****

**WYŻSZA SZKOŁA INFORMATYKI**

**I ZARZĄDZANIA**

**Z SIEDZIBĄ W RZESZOWIE**

**Inżynieria oprogramowania**

Bank Management

Prowadzący: Autorzy(**G13**):

Mgr Ewa Żesławska Kacper Żmijak 60095

Kamil Wójcik 60079

Kierunek: 6IID-ZP/2018, grupa SL03

Rzeszów 2021

Contents

[1.Opis świata rzeczywistego 3](#_Toc77522774)

[1.1 Opis zasobów ludzkich 3](#_Toc77522775)

[1.2 Przepisy i strategia firmy 3](#_Toc77522776)

[1.3 Dane techniczne 3](#_Toc77522777)

[1.4 Wymagania funkcjonalne i niefunkcjonalne 3](#_Toc77522778)

[1.6 Wymagania funkcjonalne(wraz z minimalnym zestawem atrybutów) 3](#_Toc77522779)

[1*.6 Wymagania niefunkcjonalne* 3](#_Toc77522780)

[1.7 Requirement Diagram 4](#_Toc77522781)

[2. Specyfikacja przypadków użycia za pomocą diagramu przypadków użycia 5](#_Toc77522782)

[2.1 Przykład definicji aktorów 6](#_Toc77522783)

[2.2 Przykłady scenariuszy definicji scenariuszy przypadków użycia (PU – przypadek użycia) 7](#_Toc77522784)

[3. Diagram Czynności 9](#_Toc77522785)

[4. Diagram klas 11](#_Toc77522786)

[5. Diagram komponentów 12](#_Toc77522787)

[6. Diagram sekwencji 13](#_Toc77522788)

[7. Diagram stanów 15](#_Toc77522789)

[8. Harmonogram realizacji projektu(Diagram Ganta) 16](#_Toc77522790)

[9. Testy integracyjne 17](#_Toc77522791)

[Opis techniczny 19](#_Toc77522792)

[API 19](#_Toc77522793)

[Front-end 19](#_Toc77522794)

[Prezentacja warstwy użytkowej 21](#_Toc77522795)

[Rekordy z API 26](#_Toc77522796)

# 1.Opis świata rzeczywistego

## 1.1 Opis zasobów ludzkich

**Pracownik Banku** odpowiedzialny jest za transfer środków pieniężnych, które klient w formie papierowej chce wpłacić na swoje konto.

Każdy **Klient** posiada numer ID, oraz dane osobowe. Poprzez formularz rejestracyjny będzie musiał założyć swoje konto w banku, po czym zostanie mu przypisany unikalny **numer konta**. Użytkownik po zalogowaniu do serwisu bankowego, będzie mógł wykonywać transakcje pieniężne na inne, istniejące konta. Każda transakcja pieniężna będzie miała numer ID, datę, ID wykonawcy, ID odbiorcy, tytuł transakcji oraz wartość.

## 1.2 Przepisy i strategia firmy

Aplikacja pozwala klientom banku dokonywać transakcji na istniejące konta w bazie danych, w przeciwnym wypadku klient otrzyma odpowiedni komunikat. Klient nie będzie mógł być na saldzie ujemnym.

## 1.3 Dane techniczne

Zarządzanie swoimi funduszami powinno być dostępne poprzez stronę internetową. Będzie to aplikacja web’owa, napisana w angularze przy pomocy .Net Core, C#, baza danych będzie w MSSQL, integracja z bazą danych za pomocą EFCore.

## 1.4 Wymagania funkcjonalne i niefunkcjonalne

### 1.6 Wymagania funkcjonalne(wraz z minimalnym zestawem atrybutów)

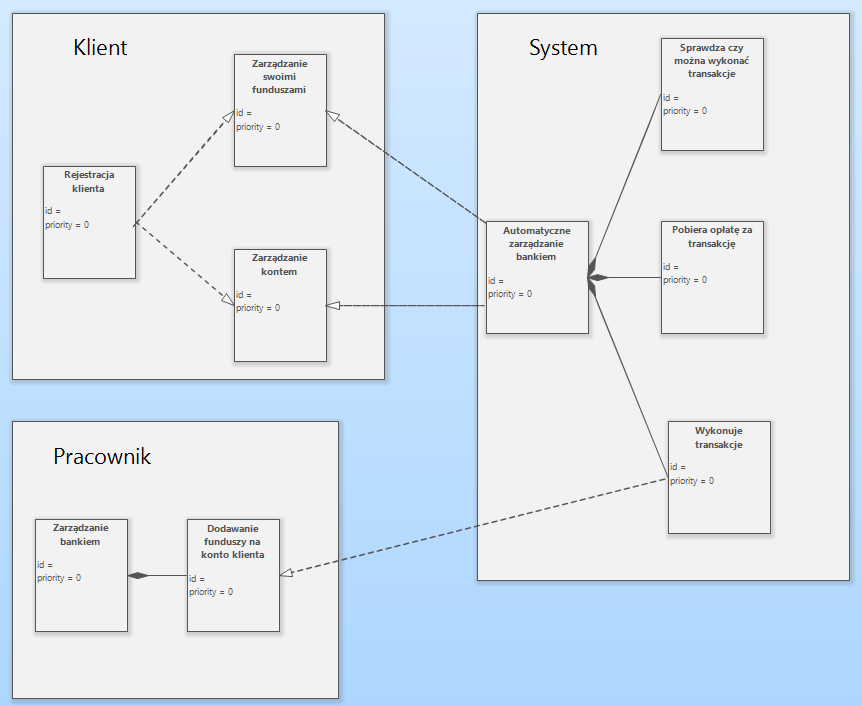
1. Utworzenie konta klienta(należy podać atrybuty)
2. Logowanie (Klient musi podać poprawne dane identyfikacyjne)
3. Dodawanie funduszy na konto klienta (Pracownik bank)
4. Zarządzanie dostępnymi funduszami, klient będzie miał możliwość transferu określonej kwoty na inne istniejące konto ( należy podać atrybuty: numer konta odbiorcy, tytuł, kwotę)
5. Dane osobowe( użytkownik może edytować swoje podstawowe dane osobowe)

### 1.6 Wymagania niefunkcjonalne

1. Aplikacja wymaga dostępu do Internetu i przeglądarki internetowej.
2. Aplikacja jest przeznaczona dla wielu klientów.
3. Niezawodność( aplikacja powinna działać sprawnie przy dużej liczbie aktywnych klientów)
4. Intuicyjność (aplikacja powinna być prosta i przejrzysta w użytkowaniu)

