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

Wstęp

Aplikacje webowe są coraz bardziej powszechnie używane. W artykule [2] zauważono, że 83.1% osób śledzi swoje wydatki, z czego 45.3% używa do tego narzędzi cyfrowych. Jednym z rozwiązań jest korzystanie z aplikacji budżetowych. W badaniach użycie tych narzędzi zadeklarowało 20.9% ludzi. 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.

W publikacji [4] dowiadujemy się, że aplikacje te wpływają na wzrost wiedzy ekonomiczno-finansowej społeczeństwa, zwiększają chęć do planowania i kontroli budżetem domowym. Usługi te są dostępne w każdym miejscu i czasie 24/7. Przeprowadzone badanie na grupie $N = 301$ polaków wykazało, że 288 korzysta z aplikacji do wspomagania budżetem, a tylko 13 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%).

Według artykułu [3] grupą, która najczęściej używa tego typu aplikacji znajduje się w przedziale wiekowym od 27 do 42 lat. Około 91% osób z tej grupy deklaruje użycie tego typu aplikacji. Kolejna grupa to osoby w wieku od 43-58 lat (80%). W przedziale od 18 do 26 roku życia jest to 68%. Według badań aktualna wielkość rynku wynosi około 441.47$ miliardów i przy aktualnym tempie wzrostu może urosnąć nawet do 906$ miliardów do 2029 roku.

Aplikacje te najczęściej cechują się interfejsem przyjaznym dla użytkownika, kategoryzacją i śledzeniem wydatków, możliwością przypominania o rachunkach oraz zapewniają ochronę informacji personalnych. Ich rozwój jest dynamiczny, na co wskazuje rosnąca liczba użytkowników.

Cel pracy

Celem pracy jest zbudowanie aplikacji do zarządzania budżetem użytkownika końcowego. Aplikacja ma za zadanie wspomagać użytkowników w kontroli wydatków, planowaniu budżetu oraz analizie danych finansowych w prosty sposób.

Zakres pracy

Zakres niniejszej pracy obejmuje zaprojektowanie i implementację aplikacji webowej umożliwiającej zarządzanie budżetem domowym. Aplikacja została oparta na architekturze klient-serwer.

Warstwa serwerowa została zaimplementowana w języku Java z wykorzystaniem frameworka Spring Boot. Wykorzystana została 21 wersja JAVY. Zapewnia obsługę logiki biznesowej, komunikację z bazą danych oraz interfejs API w technologii REST. 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.

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

Warstwa frontendowa została zrealizowana w oparciu o React + Vite. Język, wykorzystany po stronie frontendu to TypeScript, który jest językiem skupiającym się silnie na typach danych. Pozwala to na dokładne określenie wyświetlanych na stronie informacji. Elementy, z których zbudowana jest strona 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.

W ramach projektu utworzone są funkcje takie jak:

- Rejestracja - Pozwala na utworzenie nowego konta przez użytkownika
- Logowanie - Umożliwia dostęp do konta oraz funkcji strony
- Tworzenie transakcji - Zapewnia możliwość dokonywania przelewów, wpłat i wypłat z konta

- Wizualizacja danych na wykresie

Do sprawdzenia poprawności działania tych i innych funkcji wykorzystuje się narzędzie Postman, które pozwala wysyłać zapytania, sprawdzać restpointy 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.

Aplikacja wykorzystuje tez konteneryzacja, 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.

Struktura pracy (Budowa pracy)

Streszczenie wszystkiego (1 akapit) - bez wstępu

W pierwszym rozdziale zostały opisany wszystkie technologię oraz narzędzia użyte do uzyskania działającej aplikacji. Znajdują się w niej najważniejsze informacje o każdej z nich oraz pojedyncze przykłady ich zastosowań.

Drugi rozdział skupia się na wymaganiach postawionych projektowi. Przedstawia również diagramy na których możemy zobaczyć w jaki sposób mają działać funkcję.

