Politechnika Częstochowska

Wydział Informatyki i Sztucznej Inteligencji

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja wspomagająca zarządzanie budżetem

Application for Budget Management Support

Dawid Ziora

Nr albumu: 136700

Kierunek: Informatyka

Studia: stacjonarne

Poziom studiów: I

Promotor pracy:

dr inż. Bartosz Kowalczyk

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, 2025

Spis treści

1. Wstęp

W gospodarstwach domowych jednym z kluczowych obszarów decyzyjnych mających bezpośredni wpływ na poziom zaspokojenia potrzeb jest strefa finansowa. W jej ramach podejmowane są decyzję związane z konsumpcją, oszczędzaniem bądź inwestowaniem.

Jednym z celów wykorzystania aplikacji wspomagających zarządzanie budżetem domowym jest wspieranie procesów decyzyjnych konsumentów [6]. Aplikacje finansowe podnoszą świadomość społeczeństwa dotyczącą finansów oraz zwiększają chęć do planowania i kontroli budżetu domowego. Usługi finansowe są dostępne w każdym miejscu i czasie niemalże dwadzieścia cztery godziny na dobę. Aplikacje te najczęściej cechują się interfejsem przyjaznym dla użytkownika, kategoryzacją i śledzeniem wydatków, możliwością przypominania o rachunkach oraz zapewniają ochronę informacji personalnych.

W ankiecie przeprowadzonej dla AcademyBank wskazano, że 83.1% osób śledzi swoje wydatki, z czego 45.3% używa do tego narzędzi cyfrowych. Jednym z rozwiązań jest korzystanie z aplikacji budżetowych. Użycie tych narzędzi zadeklarowało 20.9% ankietowanych. Z biegiem czasu te liczby będą tylko rosły, ponieważ na rynku będzie pojawiać się coraz więcej narzędzi umożliwiających proste zarządzanie własnymi finansami. Niemal 80% osób korzystających z platform do zarządzania budżetem deklaruje, że korzysta z nich przynajmniej raz w tygodniu [2].

W celu analizy jakości aplikacji do zarządzania finansami osobistymi przeprowadzono badanie na grupie $N = 301$ polaków, z których 288 przyznało, że korzysta z aplikacji do wspomagania budżetem, a tylko 13, że nie korzysta. Główne czynności, do których Polacy wykorzystują aplikacje to Kontrola budżetu domowego (88.54%), Weryfikacja wydatków z ostatniego miesiąca (86.11%), Sprawdzanie salda rachunku/-ów (48.26%) oraz planowanie wydatków na kilka miesięcy (45.83%) [10]. **(tu ciekawy artykuł można wstawić więcej danych)**

Dane zebrane przez stronę CoinLaw przedstawiają, że aplikacje do wspomagania budżetem domowym są najczęściej wykorzystywane w przedziale wiekowym od 27 do 42 lat (91%). Kolejna grupa to osoby w wieku od 43 do 58 lat (80%). W przedziale od 18 do 26 roku życia jest to 68% [3]. Aktualna wielkość rynku dla technologii finansowych jako usługi (ang. *Fintech as a Service*) wynosi około 441.47$ miliardów i przy aktualnym tempie wzrostu może urosnąć nawet do 906$ miliardów do 2030 roku [4].

1.1. Cel pracy

Celem pracy jest zbudowanie aplikacji do zarządzania budżetem domowym. Aplikacja ma za zadanie wspomagać użytkownika w kontroli wydatków, analizie danych finansowych oraz zarządzaniu finansami osobistymi.

1.2. Zakres pracy

Niniejszy praca została utworzona w oparciu o architekturę klient-serwer, która umożliwia tworzenie aplikacji webowych. Aplikacja charakteryzuje się wysoką wydajnością, niezawodnością i czytelnym interfejsem użytkownika. Użytkownik dostaje wskazówki odnośnie korzystania z aplikacji, gdy pierwszy raz z niej korzysta. Interfejs użytkownika będzie wyposażony w przyciski pozwalające na łatwą nawigację oraz zmianę treści wyświetlanych na stronie. Użytkownicy mogą utworzyć wpłaty oraz wypłaty z konta oraz kategoryzować transakcje. Podsumowanie finansów przedstawia wykres, na którym można przeglądać wydatki za dzień, tydzień, miesiąc, rok. Aplikacja będzie dostępna na przeglądarki internetowe.

Warstwa serwerowa została zaimplementowana w języku Java z wykorzystaniem frameworka Spring Boot. Zastosowano język Java w wersji 21, który zapewnia obsługę logiki biznesowej, komunikację z bazą danych oraz interfejs API w technologii REST.

Warstwa prezentacji została zrealizowana w oparciu o React + Vite. Język, wykorzystany po stronie frontendu to TypeScript, który wspiera statyczne typowanie. Elementy, z których zbudowany jest interfejs użytkownika są wykorzystane z popularnego reactowego frameworka UI o naziwe Ant Design. Składa się on ze wstępnie zbudowanych komponentów, które łatwo jest wyświetlić na stronie internetowej.

Dane są przechowywane w nierelacyjnej bazie danych MongoDB, która zawiera dokumenty JSON ze wszystkimi ważnymi informacjami, które są związane z użytkownikami aplikacji.

Aplikacja uruchamiana jest z poziomu platformy docker.

1.3. Struktura pracy (Budowa pracy)

Streszczenie wszystkiego (1 akapit) - bez wstępu

Pierwszy rozdział opisuje architekturę aplikacji oraz technologie wykorzystane w procesie jej tworzenia. Prezentuje biblioteki oraz frameworki wraz z opisami ich zastosowań. Tłumaczy, dlaczego aplikacja została zaimplementowana z wykorzystaniem danych języków programowania oraz ukazuje sposób komunikacji poszczególnych części systemu.

Drugi rozdział przedstawia metodykę pracy oraz narzędzia, które zostały wykorzystane w procesie tworzenia oprogramowania. Opisuje metody w ramach których postępuje praca oraz prezentuje środowiska programistyczne, system kontroli wersji i narzędzia do konteneryzacji i testowania komunikacji.

Trzeci rozdział skupia się na wymaganiach, które musi spełnić projekt oraz na diagramach przedstawiających oczekiwany sposób działania wybranych funkcji aplikacji. Opisuje czym są poszczególne diagramy, do czego służą oraz sposób w jaki powinno się czytać zawarte w nich informację.

Czwarty rozdział prezentuje informację w jaki sposób został zaimplementowany system. Opisuje jak działają poszczególne warstwy systemu oraz w jaki sposób odbywa się przepływa danych między poszczególnymi warstwami. Przedstawia funkcję wraz z opisami ich działania oraz komunikaty jakie zwraca serwer.

Piąty rozdział informuje użytkownika w jaki sposób powinien poruszać się po aplikacji. Przedstawia miejsca, w których znajdują się kluczowe elementy interfejsu użytkownika oraz opisuje za jakie funkcję są odpowiedzialne.

Szósty rozdział podsumowuje efekty pracy oraz przedstawia wnioski. Informuje, czy zostały osiągnięte założone efekty pracy oraz prezentuje „Moje osiągnięci".

2. Przegląd technologii stosowanych w aplikacjach klient-serwer

W procesie wytwarzania oprogramowania częstym problemem jest wybór odpowiednich rozwiązań, które pozwolą nam w najlepszy sposób osiągnąć zamierzone cele. Znajomość kompatybilnych technologii w znaczącym stopniu przyśpiesza tworzenie aplikacji webowych. Projekt można podzielić na warstwę prezentacji, która odpowiada za interfejs graficzny oraz interakcję z użytkownikiem, warstwę serwerową odpowiadającą za logikę biznesową aplikacji oraz komunikację z bazą danych oraz warstwę dostępu do danych, która odpowiada za trwałe przechowywanie rekordów i ich udostępnianie innym warstwą.