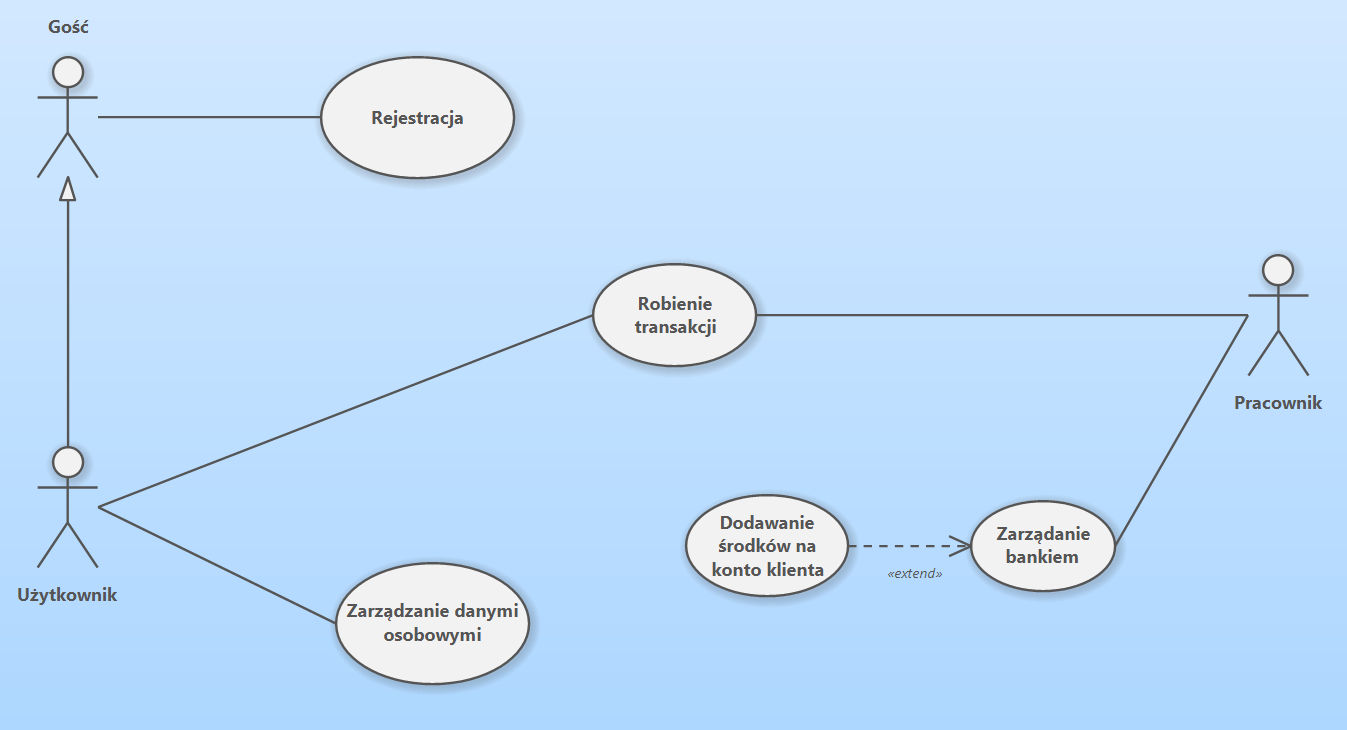
## 1.7 Requirement Diagram

W oparciu o w/w opis świata rzeczywistego oraz zdefiniowane wymagania funkcjonalne i funkcjonalne na Rys.1 przedstawiono diagram wymagań dla opisywanego systemu zarządzania kontem bankowym.



Rysunek 1 - Requirement Diagram

# 2. Specyfikacja przypadków użycia za pomocą diagramu przypadków użycia



Rysunek 2 – Case diagram

## 2.1 Przykład definicji aktorów

|  |  |  |
| --- | --- | --- |
| **AKTOR** | **OPIS** | **PRZYPADKI UŻYCIA** |
| Gość | Gość może jedynie zarejestrować konto na stronie banku. | - PU Założenie konta/Rejestracja |
| Użytkownik | Użytkownik może dodatkowo wykonywać transakcje, oraz zarządzać danymi osobowymi. | - PU Zarządzanie funduszami  - PU Zarządzanie kontem |
| Pracownik | Pracownik dodaje środki na konto Klienta | -PU Dodawanie środków na konto Klienta |

## 2.2 Przykłady scenariuszy definicji scenariuszy przypadków użycia (PU – przypadek użycia)

**PU Założenie konta**

OPIS

CEL: Rejestracja nowego klienta

* WS(warunki wstępne): Klient nie posiada konta bankowego. Klient przechodzi do formularza rejestracji.
* WK (warunki końcowe): Konto zostaje utworzone. Klient może zalogować się na swoje konto.

PRZEBIEG:

1. Należy podać dane użytkownika: imię, nazwisko, hasło.
2. System sprawdza poprawność danych.
3. System rejestruje konto klienta.
4. Klient zostaje poinformowany na przeglądarce, o swoim unikalnym numerze klienta.

**PU Zarządzanie funduszami(transakcje)**

OPIS

CEL: Robienie transakcji

* WS (warunki wstępne): Klient posiada na koncie tyle funduszy, ile chce wysłać.
* WK (warunki końcowe): Klient wysyła określoną część funduszy na inne konto, istniejące w bazie danych banku.

PRZEBIEG:

1. Klient(nadawca) musi podać numer konta odbiorcy, tytuł, kwotę.
2. System sprawdza poprawność operacji.
3. System realizuje transakcję.
4. Klient zostaje poinformowany ile funduszy zostało mu do rozdysponowania.

**PU Zarządzanie kontem**

OPIS

CEL: Zarządzanie swoimi danymi osobowymi, możliwość ich edycji.

* WS (warunki wstępne): Klient posiada konto bankowe.
* WK (warunki końcowe): Klient może edytować swoje dane.

PRZEBIEG:

1. Klient może zmienić dane takie jak: imię, nazwisko, login(ID).
2. System realizuje operacje.
3. Klient zostaje poinformowany o aktualizacji danych.