Trzeci rozdział uwzględnia informacje o tym w jaki sposób został utworzony cały projekt. Tłumaczy w jak działają poszczególne technologie, narzędzia i opisuje całą aplikację.

W czwartym rozdziale znajdują się informacje o tym jak użytkownik powinien korzystać ze strony. Jest to instrukcja dla użytkownika, aby mógł zrozumieć, gdzie należy szukać odpowiednich elementów strony. Przedstawia krótkie opisy wraz z obrazkami.

Piąty rozdział zawiera podsumowanie całej pracy. Zostały w nim wyciągnięte wnioski oraz moje osiągnięcia.

1. Technologie, z których korzysta aplikacja

1.1. Java

Java jest obiektowym wieloplatformowym językiem programowania. Jest on najpopularniejszym językiem dla deweloperów oprogramowania [1]. Korzysta z maszyn wirtualnych 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.

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. Dzięki niemu aplikacja frontendowa może komunikować się z backendem. 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 (Aspect-Oriented Programming).

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.

1.2. 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.

1.3. 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. 1.1). Połączenie bazy w aplikacji Java odbywa się za pomocą modułu Spring Data MongoDB. To właśnie ten moduł odpowiada za możliwość tworzenia repozytorium.

Listing 1.1. 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 1.2. Przykladowy dokument z kolekcji

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

1.4. Postman

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, tworzenia własnych nagłówków lub przesyłanego ciała do funkcji.

1.5. Docker

Docker pozwala na konteneryzacje aplikacji. Jego głównym elementem jest silnik (Docker Engine), który działa w formie usługi. Umożliwia on budowanie, uruchamianie i zarządzanie kontenerami. W środowisku Dockera możemy tworzyć wirtualne obrazy, które posłużą nam do wykonania jakiegoś zadania. Na bazie obrazów możemy tworzyć kontenery. Plusem takiego rozwiązania jest możliwość określenia wersji wszystkich użytych technologii, jak również perspektywa odpalenia naszego programu na dowolnym urządzeniu (przenośność), które posiada zainstalowany docker. Projekty bardziej złożone (korzystające z wielu usług) możemy połączyć za pomocą pliku compose.yml, w którym opisujemy zależności pomiędzy danymi kontenerami np.: określamy sieć w jakiej mają się znajdować, woluminy z jakich mają korzystać lub elementy od których są zależne.

1.6. Git

System kontroli wersji to jedna z najważniejszych rzeczy podczas pisania dowolnej aplikacji. Najczęściej używanym rozwiązaniem 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.

2. Wymagania i diagramy

2.1. wymagania funkcjonalne

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.