2.1. Architektura systemu

Jednym ze sposobów projektowania aplikacji sieciowych jest model klient-serwer. Klient nie posiada danych oraz funkcji, dlatego musi kontaktować się z programem posiadającym dostęp do danych oraz usług. Program ten nazywany jest serwerem.

Klient zleca serwerowi żądanie pewnej usługi, następnie serwer analizuje polecenia i wysyła odpowiedź. Model ten jest powszechnie używany w aplikacjach dostępnych za pośrednictwem internetu [8].

Przykładowe zastosowanie w aplikacji webowej

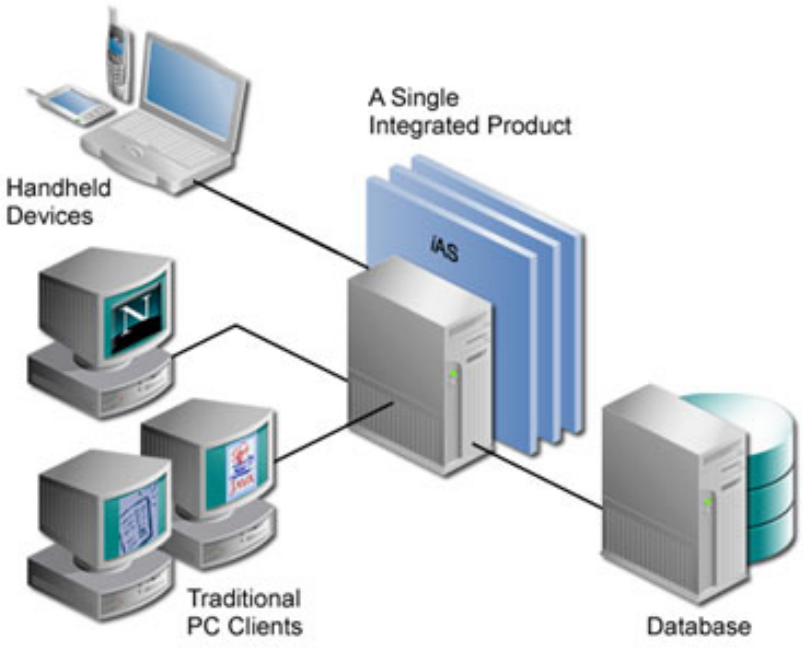
Próba logowania na stronę WWW jest równoważna z wysłaniem do serwera wiadomości z typem operacji, oraz podanymi w formularzu wartościami. Serwer otrzymując taką wiadomość wykonuje niezbędne testy na przykład: sprawdza poprawność otrzymanych danych. Po pomyślnym dokonaniu sprawdzeń serwer wykonuje żądaną operację i wysyła odpowiedź klientowi [8].

Kody odpowiedzi HTTP

Odpowiedź serwera to numeryczna dana wysłana przez serwer, która ma na celu poinformowanie użytkownika o realizacji pewnego zadania. Kody dzielą się na informacyjne, które zaczynają się od 1xx, powodzenia zaczynające się od 2xx, przekierowania (3xx), błędu aplikacji klienta (4xx), błędu serwera HTTP (5xx) [11].

Tabela 2.1. Przykładowe opisy odpowiedzi serwera

Kod	**Opis słowny**	**Znaczenie**
100	Continue	Prośba o dalsze wysłanie zapytania
200	OK	Zawartość żądania (najczęściej zwracany nagłówek)
201	Created	Wysłany dokument został zapisany na serwerze
204	No content	Serwer nie potrzebuje zwracać żadnej treści
300	Multiple Choices	Istnieje więcej niż jeden sposób obsłużenia danego zapytania
302	Found	Zasób jest chwilowo dostępny pod innym adresem
400	Bad Request	Żądanie nie może być obsłużone przez serwer z powodu nieprawidłowości postrzeganej jako błąd użytkownika
401	Unauthorized	Żądanie zasobu, który wymaga uwierzytelnienia
403	Forbidden	Serwer zrozumiał zapytanie, jednak konfiguracja bezpieczeństwa zabrana mu zwrócić żądany zasób
404	Not Found	Serwer nie znalazł zasobu według podanego adresu URL
500	Internal Server Error	Serwer napotkał niespodziewane trudności
502	Bad Gateway	Serwer spełniający rolę bramy otrzymał niepoprawną odpowiedź od serwera nadrzędnego
503	Service Unavaiable	Serwer nie jest w stanie zrealizować żądania użytkownika ze względu na przeciążenia
504	Gateway Timeout	Serwer spełniający rolę bramy nie otrzymał odpowiedzi w ustalonym czasie



Rysunek 2.1. Tymczasowe zdjęcie architektury Klient-Serwer

2.2. Java

Java jest obiektowym wieloplatformowym językiem programowania. Jest ona jednym z najpopularniejszym języków dla deweloperów oprogramowania [5]. Korzysta z wirtualnej maszyny Java (JVM), które mogą zostać zainstalowane na większości komputerów i urządzeń przenośnych. Został on stworzony z myślą „napisz raz, uruchamiaj w dowolnym miejscu".
W projekcie jest odpowiedzialny za logikę biznesową aplikacji, przetwarzanie danych oraz komunikację z bazą danych [7].

Spring Boot

Framework Spring Boot pozwala znacząco uprościć proces tworzenia aplikacji webowej w Javie. Eliminuje konieczność ręcznej konfiguracji wielu elementów. Pełni on role fundamentu dla warstwy serwerowej, udostępniając interfejs REST API, który umożliwia komunikację przy użyciu odpowiednich punktów końcowych (ang. *endpoints*). Składa się z takich rzeczy jak:

- Controller - obsługuje żądania HTML
- Repository - odpowiada za komunikacje z bazą danych
- Config - Obejmuje konfiguracje aplikacji, polityki bezpieczeństwa, ciasteczka
- Model - Reprezentuje dane w postaci klas, które są odwzorowaniem tabel w bazie danych.

- Resource - Znajdują się w nim właściwości aplikacji, oraz statyczne elementy.

Spring Boot wspiera również takie mechanizmy jak Wstrzykiwanie zależności (ang. *Dependency Injection*) oraz AOP (ang. *Aspect-Oriented Programming*) [9].

Maven

W projekcie wykorzystano narzędzie Apache Maven, które pełni role automatyzacji budowy i zarządzania zależnościami. Pozwala zdefiniować wszystkie wymagane biblioteki i frameworki w jednym centralnym pliku pom.xml (ang. Project Object Model). Działa w oparciu o repozytoria, w których przechowywane są biblioteki. Główny plik zawiera takie informacje jak:

- groupId, artifactId, version - identyfikatory projektu,
- sekcję dependencies - listę bibliotek, które mają zostać automatycznie pobrane,
- sekcję build - ustawienia dotyczące budowania aplikacji,
- sekcje plugins - narzędzia wspierające proces budowania

W przypadku aplikacji opartej na Spring Boot, Maven pobiera również tzw. startery np.:

- spring-boot-starter-web - uruchamia aplikacje serwerową (np. Tomcat) i pozwala tworzyć kontrolery REST
- spring-boot-starter-data-mongodb - Umożliwia korzystanie z bazy mongo
- spring-boot-starter-security - Dodaje mechanizmy autoryzacji i uwierzytelniania
- spring-boot-starter-test - pozwala na testowanie aplikacji

Cała konfiguracja środowiska sprowadza się do utworzenia odpowiedniego pliku z zależnościami, a Spring Boot automatycznie na jego podstawie konfiguruje je przy uruchomieniu [1].