**PU Dodawanie środków na konto klienta**

OPIS

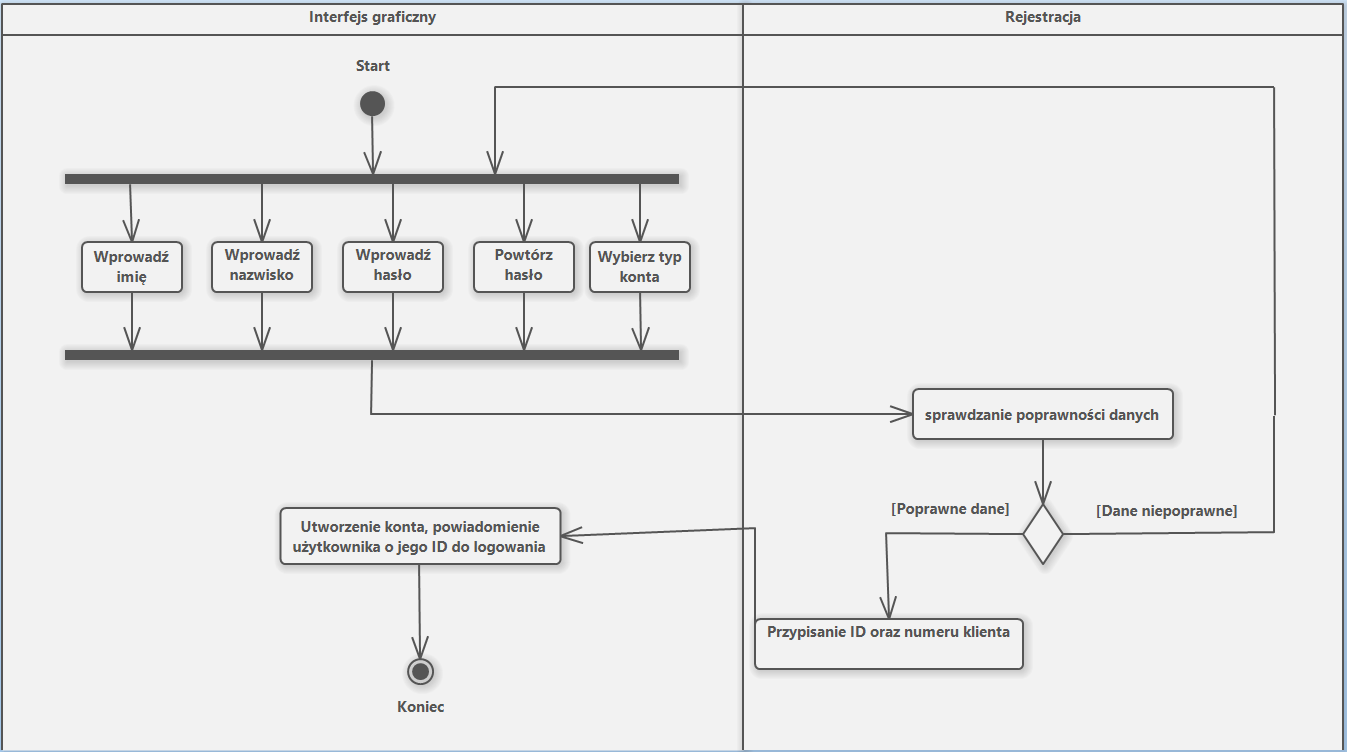
CEL: Dodanie środków płatniczych na konto klienta.

* WS (warunki wstępne) : Klient przychodzi do placówki bankowej wraz z funduszami w formie papierowej.
* WK (warunki końcowe) : Środki zostają dodane do konta Klienta.

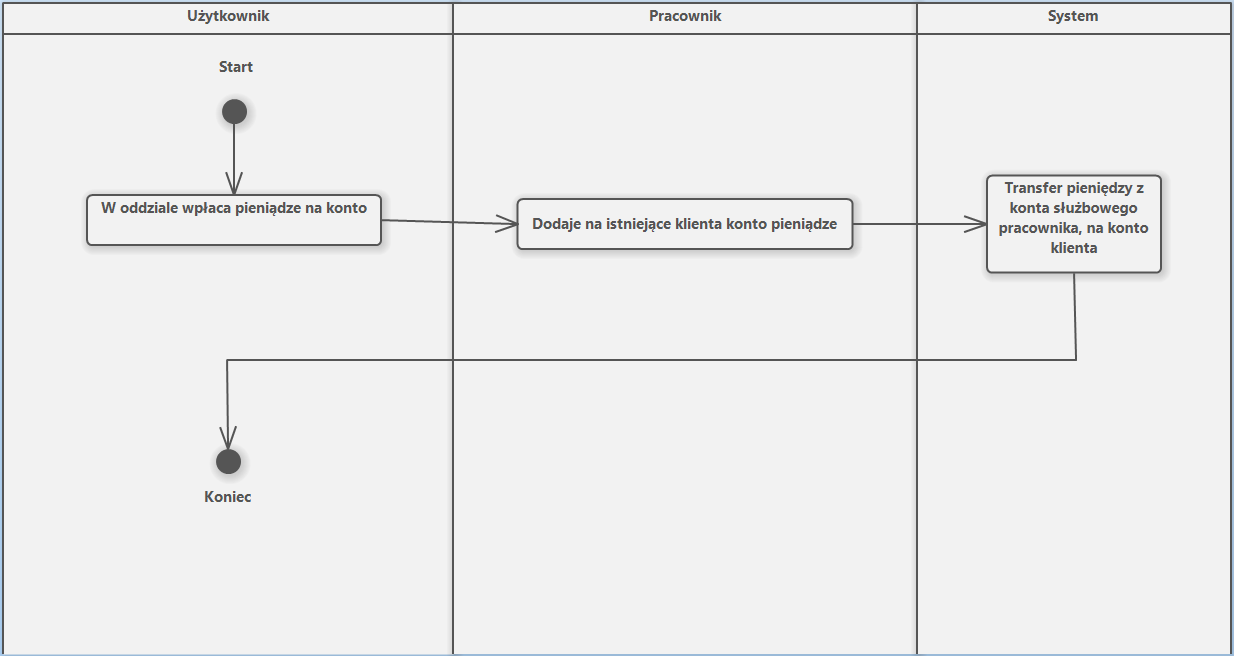
PRZEBIEG:

* Pracownik wyszukuje klienta.
* Wpisuje kwotę, ile ma być zasilone konto klienta.
* System realizuje operacje.