Tabela 2.1. Wymagania funkcjonalne systemu do zarządzania budżetem domowym

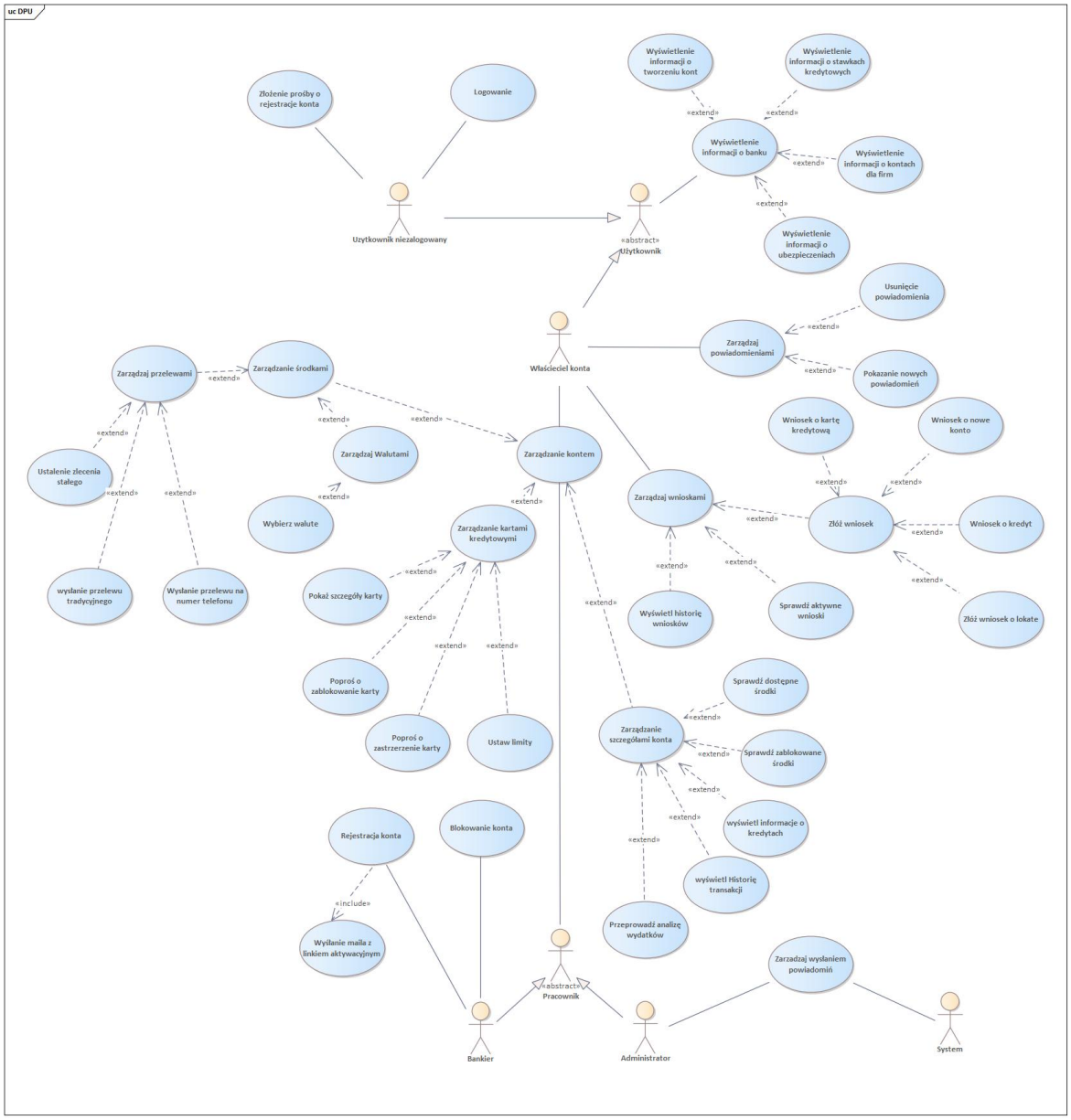
2.2. wymagania niefunkcjonalne

Nr	Wymaganie niefunkcjonalne	Opis
1.	Dostępność	aplikacja ma być dostępna przez 95% czasu działania serwera
2.	Wydajność	System reaguje na żądanie w czasie krótszym niż 30 sekund
3.	Bezpieczeństwo	Hasła muszą być przechowywane w formie zaszyfrowanej.
4.	Wdrożenie	aplikacja będzie korzystać z Java 21, Spring Boot, React, MongoDB 8.0