Dane do wstawienia w ten rozdział

Korzysta między innymi z takich zależności jak:

- Lombok - Ułatwia tworzenie klas i podstawowych funkcji w klasach.
- DevTools - Pozwalające m.in. na przeładowanie w czasie rzeczywistym.
- Web - Zawiera RESTful API, pozwala na komunikacje.

- Security - Umożliwia zastosowanie zabezpieczeń i autoryzacji do kontrolowania aplikacji.

- MongoDB - Do przechowywania dokumentów w formacie zbliżonym do JSON.

- Validation - pozwala na walidacje pól na przykład: ustawienie długości pola, pole nie może być puste i tym podobne.

2.3. React

React to jedna z bibliotek JavaScript służąca do tworzenia interfejsów użytkownika (ang. User Interface) w aplikacjach webowych. Jest oparta na koncepcji komponentów, które można łatwo ze sobą łączyć i w czytelny sposób wyświetlać na stronie. React pozwala renderować widoki na podstawie mechanizmu Virtual DOM (ang. Document Object Model). Obiekt DOM umożliwia dostęp do struktury strony w celu jej modyfikacji. W przypadku modelu wirtualnego minimalizuje operacje na drzewie rzeczywistym.

Vite

W celu usprawnienia procesu tworzenia aplikacji zastosowano narzędzie Vite, które pełni rolę bundlera, czyli łączy ze sobą wiele plików m.in. kody źródłowe i zależności. Rozwiązanie to oferuje szybkie uruchamianie środowiska, optymalizacja kodu i co najważniejsze w usprawnieniu pracy poprzez przeładowanie kodu na bieżąco (ang. Hot Module Replacement), który pozwala wyświetlać zmiany bez konieczności ponownego budowania całej aplikacji.

TypeScript

Komponenty oraz wszystkie składowe projektu są przygotowane w języku TypeScript, który jest nadzbiorem języka JavaScript. Wprowadza on statyczne typowanie, znaczy to że typy są określone i sprawdzane w czasie kompilacji. Pozwala wprowadzać typy dla zmiennych, funkcji i obiektów. Dzięki temu kod jest bardziej czytelny i zrozumiały dla developera.

Połączenie Reacta, Vite oraz TypeScript zapewnia wydajne i przyjemne środowisko dla pracy programisty, lepszą kontrole nad danymi i wydajność pracy. Technologie te zostały wybrane również dla bogatego wyboru bibliotek wspieranych przez react. Jedną z nich jest biblioteka AntD, która zawiera wstępnie przygotowane do użycia na stronie komponenty.

2.4. MongoDB

MongoDB to nierelacyjna baza typu NoSQL. W przeciwieństwie do klasycznych baz relacyjnych MongoDB nie korzysta z tabel, lecz przechowuje dane w postaci dokumentów BSON (Binary JSON), które strukturą przypominają obiekty JSON. Jeden dokument może mieć wiele różnych pól, co pozwala bardziej elastycznie modelować dane. Baza ta korzysta z kolekcji (ang. collections), które są odpowiednikami tabel w klasycznych bazach relacyjnych. Każdy wpis w bazie posiada własny unikatowy identyfikator _id. Wspiera ona wiele operacji, takich jak sortowanie, filtrowanie, czy grupowanie. Zaimplementowanie ich w kodzie programu jest proste i wystarczy do tego odpowiednio utworzone repozytorium z odpowiednimi operacjami (lst. 2.1). Połączenie bazy w aplikacji Java odbywa się za pomocą modułu Spring Data MongoDB. To właśnie ten moduł odpowiada za możliwość tworzenia repozytorium.

Listing 2.1. Przykładowe repozytorium w Javie

```
1 @RepositoryRestResource(collectionResourceRel = "users", path = "users")
2 public interface UserRepository extends MongoRepository<User, String> {
3   Optional<User> findByLogin(String login);
4   List<User> findByLoginContainingIgnoreCase(String loginPart);
5 }
```

Powyższy fragment kodu pozwala znaleźć użytkownika po loginie lub wyszukać wszystkich użytkowników zawierających podany ciąg znaków w swojej nazwie.

Listing 2.2. Przykladowy dokument z kolekcji

```
1 {
2   "id": "68d56ad95a544e07c8ebaa54",
3   "login": "ADMIN",
4   "password": "$2a$10$.itTB4jFiMBPdZSDebVE4Obtl8FpDiT7CHovqCtq8dcUnFMoe1gem",
5   "role": "ADMIN"
6 }
```

3. Narzędzia i metodyka pracy

W rozdziale przedstawiono zestaw narzędzi oraz metodykę pracy zastosowaną podczas tworzenia aplikacji webowej. Opisane rozwiązania pozwoliły na efektywne tworzenie, testowanie i wdrażanie aplikacji.

3.1. Metodyka pracy

Jaki metodyka np. Zwinna (Agile/Scrum)

3.2. Narzędzia stosowane w procesie wytwarzania aplikacji webowej

W projekcie została zastosowana masa narzędzi, które usprawniają pracę dewelopera.

Środowiska programistyczne

- IntelliJ IDEA
- Webstorm
- Node.js + npm
- Spring Boot / Maven

System kontroli wersji

System kontroli wersji to jedna z najważniejszych rzeczy podczas pisania dowolnej aplikacji.Jednym z najczęściej używanych rozwiązań jest git. Pozwala on na tworzenie lokalnych lub zdalnych repozytoriów do których zapisywane są zmiany w projekcie (commit). W dowolnym momencie możemy śledzić modyfikacje plików lub powrócić do wcześniejszej wersji projektu (rollback). Aby przesłać zmiany do zdalnego repozytorium na stronie trzeciej (np.: github, bitbucket), należy wykorzystać opcję (push), natomiast pobranie wymaga operacji (pull).

Git umożliwia również pracę w grupach dzięki systemowi tworzenia gałęzi (branches), które pozwalają na równoległą prace całego zespołu. Zakończone części kodu mogą być scalone (marge) z główną gałęzią.

Testowanie i komunikacja

Postman to narzędzie pozwalające na testowanie i analizę interfejsów API. Umożliwia wysyłanie żądań HTTP (GET, POST, PUT, DELETE) do serwera i sprawdzanie zwracanych odpowiedzi. Pozwala to na szybkie i prostą weryfikację poprawności działania endpointów. Umożliwia on tworzenie kolekcji z zapytaniami, które można używać wiele razy bez konieczności zapisu ich po każdym użyciu. Posiada też możliwość zapisywania ciasteczek oraz tworzenia własnych nagłówków.

Docker to platforma umożliwiający uruchamianie aplikacji w środowisku kontenerowym. Jego głównym elementem jest silnik (ang. Docker Engine), który działa w formie usługi. Umożliwia on budowanie, uruchamianie i zarządzanie kontenerami. W środowisku Dockera można tworzyć obrazy, które zawierają niezbędne elementy do uruchomienia aplikacji. Na podstawie tych obrazów realizują konkretne zadania w izolowanym środowisku. Zaletą zastosowania Dockera jest przenośność, czyli uruchomienie projektu na dowolnej maszynie wyposażonej w środowisko Docker. W Projektach bardziej złożonych, korzystających z wielu usług, można wykorzystać plik `compose.yml`, w którym definiuje się zależności pomiędzy kontenerami, konfigurację sieci, woluminy oraz zależności uruchamiania poszczególnych usług.

Do wykorzystania tekst

Testy jednostkowe są wykonywane przy pomocy biblioteki JUnit w wersji 5. Testy są wykonywane przed uruchomieniem aplikacji, aby sprawdzić jej poprawne działanie.

