

**Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie**

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

KATEDRA INFORMATYKI STOSOWANEJ



INŻYNIERIA OPROGRAMOWANIA

**PAWEŁ BIŁKO, ANDRZEJ KOŁAKOWSKI
STANISŁAW ŚWIATŁOCH, MATEUSZ WIĘCŁAWEK**

BANK BITCOINÓW

Kraków 2020

Spis treści

1. Wstęp	4
1.1. Cel i założenia projektu	4
2. Opis funkcjonalności	5
3. Przypadki użycia	7
4. Wymagania niefunkcjonalne	12
5. Architektura systemu	14
5.1. Schemat bazy danych	14
6. Opis interfejsów	15
6.1. Interfejsy logiczne	15
6.2. Opis API	15
6.3. Ogólny flow UI	22
7. Stos technologiczny	23
8. Projekt testów	24
9. Analiza ryzyka	25
9.1. Lista ryzyk	25
9.2. Macierz ryzyka	26
9.3. Sposoby reagowania na ryzyko	26
10. Narzędzia wspomagające realizację projektu	27
11. Dokumentacja użytkownika	28
11.1. Instrukcja użycia	28
11.2. Ekrany przed zalogowaniem	29
11.2.1. Ekran startowy	29
11.2.2. Kontakt	29
11.2.3. Formularz tworzenia konta	30
11.2.4. Formularz logowania	31
11.3. Ekrany po zalogowaniu	31
11.3.1. Ekran startowy	31
11.3.2. Przelew	32

11.3.3. Lokata	32
11.3.4. Rachunki	33
11.3.5. Odbiorcy	33
11.3.6. Historia.....	33
11.3.7. Zdarzenia.....	34
11.3.8. Ustawienia.....	34
12. Najważniejsze moduły	35
12.1. Django	35
12.1.1. Modele	35
12.1.2. Widoki.....	36
12.2. Flutter	38
12.2.1. Ekran startowy	38
12.3. Komunikacja front-endu z back-endem	41
12.3.1. Klasa requestor.....	41

1. Wstęp

W dzisiejszych czasach nie wyobrażamy sobie realizowania operacji bankowych każdorazowo wybierając się do fizycznego oddziału banku. Założenie konta, przelew, utworzenie lokaty - wszystkie te operacje wykonujemy z domowego zacisza, najczęściej przez stronę internetową banku. Nasz innowacyjny projekt obejmuje utworzenie pierwszego w kraju kwitnącej cebuli systemu bankowości opartego nie o tradycyjne waluty, a o kryptowalutę Bitcoin. Niewątpliwą zaletą dydaktyczną projektu opartego o kryptowalutę jest faktyczna możliwość realizacji przelewów na zewnątrz tworzonego systemu, dzięki komunikacji z siecią blockchain.

1.1. Cel i założenia projektu

Celem projektu jest stworzenie systemu bankowości elektronicznej, podobnego w funkcjonowaniu do systemów znanych z tradycyjnych banków. System będzie udostępniał użytkownikom adres, na który mogą odbierać przelewy (zarówno wewnętrzne jak i zewnętrzne) oraz możliwość wykonania przelewu z posiadanych środków. Będzie pokrywał się możliwościami z tzw. portfelami Bitcoin, ale także rozszerzał je o dodatkowe funkcjonalności, na przykład utworzenie lokaty czy kontakt z pomocą techniczną.