Tabela 2.2. Wymagania niefunkcjonalne systemu do zarządzania budżetem domowym

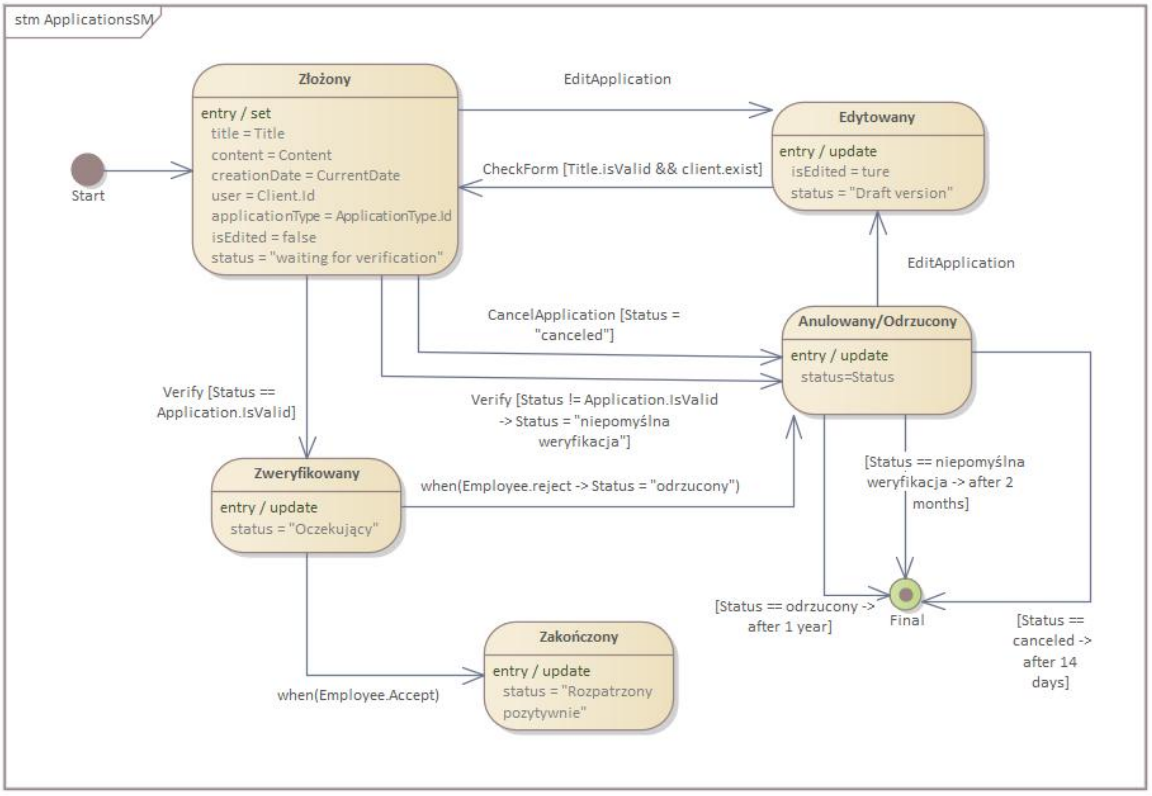
2.3. Diagramy przypadków użycia

<—Placeholder—>



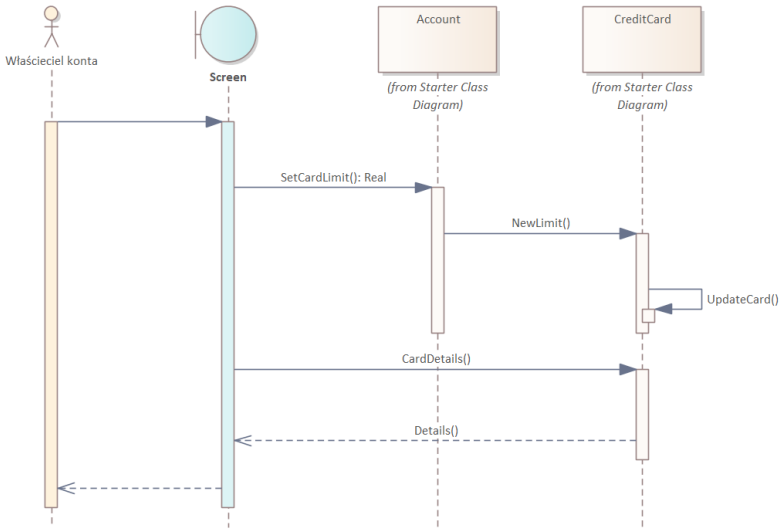
Rysunek 2.1. Diagram przypadków użycia

2.4. Diagram stanów

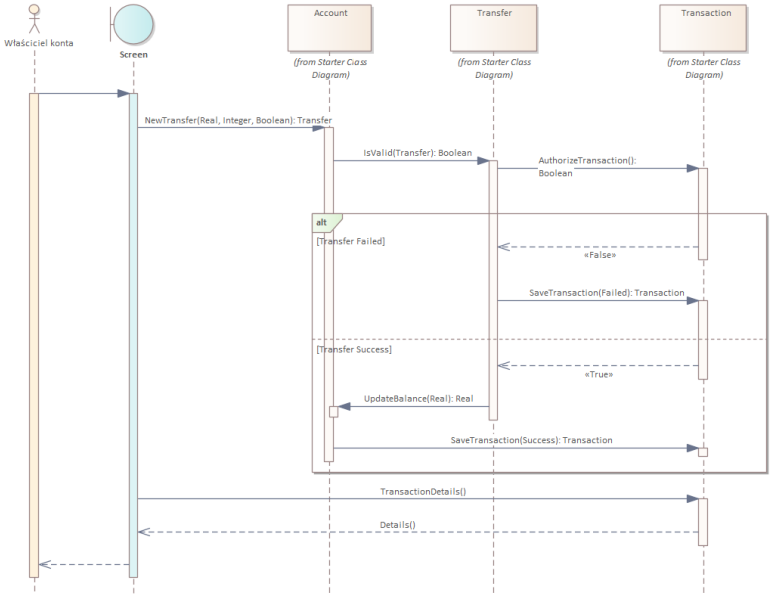


Rysunek 2.2. Diagram stanów składania wniosku

2.5. Diagramy sekwencji



Rysunek 2.3. Diagram sekwencji Limit



Rysunek 2.4. Diagram sekwencji Przelew