Do sprawdzenia poprawności działania tych i innych funkcji wykorzystuje się narzędzie Postman, które pozwala wysyłać zapytania, sprawdzać endpointy i ogólną komunikację między klientem a serwerem. Narzędzie to jest powszechnie wykorzystywane przez deweloperów, ponieważ ułatwia ich pracę i umożliwia szybszy sposób na sprawdzanie poprawności działania funkcji bez konieczności implementacji ich na stronie internetowej.

Docker ..., czyli jest zapakowane w wirtualny kontener na którym są wykonywane wszystkie operację. Do tego celu służy silnik Docker Engine, który umożliwia pobieranie wirtualnych obrazów systemów. Pozwala to na automatyzację pracy oraz niezmienność wersji programu w procesie wytwarzania oprogramowania. Daje możliwość, by daną aplikację uruchomić na dowolnym komputerze, bez konieczności posiadania zainstalowanych aplikacji takich jak mongoDB, wirtualnej maszyny Javy, czy też menadżera pakietów npm.

4. Architektura aplikacji

W rozdziale przedstawiono wymagania funkcjonalne i niefunkcjonalne, które zostały postawione aplikacji. Zaprezentowano również diagramy przypadków użycia, stanów oraz sekwencji.

4.1. wymagania funkcjonalne

Wymagania funkcjonalne określają zachowanie systemu i jego reakcji na określone zadania [12].

Nr	**Wymaganie funkcjonalne**	**Opis**
1.	Rejestracja użytkownika	System umożliwia utworzenie konta użytkownika z danymi logowania.
2.	Logowanie do systemu	System weryfikuje dane użytkownika i umożliwia dostęp do panelu aplikacji.
3.	Sprawdzenie dostępnych środków	Użytkownik może sprawdzić ile posiada balansu na koncie.
4.	Wpłata pieniędzy na konto	Użytkownik może zasilić konto.
5.	Przegląd historii wydatków	Aplikacja wyświetla listę wszystkich wydatków.
6.	Tworzenie przelewu	System pozwala stworzyć przelew użytkownikowi.
7.	Podsumowanie finansów konta	Użytkownik może sprawdzić dane o swoich wydatkach dla danego konta.
8.	Kategoryzowanie transakcji	Użytkownik może podać kategorię dla wpłat i wypłat z konta.
9.	Prezentacja wydatków na wykresie	System wyświetla wykres z wydatkami za dzień, tydzień, miesiąc oraz rok.

Tabela 4.1. Wymagania funkcjonalne systemu do zarządzania budżetem domowym

4.2. Wymagania niefunkcjonalne

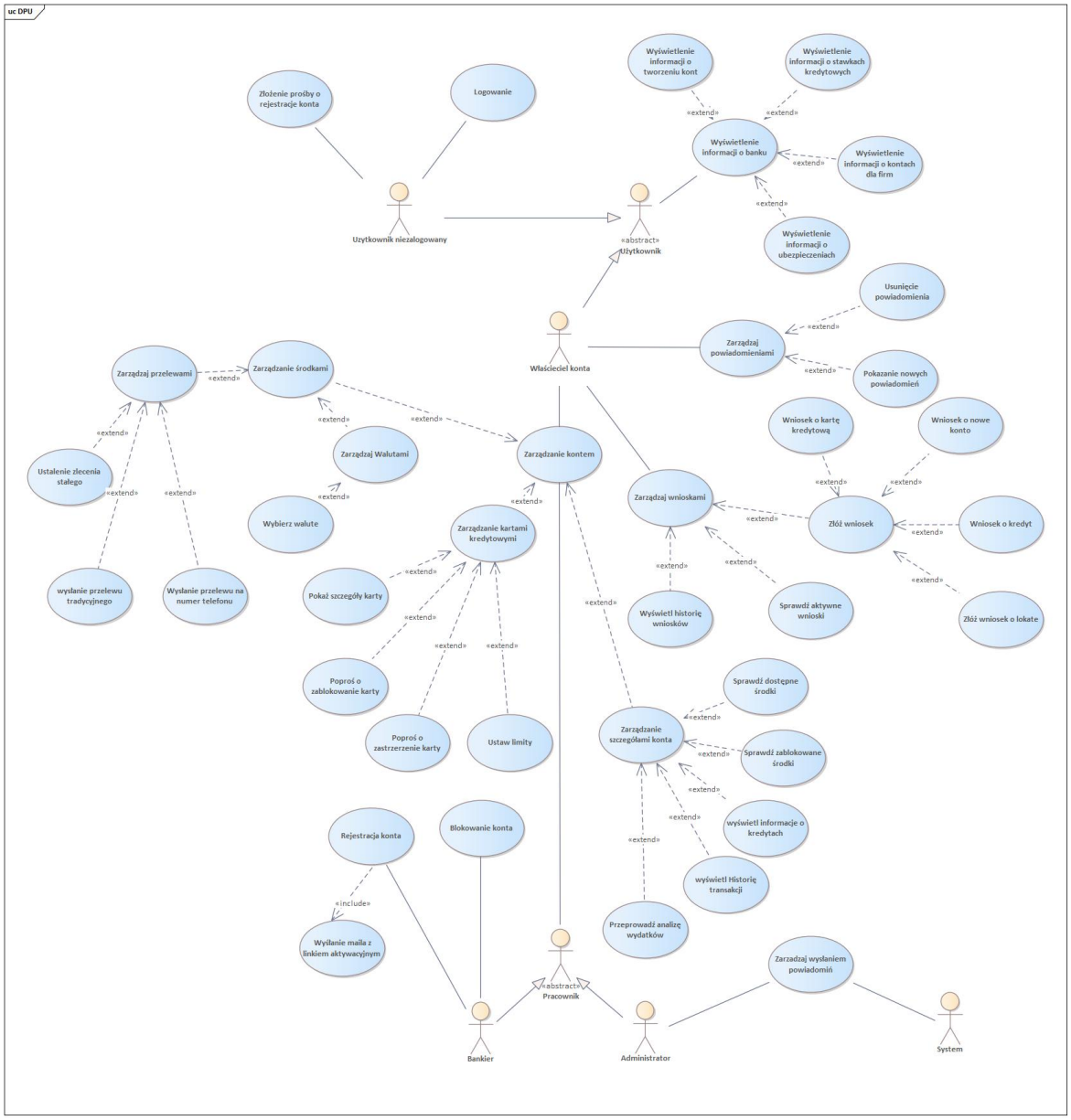
Wymaganiami funkcjonalnymi nazywamy wszystkie potrzeby, które nie dotyczą funkcjonalności produktu [12]. Wymagania te opisują m.in. jak szybko system powinien odpowiadać na żądania, w jakim czasie produkt ma zostać dostarczony, jak ma prezentować się interfejs użytkownika.