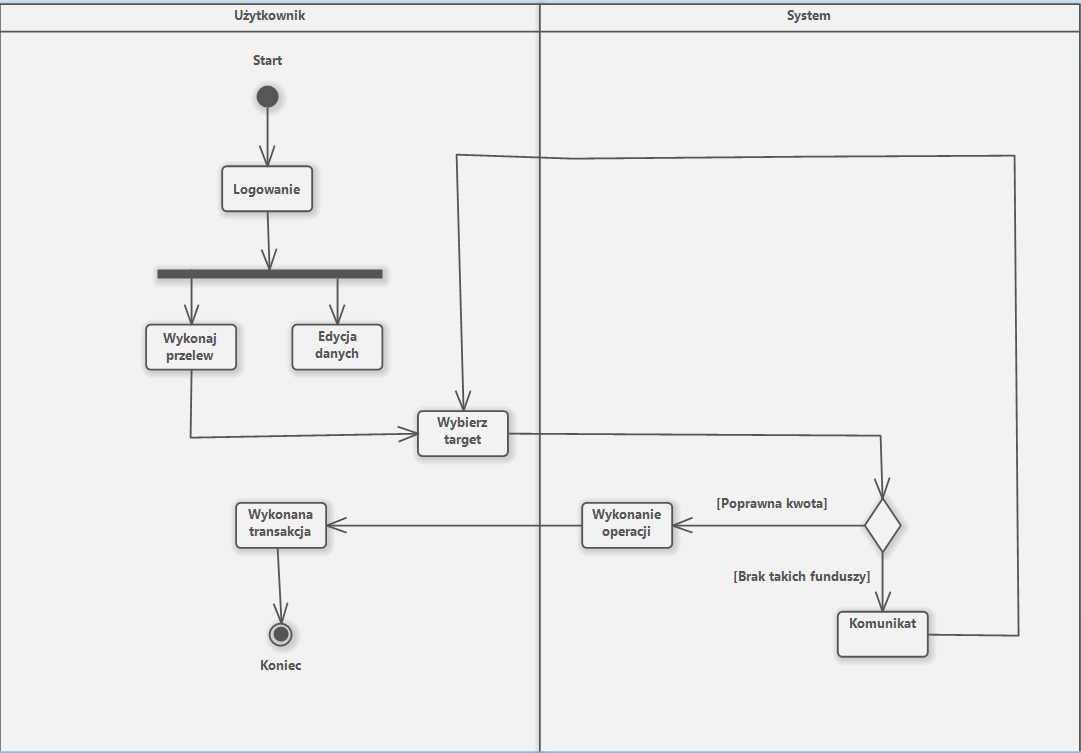
# Diagram Czynności



Rysunek 3 Activity Diagram - Rejestracja

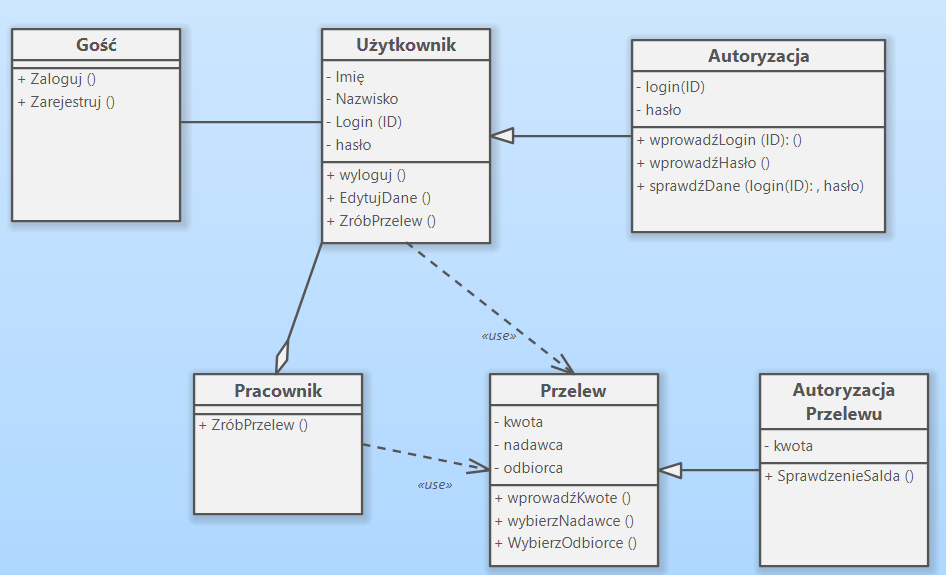


Rysunek 4 Activity Diagram - Dodawanie funduszy na konto



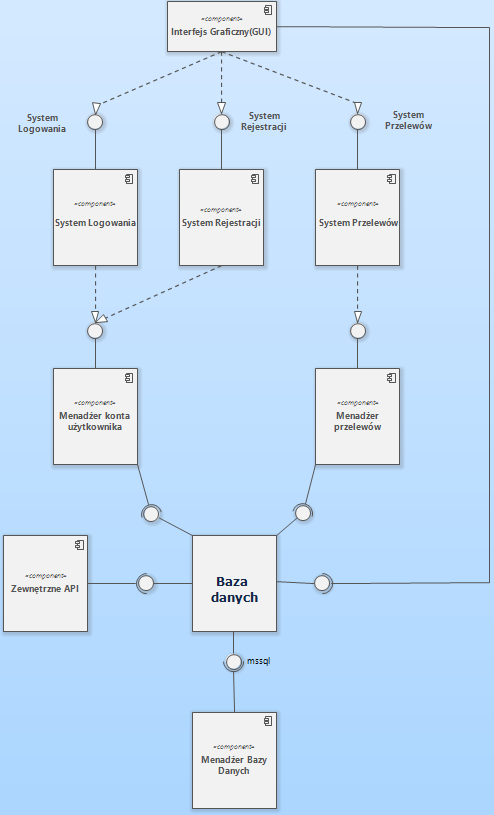
Rysunek 5 Activity Diagram - wykonanie transakcji