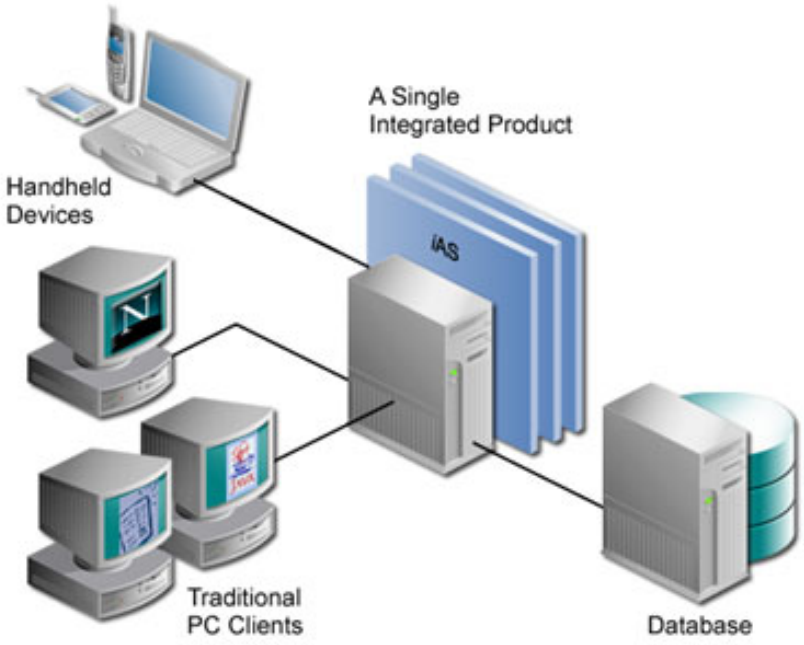
3. Implementacja systemu

3.1. Architektura systemu

Aplikacja została zaprojektowana w oparciu o architekturę klient-serwer. Składa się z takich komponentów jak:

- Frontend (Klient) - Interfejs użytkownika (UI), umożliwia użytkownikowi interakcje z systemem.
- Backend (serwer) - Warstwa logiki biznesowej, obsługuje żądania HTTP od klienta i komunikuje się z bazą.
- Baza danych (MongoDB) - Służy do przechowywania danych finansowych.