Nr	**Wymaganie niefunkcjonalne**	**Opis**
1.	Niezawodność	System w krótkim czasie odzyskuje pełną funkcjonalność w przypadku wystąpienia błędu.
2.	Dostępność	System ma być dostępna przez 99.5% czasu działania serwera.
3.	Wydajność	System reaguje na żądanie w czasie krótszym niż 1 sekunda. Może obsłużyć do 1000 użytkowników jednocześnie. System jest w stanie przetworzyć do 10 GB na godzinę. System jest skalowalny.
4.	Bezpieczeństwo	System zapewnia, że dane są dostępne tylko dla osób upoważnionych. Autoryzacja użytkowników odbywa się poprzez podanie odpowiedniego loginu oraz hasła. Hasła muszą być przechowywane w formie zaszyfrowanej.
5.	Wdrożenie	Aplikacja serwerowa musi być uruchomiona w środowisku JVM oraz korzystać z wersji Java 21 , Warstwa prezentacji uruchamiana w środowisku NodeJs z wykorzystaniem 9 wersji biblioteki React. Dane przechowywane w postaci dokumentów bazy danych MongoDB w wersji 8, Aplikacja uruchomiana jest w środowisku systemu operacyjnego Windows 11 oraz Linux

Tabela 4.2. Wymagania niefunkcjonalne systemu do zarządzania budżetem domowym

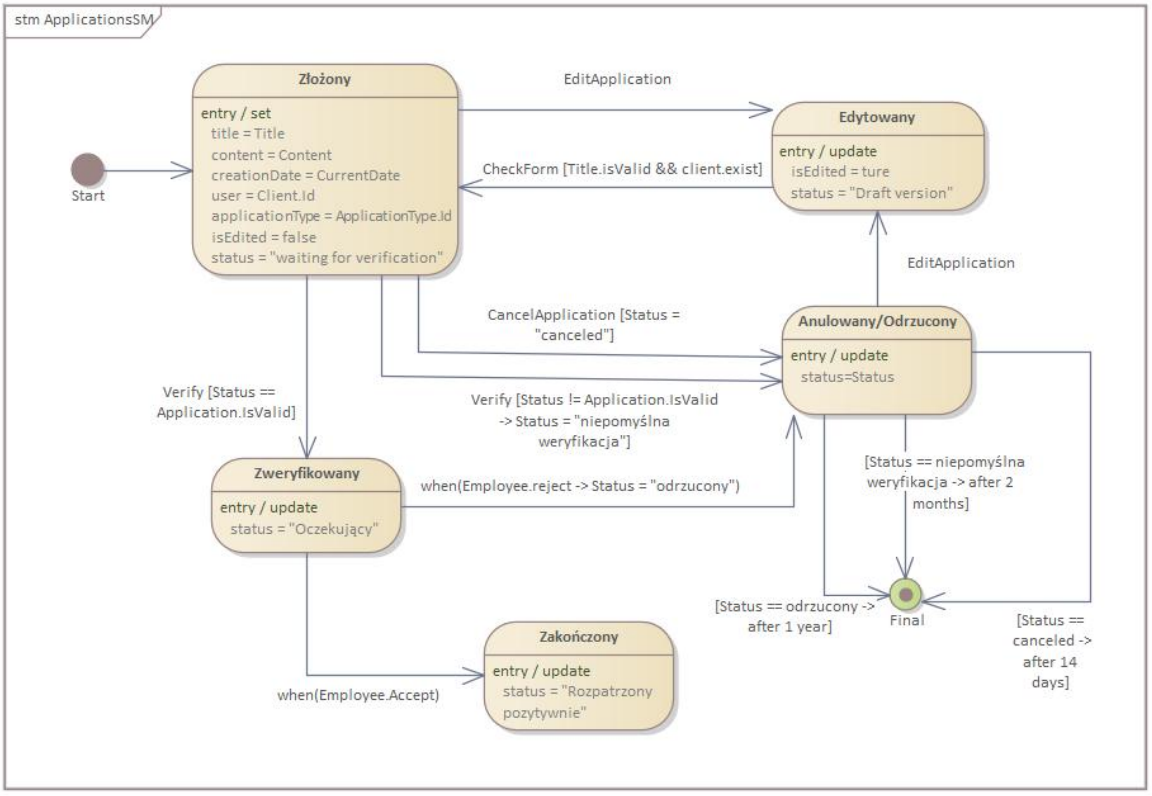
4.3. Diagramy przypadków użycia

Diagram przypadków użycia (ang. *Use Case Diagram*)



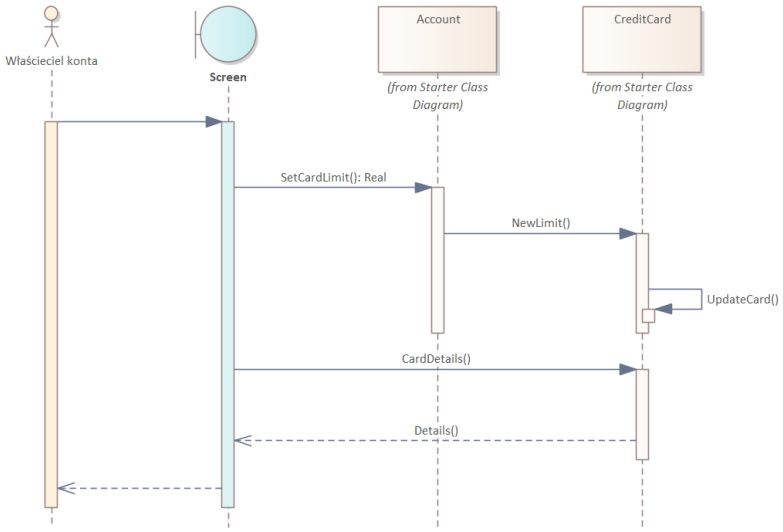
Rysunek 4.1. Diagram przypadków użycia

4.4. Diagram stanów

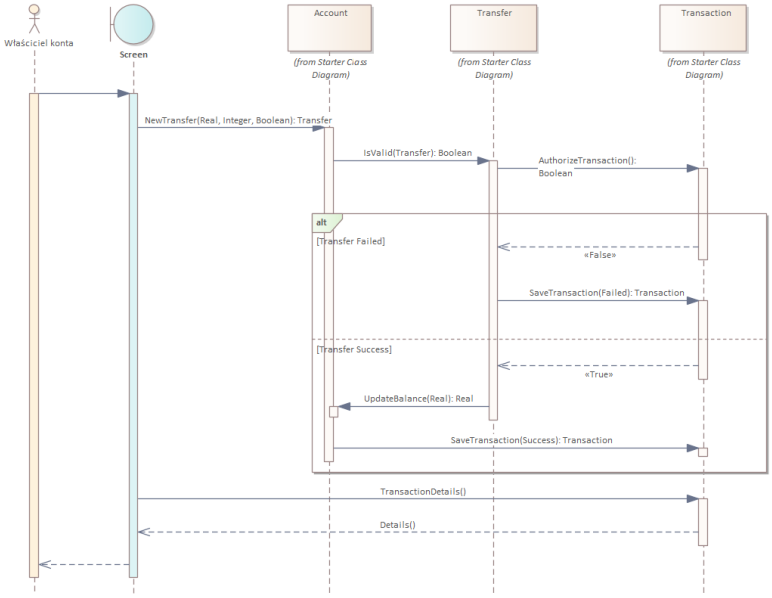


Rysunek 4.2. Diagram stanów składania wniosku

4.5. Diagramy sekwencji



Rysunek 4.3. Diagram sekwencji Limit



Rysunek 4.4. Diagram sekwencji Przelew

5. Implementacja systemu

5.1. Backend – logika biznesowa i API

Backend odpowiada ze przetwarzanie żądań od frontendu oraz za komunikację z bazą danych. Składa się on z kilku warstw m.in. warstwy kontrolerów, repozytoriów, czy Konfiguracji. Do komunikacji wykorzystywany jest standard REST API (ang. Representational State Transfer Application Programing Interface). Udostępnie on takie metody HTTP jak GET, POST, PUT, DELETE.

