Bazy danych - Projekt

Maciej Szymczak, Łukasz Mędrek

Bazy Danych - Projekt

Temat: System bankowy

Bank

Opis firmy

Bank to polski bank komercyjny o charakterze uniwersalnym tzn. świadczący usługi klientowi masowemu, przy czym bank jest nastawiony na klientów w postaci osób fizycznych.

Bank a system informatyczny

Przyszły użytkownik powinien móc założyć konto online, zaś UKB przez aplikację mieć dostęp do następujących funkcji: pogląd stanu konta, oglądanie historii transferów z i na swojego konto bankowe, wykonywać i otrzymywać przelewy wewnątrz- i międzybankowe, zakładać i zamykać lokaty oraz inne działania.

Co więcej, bank oczekuje maksymalnej automatyzacji procesów bankowych dla niwelacji kosztów generowanych przez zasoby ludzkie. Ponadto, system musi zapewniać wysoki poziom bezpieczeństwa (security) i implementować mechanizmy spójności danych (data integrity).

Podejście do użytkownika

W związku z najnowszymi odkryciami neuromarketingu i ekonomii behawioralnej, bank wymaga maksymalnego uproszczenia obsługi konta dla UKB, co zredukuje liczbę kontaktów z pracownikami infolinii. Tam, gdzie to możliwe, informcja i pomoc powinna być dostarczana przez zautomatyzowane systemy informatyczne, np. chatboty. Aplikację powinien móc obsługiwać użytkownik bez jakiegokolwiek wykształcenia technicznego.

Słowny opis wymagań

UKB - Użytkownik konta bankowego - klient banku posiadający w nim konto.

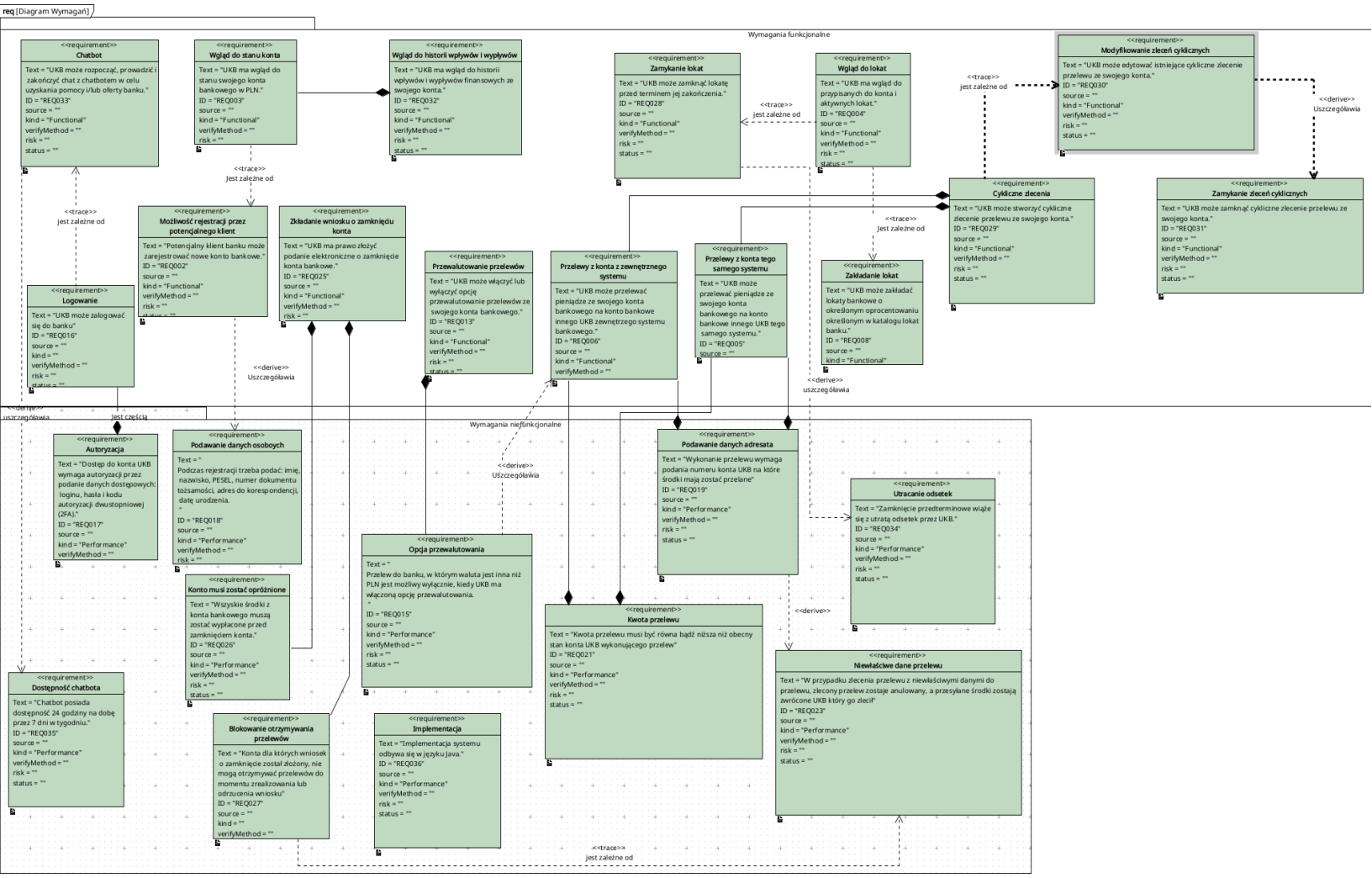

Figure 1: Diagram wymagań

Wymagania funkcjonalne

0. Potencjalny klient banku może zarejestrować nowe konto bankowe.
1. UKB ma wgląd do stanu swojego konta bankowego w PLN.
2. UKB ma wgląd do przypisanych do konta i aktywnych lokat.

3. UKB może przelewać pieniądze ze swojego konta bankowego na konto bankowe innego UKB tego samego systemu.
4. UKB może przelewać pieniądze ze swojego konta bankowego na konto bankowe innego UKB zewnętrzego systemu bankowego.
5. UKB może zakładać lokaty bankowe o określonym oprocentowaniu określonym w katalogu lokat banku.
6. UKB może włączyć lub wyłączyć opcję przewalutowanie przelewów ze swojego konta bankowego.
7. UKB ma prawo złożyć podanie elektroniczne o zamknięcie konta bankowe.
8. UKB może zamknąć lokatę przed terminem jej zakończenia.
9. UKB może stworzyć cykliczne zlecenie przelewu ze swojego konta.
10. UKB może edytować istniejące cykliczne zlecenie przelewu ze swojego konta.
11. UKB może zamknąć cykliczne zlecenie przelewu ze swojego konta.
12. UKB ma wgląd do historii wpływów i wypływów finansowych ze swojego konta.
13. UKB może rozpocząć, prowadzić i zakończyć chat z chatbotem w celu uzyskania pomocy i/lub oferty banku.