Komunikacja odbywa się przy pomocy protokołu HTTP/REST, a dane są w formacje JSON.



Rysunek 3.1. Tymczasowe zdjęcie architektury Klient-Serwer

3.2. 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 3.1. Przykładowe Endpointy

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

Ich pełny opis znajduje się dalej w listingach (3.4, 3.5, 3.6, 3.7)

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 3.2. 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 3.3. 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 3.4. 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 3.5. Lista użytkowników

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

Usuwanie użytkownika

Listing 3.6. Usuwanie użytkownika

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

Edycja użytkownika

Listing 3.7. 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 3.8. Przykładowy test

```
@Test
public void createTransferHandlesNegativeTransferAmount() throws Exception {
  this.mockMvc.perform(MockMvcRequestBuilders.post("/Transaction/create/
      transfer")
  .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 }
```

3.3. 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 3.9. Funkcja wyszukująca wszystkie konta użytkownika

```
1 export const fetchAccounts= async (): Promise<Accounts[]> =>{
2   const id = await fetchUserId();
3   try {
4     const response = await axios.get(`http://localhost:8080/Account/get/${id.
          id}`, {
5       withCredentials: true,
6       headers: {
7         '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 3.10. 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();
}, []);
```

3.4. 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 3.11. Przykład przesyłanych danych między frontendem, a backendem

```
[{
  "fromAccountNumber": "36204576739585995585483 79",
  "toAccountNumber": "12345678901222345610123 35",
  "amount": 50.0,
  "description": "test",
  "transactionId": "68dfb1d240d00a2221f78809"
},
{
  "fromAccountNumber": "36204576739585995585483 79",
  "toAccountNumber": "12345678901222345610123 35",
  "amount": 150.0,
  "description": "test",
  "transactionId": "68dfb1d740d00a2221f7880b"
}]
```

3.5. 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 3.12. Budowanie obrazu - Dockerfile

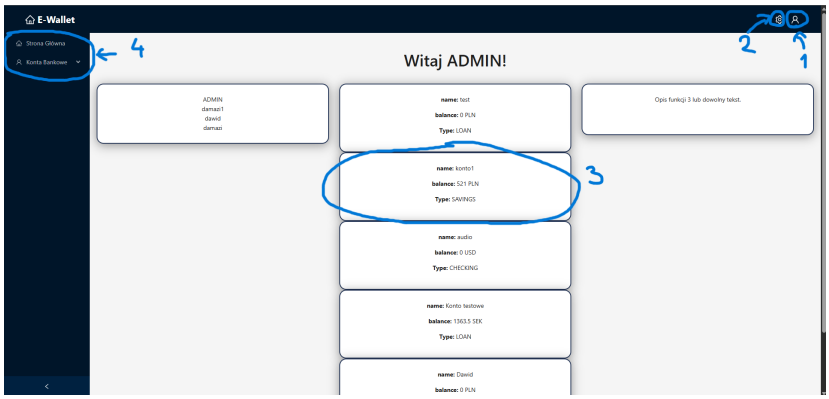
```
1 FROM maven:3.9.10-eclipse-temurin-21
2 WORKDIR /app
3 COPY pom.xml .
4 COPY src ./src
5 EXPOSE 8080
6 CMD ["mvn", "spring-boot:run"]
```