Jak to działa

1. Klient wysyła żądanie do określonego adresu URL (endpointu) serwera
2. Serwer przetwarza żądanie i wykonuje operację
3. Serwer zwraca odpowiedź w formacie JSON oraz kod stanu HTTP

Opis funkcji

Aplikacja zawiera wiele funkcji w tym Dodawanie lub usuwanie użytkowników, pobieranie listy wpłat i wypłat z konta, autoryzacja użytkowników. Wszystkie te funkcje są zawarte w odpowiednim kontrolerze. Dla każdej funkcji kontrolera został utworzony własny endpoint. Poniżej znajduje się kilka przykładów endpointów:

Listing 5.1. Przykładowe Endpointy

```
1 @RestController
2 @RequestMapping("/User") // Wszystkie endpointy zaczynają się od /User
3   @PostMapping("/register")    // Dostęp pod POST /User/register
4   @GetMapping("/search")       // Dostęp pod GET /User/search
5   @PutMapping("/update/{id}")   // Dostęp pod PUT /User/update/{id}
6   @DeleteMapping("/delete/{id}")  // Dostęp pod DELETE /User/delete{id}
```

Ich pełny opis znajduje się dalej w listingach (5.4, 5.5, 5.6, 5.7)

Połączenie z bazą danych

Połączenie z bazą danych odbywa się za pomocą specjalnego zasobu *application.properties*, czyli właściwości aplikacji. Zawiera on wszystkie potrzebne informacje jak nazwa hosta, port na którym działa baza, nazwę bazy i ogólną konfigurację:

Listing 5.2. application.properties

```
spring.data.mongodb.host=${DB_HOST:localhost}
spring.data.mongodb.port=${DB_PORT:27017}
spring.data.mongodb.database=${DB_NAME:budget_management_db}
spring.data.mongodb.auto-index-creation=true
```

Mapowanie modeli na dokumenty odbywa się dzięki zależności Spring Data MongoDB. Utworzenie klasy na przykład konta wymaga podania @Document przed klasą. Dzięki temu model będzie mógł być konwertowany do formatu dokumentu.

Listing 5.3. Fragment modelu Account

```
@Document
public class Account {
  @Id
  private String id;
  private String name;
  @Size(min = 25, max = 25)
  @Indexed(unique = true)
  private String number;
  private Currency currency;
  private String userId;
  ...
```

Konfiguracja i bezpieczeństwo

Opcje konfiguracji i bezpieczeństwa to bardzo ważna część aplikacji. Odpowiada za sposób generowania tokenów, które następnie zostaną wykorzystane do tworzenia ciasteczek. Odpowiada też za walidację oraz dostęp do metod. Określa jakie funkcje dostępne są dla użytkowników nieautoryzowanych, a które mogą zostać wywołane przez tych posiadających autoryzację.

Przykładowe fragmenty kodu

rejestracja

Listing 5.4. Rejestracja użytkownika

```
/**
* Rejestracja użytkownika.
* @param user Obiekt użytkownika zawierający dane do rejestracji w formacie
    JSON.
*             Wymagane pola: login, password, role.
*/
```

```
@PostMapping("/register")
public String register(@Valid @RequestBody User user) {
  user.setPassword(passwordEncoder.encode(user.getPassword()));
  userRepository.insert(user);
  return "Rejestracja udana";
}
```

Lista użytkowników

Listing 5.5. Lista użytkowników

```
@GetMapping("/list")
public Iterable<User> getUser() {
  return userRepository.findAll();
}
```

Usuwanie użytkownika

Listing 5.6. Usuwanie użytkownika

```
@DeleteMapping("/delete/{id}")
public void deleteUser(@PathVariable String id) {
  userRepository.deleteById(id);
}
```

Edycja użytkownika

Listing 5.7. Edycja użytkownika

```
@PutMapping("/update/{id}")
public void updateUser(@PathVariable String id, @RequestParam String login) {
  User user = userRepository.findById(id).orElseThrow(() -> new
      RuntimeException("User not found"));
  user.setLogin(login);
  userRepository.save(user);
}
```

Testy jednostkowe

Aplikacja posiada również możliwość testowania poprawności działania metod. Testowanie polega na sprawdzeniu, czy funkcja zwraca odpowiednią odpowiedź z serwera, na przykład w przypadku nieautoryzowanego dostępu do funkcji w odpowiedzi od serwera powinniśmy

otrzymać błąd 401 (Unauthorized). W przypadku, gdy wszystko zakończy się powodzeniem otrzymamy sygnał 200 (OK). Kody błędów znajdują się w tabeli (Nie ma jeszcze). Do testowania aplikacji wykorzystuje bibliotekę JUnit w wersji 5.

Listing 5.8. Przykładowy test

```
1 @Test
2 public void createTransferHandlesNegativeTransferAmount() throws Exception {
3   this.mockMvc.perform(MockMvcRequestBuilders.post("/Transaction/create/
        transfer")
4   .contentType("application/json")
5   .content("{\"fromAccountNumber\":\"12345678901222345690123335\",\"
        toAccountNumber\":\"12345678901222345690123335\",\"amount\":-50.0}"))
6   .andExpect(status().is4xxClientError())
7   .andExpect(content().string("amount: must be greater than 0"));
8 }
```

5.2. Frontend – interfejs użytkownika

Warstwa prezentacji systemu, czyli to z czym użytkownik wchodzi w interakcje nazywamy frontendem. Została zrealizowana przy pomocy biblioteki React oraz narzędzia Vite. Kod źródłowy programu został napisany w TypeScipt. Komunikacja została zapewniona dzięki protokołowi HTTP/REST. Powoduje to niezależność frontendu od backendu.

Struktura projektu

Projekt jest podzielony na moduły:

- Assets - przechowywanie statycznych elementów strony (np. ikony),
- Models - Zawiera typy utworzone na potrzeby działania funkcji,
- Services - Odpowiada za komunikacje z backendem,
- Styles - Przechowuje style w formacie .css,
- Context - Obsługuje globalny stan aplikacji,
- Components - Zawiera widoki stron i wszystkie elementy takie jak formularze

Komunikacja z API

Komunikacja z backendem odbywa się dzięki bibliotece axios, która odwołuje się do endpointów strony. Poniżej znajduje się przykładowa funkcja wyszukująca konta.

Listing 5.9. Funkcja wyszukująca wszystkie konta użytkownika

```
export const fetchAccounts= async (): Promise<Accounts[]> =>{
  const id = await fetchUserId();
  try {
    const response = await axios.get(`http://localhost:8080/Account/get/${id.
        id}`, {
      withCredentials: true,
      headers: {
        'Content-Type': 'application/json'
      }
    });
    return await response.data;
  } catch (error: any) {
  throw new Error(error.message || "Wystąpił błąd podczas pobierania kont");
  }
}
```

UI

Interfejs użytkownika jest przejrzysty i łatwy w obsłudze. Wiele elementów na stronie zostało wykorzystane z biblioteki Ant Design, która jest jedną z popularniejszych bibliotek dla React'a. Pozwala ona na tworzenie formularzy, wykresów liniowych, przycisków, ikon i wielu innych przydatnych elementów strony. Użytkownik może personalizować motyw strony (jasny/ciemny), sprawdzić szczegóły profilu lub przejrzeć historię i dane swoich kont bankowych, które ma przypisane do konta.

Stan aplikacji i interakcje

Stany są jednym z kluczowych funkcji, które wykorzystuje się w aplikacjach opartych o React. Wykorzystują funkcję haków (hook), które pozwalają używać stanów bez konieczności posiadania klasy. Do zarządzania stanami możemy użyć funkcji useState lub useEffect. Funkcja useState pozwala nam ustawić jakiś status podczas działania strony. Prostym przykładem będzie zmiana motywu strony przez użytkownika. Wtedy status zmienia się z light na dark. UseEffect pozwala wykonać jakąś funkcję lub działanie i wpłynąć na to jaki status zostanie ustawiony. Poniżej znajdują się przykłady zastosowań dla obu funkcji.