Wymagania niefunkcjonalne

0. Podczas rejestracji trzeba podać: imię, nazwisko, PESEL, numer dokumentu tożsamości, adres do korespondencji, datę urodzenia.
1. Dostęp do konta UKB wymaga autoryzacji przez podanie danych dostępowych: loginu, hasła i kodu autoryzacji dwustopniowej (2FA).
2. System obsługuje po stronie użytkownika język polski m.in., generując komunikaty o błędach czy udostępniając interfejs użytkownika w tym języku.
3. Przelew/cykliczne zlecenie między UKB są możliwe tylko pod warunkiem posiadania wystarczających środków na koncie.
4. Przelew do banku, w którym waluta jest inna niż PLN jest możliwy wyłącznie, kiedy UKB ma włączoną opcję przewalutowania.
5. Warunkiem złożenia podania elektronicznego o zamknięcie konta jest nieposiadanie aktywnych lokat.
6. Zamknięcie przedterminowe wiąże się z utratą odsetek przez UKB.
7. Chatbot posiada dostępność 24 godziny na dobę przez 7 dni w tygodniu.

Wymagania dotyczące bezpieczeństwa danych

0. Program bankowy ma dostęp do bazy danych za pomocą nieuprzywilejowanego konta system.
1. Konto system, nie może usuwać danych z tabeli przelewy i modyfikować katalogu lokat.
2. Konto system, może dodawać, modyfikować, usuwać dane w reszcie tabelach.
3. Całe oprogramowanie ma działać bez uprawnień administracyjnych.

Wymagania systemowe

0. Zaprojektowany system ma mieć możliwość łatwego skalowania.
1. System ma mieć taką samą procedurę wdrażania niezależnie od sprzętu na którym ma być uruchomiony.
2. Implementacja systemu odbywa się w języku go.

Wymagannia jakościowe

0. System ma mieć możliwość obsłużenia conajmniej 10 zapytań na sekundę.
1. System ma mieć możliwość obsłużenia conajmniej 100000 przelewów w ciągu roku.

Przypadki użycia

0. Rejestracja

   Cel: Utworzenie konta przyszłemu klientowi banku.

   Założenia wejściowe: Podmiot rejestrujący się nie jest jeszcze klientem banku i spełnia ograniczenia wiekowe.

   Założenia wyjściowe: Zostanie utworzone danemu użytkownikowi konto bankowe.

   Przebieg:

   0. Przyjęcie od użytkownika danych imię, nazwisko, PESEL, numer dokumentu tożsamości, adres do korespondencji, datę urodzenia.
   1. Sprawdzenie, czy wprowadzone dane są zgodne z odpowiadającym im formatom.
   2. Jeśli dane są nie poprawne, wyświetlany jest komunikat o błędzie i proces rejestracyjny jest przerywany.
   3. Wygenerowanie i wyświetlanie loginu oraz hasła użytkownikowi.
   4. Wygenerowanie i wyświetlenie danych potrzebnych do autoryzacji dwuetapowej (2FA).
   5. Utworzenie konta na podstawie uzyskanych danych, zapisując zmiany w systemie bazy danych.

1. Logowanie

   Cel: Autoryzacja użytkownika.

   Założenia wejściowe: PU Rejestracja został wykonany.

   Założenia wyjściowe: Użytkownik zostanie zalogowany.

   Przebieg:

   0. Przyjęcie od użytkownika loginu, hasła i kodu autoryzacji dwustopniowej (2FA).

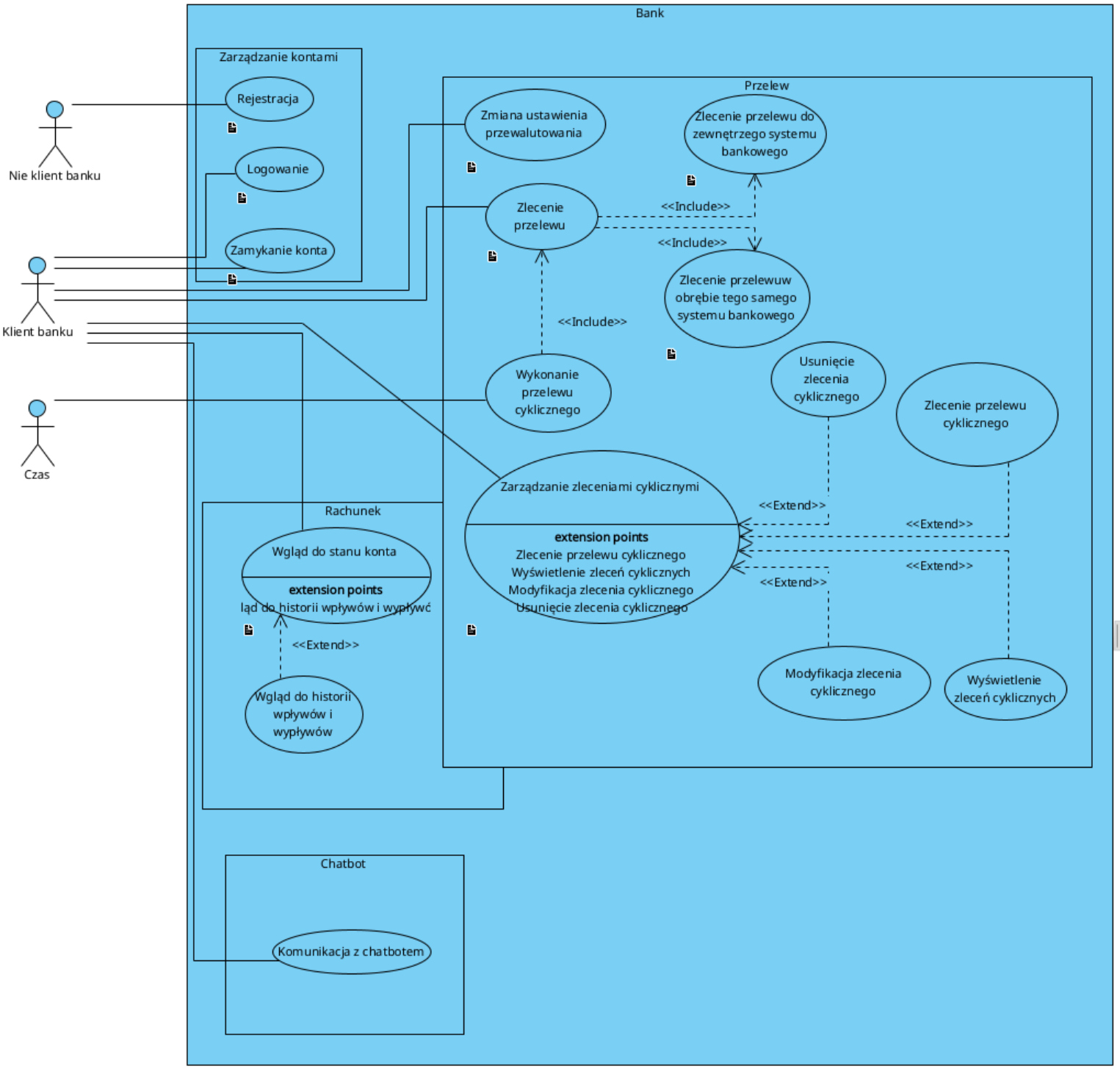

Figure 2: Diagram wymagań

1. Sprawdzenie, czy wprowadzone dane są zgodne z odpowiadającym im formatom.
2. Jeśli dane są nie poprawne, wyświetlany jest komunikat o błędzie i proces logowania jest przerywany.
3. Sprawdzenie, czy konto z podanymi danymi do logowania istnieje w systemie bazy danych.
4. Jeśli konto nie istnieje, wyświetlany jest komunikat o błędzie i proces logowania jest przerywany.
5. Użytkownik zostaje zalogowany.