# Diagram klas



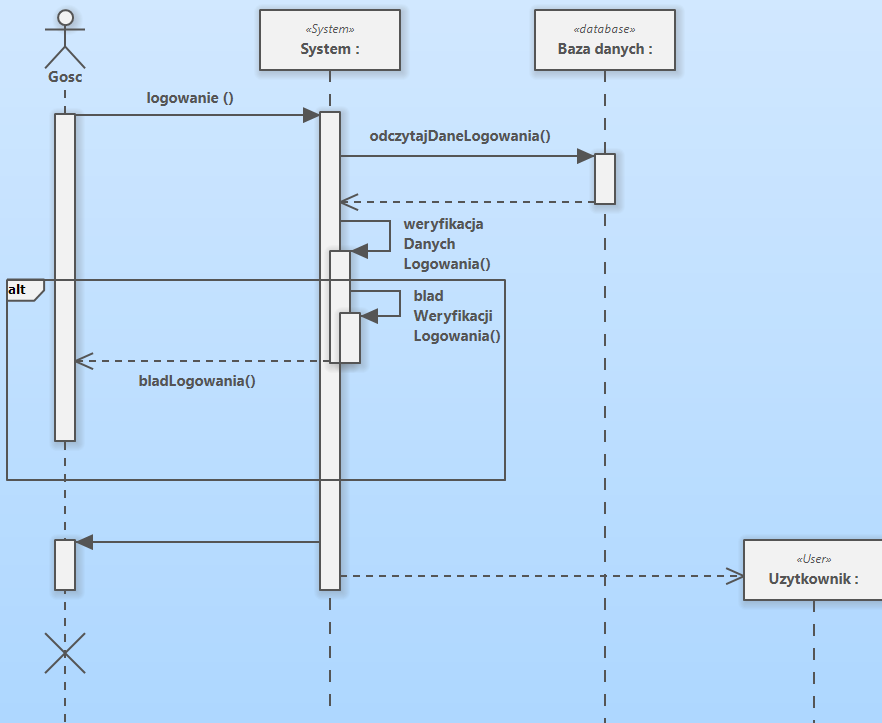
Rysunek 6 Diagram Klas

# Diagram komponentów

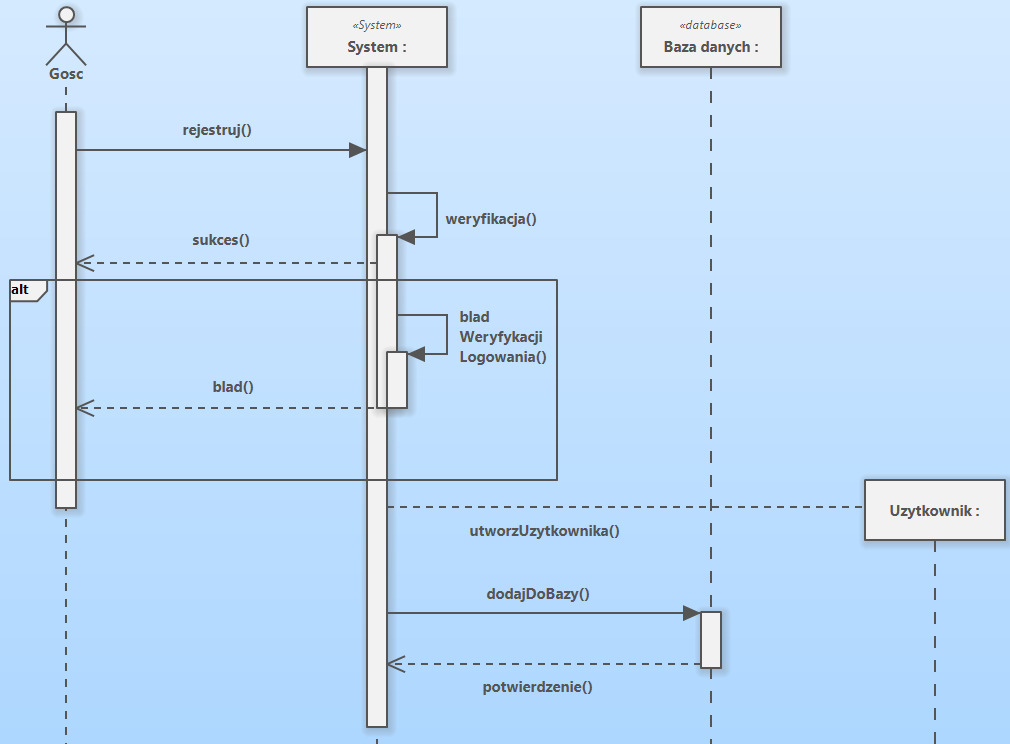


Rysunek 7 Diagram komponentów

# Diagram sekwencji

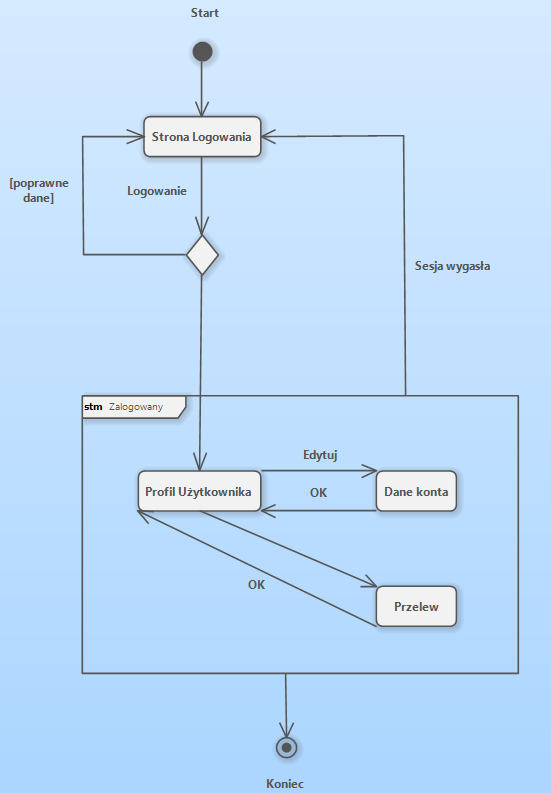


Rysunek 8 Diagram sekwencji - Logowanie



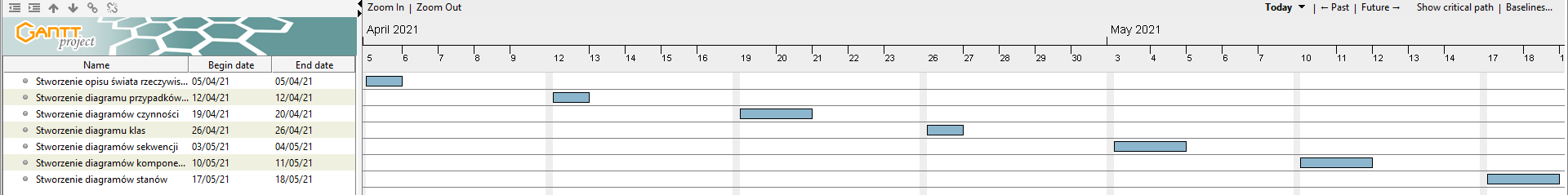
Rysunek 9 Diagram sekwencji - Rejestracja

# Diagram stanów

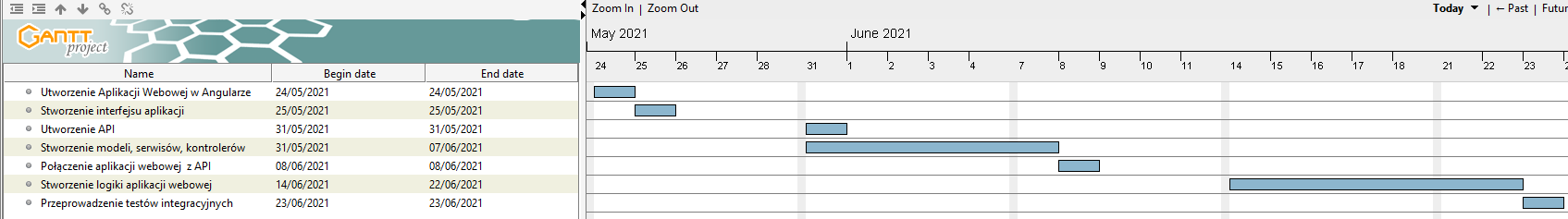


Rysunek 10 - Diagram stanów - panel użytkownika

# Harmonogram realizacji projektu(Diagram Ganta)



Rysunek 11 Diagram Gantta – Planowanie



Rysunek 12 Diagram Gantta - Implementacja

# Testy integracyjne

Ponieważ API testuje się integracyjnie.