2. Opis funkcjonalności

1. Utworzenie konta

- Użytkownik podaje login, hasło lub
- Użytkownik podaje login, hasło i e-mail
- Użytkownik podaje swój klucz prywatny / seed phrase lub
- System generuje klucz prywatny

2. Logowanie na konto

- Użytkownik podaje login / e-mail i hasło
- Wybór opcji „zapomniałem hasła” -> wysłanie wiadomości z nowym przez system jeśli e-mail jest w bazie

3. Zamknięcie konta

- Opcja zapisania klucza prywatnego
- Zamknięcie konta

4. Ekran główny

- Odnośniki na górze strony: przelew, lokata, rachunki, odbiorcy zdefiniowani, historia, kontakt, rejestr zdarzeń, ustawienia, wylogowywanie
- W części centralnej: adres publiczny, saldo, lokaty, ostatnie operacje
- Na dole strony: data ostatniego logowania

5. Przelew

- Na adres publiczny
- Na adres e-mail -> generowane nowe konto w systemie jeśli e-mail nie jest w bazie
- Wybór kwoty, daty, fee (podpowiedź optymalnego)
- Możliwość dodania tytułu w przypadku operacji wewnętrznej w systemie
- Konieczność potwierdzenia operacji kodem wysłanym na e-mail (jeśli został podany)

6. Lokata

- Wybór kwoty, czasu trwania

7. Historia transakcji

- Wyświetlenie historii operacji: tytuł, kwota, odbiorca, saldo po operacji, data zlecenia, księgowania, numer ID
- Opcja filtrowania
- Opcja powtórzenia operacji
- Eksport historii do CSV
- Eksport potwierdzenia przelewu do PDF

8. Kontakt

- Skrzynka wiadomości do kontaktu z administracją serwisu

9. Ustawienia

- Zmiana hasła
- Zmiana / dodanie adresu e-mail
- Zmiana motywu strony
- Opcja włączenia / wyłączenia powiadomień o przelewie (przychodzącym, do potwierdzenia) i nieudanym logowaniu
- Opcja włączenia / wyłączenia przesyłania wyciągów e-mailem

10. Rachunki

- Opcja stworzenia podrachunku
- Opcja zamknięcia podrachunku

11. Rejestr zdarzeń

- Daty logowań, adresy IP
- Daty operacji: przelewu, zmiany hasła itd.

12. Odbiorcy zdefiniowani

- Księga adresowa

3. Przypadki użycia

1. Tworzenie konta

- Aktorzy: Nowy użytkownik
- Warunki wstępne: Nowy użytkownik chce stworzyć konto
- Warunki końcowe: utworzenie konta
- Warunki końcowe w przypadku niepowodzenia: konto nie zostaje utworzone
- Scenariusz główny
 - System wyświetla formularz utworzenia konta
 - Użytkownik podaje login i hasło, opcjonalne jest podanie maila, w celu odzyskania hasła
 - System generuje klucz prywatny i publiczny. Publiczny zostaje podany do wiadomości użytkownika,
 - Możliwość dodania istniejącego portfela
- Scenariusz alternatywny
 - System wyświetla formularz utworzenia konta
 - Użytkownik podaje login i hasło, opcjonalne jest podanie maila, w celu odzyskania hasła
 - Weryfikacja kończy się niepowodzeniem
 - Zostaje podana informacja o niepoprawnych danych

2. Sprawdzenie stanu konta

- Aktorzy: Użytkownik
- Warunki wstępne: zalogowany użytkownik chce sprawdzić stan konta
- Warunki końcowe: Sprawdzenie stanu konta
- Warunki końcowe w przypadku niepowodzenia: użytkownik jest niezalogowany
- Scenariusz główny
 - Użytkownik loguje się na swoje konto,
 - Po zalogowaniu na panelu startowym widać aktualny stan portfela
 - Użytkownik sprawdza stan konta
- Scenariusz alternatywny
 - Logowanie na konto kończy się niepowodzeniem

3. Logowanie na konto

- Aktorzy: Użytkownik
- Warunki wstępne: Użytkownik chce zalogować się na konto
- Warunki końcowe: Logowanie zakończone powodzeniem
- Warunki końcowe w przypadku niepowodzenia: Logowanie zakończone niepowodzeniem
- Scenariusz główny
 - System wyświetla formularz zalogowania
 - Użytkownik podaje e-mail lub login i hasło
 - System weryfikuje dane
 - Użytkownik zostaje zalogowany
 - Wyświetla się panel startowy
- Scenariusz alternatywny
 - System wyświetla formularz zalogowania
 - Użytkownik podaje e-mail lub login i hasło
 - System weryfikuje dane
 - Weryfikacja kończy się niepowodzeniem
 - Wyświetla się ponownie ekran logowania