2. Wgląd do stanu konta.

   Cel: Użytkownik ma wgląd do stanu swojego konta.

   Założenia wejściowe: PU Logowanie został wykonany.

   Założenia wyjściowe: Wyświetlenie stanu konta.

   Przebieg:

   0. Zgodnie z identyfikatorem użytkownika, odpytywany jest system bazy danych o stan konta.
   1. Dane o stanie konta są wyświetlane UKB.

3. Wgląd do historii wpływów i wypływów.

   Cel: Użytkownik ma wgląd do historii transakcyjnych.

   Założenia wejściowe: PU Logowanie został wykonany.

   Założenia wyjściowe: Wyświetlenie stanu konta.

   Przebieg:

   0. Zgodnie z identyfikatorem użytkownika, odpytywany jest system bazy danych o stan konta.
   1. Dane o historii wpływów i wypływów są wyświetlane UKB.

4. Zamykania konta

   Cel: Usunięcie niechcianego przez UKB konta które posiada.

   Założenia wejściowe: PU logowanie został wykonany.

   Założenia wyjściowe: Konto UKB zostaje oznaczone jako do usunięcia

   Przebieg:

   0. Przyjęcie od UKB danych z formularza usuwania konta.
   1. Jeśli dane są nie poprawne, wyświetlany jest komunikat o błędzie i proces zamykania konta jest przerywany.
   2. Jeśli konto UKB posiada jakieś środki, wyświetlany jest komunikat o konieczności opróżnienia konta i proces zamykania konta jest przerywany.

3. Jeśli konto UKB posiada jakieś nieuregulowane rachunki, wyświetlany jest komunikat o konieczności uregulowania rachunków i proces zamykania konta jest przerywany.
4. Zgodnie z identyfikatorem użytkownika, wysyłane jest polecenie do systemu bazy danych o oznaczenie konta UKB do usunięcia.
5. Wyświetlony zostanie komunikat o pomyślnym przeznaczeniu konta do usunięcia.

5. Zmiana ustawienia przewalutowania

   Cel: UKB włącza przewalutowanie, bądź je wyłącza.

   Założenia wejściowe: PU Logowanie został wykonany.

   Założenia wyjściowe: Nastąpiła zmiana stanu opcji automatycznego przewalutowania.

   Przebieg:

   0. Przyjęcie od UKB danych z formularza przewalutowania.
   1. Jeśli dane są nie poprawne, wyświetlany jest komunikat o błędzie i proces zmiany przewalutowania jest przerywany.
   2. Zgodnie z identyfikatorem użytkownika, wysyłane jest polecenie do systemu bazy danych o zmianę.
   3. Wyświetlony zostanie komunikat o pomyślnej zmianie stanu ustawienia przewalutowania.

6. Zlecenie przelewu

   Cel: UKB przelewa określoną ilość środków ze swojego konta na konto adresata.

   Założenia wejściowe: PU Logowanie został wykonany lub nadszedł czas aktywacji PU Wykonania przelewu cyklicznego.

   Założenia wyjściowe: Przelew został zlecony.

   Przebieg:

   0. Przyjęcie od UKB danych z formularza zlecenia przelewu.
   1. Jeśli kwota przelewu jest wyższa niż stan konta UKB zlecającego przelew, wyświetlany jest komunikat o błędzie i proces zlecania przelewu jest przerywany.
   2. Jeśli adresat przelewu jest w zewnętrznym banku, wykonywany jest PU Zlecenie przelewu do zewnętrznego systemu bankowego.
   3. Jeśli adresat przelewu nie jest w zewnętrznym banku, wykonywany jest przypadek użycia zlecenia przelewu w obrębie tego samego systemu bankowego.
   4. Jeśli wykonana procedura zwróciła błąd, wyświetlany jest błąd i przerywana jest procedura zlecania przelewu.

7. Zlecenie przelewu do zewnętrznego systemu bankowego

   Cel: UKB przelewa określoną ilość środków swojego konta na konto adresata

   Założenia wejściowe: Wykonywany jest przypadek użycia zlecenia przelewu

   Założenia wyjściowe: Przelew został zlecony

   Legenda:

   - OSB - obcy system bankowy - system bankowy działający niezależnie od tego systemu bankowego.
   - LSB - lokalny system bankowy - ten system bankowy.

   Przebieg:

   0. OSB odpytany jest o walutę której używa.
   1. Jeśli waluta OSB nie jest taka sama jak LSB, sprawdzana jest opcja automatycznego przewalutowania UKB zlecającego przelew.
   2. Jeśli opcja automatycznego przewalutowania nie jest włączona, wyświetlany jest komunikat błędu i proces zlecania przelewu jest przerywany z błędem.
   3. Kwota przelewu na koncie UKB zlecającego przelew jest zamrażana.
   4. Jeśli waluta OSB nie jest taka sama jak LSB, podana kwota środków przelewu jest przewalutowywana.
   5. Jeśli waluta OSB nie jest taka sama jak LSB, przewalutowana kwota środków przelewu jest przesyłana do OSB z danymi adresata.
   6. Jeśli waluta OSB jest taka sama jak LSB, kwota środków przelewu jest przesyłana do OSB z danymi aadresata.
   7. Zamrożona kwota środków zostaje usunięta z konta UKB zlecającego przelew.

8. Zlecenie przelewu w obrębie tego samego systemu bankowego.

   Cel: UKB przelewa określoną ilość środków swojego konta na konto adresata

   Założenia wejściowe: Wykonywany jest przypadek użycia zlecenia przelewu

   Założenia wyjściowe: Przelew został zlecony

   LSB - lokalny system bankowy - ten system bankowy.

   Przebieg:

   0. Kwota przelewu na koncie UKB zlecającego przelew jest zamrażana.
   1. Zamrożona kwota przelewu zostaje dodana do konta UKB będącego adresatem przelewu.
   2. Zamrożona kwota środków zostaje usunięta z konta UKB zlecającego przelew.

3. Zamrożona kwota środków zostaje odmrożona z konta UKB będącego adresatem przelewu.

9. Zarządzanie przelewami cyklicznymi

   Cel: UKB ma możliwość zarządzania przelewami cyklicznymi na swoim koncie bankowym.

   Założenia wejściowe: PU Logowanie został wykonany.

   Założenia wyjściowe: Wykonanie wybranego przez UKB przypadku użycia.

   Przebieg:

   0. UKB wybiera jedną z opcji zarządzania swoimi przelewami cyklicznymi.
   1. Realizowany jest odpowiadający wybranej opcji przypadek użycia z następujących: PU Zlecenie przelewu cyklicznego, PU Modyfikacja przelewu cyklicznego, PU Usunięcie przelewu cyklicznego.

