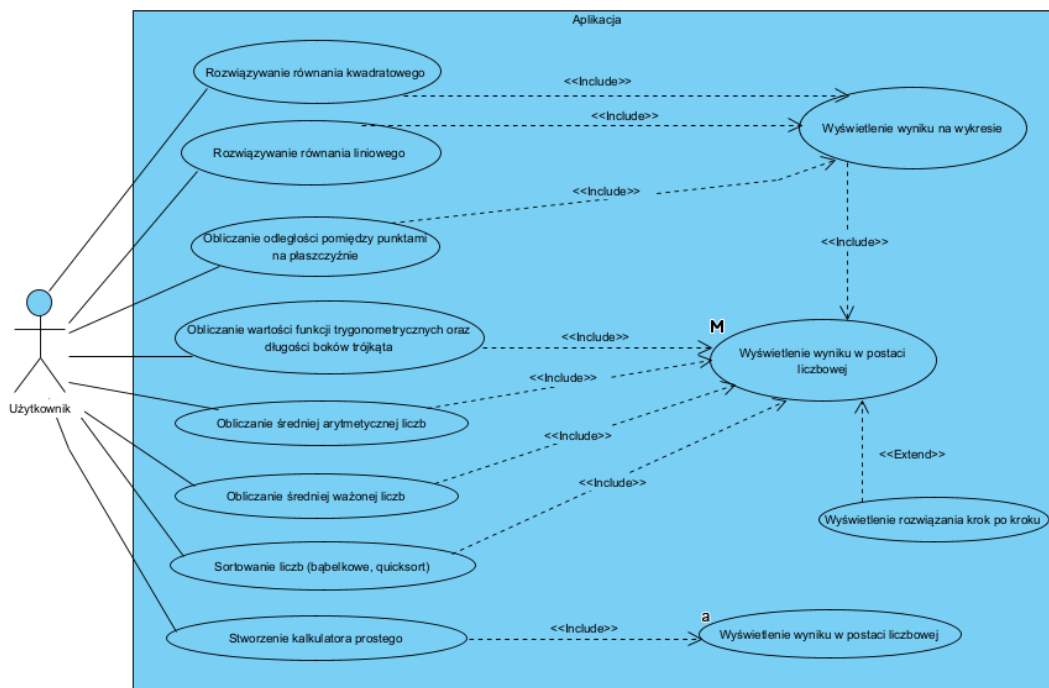
2.4.2. Wymagania niefunkcjonalne

- a) Stworzona aplikacja powinna być aplikacją webową.
- b) Aplikacja powinna być tak zaimplementowana, aby możliwe było korzystanie z niej przy pomocy najpopularniejszych przeglądarek, takich jak: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera.
- c) Aplikacja powinna być stworzona przy pomocy następujących narzędzi:
 - Python,
 - Django,
 - JavaScript,
 - TypeScript,
 - HTML5,
 - CSS3.
- d) Aplikacja powinna być wdrożona w serwisie chmurowym Microsoft Azure przy użyciu usługi AppService.

2.5. Modelowanie UML

2.5.1. Diagram przypadków użycia

Na rysunku 2.4 umieszczono diagram przypadków użycia projektowanej aplikacji. Do jego wykonania zostało wykorzystane narzędzie Visual Paradigm [19].



Rys. 2.4: Diagram przypadków użycia aplikacji

2.5.2. Opisy przypadków użycia

Przypadek użycia: Rozwiązywanie równania liniowego

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę z równaniem liniowym.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać równanie (krok opcjonalny).
2. Użytkownik wpisuje równanie liniowe.
3. Użytkownik klika przycisk „Rozwiąż”.
4. Wynik wyświetlany jest w postaci liczbowej oraz na wykresie.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Rozwiązywanie równania kwadratowego

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę z równaniem kwadratowym.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać równanie (krok opcjonalny).
2. Użytkownik wpisuje równanie kwadratowe.
3. Użytkownik klika przycisk „Rozwiąż”.
4. Wynik wyświetlany jest w postaci liczbowej oraz na wykresie.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Stworzenie kalkulatora prostego

Aktor: Użytkownik

Opis: Kalkulator prosty, w którym działanie można wpisać przy pomocy przycisków, dostępne operatory to mnożenie, dzielenie, dodawanie, odejmowanie, procent.

Warunki wstępne: Użytkownik wszedł na podstronę z kalkulatorem.

Przebieg:

1. Użytkownik wpisuje działanie, używając przycisków.
2. Użytkownik klika przycisk „=”.
3. Wyświetla się wynik.

Przypadek użycia: Obliczanie średniej arytmetycznej liczb

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę ze średnią arytmetyczną.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać liczby (krok opcjonalny).
2. Użytkownik dodaje liczby, rozdzielając je średnikiem.
3. Użytkownik klika przycisk „Oblicz”.
4. Wynik wyświetlany jest w postaci liczbowej.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Obliczanie średniej ważonej liczb

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę ze średnią ważoną.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać liczby oraz wagi liczb (krok opcjonalny).

2. Użytkownik dodaje liczby z wagami, rozdzielając je średnikiem lub poprzez osobne formularze wpisuje liczbę oraz wagę liczby.
3. Użytkownik klika przycisk „Oblicz”.
4. Wynik wyświetlany jest w postaci liczbowej.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Sortowanie liczb (bąbelkowe, quicksort)

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę z sortowaniem bąbelkowym/szybkim.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać liczby (krok opcjonalny).
2. Użytkownik dodaje liczby, rozdzielając je średnikiem.
3. Użytkownik klika przycisk „Oblicz”.
4. Wynik wyświetlany jest w postaci liczbowej.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Obliczanie odległości pomiędzy punktami na płaszczyźnie

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę z obliczaniem odległości pomiędzy punktami.

Przebieg:

1. Użytkownik wpisuje wartości współrzędnych x oraz y punktów p1 oraz p2.
2. Użytkownik klika przycisk „Rozwiąż”.
3. Wynik wyświetlany jest w postaci liczbowej oraz na wykresie.
4. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Przypadek użycia: Obliczanie wartości funkcji trygonometrycznych oraz długości boków trójkąta

Aktor: Użytkownik

Warunki wstępne: Użytkownik wszedł na podstronę z obliczaniem właściwości trójkąta.

Przebieg:

1. Użytkownik wyświetla informację, jak prawidłowo wpisać długości boków trójkąta (krok opcjonalny).
2. Użytkownik wpisuje długości minimum dwóch boków.
3. Użytkownik klika przycisk „Rozwiąż”.
4. Wynik wyświetlany jest w postaci liczbowej.
5. Użytkownik wyświetla rozwiązanie krok po kroku poprzez kliknięcie przycisku (krok opcjonalny).

Rozdział 3

Implementacja

3.1. Frontend

Część frontendowa aplikacji została wykonana przy pomocy języków JavaScript [5] oraz TypeScript [18], z wykorzystaniem biblioteki React [15]. Zdecydowano się na te technologie ze względu na ich popularność, prostotę oraz szerokie zastosowanie. JavaScript jest językiem, który potrafią interpretować wszystkie najpopularniejsze przeglądarki. TypeScript jest językiem rozszerzającym język JavaScript, statycznie typowanym w przeciwieństwie do dynamicznie typowanego języka JavaScript, co pozwala na łatwiejsze wykrywanie błędów [24]. Biblioteka React jest obecnie najpopularniejszą biblioteką używaną do tworzenia frontendu aplikacji webowych. Jej zalety to modularność, szybkość działania, duże wsparcie społeczności oraz prostota [23]. W podrozdziale znajdują się fragmenty kodu części frontendowej aplikacji oraz zrzuty ekranu, na których zaprezentowano wygląd stron powstałej aplikacji.

3.1.1. Główne fragmenty kodu

Na listingu 3.1 przedstawiono główny komponent aplikacji. Znajdują się w nim osobne komponenty dla każdej ze stron; przejście pomiędzy stronami zrealizowano przy pomocy biblioteki React Router [16]. Aplikacja została zabezpieczona przed błędem braku zasobu. W przypadku wejścia użytkownika na podstronę, która nie istnieje, otwiera się strona z napisem „Nie znaleziono zasobu! Przejdź do strony głównej.”.