4. Wysyłanie przelewu

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik chce przelać walutę
- Warunki końcowe: Przelew zostaje wysłany
- Warunki końcowe w przypadku niepowodzenia: przelew nie zostaje wysłany
- Scenariusz główny
 - System wyświetla formularz przesłania waluty
 - Użytkownik podaje adres publiczny odbiorcy bądź adres e-mail, kwotę przelewu, datę i w przypadku transakcji w obrębie naszego banku- tytuł,
 - Wprowadzone dane są sprawdzane w systemie, w przypadku przelewu na e-mail, jeżeli nie jest on w bazie użytkowników, generowany jest e-mail z linkiem do utworzenia konta w naszym banku,
 - Pojawia się informacja o udanym przelewie, dane zostają wpisane do historii operacji
- Scenariusz alternatywny
 - System wyświetla formularz przesłania waluty
 - Użytkownik podaje adres publiczny odbiorcy bądź adres e-mail, kwotę przelewu, datę i w przypadku transakcji w obrębie naszego banku- tytuł,

- Wprowadzone dane są odrzucone przez system; powodem może być niewystarczający stan konta, błędny adres
- Pojawia się informacja o nieudanym przelewie

5. Przeglądanie historii transakcji i eksport do CSV

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik
- Warunki końcowe: Wyświetlona historia transakcji i wygenerowany jest plik CSV
- Warunki końcowe w przypadku niepowodzenia: Historia nie zostanie wyświetlona, a plik nie zostanie wygenerowany
- Scenariusz główny
 - Pojawia się panel historii transakcji, zawierająca informacje dotyczące przelewów przychodzących jak i wychodzących, takie jak kwota, data, odbiorca/nadawca, saldo po operacji
 - Użytkownik wybiera operacje do eksportu,
 - System generuje plik CSV
- Scenariusz alternatywny
 - Pojawia się panel historii transakcji, zawierająca informacje dotyczące przelewów przychodzących jak i wychodzących, takie jak kwota, data, odbiorca/nadawca, saldo po operacji
 - Użytkownik wybiera operacje do eksportu,
 - System generuje informację o wygaśnięciu sesji

6. Edycja konta- ustawienia

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik chce zmienić ustawienia
- Warunki końcowe: Zmiany zostają zastosowane
- Warunki końcowe w przypadku niepowodzenia: Zmiany zostają odrzucone
- Scenariusz główny
 - System wyświetla panel ustawień
 - Użytkownik ma możliwość zmiany pól takich jak: e-mail, zmiana motywu strony, powiadomienia, wiadomości przychodzące e-mail,
 - System weryfikuje dane, takie jak poprawność e-maila,
 - System generuje informację o poprawnych zmianach wprowadzonych do systemu
- Scenariusz alternatywny
 - System wyświetla panel ustawień

- Użytkownik ma możliwość zmiany pól takich jak: e-mail, zmiana motywu strony, powiadomienia, wiadomości przychodzące e-mail,
- System weryfikuje dane, takie jak poprawność e-maila,
- System generuje informację o niepoprawnych danych

7. Zakup lokaty

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik
- Warunki końcowe: Lokata zostaje zakupiona
- Warunki końcowe w przypadku niepowodzenia: Lokata nie zostaje zakupiona
- Scenariusz główny
 - System wyświetla panel zakupu lokaty
 - Użytkownik wybiera spośród dostępnych lokat, np. jednodniowa, tygodniowa, itd; wybiera kwotę do inwestycji,
 - System weryfikuje wprowadzone dane i stan konta
 - System tworzy lokatę i generuje informację o udanym założeniu lokaty
- Scenariusz alternatywny
 - System wyświetla panel zakupu lokaty
 - Użytkownik wybiera spośród dostępnych lokat, np. jednodniowa, tygodniowa, itd; wybiera kwotę do inwestycji,
 - System weryfikuje wprowadzone dane i stan konta
 - System odrzuca prośbę i generuje informację o odrzuconym żądaniu