10. Zlecenie przelewu cyklicznego

    Cel: UKB przelewa określoną ilość środków swojego konta na konto adresata określoną ilość razy, w określonych odstępach czasowych.

    Założenia wejściowe: PU logowanie został wykonany.

    Założenia wyjściowe: Przelew cykliczny został zapisany.

    Przebieg:

    0. Przyjęcie od UKB danych z formularza zlecenia przelewu cyklicznego.
    1. Przesłanie danych do systemu bazy danych.

11. Wykonanie przelewu cyklicznego

    Cel: System informatyczny zleca wykonania przelewów na podstawie zapisanych informacji podanych przez przypadek użycia przelewu cyklicznego.

    Założenia wejściowe: Przypadek jest wykonywany o północy każdego dnia roboczego.

    Założenia wyjściowe: Przelew został zlecony

    Przebieg:

    0. System bazy danych jest odpytany o zapisane zlecenia przelewów cyklicznych.
    1. Dla każdego zlecenia, wykonaj PU wykonania przelewu.
    2. Jeśli PU wykonania przelewu nie został wykonany pomyślnie, prześlij błąd UKB który zlecił zlecenie i usuń zlecenie.

12. Wyświetlenia zleceń cyklicznych

    Cel: UKB dowiaduje się o swoich informacji podanych przez przypadek użycia przelewu cyklicznego.

    Założenia wejściowe: PU logowanie zostało wykonane.

    Założenia wyjściowe: UKB dowiaduje się o swoich zleceniach cyklicznych.

    Przebieg:

    0. System bazy danych jest odpytany o zlecenia cykliczny przypisane do identyfikatora UKB.
    1. Zlecenia cykliczne sią wyświetlane UKB

13. Modyfikacja zlecenia cyklicznego

    Cel: UKB dowiaduje się o swoich informacji podanych przez przypadek użycia przelewu cyklicznego.

    Założenia wejściowe: PU Logowanie zostało wykonane.

    Założenia wyjściowe: UKB dowiaduje się o swoich zleceniach cyklicznych.

    Przebieg:

    0. Przyjęcie od UKB danych z formularza modyfikacji przelewu cyklicznego.
    1. Przesłanie danych do systemu bazy danych.

14. Usunięcie zlecenia cyklicznego

    Cel: UKB usuwa zlecone przez siebie zlecenie cykliczne

    Założenia wejściowe: PU logowanie zostało wykonane.

    Założenia wyjściowe: Zlecenie cykliczne zostało usunięte.

    Przebieg:

    0. Przyjęcie od UKB danych z formularza usunięcia przelewu cyklicznego.
    1. Przesłanie informacji o usunięciu zlecenia do systemu bazy danych.

15. Komunikacja z chatbotem

    Cel: UKB nawiązuje komunikację z chatbotem

    Założenia wejściowe: PU logowanie zostało wykonane.

    Założenia wyjściowe: Komunikaca z chatbotem została nawiązana

    Przebieg:

    0. Indentyfikator użytkownika zostaje przesłany do systemu chatbota.
    1. Tworzony jest kanał tekstowy umożliwający komunikację między chatbotem a użytkownikiem.
    2. Użytkownik jest przekierowywany do kanału tekstowego.

Projekt bazy danych

Utworzono projekt bazy danych mający na celu umożliwić jednoznaczne przechowywanie informacji o stanie kont użytkowników banku.

Implementacja bazy danych

System bankowy używa zaprojektowanej bazy danych zaimplemnetowanej na systemie bazodanowym postgres. Aby obniżyć koszty utrzymania, baza danych będzie skonteneryzowana i uruchamiana przez system podman, w celu uruchomiania jej bez potrzeby posiadania uprawnień administracyjnych.

Serwer postgres będzie uruchomiony z obrazu bazy postgres opartego o alpine linux, aby maksymalnie zmniejszyć rozmiar kontenera, oraz zmniejszając możliwości środowiska które on udostępnia, w celu zminimalizowania obszaru który jest podatny na ataki.

Aby umożliwić sprawne tworzenie kopii zapasowych bazy danych, oraz zapewnić trwałość danych niezależną od kontenera z której uruchomiony jest serwer, katalog przechowujący pliki bazodanowe serwera mają być podmontowanymi voluminami.

Aby zwiększyć izolację między bazą danych a użytkownikami, baza jest skonfigurowana domyślnie aby, dostęp do niej był możliwy tylko z tworzonej przez podmana sieci wewnętrznej, w której znajduje się również serwer pełniący rolę inferfejsu graficznego systemu bankowego.

System zarządzania kontenerami	podman	
Obraz	docker.io/postgres:12.17-alpine3.19	
Podmontowane katalogi	kontener	system
	/var/lib/postgresql/data	volumin

Szkielet bazy danych, tabele, wymagania, mogą być zainicjalizowane prosto przy użyciu dołączonego pliku baza.sql. Aby po raz pierwszy utworzyć bazę, należy uruchomić serwer bazo danowy, a następnie podłączyć się do niego, np. przy użyciu standardowego programu psql do użytkownika admin, z hasłem password do bazy bazy i wykonać zawartość dołączonego pliku.

Plik baza.sql został wygenerowany narzędziem pq_dump służącym do generowania zapytań sql do odtwarzania całej bazy danych. Pierwotna baza została utworzona na podstawie zaprojektowanego diagramu relacji encji.

Baza ma zdefiniowanego użytkownika admin, i użytkownika system, który pozwala serwerowi bankowemu na modyfikację i wstawianie nowych rekordów.

Przed wdrożeniem systemu, zalecana jest zmiana hasła kontom bazodanowym.