Test który sprawdza, czy można dokonać transakcji, nie mając środków na koncie:

public class TransactionControllerTests : IClassFixture<BmWebApplicationFactory<Startup>>

{

private readonly BmWebApplicationFactory<Startup> \_factory;

private readonly HttpClient \_httpClient;

public TransactionControllerTests(BmWebApplicationFactory<Startup> factory)

{

\_factory = factory;

\_httpClient = factory.CreateDefaultClient();

}

[Fact]

public async Task Transaction\_with\_insufficient\_funds\_Should\_be\_declined()

{

string transactionUrl = "api/transactions/transfer";

// Arrange

Account sender = new Account

{

Balance = 0,

Number = "1111111",

UserId = 60095,

};

Account receiver = new Account

{

Balance = 0,

Number = "2222222",

UserId = 60095,

};

using (var scope = \_factory.Services.CreateScope())

{

var context = scope.ServiceProvider.GetRequiredService<BmDbContext>();

await context.AddRangeAsync(sender, receiver);

await context.SaveChangesAsync();

}

TransferRequest transferRequest = new TransferRequest

{

ReceiverNumber = "2222222",

Amount = 100,

SenderNumber = "1111111",

Title = "Test",

};

// Act

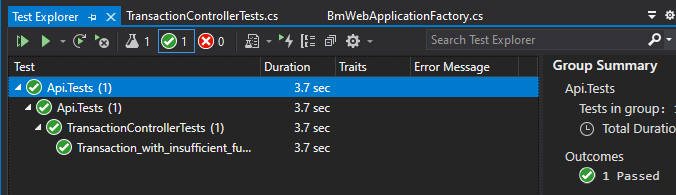
var response = await \_httpClient.PatchAsync(transactionUrl, new StringContent(JsonConvert.SerializeObject(transferRequest), Encoding.UTF8, "application/json"));

// Assert

response.StatusCode.Should().Be(HttpStatusCode.BadRequest);

}

}



# Opis techniczny

## API

Do projektu zostało zrobione API w języku C#, w technologii ASP.NET. Które odpowiada za:

* Rejestrację
* Logowanie
* Walidację danych
* Transakcje
* Edycję danych

Baza danych została utworzona za pomocą podejścia Code-First, które polega na odwzorowywaniu tabel i encji w bazie danych, za pomocą modeli w projekcie. W projekcie do przechowywania danych została wykorzystana baza danych Mssql.

Do przeprowadzenia testu integracyjnego został wykorzystany xunit.

## Front-end

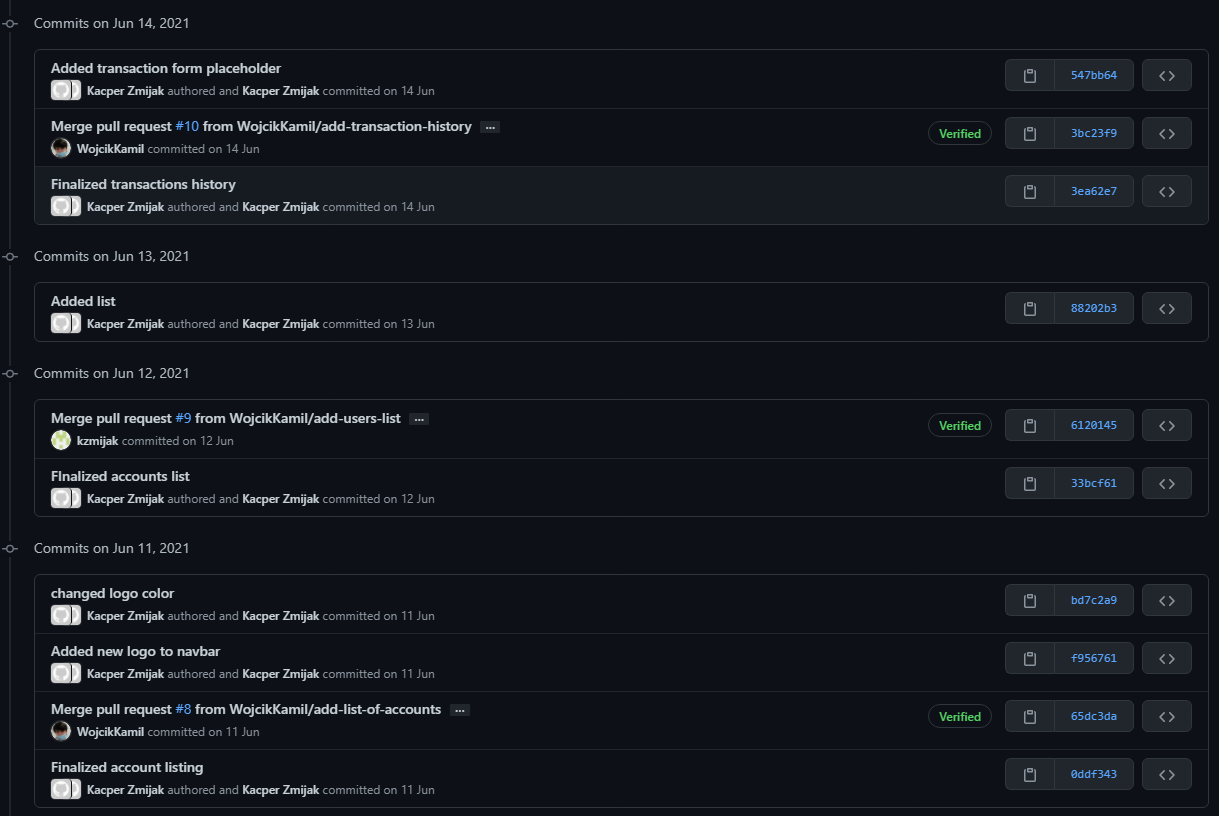
Front – end projektu został zrobiony w technologii Angular. Składa się z :

* Html
* Scss
* TypeScript

Dodatkowo został użyty Angularowy bootstrap. Czyli technologia Angular-Material.