3.6. Przykładowe fragmenty kodu i funkcje

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

4. Prezentacja aplikacji

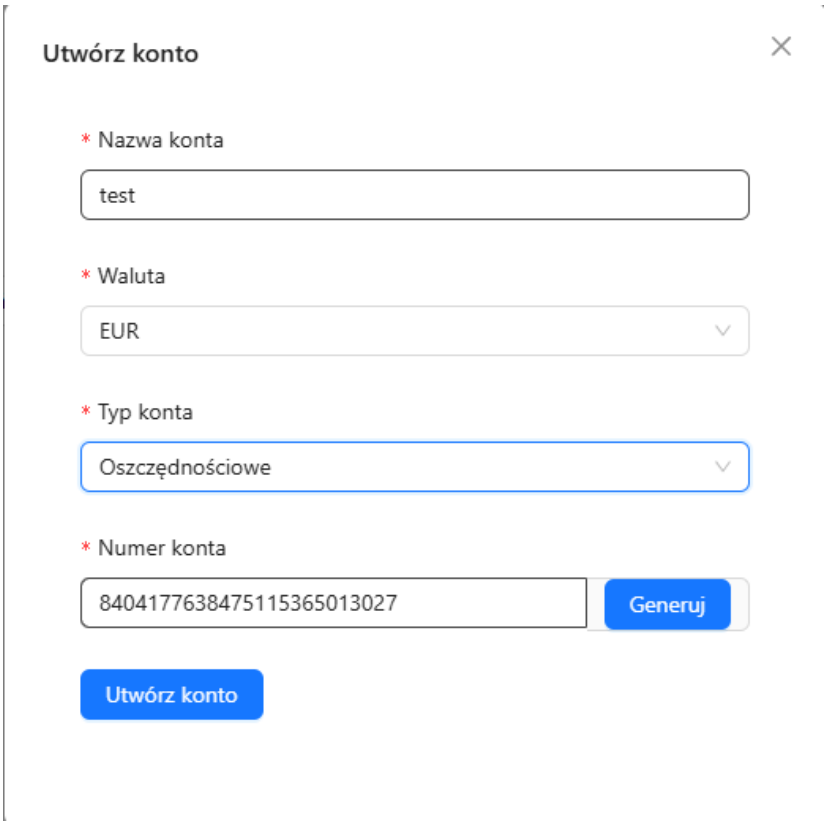
Aplikacja po uruchomieniu wyświetla domyślną stronę (Home), na której wyświetlają się informację. Pierwszą rzeczą jaką należy zrobić po uruchomieniu jest zalogowanie się lub rejestracja.



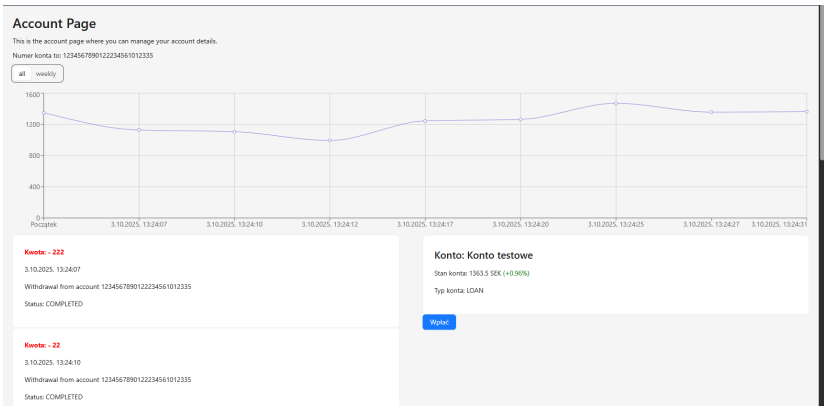
Rysunek 4.1. Strona startowa po zalogowaniu

- 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.4.2), 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. 4.3). Będziemy mieć również opcję wpłacenia gotówki na konto.



Rysunek 4.2. Formularz dodania konta bankowego



Rysunek 4.3. Informacje o koncie użytkownika

5. 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.

Bibliografia

[1] Co to jest java? *Azure Microsoft*, ??

[2] ?? Banking trends in 2025 & beyond: Budgeting apps for financial success. *Academy Bank*, 2025.

[3] Kathleen Kinder Barry Elad. Fintech adoption statistics 2025: What's driving it (and what's holding it back). *CoinLaw*, 2025.

[4] Anna Warchlewska Krzysztof Waliszewski. *Innowacje finansowe w gospodarce 4.0.* researchgate, 2021.

Spis rysunków

Spis tabel

Spis listingów