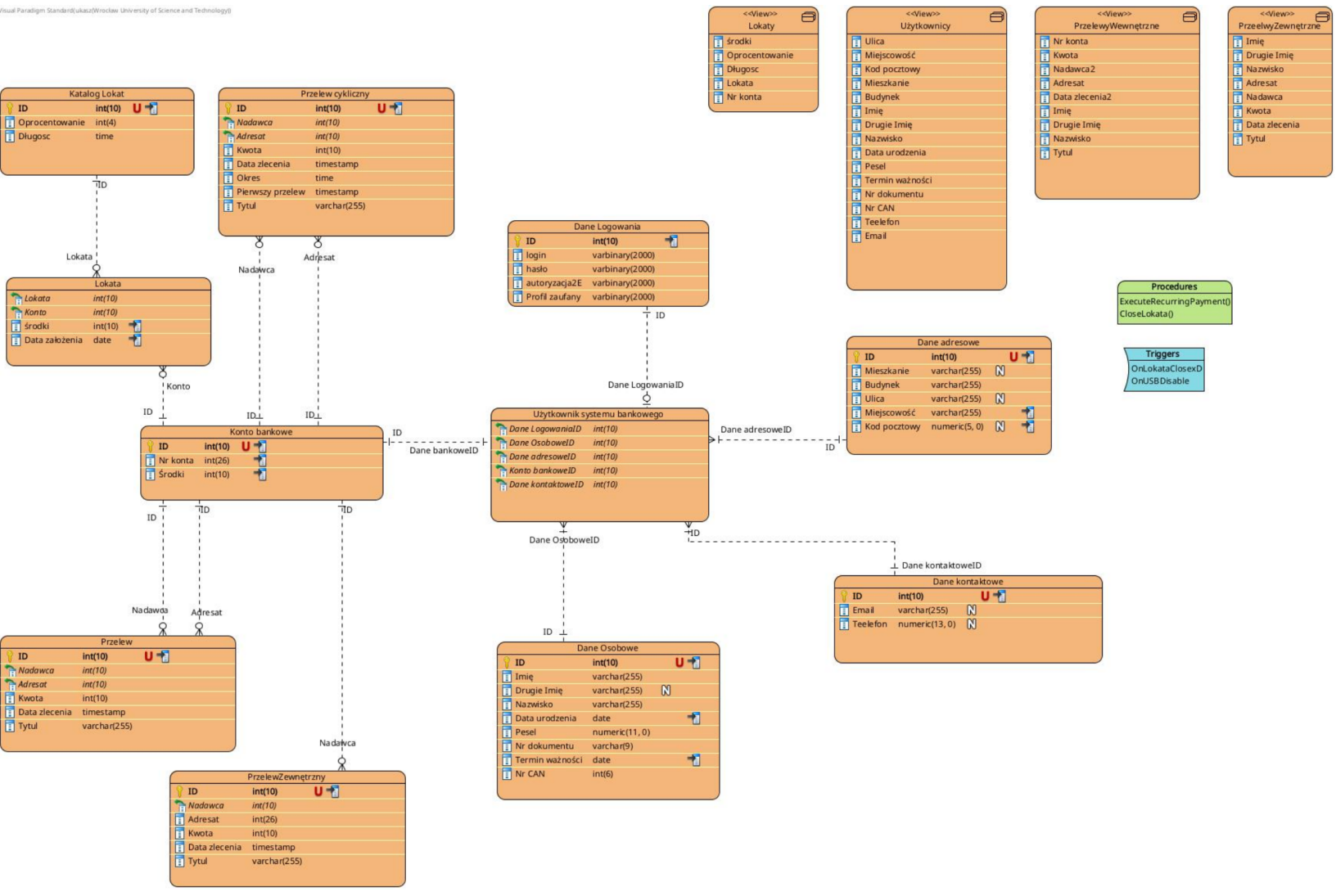

Figure 3: Diagram relacji encji

użytkownik	dostęp
admin	wszystko, wszędzie
system	odczyt, zapis, wszędzie

Polecenia SQL

Niżej zaprezentowane są komendy SQL, przy których utworzono bazę danych.

- tabele

```
CREATE TABLE public.konto (
    id integer NOT NULL,
    nr numeric(24,0) NOT NULL,
    srodki integer NOT NULL
);

CREATE TABLE public.lokata (
    lokata integer NOT NULL,
    konto integer NOT NULL,
    srodki integer NOT NULL,
    data date NOT NULL
);

CREATE TABLE public.adresowe (
    id integer NOT NULL,
    mieszkanie character varying(255),
    budynek character varying(255) NOT NULL,
    ulica character varying(255),
    miejscowosc character varying(255) NOT NULL,
    poczta integer
);

CREATE TABLE public.kontaktowe (
    id integer NOT NULL,
    email character varying(255),
    teelefon integer
);

CREATE TABLE public.logowania (
    id integer NOT NULL,
    login character varying NOT NULL,
    haslo character varying NOT NULL,
    autoryzacja2e character varying NOT NULL,
```

```
    zaufany character varying NOT NULL
);

CREATE TABLE public.lokaty (
    id integer NOT NULL,
    oprocentowanie integer NOT NULL,
    dlugosc interval NOT NULL
);

CREATE TABLE public.osobowe (
    id integer NOT NULL,
    imie character varying(255) NOT NULL,
    imie2 character varying(255),
    nazwisko character varying(255) NOT NULL,
    data date NOT NULL,
    pesel integer NOT NULL,
    dokument character varying(9) NOT NULL,
    termin date NOT NULL,
    can integer NOT NULL
);

CREATE TABLE public.przelew (
    nadawca numeric(24,0) NOT NULL,
    adresat numeric(24,0) NOT NULL,
    kwota integer NOT NULL,
    data timestamp without time zone NOT NULL,
    tytul character varying(255) NOT NULL
);

CREATE TABLE public.przelewc (
    id integer NOT NULL,
    nadawca numeric(24,0) NOT NULL,
    adresat numeric(24,0) NOT NULL,
    kwota integer NOT NULL,
    data timestamp without time zone NOT NULL,
    okres interval NOT NULL,
    faza timestamp without time zone NOT NULL,
    tytul character varying(255) NOT NULL
);

CREATE TABLE public.przelewz (
    id integer NOT NULL,
    nadawca numeric(24,0) NOT NULL,
    adresat numeric(24,0) NOT NULL,
    kwota integer NOT NULL,
    data timestamp without time zone NOT NULL,
```

```
    tytul character varying(255) NOT NULL
);

CREATE TABLE public.usb (
    d_logowania integer NOT NULL,
    d_osobowe integer NOT NULL,
    d_adresowe integer NOT NULL,
    konto integer NOT NULL,
    d_kontaktowe integer NOT NULL
);

CREATE TABLE public.konto (
    id integer NOT NULL,
    nr numeric(24,0) NOT NULL,
    srodki integer NOT NULL
);

CREATE TABLE public.lokata (
    lokata integer NOT NULL,
    konto integer NOT NULL,
    srodki integer NOT NULL,
    data date NOT NULL
);

CREATE TABLE public.adresowe (
    id integer NOT NULL,
    mieszkanie character varying(255),
    budynek character varying(255) NOT NULL,
    ulica character varying(255),
    miejscowosc character varying(255) NOT NULL,
    poczta integer
);

CREATE TABLE public.kontaktowe (
    id integer NOT NULL,
    email character varying(255),
    teelefon integer
);

CREATE TABLE public.logowania (
    id integer NOT NULL,
    login character varying NOT NULL,
    haslo character varying NOT NULL,
    autoryzacja2e character varying NOT NULL,
    zaufany character varying NOT NULL
);
```

- Uprawnienia dla użytkownika system

```
GRANT INSERT ON TABLE public.konto TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.lokata TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.adresowe TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.kontaktowe TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.logowania TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.osobowe TO system;
GRANT SELECT,INSERT ON TABLE public.przelew TO system;
GRANT SELECT,INSERT,DELETE ON TABLE public.przelewc TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.usb TO system;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.uzytkownicy TO system;
```