Listing 5.10. Wykorzystanie stanów

```
const [loginData, setLogin] = useState<string | null>(null);

useEffect(() => {
  const fetchLoginData = async () => {
```

```
    try {
      const users = await fetchUsers();
      setLogin(users.join('\n'));
    } catch (err: any) {
      message.error(err.response?.data?.error || err.message);
    }
  };
  fetchLoginData();
}, []);
```

5.3. Przepływ danych

Aplikacja przechowuje dane w postaci dokumentów bazy NoSql MongoDB. Backend komunikuje się z bazą poprzez warstwę repozytoriów i pobiera z niej informacje w formacie JSON, następnie odpowiednio sformatowane dane przesyła odpowiednim endpointem do frontendu, gdy zostanie wywołana odpowiednia metoda.

Przykładowy przepływ danych

Poniżej znajduje się sposób przepływu danych dla funkcji logowania aplikacji:

1. Użytkownik odpala formularz na stronie i wprowadza dane.
2. Frontend wysyła żądanie HTTP do backendu.
3. Backend przetwarza dane i porównuje je z tymi znajdującymi się w bazie.
4. Wynik zwraca w formacie JSON.
5. Frontend aktualizuje widok użytkownika.

Listing 5.11. Przykład przesyłanych danych między frontendem, a backendem

```
[{
  "fromAccountNumber": "36204576739585995585483 79",
  "toAccountNumber": "123456789012223456101 2335",
  "amount": 50.0,
  "description": "test",
  "transactionId": "68dfb1d240d00a2221f78809"
},
{
  "fromAccountNumber": "36204576739585995585483 79",
  "toAccountNumber": "123456789012223456101 2335",
  "amount": 150.0,
```

```
12   "description": "test",
13   "transactionId": "68dfb1d740d00a2221f7880b"
14 }]
```

5.4. Integracja komponentów

Komponenty są uruchamiane w oddzielnych kontenerach Dockera. Plik compose.yml pozwala jednocześnie uruchomić bazę danych, backend i frontend przy pomocy jednego polecenia: *docker compose up*.

Listing 5.12. Budowanie obrazu - Dockerfile

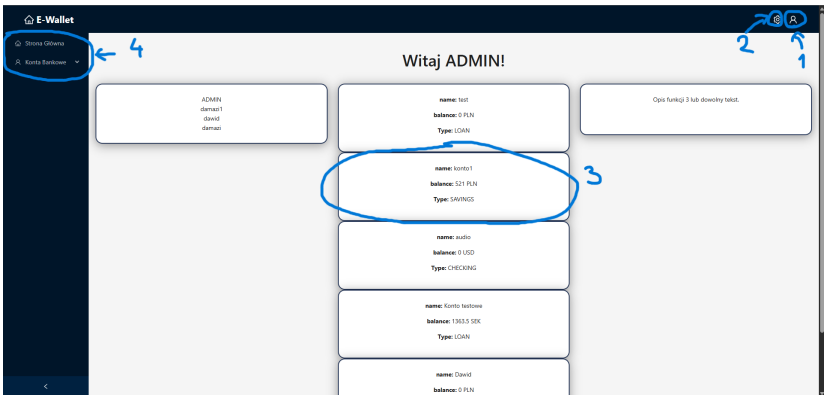
```
1 FROM maven:3.9.10-eclipse-temurin-21
2 WORKDIR /app
3 COPY pom.xml .
4 COPY src ./src
5 EXPOSE 8080
6 CMD ["mvn", "spring-boot:run"]
```

5.5. Przykładowe fragmenty kodu i funkcje

Wywalić przykłady z wcześniej i wrzucić je tutaj? Czy tutaj umieścić coś jeszcze? wszystkie przykłady dałem wcześniej.

6. Prezentacja aplikacji

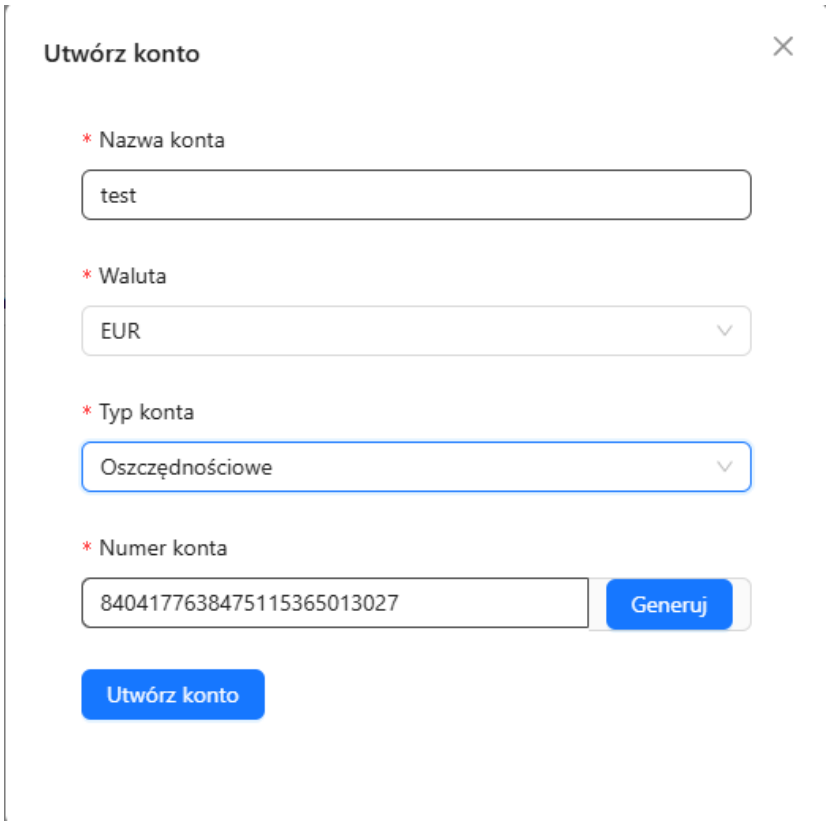
Aplikacja po uruchomieniu wyświetla domyślną stronę (Home), na której wyświetlają się informację. Pierwszą rzeczą jaką należy zrobić po uruchomieniu jest zalogowanie się lub rejestracja.



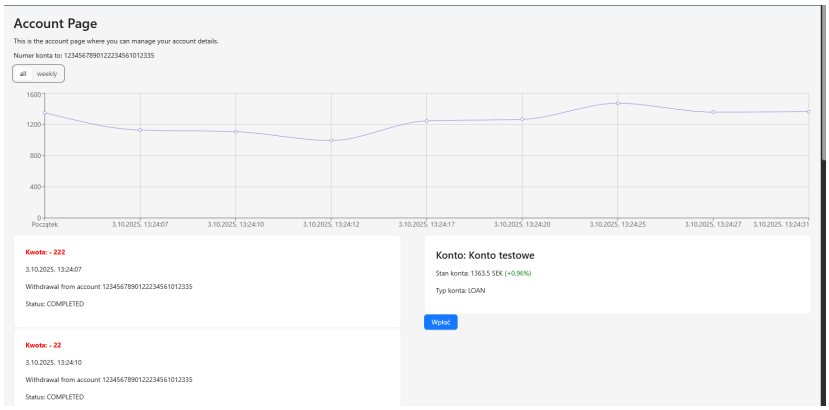
Rysunek 6.1. Strona startowa po zalogowaniu

- Pierwszy zaznaczony element na stronie to aktualnie zalogowany użytkownik, po kliknięciu w ikonę użytkownika wyświetli się informacja o profilu oraz możliwość wylogowania w liście wysuwanej (Dropdown Menu).
- Drugim elementem są ustawienia. Możemy tu ustawić motyw strony.
- Trzeci element to kafelki znajdujące się po środku (Card). Po kliknięciu w nie zostaniemy przeniesieni do odpowiedniego konta.
- Czwarty element to lista wysuwana z boku. Możemy ją schować, by nie zajmowała miejsca na stronie. Dzięki niej możemy powrócić do strony lub uruchomić jedno z naszych kont.