# System kontroli wersji

Korzystaliśmy z githuba - https://github.com/WojcikKamil/Bank-Management



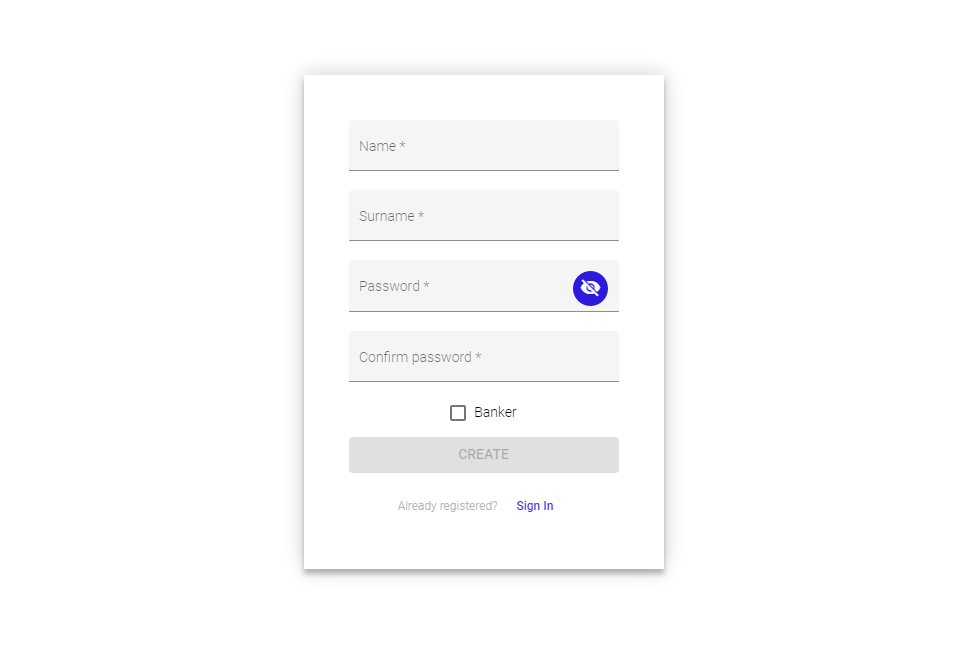
# Prezentacja warstwy użytkowej

Strona startowa, tuż po uruchomieniu aplikacji:



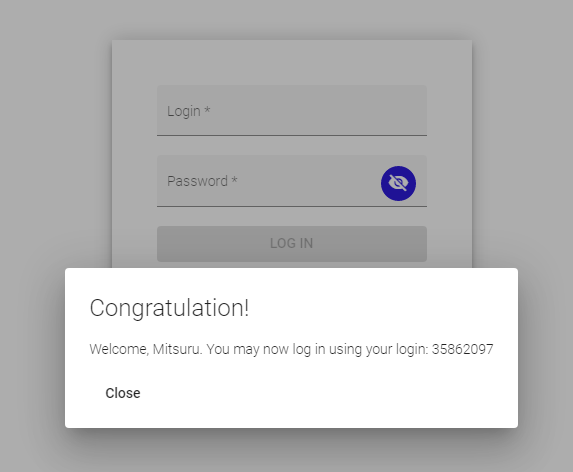
Rysunek 13 Startowy widok – Login

Widok rejestracji, po kliknięciu „Sign up” z rysunku 8



Rysunek 14 widok Rejestracji

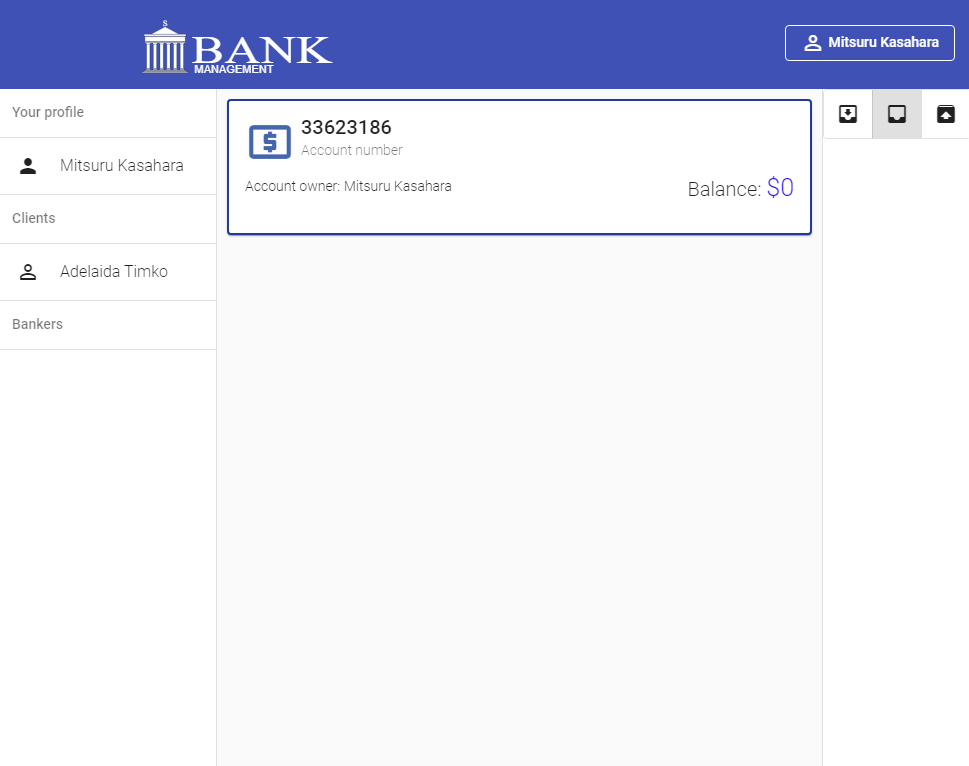
Widok po zarejestrowaniu, od razu po wprowadzeniu danych do formularza, i kliknięciu „Create” , użytkownik zostaje przeniesiony do strony z logowaniem, i pop-upem ze swoim unikalnym ID, niezbędnym do zalogowania



Rysunek 15 Widok po zarejestrowaniu

Po pomyślnym zalogowaniu, użytkownik zostaje przeniesiony na jego panel. Z tego widoku, użytkownik może:

* Kliknąć w innego klienta, po tym zostaje wyświetlony numer konta klienta który nas intertesuje.
* Kliknąć w prawym górnym rogu na swoje imię i nazwisko, wtedy wchodzi w ustawienia.
* Kliknąć w swój numer konta, dzięki czemu uruchomi funkcję przelewu.