- Procedury

```
create or replace procedure run_reccurent_payments()
language plpgsql
as $$
 declare my_cur cursor for
    select
        nadawca,
        adresat,
        kwota,
        tytul,
        extract(epoch from data),
        extract(epoch from faza),
        extract(epoch from okres)
    from przelewc;

 declare r_nadawca numeric(24);
 declare r_adresat numeric(24);
 declare r_kwota int;
 declare r_tytul varchar;
 declare r_data numeric;
 declare r_faza numeric;
 declare r_okres numeric;
begin
 open my_cur;
 fetch my_cur into r_nadawca, r_adresat, r_kwota, r_tytul, r_data, r_faza, r_okres;
 while FOUND loop
  if (r_data - r_faza) / r_okres >= 1 then
   call przelew(r_nadawca, r_adresat, r_kwota, r_tytul);
  end if;
  fetch my_cur into r_nadawca, r_adresat, r_kwota, r_tytul, r_data, r_faza, r_okres;
 end loop;
```

```
 close my_cur;
end; $$;

CREATE PROCEDURE public.przelew(IN p_nadawca numeric, IN p_adresat numeric, IN p_kwota integ
    LANGUAGE plpgsql
    AS $$
begin
 if exists (select 1 from konto where nr=p_nadawca and srodki >= p_kwota)  then
   insert into przelew(nadawca, adresat, kwota, data, tytul) values (p_nadawca, p_adresat, p
   update konto set srodki=srodki-p_kwota where konto.nr = p_nadawca;
  end if;
end; $$;

CREATE PROCEDURE public.run_reccurent_payments()
    LANGUAGE plpgsql
    AS $$
 declare my_cur cursor for
select
nadawca,
adresat,
kwota,
tytul,
extract(epoch from data),
extract(epoch from faza),
extract(epoch from okres)
from przelewc;

 declare r_nadawca numeric(24);
 declare r_adresat numeric(24);
 declare r_kwota int;
 declare r_tytul varchar;
 declare r_data numeric;
 declare r_faza numeric;
 declare r_okres numeric;
begin
 open my_cur;
 fetch my_cur into r_nadawca, r_adresat, r_kwota, r_tytul, r_data, r_faza, r_okres;
 while FOUND loop
  if (r_data - r_faza) / r_okres >= 1 then
   call przelew(r_nadawca, r_adresat, r_kwota, r_tytul);
  end if;
  fetch my_cur into r_nadawca, r_adresat, r_kwota, r_tytul, r_data, r_faza, r_okres;
 end loop;
 close my_cur;
end; $$;
```

```
CREATE PROCEDURE public.close_lokata(IN p_konto integer, IN p_lokata integer)
    LANGUAGE sql
    BEGIN ATOMIC
 UPDATE public.konto SET srodki = (konto.srodki + ( SELECT lokata.srodki
           FROM public.lokata
          WHERE ((lokata.konto = close_lokata.p_konto) AND (lokata.lokata = close_lokata.p_
   WHERE (konto.id = close_lokata.p_konto);
 DELETE FROM public.lokata
   WHERE ((lokata.lokata = close_lokata.p_lokata) AND (lokata.konto = close_lokata.p_konto)
END;
```

- Klucze główne i obce

```
ALTER TABLE ONLY public.adresowe
    ADD CONSTRAINT adresowe_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.konto
    ADD CONSTRAINT fk_konto_nr UNIQUE (nr);
ALTER TABLE ONLY public.kontaktowe
    ADD CONSTRAINT kontaktowe_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.konto
    ADD CONSTRAINT konto_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.logowania
    ADD CONSTRAINT logowania_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.lokaty
    ADD CONSTRAINT lokaty_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.osobowe
    ADD CONSTRAINT osobowe_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.przelewc
    ADD CONSTRAINT przelewc_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.przelewz
    ADD CONSTRAINT przelewz_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.przelewc
    ADD CONSTRAINT fk_przelewc_adresat FOREIGN KEY (adresat) REFERENCES public.konto(nr);
ALTER TABLE ONLY public.lokata
    ADD CONSTRAINT fklokatakonto FOREIGN KEY (konto) REFERENCES public.konto(id);
ALTER TABLE ONLY public.lokata
    ADD CONSTRAINT fklokatalokata FOREIGN KEY (lokata) REFERENCES public.lokaty(id);
ALTER TABLE ONLY public.usb
    ADD CONSTRAINT fkusbadresowe FOREIGN KEY (d_adresowe) REFERENCES public.adresowe(id);
ALTER TABLE ONLY public.usb
    ADD CONSTRAINT fkusbkontaktowe FOREIGN KEY (d_kontaktowe) REFERENCES public.kontaktowe(
ALTER TABLE ONLY public.usb
    ADD CONSTRAINT fkusbkonto FOREIGN KEY (konto) REFERENCES public.konto(id);
ALTER TABLE ONLY public.usb
    ADD CONSTRAINT fkusblogowania FOREIGN KEY (d_logowania) REFERENCES public.logowania(id);
```

```
ALTER TABLE ONLY public.usb
    ADD CONSTRAINT fkusbosobowe FOREIGN KEY (d_osobowe) REFERENCES public.osobowe(id);
```

- Sekwencyje

```
CREATE SEQUENCE public.adresowe_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.kontaktowe_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.konto_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.logowania_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.lokaty_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.osobowe_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
```

```
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.przelewc_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE SEQUENCE public.przelewz_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

Testy bazy danych

Do bazy danych utworzono zautomatyzowane testy jednostkowe sprawdzające jest spójność. Testy zostały zaprogramowane w języku go. Aby uruchomić testy, należy wykonać

```
go test
```

Przygotowano trzy rodzaje testów:

- Test czy da się użyć bazy.

Niżej przygotowano przykładowy test i test do niego przeciwny

```
// język go

func TestTransferIllegalAddress(t *testing.T) {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()
```

```
    if _, err := db.Exec(`call przelew($1, $2, $3, $4))`,
    "ble", "ble", int(1 * 100), "Shoudl not work"); err == nil {
        t.Fatalf("Could execute malformend command")
    }
}

func TestTransfer(t *testing.T) {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
    host, port, user, password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()


    if _, err := db.Exec(`call przelew($1, $2, $3, $4)`,
    "00001111222233334444555",
    "00001111222233334444556",
    int(1 * 100), "Shoudl work"); err != nil {
        t.Fatalf("%v", err)
    }
}
```

- Test czy prawa dostępu są poprawne.

```
// język go

func TestAccessIllegal(t *testing.T) {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
    host, port, service_user, service_password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()


    if _, err := db.Exec(`delete from przelew`); err == nil {
```

```
        t.Fatalf("Could execute illegal command")
    }
}

func TestAccess(t *testing.T) {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
    host, port, service_user, service_password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()


    if _, err := db.Exec(`call przelew($1, $2, $3, $4)`,
    "00001111222233344445555",
    "00001111222233344445556",
    int(1 * 100), "Shoudl work"); err != nil {
        t.Fatalf("%v", err)
    }
}
```

- Test wydajnościowy w przy dużym wypełnieniu bazy, w zależności od liczby zapytań.

Bazę wypełniono danymi

```
// język go

func TestOnce(t *testing.T) {
    return; // return aby nie dodawać więcej danych niż już dodano
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
    host, port, service_user, service_password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()


    for i := 0; i < 100000; i++ {
```

```
        if _, err := db.Exec(`call przelew($1, $2, $3, $4)`, "00001111222233334444555",
        "00001111222233334445556", int(1 * 1), "Shoudl work"); err != nil {
            t.Fatalf("%v", err)
        }
    }
}
```

A następnie

```
// język go
func TestSelect(t *testing.T) {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
    host, port, service_user, service_password, dbname)