Po uruchomieniu Szczegółów profilu zostaniemy przekierowani do strony na której możemy zobaczyć wszystkie informacje o użytkowniku oraz dodać nowe konto bankowe. Dodanie go wyświetli formularz (Rys.6.2), gdzie możemy dodać nowe konto. Po utworzeniu konta możemy wejść w szczegóły naszego konta, gdzie zobaczymy wykresy z transakcjami, wszystkie transakcje i będziemy mogli posortować je dziennie lub wyświetlić wszystkie (Rys. 6.3). Będziemy mieć również opcję wpłacenia gotówki na konto.



Rysunek 6.2. Formularz dodania konta bankowego



Rysunek 6.3. Informacje o koncie użytkownika

7. Podsumowanie i wnioski

Celem niniejszej pracy było zaprojektowanie i implementacja aplikacji webowej wspomagającej zarządzanie finansami. Aplikacja powstała w oparciu o architekturę klient-serwer i posiada takie funkcjonalności jak:

- Rejestrowanie i logowanie użytkownika,
- Tworzenie różnego rodzaju transakcji,
- Sprawdzanie szczegółów profilu lub kont,
- Analiza danych finansowych przy pomocy wykresów,
- Komunikacja między frontendem, a backendem przy pomocy REST API,
- Przechowywanie danych w bazie nierelacyjnej MongoDB.

Aplikacja umożliwia elastyczne zarządzanie danymi, dzięki bazie MongoDB. Jej uniwersalny interfejs jest prosty w obsłudze co ułatwia poruszanie się po stronie dla każdej grupy wiekowej. Projekt pozwoli użytkownikowi w łatwiejszy sposób robić takie rzeczy jak:

- Zarządzać własnym budżetem domowym,
- Śledzić swoje wydatki,
- Planować przyszłe płatności,
- Wykonywać kroki w celu inwestycji środków.

Wnioski

Nowoczesne technologie webowe pozwalają na tworzenie skalowalnych i łatwych w rozwoju aplikacji. Architektura klient-serwer sprawdziła się w rozdzieleniu warstwy frontendu od backendu.

W przyszłości aplikację będzie można rozszerzyć o:

- Funkcje prognozowania wydatków z wykorzystaniem algorytmów uczenia maszynowego,
- Możliwość eksportu danych do formatu PDF lub CSV.

Projekt umożliwił rozwinięcie umiejętności analitycznych oraz technicznych związanych z tworzeniem aplikacji webowych.

Co osiągnąłem

Podczas realizacji projektu osiągnięto wiele rezultatów. Stworzono kompletną aplikację webową umożliwiającą zarządzanie budżetem domowym, w której połączono wiele nowoczesnych rozwiązań technologicznych. Do najważniejszych osiągnięć należą:

- Wykorzystanie architektury klient-serwer,
- Interfejs oparty o technologię React,
- Opracowanie logiki biznesowej przy pomocy Spring Boot,
- Uzyskanie płynnie działającego przepływu danych w bazie MongoDB,
- Wdrożenie środowiska do uruchamiania w oparciu o Docker,
- Opracowanie testów funkcjonalnych przy użyciu narzędzia Postman,
- Umożliwienie dalszej rozbudowy systemu.

Realizacja powyższych elementów pozwoliła na stworzenie w pełni działającego systemu, który spełnia założenia projektowe.

W trakcie tworzenia projektu autor rozwinął swoje umiejętności w zakresie programowania w językach TypeScript i Java, projektowania aplikacji webowych oraz pracy z bazami typu NoSQL. Zdobyte doświadczenie obejmuje również prace z narzędziami takimi jak Git, Docker oraz Postman.

Streszczenie

zawartość...

Abstract

Content...

Częstochowa, dn. .....

Imię i nazwisko: .....
Nr albumu: .....
Kierunek: .....
Wydział: .....

Oświadczenie autora pracy dyplomowej*

Oświadczam pod rygorem odpowiedzialności karnej, że złożona przeze mnie praca dyplomowa pt. .....
.....
jest moim samodzielnym opracowaniem i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Jednocześnie oświadczam, że moja praca (w całości ani we fragmentach) nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w Politechnice Częstochowskiej.

Wyrażam/nie wyrażam** zgody/zgody na nieodpłatne wykorzystanie przez Politechnikę Częstochowską całości lub fragmentów wyżej wymienionej pracy w publikacjach Politechniki Częstochowskiej.

.....
czytelny podpis studenta

---

*W przypadku zbiorowej pracy dyplomowej dołącza się Oświadczenie każdego ze współautorów.

**Niepotrzebne skreślić.

Bibliografia

[1] Apache. Apache maven documentation. https://maven.apache.org/guides/index.html. stan na dzień: 14.10.2025.

[2] Academy Bank. *Banking Trends in 2025 & Beyond: Budgeting Apps for Financial Success.* https://www.academybank.com/article/banking-trends-in-2025-and-beyond-budgeting-apps-for-financial-success. stan na dzień: 13.10.2025.

[3] Elad B., Kinder K. *Fintech Adoption Statistics 2025: What's Driving It (And What's Holding It Back).* https://coinlaw.io/fintech-adoption-statistics/, 2025. stan na dzień: 14.10.2025.

[4] markets and markets. *Fintech as a Service Market.* https://www.marketsandmarkets.com/Market-Reports/fintech-as-a-service-market-9388805.html, 2025. stan na dzień: 14.10.2025.

[5] Azure Microsoft. Co to jest java? https://azure.microsoft.com/pl-pl/resources/cloud-computing-dictionary/what-is-java-programming-language#:~:text=Jak%20dzia%C5%82a%20j%C4%99zyk%20Java?%20Kod%20Java%20jest,i%20udost%C4%99pnia%20%C5%9Brodowisko%20uruchomieniowe%20dla%20aplikacji%20Java. stan na dzień: 14.10.2025.

[6] Abgilas D. Aishwarya D. Lipsa D. Siddharth M. Supriyo Ghose Nitin B. Robo-advisory: An investor's perception. *International Journal of Psychosocial Rehabilitation*, 23(3), 2019.

[7] Oracle. *JDK 21 Documentation.* https://docs.oracle.com/en/java/javase/21/. stan na dzień: 14.10.2025.

[8] Majkowski A., Rak R. *Systemy pomiarowe*, pages 35–37. Ośrodek kształcenia na Odległość OKNO, 2018.

[9] VMware Tanzu. Spring boot documentation. https://docs.spring.io/spring-boot/documentation.html. stan na dzień: 14.10.2025.

[10] w Perez K. (red.) Waliszewski K., Warchlewska A. *Innowacje finansowe w gospodarce 4.0*, pages 120–140. UEP, 2021.

[11] Wikipedia. Kod odpowiedzi http. https://pl.wikipedia.org/wiki/Kod_odpowiedzi_HTTP. stan na dzień: 14.10.2025.

[12] Łuczak K. Identyfikowanie wymagań użytkowników jako podstawa projektowania user experience. *Uniwersytet Ekonomiczny we wrocławiu*.

Spis rysunków

Spis tabel

Spis listingów