8. Tworzenie podrachunku

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik
- Warunki końcowe: Utworzenie podrachunku
- Warunki końcowe w przypadku niepowodzenia: podrachunek nie zostaje utworzony
- Scenariusz główny
 - Pojawia się panel tworzenia podrachunku do jednego portfela
 - Użytkownik podaje nazwę podrachunku
 - System weryfikuje nazwę pod kontem niewłaściwych znaków
 - Podrachunek zostaje utworzony i wyświetlony stosowny komunikat
- Scenariusz alternatywny
 - Pojawia się panel tworzenia podrachunku do jednego portfela
 - Użytkownik podaje nazwę podrachunku
 - System weryfikuje nazwę pod kontem niewłaściwych znaków

- Podrachunek nie zostaje utworzony i wyświetlony zostaje stosowny komunikat

9. Kontakt z Pomocą

- Aktorzy: Użytkownik
- Warunki wstępne: Zalogowany użytkownik
- Warunki końcowe: Wysłanie wiadomości
- Warunki końcowe w przypadku niepowodzenia: Wiadomość nie zostaje wysłana
- Scenariusz główny
 - System wyświetla panel kontaktu
 - Użytkownik wybiera powód kontaktu z listy
 - Użytkownik wpisuje wiadomość
 - Wiadomość zostaje wysłana i wyświetlony stosowny komunikat
- Scenariusz alternatywny
 - System wyświetla panel kontaktu
 - Użytkownik wybiera powód kontaktu z listy
 - Użytkownik wpisuje wiadomość
 - Wiadomość nie zostaje wysłana z powodu wygaśnięcia sesji, wyświetlony stosowny komunikat

10. Odpowiedź na pytanie użytkownika

- Aktorzy: Administrator
- Warunki wstępne: Zalogowany administrator
- Warunki końcowe: Wysłanie wiadomości
- Warunki końcowe w przypadku niepowodzenia: Wiadomość nie zostaje wysłana
- Scenariusz główny
 - System wyświetla panel kontaktu
 - Administrator wybiera wiadomość z listy otrzymanych
 - Administrator wpisuje swoją odpowiedź
 - Wiadomość zostaje wysłana i wyświetlony stosowny komunikat
- Scenariusz alternatywny
 - System wyświetla panel kontaktu
 - Administrator wybiera wiadomość z listy otrzymanych
 - Administrator wpisuje swoją odpowiedź
 - Wiadomość nie zostaje wysłana z powodu wygaśnięcia sesji, wyświetlony stosowny komunikat

4. Wymagania niefunkcjonalne

1. Funkcjonalność

- Poprawne działanie na przeglądarkach internetowych w wersji nie starszej niż: Chrome 85, Firefox 81, Edge 85
- Minimalna obsługiwana rozdzielczość ekranu: 1366x768
- Nowoczesność i zgodność z aktualnymi standardami na rynku
- Budowa systemu za pomocą narzędzi umożliwiających rozwój aplikacji
- Skalowalność wydajnościowa

2. Estetyka

- Projekt logo serwisu
- Logo serwisu pojawiające się na stronie internetowej oraz w eksportowanych plikach PDF

3. Użyteczność

- Prostota i intuicyjność obsługi, pozwalająca na obsługę osobom nie posiadającym umiejętności technicznych
- Polska wersja językowa interfejsu

4. Niezawodność

- Uptime na poziomie minimum 90%
- Dostępność 24/7/365

5. Wydajność

- Umożliwienie korzystania z aplikacji równocześnie przez co najmniej 4 użytkowników na raz
- Maksymalny czas odpowiedzi systemu na akcję użytkownika nie dłuższy niż 4 sekundy
- Minimalizacja zużycia procesora, pamięci RAM i przestrzeni dyskowej