Listing 3.1: Główny komponent aplikacji frontendowej

```
<React.StrictMode>
  <BrowserRouter>
    <ThemeProvider theme={theme}>
      <Grid
        container
        width={a.width > 1200 ? a.width : 1200}
        justifyContent="center"
        alignItems="center"
      >
        <Grid
          item
          height="100%"
          width={a.width > 1200 ? a.width : 1200}
          justifyContent="center"
          alignItems="center"
        >
          <AppBar />
        <Routes>
```

```

<Route path="/" element={<MainPage />} />
<Route path="/square-equation" element={<SquareEquation />}
  ↪ />
<Route path="/linear-equation" element={<LinearEquation />}
  ↪ />
<Route
  path="/arithmetic-average"
  element={<ArithmeticAverage />}
/>
<Route path="/weighted-average" element={<WeightedAverage
  ↪ />} />
<Route path="/points-distance" element={<PointsDistance />}
  ↪ />
<Route path="/bubble-sort" element={<BubbleSort />} />
<Route path="/quick-sort" element={<QuickSort />} />
<Route path="/calculator" element={<Calc />} />
<Route
  path="/triangle-calculations"
  element={<TriangleCalculations />}
/>
<Route path="/*" element={<NotFoundPage />} />
</Routes>
</Grid>
<Footer width={a.width > 1200 ? a.width : 1200} />
</Grid>
</ThemeProvider>
</BrowserRouter>
</React.StrictMode>

```

Na listingu 3.2 zaprezentowano sposób zapamiętywania stanu aplikacji. Zostały do tego wykorzystane hook'i `useState` z biblioteki `React` [15]. Użyto je między innymi do zapamiętania aktualnie wpisanych danych do formularzy oraz wyników obliczeń.

Listing 3.2: Przykładowe hook'i

```

export interface Response {
  error?: string;
  firstSide?: string;
  secondSide?: string;
  hypotenuse?: string;
  sinL?: string;
  cosL?: string;
  tgL?: string;
  ctgL?: string;
  hint?: string;
}

const TriangleCalculations = () => {
  const [res, setRes] = useState<Response>({
    error: "",
    firstSide: "",
    secondSide: "",
    hypotenuse: "",
    sinL: "",
    cosL: "",
    tgL: "",
    ctgL: "",
    hint: "",
  });

  const [sides, setSides] = useState<TriangleSides>({

```

```

    firstSide: "",
    secondSide: "",
    hypotenuse: "",
  });

  const [errors, setErrors] = useState<TriangleSides>({
    firstSide: "",
    secondSide: "",
    hypotenuse: "",
  });

```

Na listingu 3.3 pokazano sposób wysyłania zapytań od klienta do serwera. Wykorzystano do tego celu bibliotekę Axios [1]. Zapytanie jest wysyłane przy użyciu metody `get`. Wszystkie przesyłane dane są umieszczane w parametrach zapytania.

Listing 3.3: Wysłanie zapytania przy pomocy biblioteki Axios

```

import axios from "axios";
import { address } from "../address";

export interface TriangleSides {
  firstSide: string;
  secondSide: string;
  hypotenuse: string;
}

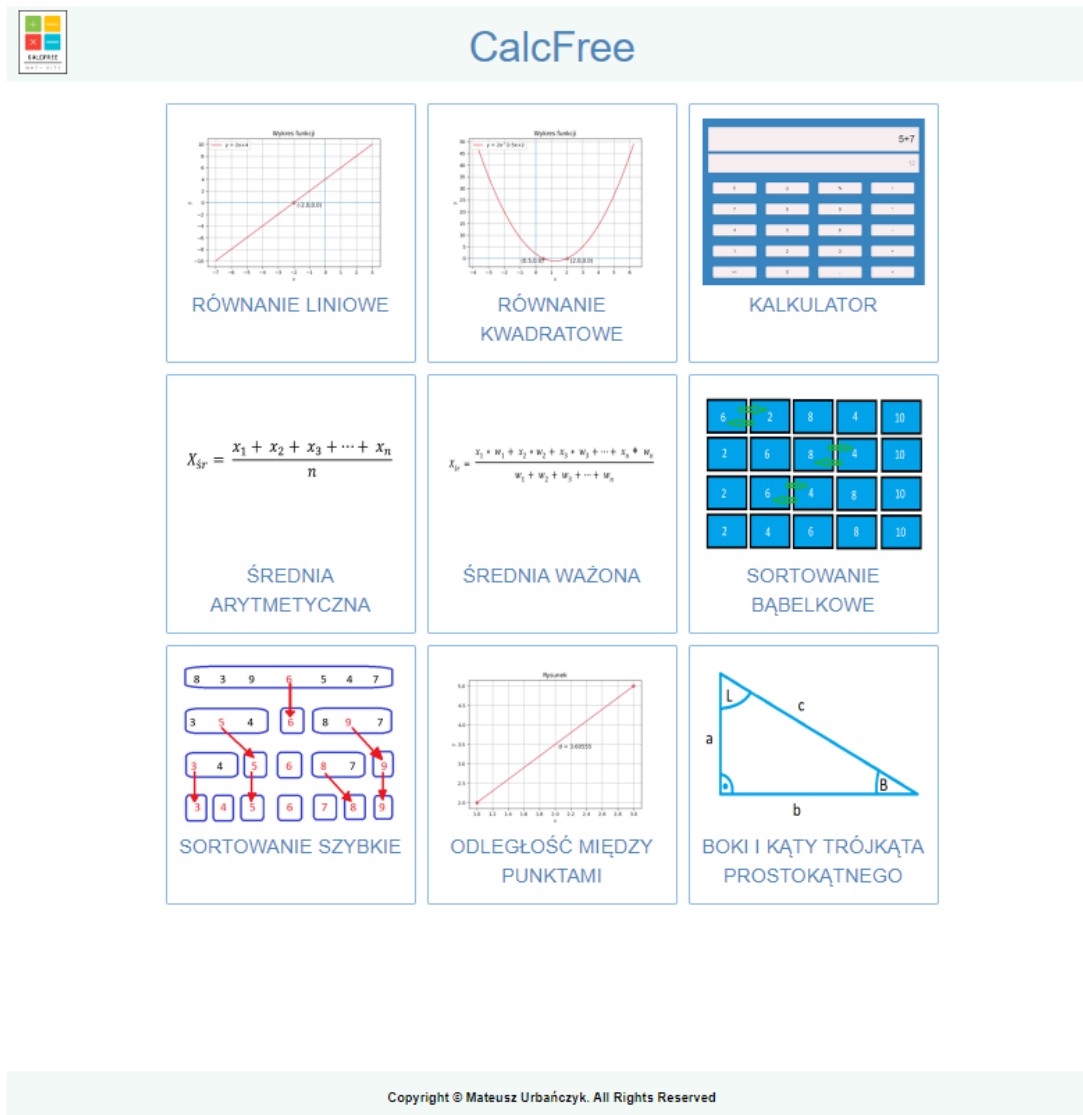
class TriangleService {
  solveTriangle(triangleSides: TriangleSides) {
    return axios.get(`${address}/triangle/`, {
      params: {
        ...(triangleSides.firstSide.length > 0
          ? { firstSide: triangleSides.firstSide }
          : {}),
        ...(triangleSides.secondSide.length > 0
          ? { secondSide: triangleSides.secondSide }
          : {}),
        ...(triangleSides.hypotenuse.length > 0
          ? { hypotenuse: triangleSides.hypotenuse }
          : {}),
      },
    });
  }
}

export default new TriangleService();

```

3.1.2. Wygląd stron

Na rysunkach 3.1 - 3.10 przedstawiono wygląd wszystkich stron powstałej aplikacji. Strona nie jest w pełni responsywna, ale korzystanie z niej na urządzeniach mobilnych jest możliwe. Na rysunku 3.1 zaprezentowano wygląd strony głównej powstałej aplikacji. Znajduje się na niej dziewięć przycisków, które przenoszą na podstronę z wybranym typem obliczeń.



Rys. 3.1: Strona główna

Na rysunku 3.2 umieszczono wygląd podstrony z równaniem liniowym. Znajduje się na niej pole, w którym użytkownik może wpisać swoje równanie. Istnieje możliwość sprawdzenia, jak poprawnie je wprowadzić. Po kliknięciu przycisku „info” obok pola z równaniem wyświetla się informacja, jak to poprawnie zrobić. Wynik wyświetlany jest w postaci liczbowej oraz na wykresie. Po kliknięciu przycisku „POKAŻ WSKAZÓWKĘ” pojawia się także wskazówka.