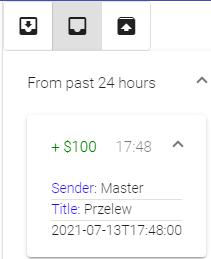
Rysunek 16 Widok po zalogowaniu

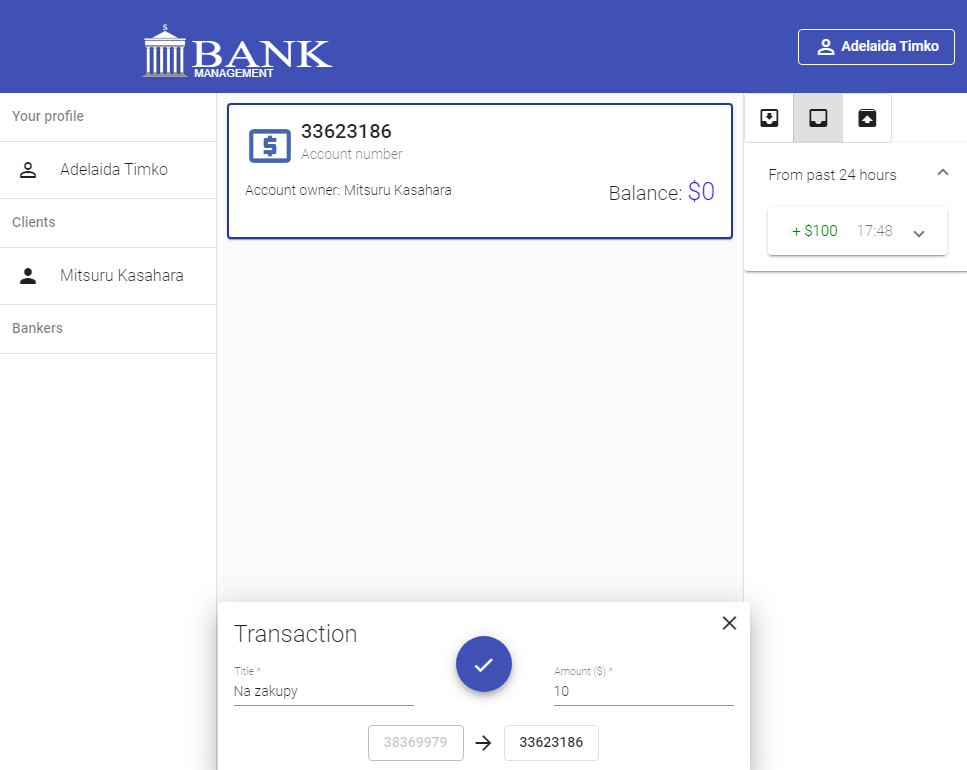
Użytkownik kliknął w swój numer klienta, uzupełnił pole:

* Tytuł
* Wartość

Następnie kliknął „target” i wybrał konto innego klienta.

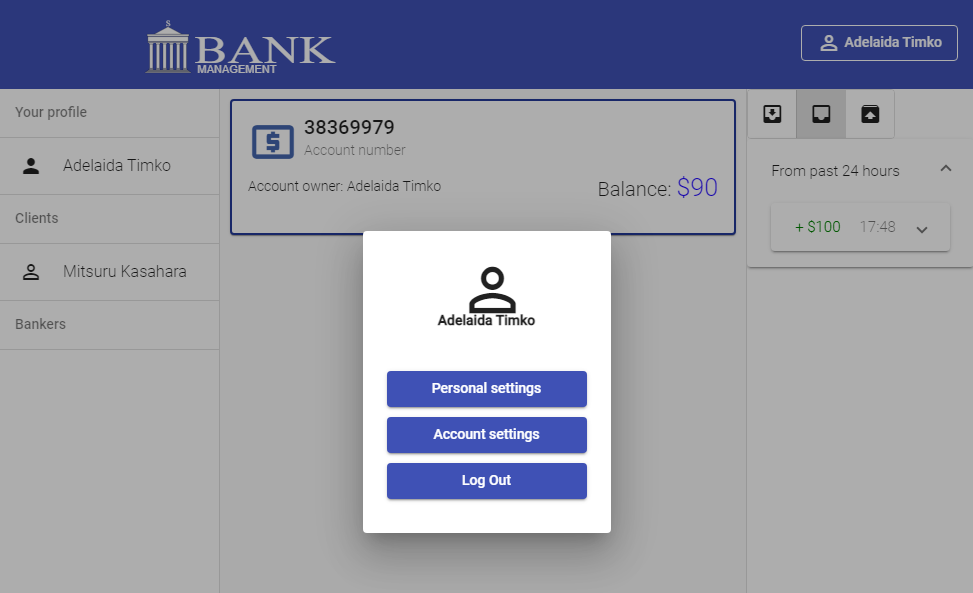
W tym przypadku z konta Adelaida( poniważ miała dodane 100$ ), przelewa 10$ na konto Mitsuru, po prawej stronie widać przelew na 100$, po rozwinięciu można zobaczyć szczegóły operaccji:



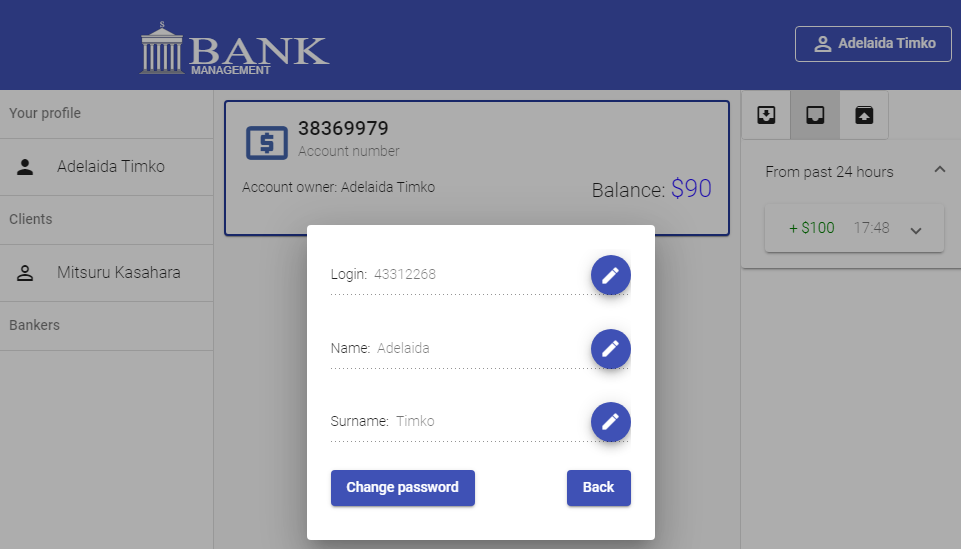


Rysunek 17 Wykonywanie przelewu

Widok po naciśnięciu w prawym górnym rogu imienia i nazwiska klienta.



Rysunek 18 Widok zarządzania kontem



Rysunek 19 Ustawienia osobiste

# Rekordy z API

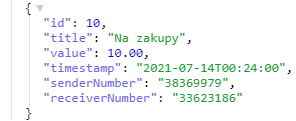
Rekordy z prezentacji warstwy użytkowej:

Na potrzeby prezentacji zostało dodanych 2 klientów:



Rysunek 20 GET - api/users

Po przelewie z konta Adelaida na konto Mitsuru:



Rysunek 21 - GET - api/transaction/60100