6. Bezpieczeństwo

- Ochrona przed nieautoryzowanym dostępem do danych użytkowników

7. Wsparcie

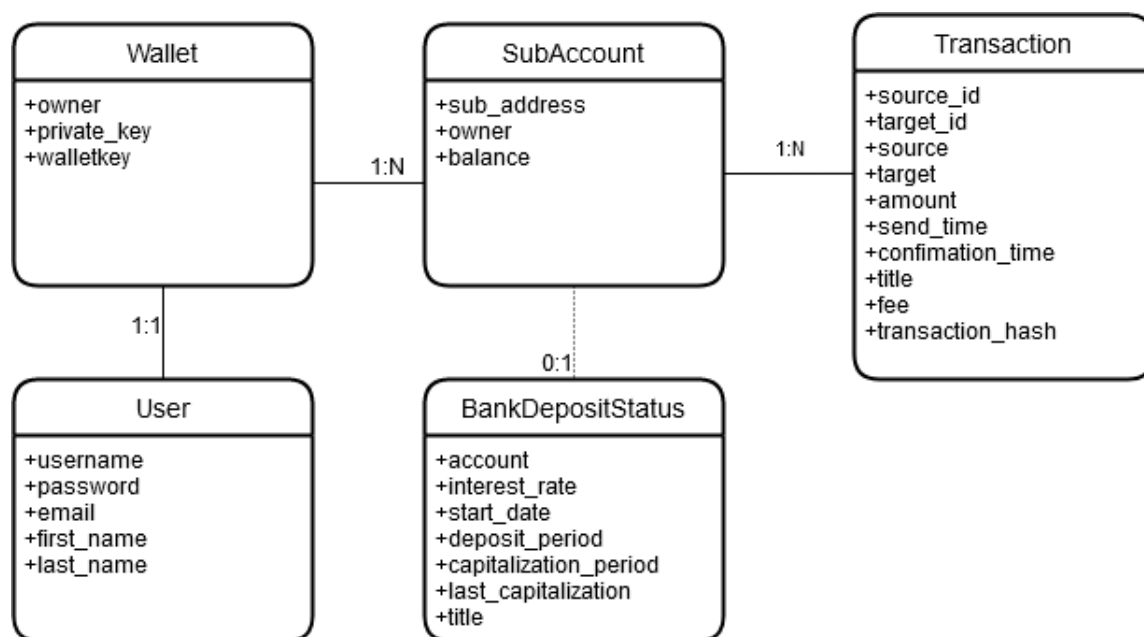
- Naprawa błędów krytycznych aplikacji w ciągu 48 godzin

8. Sposób wdrożenia

- Dostarczenie aplikacji w terminie do 17 stycznia 2020
- Dokumentacja zawierająca zrzuty ekranu z opisem użytkowania aplikacji
- Uwzględnienie w dokumentacji perspektywy użytkownika i programisty

5. Architektura systemu

5.1. Schemat bazy danych



6. Opis interfejsów

6.1. Interfejsy logiczne

- **Aplikacje backendowe <-> Baza danych PostgreSQL** - moduł Python `psycopg2`
- **Aplikacje backendowe <-> Blockchain** - moduł Python `bit`
- **Serwer HTTP <-> Aplikacje backendowe** - komunikacja za pomocą Mod WSGI
- **Aplikacje frontendowe <-> Aplikacje backendowe** - zakończenia REST API zdefiniowane przy użyciu Django REST Framework

6.2. Opis API

Zakończenia REST API pozwalające na dostęp do bazy danych. Aby uzyskać dostęp do zasobów, trzeba w nagłówku *Authorization* podać wartość *JWT {access token}*.