Równanie liniowe

 $3x+9$
 $= 0$ ⓘ

ROZWIĄŻ

Wynik

 $x = -3,0$

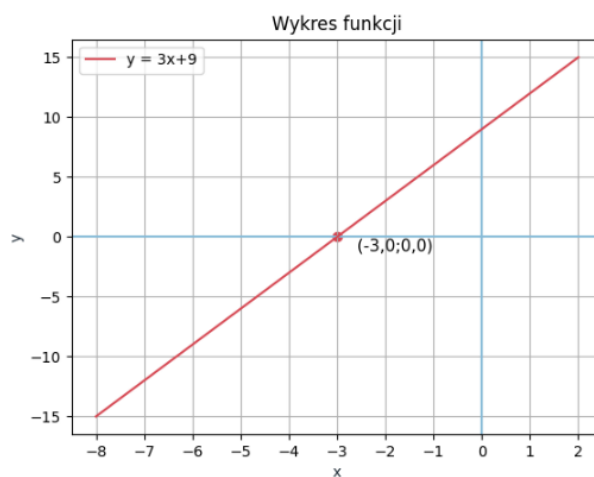
POKAŻ WSKAZÓWKĘ

Wskażówka

Wynik równania $ax + b = 0$ obliczamy, przekształcając równanie.

Wzór: $x = -b / a$

Wynik: $x = -9,0 / 3,0 = -3,0$



Rys. 3.2: Strona z równaniem liniowym

Na rysunku 3.3 znajduje się wygląd podstrony z równaniem kwadratowym. Jest on identyczny, jak dla równania liniowego.

Równanie kwadratowe

$$6x^2 + 15x + 3$$

= 0 ⓘ

ROZWIĄŻ

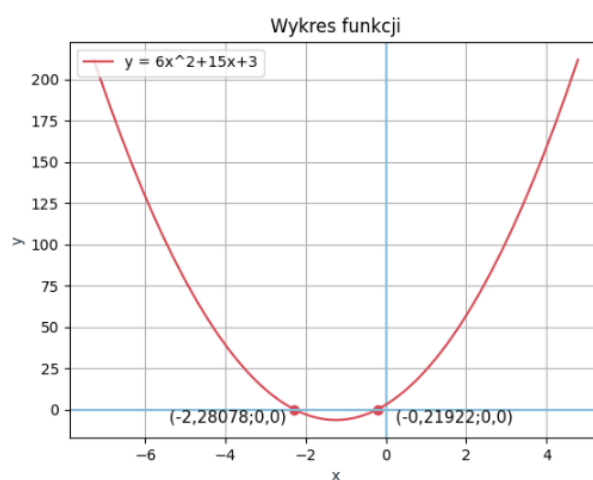
Wynik

x1 = -2,280776406404415
x2 = -0,21922359359558494

POKAŻ WSKAZÓWKĘ

Wskazówka

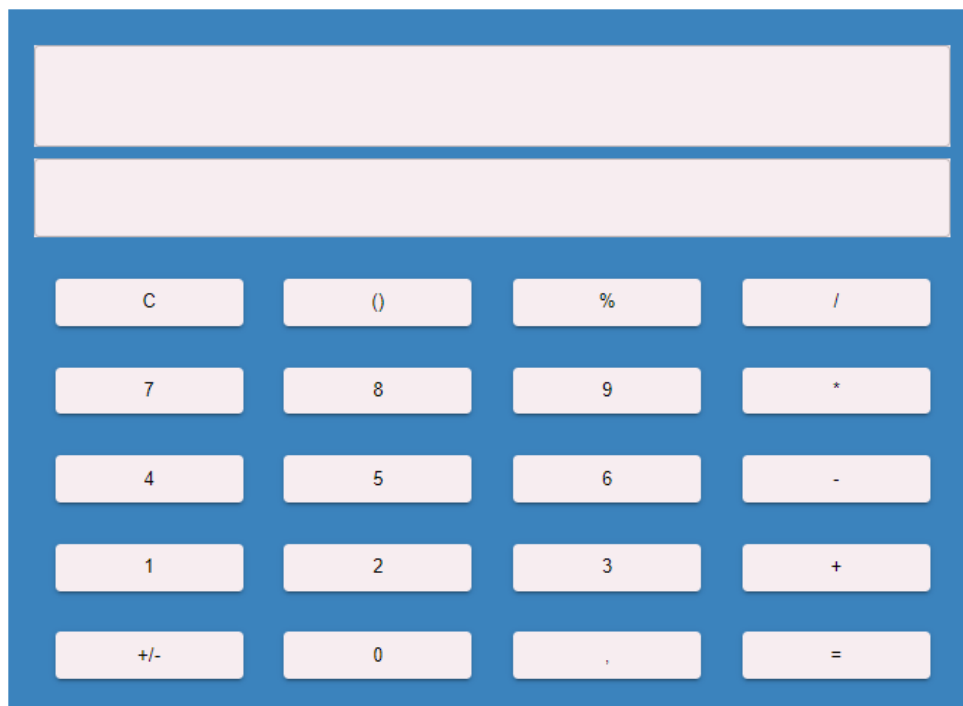
Obliczamy deltę ze wzoru $\Delta = b^2 - 4 \cdot a \cdot c$.
 $\Delta = 15,0^2 - 4 \cdot 6,0 \cdot 3,0 = 153,0$
 $\Delta > 0$, zatem istnieją dwa rozwiązania
 Wzory:
 $x_1 = \frac{-b - \Delta^{(1/2)}}{2 \cdot a}$
 $x_2 = \frac{-b + \Delta^{(1/2)}}{2 \cdot a}$
 Wynik:
 $x_1 = \frac{(-15,0 - 153,0^{(1/2)})}{(2 \cdot 6,0)} = -2,280776406404415$
 $x_2 = \frac{(-15,0 + 153,0^{(1/2)})}{(2 \cdot 6,0)} = -0,21922359359558494$



Rys. 3.3: Strona z równaniem kwadratowym

Na rysunku 3.4 przedstawiono wygląd podstrony z kalkulatorem. Wpisywanie działania możliwe jest przy pomocy przycisków. Kalkulator obsługuje dodawanie, odejmowanie, dzielenie, mnożenie oraz przeliczanie procentów na liczby.

Kalkulator



Rys. 3.4: Strona z kalkulatorem

Na rysunku 3.5 zaprezentowano wygląd podstrony ze średnią arytmetyczną. Liczby, których użytkownik chce obliczyć średnią, należy wpisać, rozdzielając je średnikiem. Po dodaniu liczb i kliknięciu „Oblicz” wynik pojawi się w postaci liczbowej. Istnieje możliwość także wyświetlenia wskazówki.

Średnia arytmetyczna

Twoje liczby

Wynik

Wskazówka

Średnia arytmetyczna to suma liczb podzielona przez ich liczbę.
Wzór: $(x_1 + x_2 + x_3 + \dots + x_n) / n$
Wynik: $(4,0 + 5,0 + 6,0 + 1,0 + 3,0 + 8,0 + 9,0 + 76,0 + 54,0 + 34,0 + 57,0 + 31,0 + 12,0) / 13 = 23,076923076923077$

Rys. 3.5: Strona ze średnią arytmetyczną

Na rysunku 3.6 ukazano wygląd podstrony ze średnią ważoną. Liczbę oraz wagę liczby można dodać w osobnych polach lub rozdzielając je średnikiem w polu „Twoje liczby”.

Średnia ważona

Twoje liczby

Wynik

Wskazówka

Średnia ważona to suma liczb pomnożonych przez wagi tych liczb, podzielona przez sumę wag.
Wzór: $(x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots + x_n * w_n) / (w_1 + w_2 + w_3 + \dots + w_n)$
Wynik: $(2,0 * 3,0 + 1,0 * 3,0 + 4,0 * 3,0 + 3,0 * 4,0 + 5,0 * 6,0 + 3,0 * 5,0 + 1,0 * 4,0 + 5,0 * 5,0) / (3,0 + 3,0 + 3,0 + 4,0 + 6,0 + 5,0 + 4,0 + 5,0) = 3,242424242424242$

Rys. 3.6: Strona ze średnią ważoną

Na rysunkach 3.7 oraz 3.8 znajduje się wygląd podstron dla sortowania bąbelkowego oraz szybkiego. Liczby do posortowania należy wpisać, rozdzielając je średnikami. Istnieje możliwość wybrania, czy sortowanie ma być niemalejąco, czy nierosnąco.

Sortowanie bąbelkowe

DODAJ LICZBY i

Twoje liczby
 WYCZYŚĆ

☒ Niemalejąco

SORTUJ

Wynik

POKAŻ WSKAZÓWKĘ

Wskazówka

Sortowanie bąbelkowe polega na cyklicznej zamianie dwóch sąsiadujących elementów, jeżeli nie są ułożone w kolejności, w której sortujemy. Przebieg sortowania:

Zamiana elementów (nr iteracji pętli zewnętrznej: 1, nr iteracji pętli wewnętrznej: 2)
Stan po zamianie: [3,5; 4,0; 5,0; 5,0; 3,5]

Zamiana elementów (nr iteracji pętli zewnętrznej: 1, nr iteracji pętli wewnętrznej: 4)
Stan po zamianie: [3,5; 4,0; 5,0; 3,5; 5,0]

Zamiana elementów (nr iteracji pętli zewnętrznej: 2, nr iteracji pętli wewnętrznej: 3)
Stan po zamianie: [3,5; 4,0; 3,5; 5,0; 5,0]

Zamiana elementów (nr iteracji pętli zewnętrznej: 3, nr iteracji pętli wewnętrznej: 2)
Stan po zamianie: [3,5; 3,5; 4,0; 5,0; 5,0]

Liczba wykonanych zamian: 4.