    // open database
    db, err := sql.Open("postgres", psqlconn)
    if err != nil {
        t.Fatalf("%v", err.Error())
    }
    defer db.Close()


    if _, err := db.Query(`select * from przelew`); err != nil {
        t.Fatalf("%v", err)
    }
}
```

Testy wydajnościowe pokazały, że czas wykonywania apytania rośnie liniowo i dla 16k rekordów, selekt wykonuje się 0.04s; odczyt i przetworzenie 16k rekordów do formatu HTML zajmuje około 5s, niezależnie dla małej liczby zapytań na sekundę (2/s). 40k wykonuje się 0.05s, a odczyt i przetworzenie do formatu HTML około 25s. Jak widać, czas przygotowania zapytania jest zawsze większy niż szybkość jego przetwarzania.

Specyfikacja

Specyfikacja techniczna serwera systemu bankowego

Interakcja z systemem bankowym ma opierac się o protokół HTTP, mający zapewnić możliwość połączenia się każdego użądzenia komputerowego posiadającego kartę sieciową, bez żadnej dodatkowej konfiguracji.

Sam serwer jest zaprojektowany w języku golang, jego standardowej biblioteki i modułu gorilla/sessions wraz z gorilla/csrf mające zapewnić wystarczający

poziom bezpieczeństwa przed zautomatyzowanymmi atakammi na udostępniany interfejs.

Dany język i zestaw bibliotek zostały wybrane ze względu na prostotę składni, jawną obsługę błędów i architekturę wspierającą tworzenie programów asynchronicznych na poziomie samej składni języka.

Działanie samego serwer opiera się o przyjmowanie zapytań HTTP, i w zależności od url w nich zawartego wraz z typem zapytania (GET, POST lub DELETE) sprawdzania danych sesji użytkownika, przesłanego formularza i wykonywanai odpowiedniego zapytanai sql jeśli wszystko jest ok.

Każda obsługa zapytania kończy się przesłaniem odpowiedniej odpowiedzi HTTP zapierającej przekierowanie, lub prekazującej minimalistyczny kod HTML zawierający dane których użytkownik żądał. Aby zwiększyć interaktywność intergejsu, strona główna interfejsu zawiera dołączenie biblioteki htmx, umożliwiającej po stronie klienta dynamiczne ładowanie danych z serwera, przy pomocy dodawania do zwykłych tagów HTML specjalnych atrybutów zwiększających ich interaktywność.

Specyfikacja infrastruktury systemu bankowego

Cały system jest oparty o technologię konteneryzacji opartą o kontenery OCI, aby umożliwić prostą instalację i integrację systemu niezależnie od maszyn na których ma być on hostowany.

Sam system jest zdefiniowany w pliku compose, umożliwiając jego całkowitą reprodukcję i definiując środowisko w którym ma być on uruchamiany.

System nie ma być bezpośrednio łączony z internetem, do infrastruktury umożliwiającej działanie systemu powinien wchodzić serwer reverse proxy z load balancerem, który będzie wymuszał komunikację szyfrowaną między internetem a infrastrukturą, oraz umożliwiał skalowanie w szerz systemu bankowego.

Stawianie serwera

```
podman-compose up
```

Jeśli baza danych nie została jeszcze zainicjalizowana, w kontenerze bazy

```
psql -U admin -d bazy < baza.sql
```

Specyfikacja kliencka

Każdy klient banku ma udostępniony interfejs webowy do banku. Interfejs webowy jest zaprojektowany w sposób minimalistyczny, umożliwiający proste

zarządzanie stanem konta użytkownika, jak i minimalizujący transfer danych między użytkownikiem a systemem bankowym.

Minimalistyczny interfejs ma również zapewnić łatwość tworzenia różnych nakładek graficznych przez osoby trzecie, w wypadku gdyby mmiały one specjalne potrzeby wynikające z ich stanu zdrowotnego.

Wszystkie funkcje bankowe wymagają uprzedniego poprawnego wypełnienia formularza logowania do systemu bankowego.

W przypadku podania błędnych danych przez użytkownika, zostaje on wylogowany i zmuszany do ponownego zalogowania się aby chronić jego dane przez neupowarznionym dostępem.

Mini instrukcja obsługi

- Aby utworzyć lokatę, kliknij przycisk załóż obrazek obrazek 4

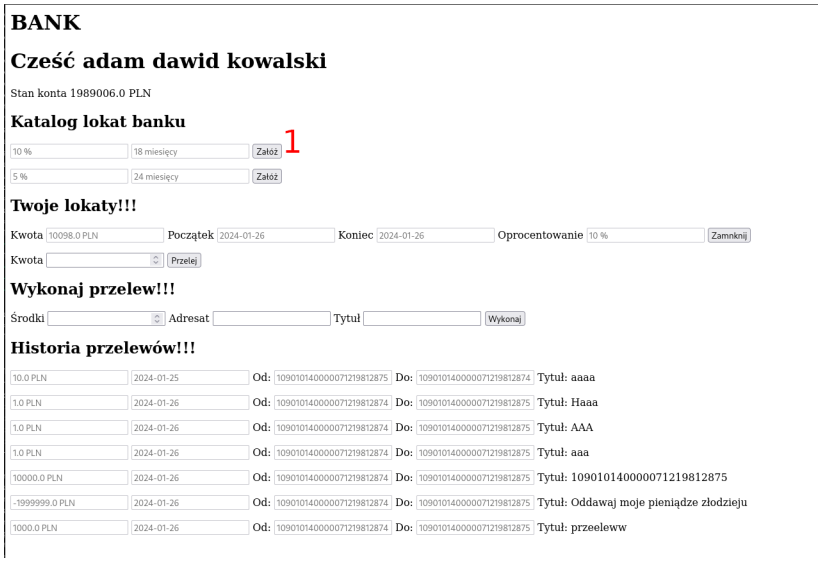

Figure 4: Tworzenie lokat

- Aby wpłacić/wypłacić pieniądze z lokaty, wpisz kwotę do przelania (1), i kliknij przelej (2) obrazek 5

- Aby wykonać przelew, wpisz 26 cyfrowy numer konta bankowego adresata (2), wpisz kwotę przelewu (1), wpisz tytuł przelewu (3) i kliknij przrycisk (4) obrazek 6

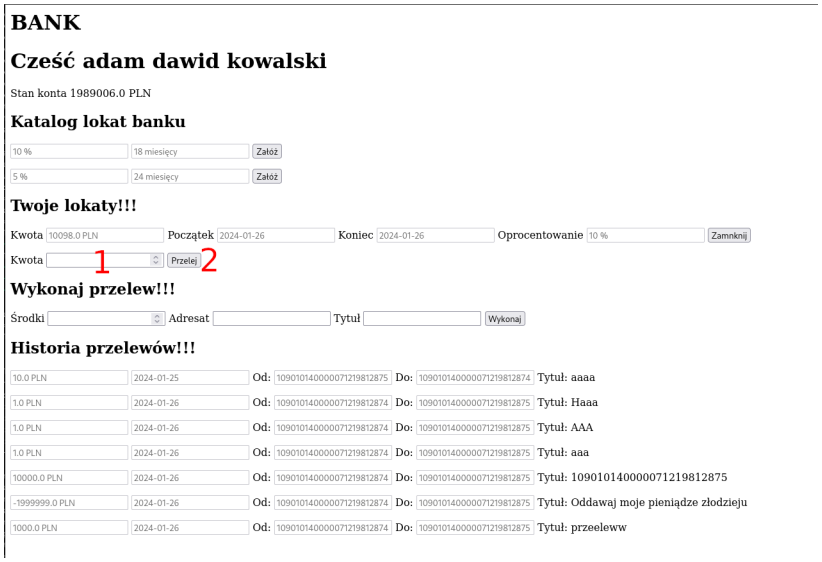

Figure 5: Przelewanie na lokate

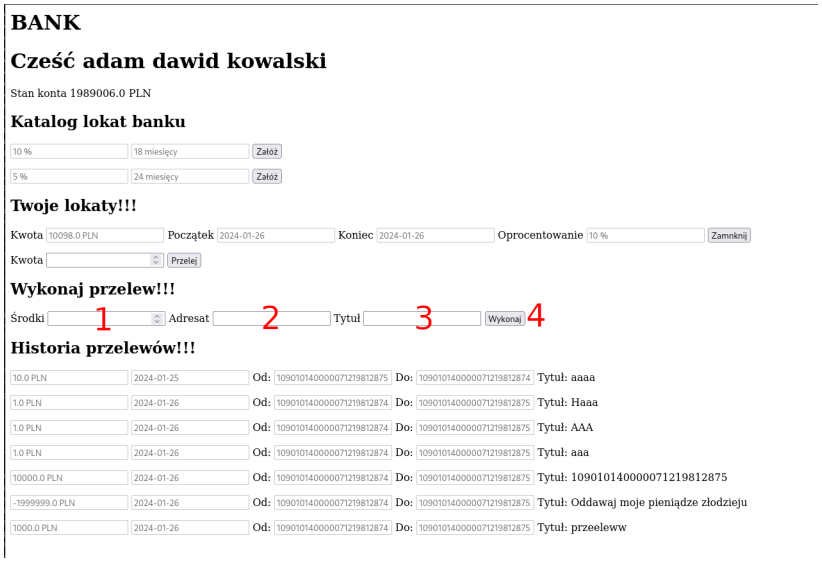

Figure 6: Przelew

Podsumowanie użytych technologii

strona	język	oprogramowanie	cel
serwer	golang	serwer bankowy	utworzenie bezpiecznego interfejsu do komunikacji między bazą danych a klientem
serwer	golang	pq	połączenie z postgres
serwer	golang	gorilla/sessions	zapewnienie sesji przeglądania dla użytkowników
serwer	golang	gorilla/csrf	zabezpieczenie formularzy przed atakiem cross site request forgery
serwer	golang	templ	generowanie funkcji na posdstawie szablonów HTML generujących HTML z podanymi parametrami
infrastruktura	niedotyczy	postgres	używana baza danych
infrastruktura	compose	niedotyczy	zapewnienie jednoznaczej i przenośnej definicji systemu bankowego
infrastruktura	niedotyczy	podman	zarządzanie kontenerami
infrastruktura	niedotyczy	reverse proxy + load balancer przyjmujący połączenia tls	umożliwienie rozszerzania w szerz infrastruktury
klient	javascript	htmx	zapewnienie interaktywnej strony, bez ciągłego odświerzania

Biblioteki

nazwa	wersja
go	1.21.5
github.com/lib/pq	v1.10.9
github.com/gorrila/csrf	v1.8.2
github.com/gorrila/session	v1.2.9
github.com/a-h/templ	v0.2.513

Rozwiązania technologiczne

Obsługa endpointa na przykładzie wyświetlania lokat przypisanych do konta

- Walidacja danych