Rys. 3.7: Strona z sortowaniem bąbelkowym

Sortowanie szybkie (quicksort)

DODAJ LICZBY i

Twoje liczby

3;1;2

WYCZYŚĆ

☒ Niemalejąco

SORTUJ

Wynik

1,0;2,0;3,0

POKAŻ WSKAZÓWKĘ

Wskazówka

Sortowanie szybkie (quicksort) polega na wybraniu pewnej liczby z sortowanych liczb oraz podziale elementów na dwie grupy. W jednej umieszczamy liczby mniejsze od wybranej wartości, a w drugiej większe lub równe. Procedurę należy powtórzyć rekurencyjnie dla obu grup.

=====

Oś: 2,0

Stan początkowy:

[3,0; 1,0; 2,0]

Zamiany:

[1,0; 3,0; 2,0]

[1,0; 2,0; 3,0]

Stan końcowy:

[1,0; 2,0; 3,0]

Liczba wykonanych zamian: 2.

Rys. 3.8: Strona z sortowaniem szybkim

Na rysunku 3.9 przedstawiono wygląd podstrony z obliczaniem odległości pomiędzy punktami. Na stronie należy wpisać w osobnych polach wartości współrzędnych x oraz y dwóch punktów. Wynik wyświetlany jest w postaci liczbowej oraz na wykresie.

Odległość pomiędzy punktami

$p1 =$

 $p2 =$

ROZWIĄŻ

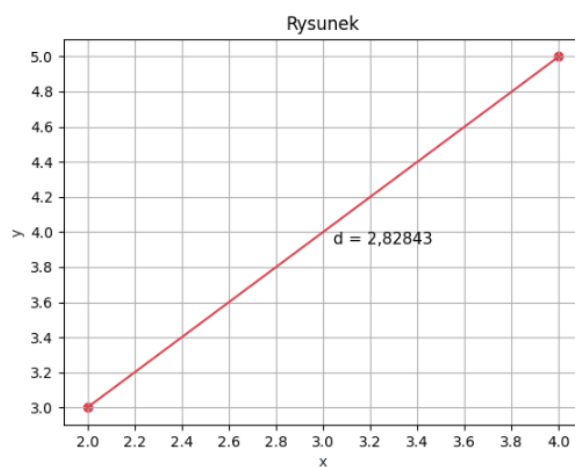
Wynik

$d = 2,8284271247461903$

POKAŻ WSKAZÓWKĘ

Wskazówka

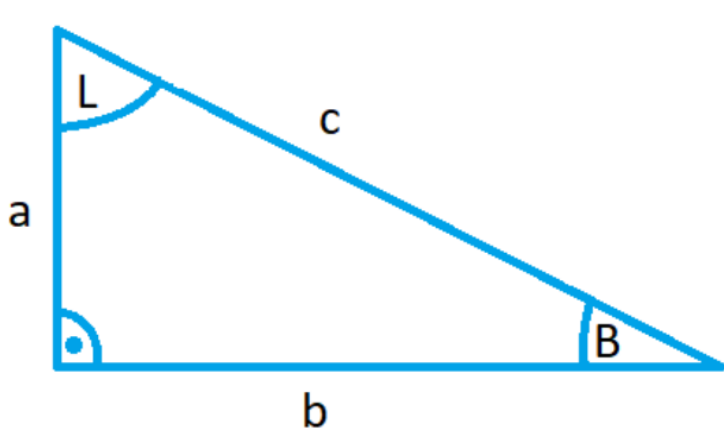
Odległość pomiędzy punktami można wyliczyć z twierdzenia Pitagorasa. Jest on równa długości przeciwprostokątnej trójkąta, którego przyprostokątne mają długość równą wartości bezwzględnej z różnicy współrzędnych x oraz y punktów.
 Wzór: $d = ((x1 - x2)^2 + (y1 - y2)^2)^{(1 / 2)}$
 Wynik: $d = ((2,0 - 4,0)**2 + (3,0 - 5,0)**2)^{(1 / 2)} = 2,8284271247461903$



Rys. 3.9: Strona z obliczaniem odległości pomiędzy punktami

Rysunek 3.10 zawiera podstronę z obliczaniem długości trzeciego boku trójkąta oraz wartości funkcji trygonometrycznych jego kątów ostrych. Wymagane jest podanie długości minimum dwóch boków trójkąta. W przypadku niewprowadzenia długości trzeciego boku zostanie ona obliczona automatycznie.

Boki i kąty trójkąta prostokątnego



a =

b =

c =

ROZWIĄZ i

sin(L) = <input style="width: 150px;" type="text" value="0,8"/>	sin(B) = <input style="width: 150px;" type="text" value="0,6"/>
cos(L) = <input style="width: 150px;" type="text" value="0,6"/>	cos(B) = <input style="width: 150px;" type="text" value="0,8"/>
tg(L) = <input style="width: 150px;" type="text" value="1,3333333333333333"/>	tg(B) = <input style="width: 150px;" type="text" value="0,75"/>
ctg(L) = <input style="width: 150px;" type="text" value="0,75"/>	ctg(B) = <input style="width: 150px;" type="text" value="1,3333333333333333"/>

POKAŻ WSKAZÓWKĘ

Rys. 3.10: Strona z obliczaniem właściwości trójkąta

3.2. Backend

Część backendowa aplikacji została napisana w języku Python [14]. Autor zdecydował się na wykorzystanie tego języka ze względu na wcześniejsze doświadczenie z nim związane oraz jego prostotę polegającą na tym, że jest językiem wysokiego poziomu pozwalającym na tworzenie aplikacji dużo szybciej niż w wielu innych językach oraz wymaga napisania mniejszej ilości kodu w celu uzyskania tego samego efektu. W celu uproszczenia implementacji zdecydowano się skorzystać z frameworka Django [2]. Jest on odpowiedni do tworzenia wielostronowych aplikacji. Jest najczęściej wykorzystywanym frameworkiem tego języka do tworzenia serwisów webowych. Jego zalety to:

- szybkość tworzenia aplikacji internetowych,
- dostępność gotowych rozwiązań oraz komponentów,
- wszechstronność,
- bezpieczeństwo,
- skalowalność [25].

3.2.1. Główne fragmenty kodu

Listing 3.4 zawiera listę wszystkich endpointów serwera. Dwa *endpointy* służą do zwracania strony głównej aplikacji, pozostałe służą do wykonywania obliczeń.

Listing 3.4: Lista endpointów serwera

```
urlpatterns = [
    path('', views.return_page),
    path('linear-equation/', views.solve_linear_equation),
    path('square-equation/', views.solve_square_equation),
    path('arithmetic-average/', views.arithmetic_average),
    path('weighted-average/', views.weighted_average),
    path('points-distance/', views.distance_between_points),
    path('bubble-sort/', views.bubble_sort),
    path('quick-sort/', views.quick_sort),
    path('triangle/', views.calculate_triangle_properties),
    path('<path:resource>', views.return_page2)
]
```

Listing 3.5 zawiera kod funkcji, w której odbywa się walidacja danych wejściowych zadania obliczenia odległości pomiędzy punktami. Funkcja ta ma ustawiony limit zapytań na 1 000 000/s, aby nie przeciążyć zbyt mocno serwera. Na początku funkcji znajduje się sprawdzenie, czy zapytanie jest typu GET. Jeżeli nie, to zwracany jest błąd 403 - Forbidden. Następnie odbywa się sprawdzenie parametrów zapytania. Jeżeli nie spełniają one wymagań, to zwracany jest błąd 400 - Bad request. Na końcu wywoływana jest funkcja wykonująca obliczenia.

Listing 3.5: Walidacja początkowa argumentów

```
@ratelimit(key='get:q', rate='1000000/s')
def distance_between_points(request):
    if request.method=="GET":
        first = request.GET.get('firstPoint', None)
        second = request.GET.get('secondPoint', None)
        if first is None or second is None:
            return HttpResponse(status=400)
        first = list(first.split(";"))
        second = list(second.split(";"))
        if len(first) != 2 or len(second) != 2:
            return HttpResponse(status=400)
        first[0] = first[0].replace(",", ".")
        first[1] = first[1].replace(",", ".")
        second[0] = second[0].replace(",", ".")
        second[1] = second[1].replace(",", ".")
        try:
            first = [float(x) for x in first]
            second = [float(x) for x in second]
        except ValueError:
            return HttpResponse(status=400)
        response = points_on_surface.find_distance_between_points(first,
            ↪ second, matplotlib_lock)
        if response is None:
            return HttpResponse(status=400)
        return HttpResponse(response, status=200)
    return HttpResponse(status=403)
```

Na listingu 3.6 umieszczono funkcję obliczającą odległość między punktami położonymi na płaszczyźnie. Funkcja oblicza rozwiązanie, generuje wskazówkę oraz wykres przy użyciu biblioteki matplotlib [8].

Listing 3.6: Funkcja obliczająca odległość między punktami

```
def find_distance_between_points(first_point, second_point, matplotlib_lock
    ↪ ):
    response = {}
    result = ((first_point[0] - second_point[0]) ** 2 + (first_point[1] -
    ↪ second_point[1]) ** 2) ** 0.5
    response["solution"] = "d = " + str(result).replace(".", ",")
    hint = "Odległość pomiędzy punktami można wyliczyć z twierdzenia
    ↪ Pitagorasa.\n" + \
    "Jest on równa długości przeciwprostokątnej trójkąta, którego
    ↪ przyprostokątne\n" + \
    "mają długość równą wartości bezwzględnej z różnicy współrzędnych x
    ↪ oraz y punktów.\n" + \
    "Wzór: d = ((x1 - x2)^2 + (y1 - y2)^2)^(1 / 2)\nWynik: d = ((" + \
    str(first_point[0]).replace(".", ",") + " - " + str(second_point
    ↪ [0]).replace(".", ",") + ")**2 + (" + \
    str(first_point[1]).replace(".", ",") + " - " + \
    str(second_point[1]).replace(".", ",") + ")**2)^(1 / 2) = "
    ↪ + str(result).replace(".", ",")
    response["hint"] = hint
    matplotlib_lock.acquire()
    plt.plot([first_point[0], second_point[0]], [first_point[1],
    ↪ second_point[1]], color='#d64550')
    plt.title("Rysunek")
    plt.xlabel('x', color='#1C2833')
    plt.ylabel('y', color='#1C2833')
    plt.locator_params(nbins=12)
    plt.grid(True)
    plt.scatter(first_point[0], first_point[1], color='#d64550')
    plt.scatter(second_point[0], second_point[1], color='#d64550')
    x_half = (first_point[0] + second_point[0])/2.0
    y_half = (first_point[1] + second_point[1])/2.0
    round_result = round(result, 5)
    if first_point[1] >= second_point[1]:
        plt.annotate(" d = " + str(round_result).replace(".", ","),(x_half
        ↪ , y_half), fontsize = 11, verticalalignment='bottom')
    else:
        plt.annotate(" d = " + str(round_result).replace(".", ","),(x_half
        ↪ , y_half), fontsize = 11, verticalalignment='top')
    fig = plt.gcf()
    img_data = io.BytesIO()
    fig.savefig(img_data, format = 'png')
    plt.clf()
    matplotlib_lock.release()
    img_data.seek(0)
    image_bytes = base64.b64encode(img_data.read())
    response["graph"] = image_bytes.decode("raw-unicode-escape")
    return json.dumps(response)
```

3.3. Opis pozostałych użytych technologii

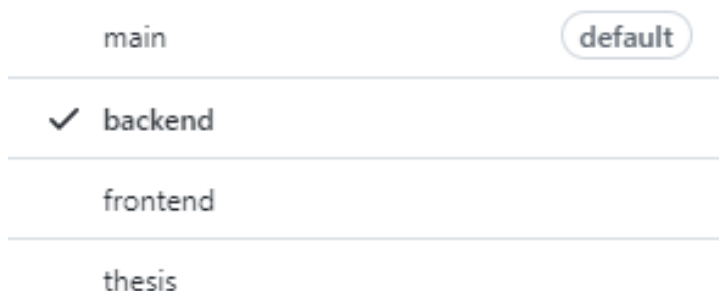
3.3.1. Visual Studio Code

Do napisania kodu źródłowego, zarówno części frontendowej, jak i backendowej zostało wykorzystane narzędzie Visual Studio Code [20]. Narzędzie to jest najczęściej używanym przez programistów środowiskiem programistycznym. Obsługuje setki języków. Jest dostępne na naj-

popularniejszych systemach operacyjnych. Narzędzie to jest proste w obsłudze, intuicyjne oraz przyjazne dla użytkownika.

3.3.2. GitHub

Stosowanie systemów kontroli wersji ułatwia pracę przy pisaniu kodu. Pozwala na wprowadzanie zmian w kodzie oraz śledzenie wszystkich zmian, także wprowadzanych przez innych użytkowników. Dzięki temu każdy programista uczestniczący w projekcie jest w stanie przywrócić dowolną, wcześniejszą wersję kodu [22]. Do wersjonowania kodu źródłowego zostało wykorzystane narzędzie GitHub [4]. Na rysunku 3.11 przedstawiono stworzone gałęzie, które były wykorzystywane przy realizacji projektu.



Rys. 3.11: Stworzone gałęzie na GitHubie

Gałąź **main** posłużyła do umieszczenia finalnej wersji całego projektu, na gałęzi **frontend** był przechowywany kod części frontendowej aplikacji, na gałęzi **backend** kod części backendowej aplikacji, a na gałęzi **thesis** dokument pracy dyplomowej. Na rysunku 3.12 pokazano przykładowe commity do zdalnego repozytorium.



Rys. 3.12: Przykładowe commity na GitHubie

Rozdział 4

Wdrożenie

4.1. Wdrażanie aplikacji w chmurze

Aplikację wdrożono w chmurze Microsoft Azure [9]. Autor wybrał to rozwiązanie ze względu na własne doświadczenie z wdrażaniem innej aplikacji w tej usłudze oraz brak kosztów związanych z usługą ze względu na posiadanie licencji studenckiej. Przy wdrażaniu zostały wykorzystane usługi ContainerRegistry oraz AppService. Usługa rejestru kontenerów - ContainerRegistry była konieczna do wdrożenia ze względu na decyzję o wdrażaniu przy pomocy kontenera Docker. Na rysunkach 4.1 oraz 4.2 przedstawiono dane włączonych usług.

Serwer logowania
engineercontainer.azurecr.io

Data utworzenia
6.12.2022, 19:01 CET

Jednostka SKU
Standardowa

Stan aprowizacji
Powodzenie

Usuwanie nietrwałe (wersja zapoznawcza)
Wyłączone

Rys. 4.1: Widok działającej usługi ContainerRegistry

Adres URL : <https://calcfree.azurewebsites.net>
 Plan usługi App Service : [ASP-EngineeringThesis-9d52 \(B1: 1\)](#)
 System operacyjny : Linux
 Sprawdzanie kondycji : [Nieskonfigurowane](#)

Rys. 4.2: Widok działającej usługi AppService

Wdrożona aplikacja znajduje się pod adresem <https://calcfree.azurewebsites.net/>.

4.2. Docker

Do wdrożenia została wykorzystana konteneryzacja przy pomocy narzędzia Docker [3]. Narzędzie to pozwala na bardzo proste wdrożenie aplikacji na Microsoft Azure oraz umożliwia przetestowanie aplikacji przed wdrożeniem poprzez uruchomienie w kontenerze. Do wdrożenia wymagane jest stworzenie pliku Dockerfile, zalogowanie się na konto Microsoft Azure oraz wykonanie kilku komend. Listing 4.1 zawiera zawartość pliku konfiguracyjnego Dockerfile, który służy do konfiguracji narzędzia. W pliku tym znajduje się informacja o wersji języka Python, ustawiony jest folder, w którym znajdują się pliki, przypisane są zmienne środowiskowe oraz umieszczone są komendy służące do zainstalowania wszystkich potrzebnych zależności i uruchomienia serwera na porcie nr 8000.