```
// język go
http.HandleFunc("/moje_lokaty", func(w http.ResponseWriter, r *http.Request) {
    if err := r.ParseForm(); err != nil {
        log.Fatal(err)
    }

    cookie, err := r.Cookie("account_id")
    if err != nil {
        fmt.Printf("%v", err)
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }
    account, err := strconv.ParseInt(cookie.Value, 10, 0)
    if err != nil {
        fmt.Printf("%v", err)
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }
```

- Odczyt danych z dazy banych.

```
// język go
    rows, err := db.Query(`select
lokata.lokata, lokata.srodki, extract(epoch from lokata.data)::int,
extract(epoch from (lokata.data + lokaty.dlugosc))::int, lokaty.oprocentowanie
from lokata
inner join lokaty on lokata.lokata=lokaty.id
where lokata.konto = $1`, account)
    if err != nil {
        fmt.Printf("%v", err)
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }
```

- Wyświetlanie HTML zawierającego zwrócone rekordy

```
// język go
    fmt.Fprintf(w, "<html><body>")
```

```
    for rows.Next() {
        var (
            lokata int
            srodki int
            data int64
            koniec int64
            oprocentowanie int
        )
        rows.Scan(&lokata, &srodki, &data, &koniec, &oprocentowanie);

        fmt.Println(data, koniec)
        if err := templates.MojaLokata(
            models.Lokata{
                Lokata: lokata,
                Srodki: srodki,
                Data: time.Unix(data, 0),
                Koniec: time.Unix(data, 0),
            },
            models.LokataTyp{
                Oprocentowanie: oprocentowanie,
            },
        ).Render(context.Background(), w); err != nil {
            log.Fatal(err)
        }
    }
    fmt.Fprintf(w, "</body></html>")
}
```

Podsumowanie

System świadczy podstawowe wymagania w temacie bezpieczeństwa, bezpiecznie przechowując dane użytkowników i ich środki.

System opiera się na interfejsie dostarczanym przez serwer HTTP, umożliwiającym na interakcje z bazą danych. Sama architektura systemu jest otwarta na rozszeżanie jej w ramach rosnących wymagań klienta.

Baza danych, utworzony program są dostępne na serwisie github.com github.com/Lukasz825700516/PwrBazyAleBezUML.

Wnioski

Wybór języka go, wypecjalizowanego do szybkiego tworzenia aplikacji sieciowych faktycznie przyśpieszył pracę nad implementacją serwera HTTP,

zmniejszając do minimum czas wymagany do skonfigurowania poprawnie połączeń i ich optymalnej obsługi, biorąc pod uwagę wcześniejsze doświadczenie zespołu programistycznego w językach programowania takich jak php, rust, javascript, c.

Realizacja projektu pokazała również, że etap projektowania powinien przewidywać kilka faz podczas których opracowany projekt będzie sprawdzony pod kątem wymagań narzucanych przez jego impleentację, tak aby po wykryciu problemu, dało się zmodyfikować tylko jego małą część, zamiast wszyskiego.

W czasie pisania oprogramowania, skupienie się na samej funkcjonalności jednocześnie pozostawiając łatwy do edycji szkielet interfejsu użytkownika, przez udostępnienie interfejsu HTML obsługiwanego przez zwykłe zapytania GET i POST, umożliwiły skrócenie cyklu rozwoju całego oprogramowania do minimum.

Literatura, głównie dokumentacja

- Dokumentacja go go.dev.
- Dokumentacja templ github.com/a-h/templ.
- Dokumentacja gorrila gorilla.github.io.
- Dokumentacja postgres www.postgresql.org/docs/.
- Podman strona główna podman.io.
- HTMX strona główna htmx.org.