Listing 4.1: Dockerfile

```
FROM python:3.10

# set work directory
WORKDIR /usr/src/app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN pip install --upgrade pip
COPY ./requirements.txt /usr/src/app
RUN pip install -r requirements.txt

# copy project
COPY . /usr/src/app

EXPOSE 8000

CMD ["cd", "calc_free_backend"]
CMD python calc_free_backend/manage.py runserver 0.0.0.0:8000
```

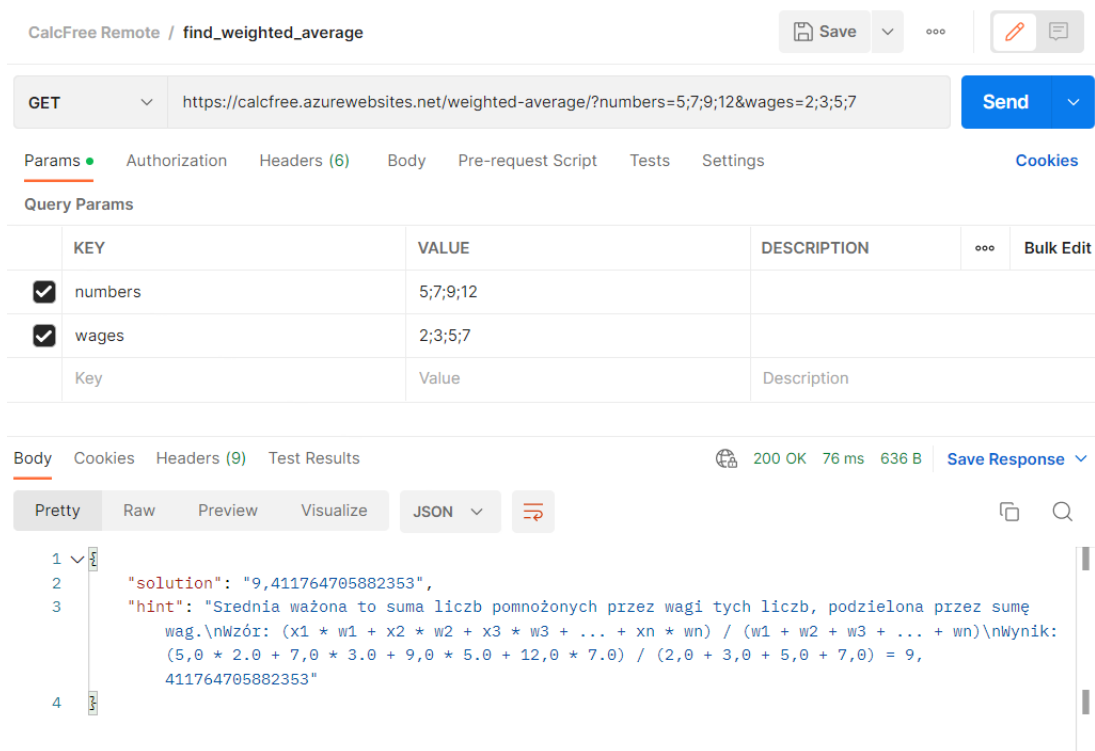
Pełna instrukcja wdrożeniowa znajduje się w dodatku A.

Rozdział 5

Testowanie

5.1. Sposób testowania aplikacji

Ze względu na dosyć dużą złożoność aplikacji zdecydowano się na wykonanie wraz z dodaniem każdej nowej funkcji zarówno testów manualnych, jak i automatycznych. Testy manualne opierały się na debuggowaniu aplikacji, ręcznym sprawdzaniu możliwości przechodzenia pomiędzy stronami, działania przycisków i formularzy na stronie, reakcji aplikacji na wprowadzone dane oraz ręcznym wysyłaniu zapytań do serwera. Do ostatniego wymienionego sposobu testowania wykorzystano narzędzie Postman [12]. Posiada ono bardzo przyjazny interfejs i ułatwia wysyłanie zapytań. Na rysunku 5.1 pokazano zrzut ekranu z aplikacji desktopowej. Można zobaczyć na nim, na jaki adres wysyłane jest zapytanie, jakie parametry są dodane do zapytania oraz jaka jest odpowiedź serwera (kod odpowiedzi i jej zawartość). Rysunek zawiera zapytanie służące do sprawdzenia, czy serwer poprawnie oblicza średnią ważoną liczb.



Rys. 5.1: Zrzut ekranu z aplikacji Postman

Testy automatyczne były tworzone w czasie powstawania aplikacji. Po dodaniu każdej funkcji był pisany do niej test. W przypadku wykrytych przez niego błędów, błędy te były poprawiane. Po zaimplementowaniu kolejnej funkcji były uruchamiane testy także dla wcześniej stworzonych komponentów aplikacji. Zdecydowano się na wykonanie trzech rodzajów testów automatycznych - jednostkowych, integracyjnych oraz funkcjonalnych.

5.2. Testy jednostkowe

Test jednostkowy (ang. *unit test*) – metoda testowania tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania pojedynczych elementów (jednostek) programu, np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Testowany fragment programu poddawany jest testowi, który wykonuje go i porównuje wynik (np. zwrócone wartości, stan obiektu, zgłoszone wyjątki) z oczekiwanymi wynikami – tak pozytywnymi, jak i negatywnymi (niepowodzenie działania kodu w określonych sytuacjach również może podlegać testowaniu). Zaletą testów jednostkowych jest możliwość wykonywania na bieżąco w pełni zautomatyzowanych testów na modyfikowanych elementach programu, co umożliwia często wychwycenie błędu natychmiast po jego pojawieniu się i szybką jego lokalizację, zanim dojdzie do wprowadzenia błędnego fragmentu do programu [28].

W celu sprawdzenia poprawności działania oprogramowania serwera napisano 29 testów jednostkowych funkcji odpowiedzialnych za wykonywanie obliczeń. Przetestowano wyniki tych funkcji dla typowych danych wejściowych oraz dla danych, które mogłyby prowadzić do błędów, takich jak dzielenie przez 0.

Na listingu 5.1 zamieszczono kod testu sprawdzającego poprawność funkcji służącej do obliczania odległości pomiędzy punktami. Test ma za zadanie potwierdzić, czy funkcja zwraca prawidłowy wynik obliczeń oraz wskazówkę dla różnych danych wejściowych, wygenerowanych pseudolosowo.

Listing 5.1: Test jednostkowy funkcji obliczającej odległość między punktami

```
def test_correct_result(self):
    """Test should check if functions return correct distance between
    points"""
    for i in range(200):
        first_point = []
        second_point = []
        first_point.append(random.uniform(-1000.0, 1000.0))
        first_point.append(random.uniform(-1000.0, 1000.0))
        second_point.append(random.uniform(-1000.0, 1000.0))
        second_point.append(random.uniform(-1000.0, 1000.0))
        response = points_on_surface.find_distance_between_points(
            ↪ first_point, second_point, self.lck)
        self.assertTrue(type(response) == str, msg="Function should
            ↪ return string")
        temp = json.loads(response)
        self.assertTrue(type(temp) == dict, msg="Json should be valid")
        self.assertTrue("solution" in temp.keys(), "Solution should be
            ↪ in json")
        sol = ((first_point[0] - second_point[0])**2 + (first_point[1]
            ↪ - second_point[1])**2)**(1/2.0)
        self.assertTrue(abs(sol - float(temp["solution"])[4:].replace
            ↪ (" ", ".", ""))) < 0.0000001,
            msg="Distance should be calculated correctly")
        self.assertTrue("hint" in temp.keys(), msg="Hint should be in
            ↪ json")
```



```

self.assertTrue(len(temp["hint"]) > 0, msg = "Hint should be
    ↪ valid(length greater than 0)")
self.assertTrue("Wzór" in temp["hint"], msg = "Hint should
    ↪ contain formula")
self.assertTrue("Wynik" in temp["hint"], msg = "Hint should
    ↪ contain result")

```

Na rysunku 5.2 umieszczono wynik uruchomienia testów. Wszystkie testy potwierdziły poprawność implementacji testowanych funkcji.

```

Found 29 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 29 tests in 21.537s

OK

```

Rys. 5.2: Wynik testów jednostkowych

5.3. Testy integracyjne

Testy integracyjne (ang. *integration tests*) – jeden z etapów testowania oprogramowania, wykonywany w celu wykrycia defektów w interfejsach i interakcjach pomiędzy modułami lub systemami. Testy integracyjne są przeprowadzane w celu oceny zgodności systemu lub komponentu z określonymi wymaganiami funkcjonalnymi. Występują po testach jednostkowych, jednakże przed testami walidacyjnymi. Współczesne aplikacje składają się przeważnie z wielu współpracujących systemów, należy więc sprawdzić, czy komunikacja pomiędzy nimi nie jest zakłócona [29]. W celu sprawdzenia integracji części frontendowej aplikacji z częścią backendową, wykonano 19 testów integracyjnych. Wszystkie testy opierały się na wygenerowaniu pewnych danych, wysłaniu żądania do serwera, umieszczając w nim te dane oraz zbadaniu poprawności odpowiedzi otrzymanej z serwera. Na listingu 5.2 umieszczono test integracyjny, sprawdzający odpowiedź serwera na żądanie wysłane na endpoint służący do obliczania odległości pomiędzy dwoma punktami. W teście tym generowane są pseudolosowo współrzędne punktów oraz wysyłane jest żądanie. Po uzyskaniu odpowiedzi następuje sprawdzenie, czy uzyskany kod odpowiedzi to 200 (OK) oraz czy odpowiedź zawiera rozwiązanie, wskazówkę i wykres.

Listing 5.2: Test integracyjny obliczania odległości między punktami poprzez wysłanie żądania do serwera

```

def test_correct_numbers(self):
    """Checking server answer, when params are correct"""
    for i in range(3):
        x1 = random.uniform(-1000.0, 1000.0)
        y1 = random.uniform(-1000.0, 1000.0)
        x2 = random.uniform(-1000.0, 1000.0)
        y2 = random.uniform(-1000.0, 1000.0)
        params = {"firstPoint": str(x1).replace(".", ",") + ";" + str(
            ↪ y1).replace(".", ","),
            "secondPoint": str(x2).replace(".", ",") + ";" + str(y2).
            ↪ replace(".", ",")}
        try:
            server_answer = requests.get(self.address + "points-
            ↪ distance/", params = params)

```

```

except ConnectionError:
    self.fail("No connection with server")
self.assertTrue(server_answer.status_code == 200, msg = "Server
    ↪ should return status code 200")
answer = json.loads(server_answer.text)
self.assertTrue(type(answer) == dict, "Answer should be valid
    ↪ json")
self.assertTrue("solution" in answer.keys(), "Solution should
    ↪ be in json")
self.assertTrue("hint" in answer.keys(), "Hint should be in
    ↪ json")
self.assertTrue("graph" in answer.keys(), "Graph should be in
    ↪ json")

```

Na rysunku 5.3 umieszczono wynik testów integracyjnych. Wszystkie testy wykazały, że część frontendowa aplikacji z częścią backendową są poprawnie zintegrowane.

```

Found 19 test(s).
System check identified no issues (0 silenced).
.....
-----
Ran 19 tests in 28.507s

OK

```

Rys. 5.3: Wynik testów integracyjnych

5.4. Testy funkcjonalne

Testy funkcjonalne (ang. *functional tests*) - rodzaj testów, które umożliwiają dokonanie oceny funkcji, jakie system powinien realizować. Podstawą tego typu testów zwykle bywa dokumentacja pod postacią specyfikacji funkcjonalnej, w której można znaleźć informacje dotyczące funkcji, jakie powinny być dostępne w systemie oraz wymagania im stawiane. Wymagania funkcjonalne można znaleźć także w takich dokumentach projektowych, jak: specyfikacja wymagań biznesowych, scenariusze testowe, historyjki użytkownika czy przypadki użycia [27]. Do wykonania testów funkcjonalnych została wykorzystana biblioteka Selenium [17]. W celu sprawdzenia, czy aplikacja jest zgodna z wymaganiami funkcjonalnymi oraz z przypadkami użycia, wykonano 10 testów. Na listingu 5.3 przedstawiono test funkcjonalny podstrony służącej do obliczania odległości pomiędzy punktami. Test służy do sprawdzenia, czy da się wejść na podstronę, czy da się wpisać współrzędne punktów, czy przycisk „Rozwiąż” działa oraz poprawności wyświetlonego wyniku i wskazówki.

Listing 5.3: Test funkcjonalny przy użyciu narzędzia Selenium podstrony służącej do obliczania odległości pomiędzy punktami

```

def test_page(self):
    "Test checks if all of page functions works"
    page = self.driver.find_element(By.ID, "pointsDistance")
    page.click()
    time.sleep(self.time_sleep)
    x1 = self.driver.find_element(By.ID, "x1")
    y1 = self.driver.find_element(By.ID, "y1")
    x2 = self.driver.find_element(By.ID, "x2")
    y2 = self.driver.find_element(By.ID, "y2")
    x1_value = x1.get_attribute("value")

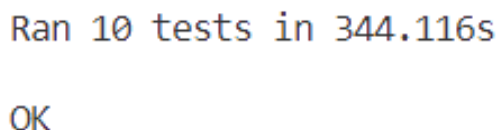
```

```

y1_value = y1.get_attribute("value")
x2_value = x2.get_attribute("value")
y2_value = y2.get_attribute("value")
self.assertEqual(len(x1_value), 0, "X1 should be empty on start")
self.assertEqual(len(y1_value), 0, "Y1 should be empty on start")
self.assertEqual(len(x2_value), 0, "X2 should be empty on start")
self.assertEqual(len(y2_value), 0, "Y2 should be empty on start")
x1.send_keys("5,2")
time.sleep(self.time_sleep)
y1.send_keys("3,1")
time.sleep(self.time_sleep)
x2.send_keys("8,2")
time.sleep(self.time_sleep)
y2.send_keys("7,0")
time.sleep(self.time_sleep)
result = self.driver.find_element(By.ID, "result")
result_text = result.text
self.assertEqual(len(result_text), 0, "Result should be empty on
    ↪ start")
hint = self.driver.find_elements(By.ID, "hint")
self.assertTrue(len(hint) == 0, msg = "Hint text area should not be
    ↪ on the page")
graph = self.driver.find_element(By.ID, "image")
graph_old = graph.screenshot_as_png
solve_button = self.driver.find_element(By.ID, "solve")
solve_button.click()
time.sleep(self.time_sleep)
result_text = result.text
self.assertTrue(len(result_text) > 0, msg = "Result should appear")
self.assertEqual(result_text, "d = 4,920365840057017", msg = "
    ↪ Result should be correct")
hint_button = self.driver.find_element(By.ID, "hintButton")
hint_button.click()
time.sleep(self.time_sleep)
hint = self.driver.find_element(By.ID, "hint")
hint_text = hint.text
self.assertTrue(len(hint_text) > 0, msg = "Hint text should appear
    ↪ ")
hint_button.click()
time.sleep(self.time_sleep)
hint = self.driver.find_elements(By.ID, "hint")
self.assertTrue(len(hint) == 0, msg = "Hint text area should not be
    ↪ on the page")
graph_new = graph.screenshot_as_png
self.assertNotEqual(graph_old, graph_new, "New graph should appear
    ↪ ")

```

Na rysunku 5.4 zamieszczono wynik testów funkcjonalnych.



Ran 10 tests in 344.116s

OK

Rys. 5.4: Wynik testów funkcjonalnych

Wszystkie testy przeszły bez żadnych błędów.

Rozdział 6

Podsumowanie

6.1. Podsumowanie pracy

Celem niniejszej pracy dyplomowej było stworzenie aplikacji o architekturze klient-serwer, która umożliwi użytkownikom z niej korzystającym wykonywanie podstawowych obliczeń matematycznych oraz ich naukę. Aplikacja miała za zadanie rozwiązać problemy konkurencyjnych rozwiązań, takie jak niedostateczna wizualizacja wskazówek zawierających sposób osiągnięcia rozwiązania oraz konieczność posiadania licencji premium, aby uzyskać dostęp do niektórych typów obliczeń i rozwiązania krok po kroku. Zdecydowano, że w powstałym serwisie będzie możliwe wykonywanie 9 różnych typów obliczeń: rozwiązywanie równania liniowego oraz kwadratowego, kalkulator, obliczanie średniej arytmetycznej oraz ważonej liczb, sortowanie bąbelkowe oraz szybkie (quicksort), obliczanie odległości pomiędzy punktami na płaszczyźnie oraz obliczanie długości boków i wartości funkcji trygonometrycznych kątów trójkąta prostokątnego. Do wykonania frontendu aplikacji autor zdecydował się na użycie następujących technologii: JavaScript, TypeScript, React, HTML5, CSS3, natomiast w części backendowej został użyty Python z frameworkiem Django. Aplikację udało się wdrożyć w usłudze chmurowej Microsoft Azure z wykorzystaniem konteneryzacji przy pomocy narzędzia Docker. Aplikacja dostępna jest pod adresem <https://calcfree.azurewebsites.net/>. Z założonych podczas projektowania funkcji, jakie powinny być dostępne w serwisie, udało się wszystkie zrealizować. Powstałe oprogramowanie zarówno części frontendowej, jak i backendowej zostało przetestowane przy pomocy testów manualnych oraz automatycznych. Wykonano testy jednostkowe, integracyjne oraz funkcjonalne. Wszystkie testy przeszły bez żadnych błędów. Stworzoną aplikację można wykorzystać głównie do nauki matematyki oraz w codziennym życiu, gdy istnieje potrzeba wykonania obliczeń, które są dostępne w aplikacji.

6.2. Perspektywy dalszego rozwoju aplikacji

Aplikację można dalej rozwijać poprzez dodanie nowych funkcji/typów obliczeń, które będą dostępne. Użyty framework django oraz biblioteka React umożliwiają bardzo łatwe rozszerzanie aplikacji. Aby aplikacja była możliwa do znalezienia przy użyciu wyszukiwarki, trzeba wykupić domenę. W przypadku dużej liczby użytkowników jednocześnie korzystających z serwisu należałoby zakupić także usługę, która będzie udostępniać więcej zasobów obliczeniowych. Obecna usługa pozwala na wydajne działanie przy niewielkiej liczbie użytkowników. W celu optymalizacji czasu odpowiedzi serwera można do aplikacji dodać bazę danych, w której będą przechowywane wyniki najczęstszych obliczeń wykonywanych przez użytkowników oraz zmodyfikować oprogramowanie serwera tak, żeby najpierw szukał rozwiązania w bazie danych. Aby

wyszukiwanie było szybkie i dodanie bazy nie zwiększyło czasu odpowiedzi serwera, baza ta może mieć architekturę opartą, np. o B-drzewo.

Literatura

- [1] Axios. <https://axios-http.com/docs/intro> [dostęp 15 stycznia 2023].
- [2] Django. <https://www.djangoproject.com/> [dostęp 17 stycznia 2023].
- [3] Docker. <https://www.docker.com/> [dostęp 18 stycznia 2023].
- [4] Github. <https://github.com/> [dostęp 15 stycznia 2023].
- [5] Javascript. <https://www.javascript.com/> [dostęp 16 stycznia 2023].
- [6] Klient-serwer. <https://vavatech.pl/technologie/architektura/Klient-Server> [dostęp 14 stycznia 2023].
- [7] Klient-serwer. <https://pl.wikipedia.org/wiki/Klient-serwer> [dostęp 14 stycznia 2023].
- [8] Matplotlib. <https://matplotlib.org/> [dostęp 18 stycznia 2023].
- [9] Microsoft azure. <https://azure.microsoft.com/pl-pl> [dostęp 18 stycznia 2023].
- [10] Microsoft math solver. <https://math.microsoft.com/pl> [dostęp 14 stycznia 2023].
- [11] Microsoft math solver-photo. <https://play.google.com/store/apps/details?id=com.microsoft.math> [dostęp 1 lutego 2023].
- [12] Postman. <https://www.postman.com/> [dostęp 20 stycznia 2023].
- [13] Programowanie klient-serwer z użyciem gniazd i protokołów sieciowych. <https://edu.pjwstk.edu.pl/wyklady/mpr/scb/W8/W8.htm> [dostęp 14 stycznia 2023].
- [14] Python. <https://www.python.org/> [dostęp 17 stycznia 2023].
- [15] React. <https://pl.reactjs.org/> [dostęp 15 stycznia 2023].
- [16] React router. <https://reactrouter.com/en/main> [dostęp 15 stycznia 2023].
- [17] Selenium. <https://pypi.org/project/selenium/> [dostęp 20 stycznia 2023].
- [18] Typescript. <https://www.typescriptlang.org/> [dostęp 15 stycznia 2023].
- [19] Visual paradigm. <https://www.visual-paradigm.com/> [dostęp 15 stycznia 2023].
- [20] Visual studio code. <https://code.visualstudio.com/> [dostęp 15 stycznia 2023].
- [21] Wolframalpha main page. <https://www.wolframalpha.com/> [dostęp 14 stycznia 2023].
- [22] W. Kawa, W. Staszewska. Wersjonowanie kodu czyli github + tia portal, June 2022. <https://automatyk.pwr.edu.pl/wersjonowanie-kodu-czyli-github-tia-portal/> [dostęp 21 stycznia 2023].
- [23] M. Kuba. Dlaczego twoja aplikacja powinna być napisana w react.js?, November 2021. <https://boringowl.io/blog/dlaczego-twoja-aplikacja-powinna-byc-napisana-w-reactjs> [dostęp 21 stycznia 2023].


-
- [24] K. Myśliwiec. Typescript – wprowadzenie, typy proste i złożone, November 2016. <https://kamilmysliwiec.com/typescript-wprowadzenie-typy-proste-zlozone> [dostęp 21 stycznia 2023].
- [25] Radek. Aplikacje napisane w django. jakie firmy używają django? <https://thestory.is/pl/journal/popularne-aplikacje-django/> [dostęp 21 stycznia 2023].
- [26] Sieci. Klient - serwer i sieć p2p | wady i zalety. <http://wszystkoosieciach.blogspot.com/2011/09/klient-serwer-i-siec-p2p-wady-i-zalety.html> [dostęp 24 stycznia 2023].
- [27] T. Stelmach. Typy testów, August 2021. <https://qualityisland.pl/typy-testow/> [dostęp 20 stycznia 2023].
- [28] wikipedysta:CiaPan. Test jednostkowy, 2017. https://pl.wikipedia.org/wiki/Test_jednostkowy [dostęp 20 stycznia 2023].
- [29] wikipedysta:Stok. Testy integracyjne, 2022. https://pl.wikipedia.org/wiki/Testy_integracyjne [dostęp 20 stycznia 2023].

Dodatek A

Instrukcja wdrożeniowa

Wdrożenie aplikacji może znacząco różnić się w zależności od miejsca, w jakim aplikacja ma zostać wdrożona. Do wdrożenia można użyć narzędzia Docker, które to ułatwia, ale nie jest to konieczne. W celu wdrożenia aplikacji na Microsoft Azure należy wykonać następujące kroki:

1. Zainstalowanie narzędzia Docker, które jest dostępne na systemy operacyjne Windows, Linux oraz Mac. Dobra instrukcja, jak to zrobić znajduje się pod adresem <https://docs.docker.com/get-docker/>.
2. Uruchomienie Dockera.
3. Skopiowanie na dysk komputera kodu części backendowej aplikacji z załączonej płyty CD.
4. Stworzenie konta na Microsoft Azure, zalogowanie się oraz aktywowanie darmowej licencji, jeżeli ma się do niej dostęp lub wpisanie danych karty kredytowej.
5. Uruchomienie usługi ContainerRegistry.
6. Uruchomienie usługi AppService na Microsoft Azure. W opcji „Publikuj” należy wybrać „Kontener Docker”, a w opcji „System operacyjny” wybrać „Linux”. Trzeba także podać dane wcześniej utworzonego kontenera (usługa ContainerRegistry), z którego ma być pobierany obraz do usługi. Po wejściu na stronę uruchomionej usługi AppService, w miejscu w którym znajdują się właściwości usługi, jest widoczny atrybut „Obraz kontenera”. Należy go zapamiętać. Przedstawione jest to na rysunku A.1.

Właściwości	Monitorowanie	Dzienniki	Możliwości	Powiadomienia	Rekomendacje
 Aplikacja internetowa					
Nazwa	calcfree				
Model publikowania	Kontener				
Obraz kontenera	engineercontainer.azurecr.io/freecalc:latest				

Rys. A.1: Właściwości uruchomionej usługi AppService

7. Otwarcie konsoli systemu.
8. Przejście do folderu, w którym znajduje się plik Dockerfile, w systemie Windows oraz Linux wykonuje się to przy pomocy komendy „cd”.
9. Zalogowanie się do utworzonego w Azure rejestru kontenerów przy pomocy komendy „docker login”.

10. Utworzenie obrazu oprogramowania serwera przy pomocy komendy „docker build . -t backend”.
11. Powiązanie utworzonego obrazu z rejestrem kontenerów na Azure przy pomocy komendy „docker tag backend <Zapamiętany obraz kontenera na Azure>”.
12. Wysłanie obrazu przy pomocy komendy „docker push <Zapamiętany obraz kontenera na Azure>”.

Po kilku minutach od wdrożenia aplikacja powinna być dostępna w internecie pod adresem ustawionym w czasie tworzenia usługi AppService.

Dodatek B

Opis załączonej płyty CD/DVD

Na załączonej płycie znajduje się praca dyplomowa w pliku z rozszerzeniem pdf oraz archiwum zip, w którym został umieszczony plik „Readme.txt” zawierający informacje o autorze i dane pracy dyplomowej oraz trzy foldery. W folderze o nazwie „Praca dyplomowa-Latex” znajduje się praca w postaci kodu Latex skompresowanego do archiwum zip. W folderze o nazwie „Frontend” został umieszczony kod części frontendowej aplikacji, a w folderze o nazwie „Backend” kod części backendowej aplikacji.