– Logowanie się/tworzenie access tokenu

- URL: `/auth/jwt/create/`
- Metoda: POST
- Request:
 - * `username`
 - * `password`
- Response:
 - HTTP 201 CREATED
 - * `refresh`
 - * `access`

– Tworzenie Usera

- URL: `/auth/users/`
- Metoda: POST

- Request:
 - * username
 - * password
 - * email
- Response:
HTTP 201 CREATED
 - * username
 - * email

– Aktualizacja Usera

- URL: /api/user/{username}/
- Metoda: PUT
- Request:
 - * username
 - * [email]
 - * [first_name]
 - * [last_name]
- Response:
HTTP 200 OK
 - * username
 - * email
 - * first_name
 - * last_name

– Pobieranie danych aktualnie zalogowanego Usera

- URL: /api/user/
- Metoda: GET
- Response:
HTTP 200 OK
 - * username
 - * email
 - * first_name
 - * last_name

– Tworzenie Walleta dla aktualnie zalogowanego Usera

- URL: /api/wallet/

- Metoda: POST
- Request:
 - * [private_key]
- Response:
HTTP 201 CREATED
 - * owner
 - * private_key
 - * wallet_address

– **Pobieranie adresu portfela dla aktualnie zalogowanego Usera**

- URL: /api/wallet/
- Metoda: GET
- Response:
HTTP 200 OK
 - * wallet_address

– **Pobieranie adresu portfela dla aktualnie zalogowanego Usera**

- URL: /api/wallet/
- Metoda: GET
- Response:
HTTP 200 OK
 - * wallet_address

– **Tworzenie nowego podkonta dla zalogowanego Usera**

- URL: /api/subacc/
- Metoda: POST
- Response:
HTTP 201 CREATED
 - * owner
 - * balance
 - * sub_address

– **Pobieranie listy podkont zalogowanego Usera**

- URL: /api/subacc/
- Metoda: GET

- Response:

HTTP 200 OK

Lista:

- * owner
- * balance
- * sub_address

– **Pobieranie danych podkonta o podanym adresie**

- URL: /api/subacc/{sub_address}/

- Metoda: GET

- Response:

HTTP 200 OK

- * owner
- * balance
- * sub_address

– **Tworzenie lokaty dla zalogowanego Usera**

- URL: /api/deposit/

- Metoda: POST

- Request:

- * account – adres podkonta
- * title
- * [interest_rate]
- * [start_date]
- * [deposit_period]
- * [capitalization_period]

- Response:

HTTP 201 CREATED

- * account – adres podkonta
- * title
- * interest_rate
- * start_date
- * deposit_period
- * capitalization_period
- * last_capitalization

– Pobieranie listy lokat zalogowanego Usera

- URL: /api/deposit/
- Metoda: GET
- Response:
HTTP 200 OK
Lista:
 - * account – adres podkonta
 - * title
 - * interest_rate
 - * start_date
 - * deposit_period
 - * capitalization_period
 - * last_capitalization

– Pobieranie danych lokaty o podanym adresie podkonta

- URL: /api/deposit/{sub_address}/
- Metoda: GET
- Response:
HTTP 200 OK
 - * owner
 - * balance
 - * sub_address

– Tworzenie transakcji

- URL: /api/trans/
- Metoda: POST
- Request:
 - * source – adres podkonta użytkownika
 - * target – adres podkonta lub adres portfela Bitcoin
 - * amount
 - * title
 - * fee
- Response:
HTTP 201 CREATED
 - * source – adres podkonta użytkownika

- * target – adres podkonta lub adres portfela Bitcoin
- * amount
- * title
- * fee
- * sent_time
- * transaction_hash

– **Pobieranie listy transakcji zalogowanego Usera**

- URL: /api/trans/
- Metoda: GET
- Response:

HTTP 200 OK

Lista:

- * source – adres podkonta użytkownika
- * target – adres podkonta lub adres portfela Bitcoin
- * amount
- * title
- * fee
- * sent_time
- * transaction_hash

– **Pobieranie listy transakcji wychodzących zalogowanego Usera**

- URL: /api/trans/out/
- Metoda: GET
- Response:

HTTP 200 OK

Lista:

- * source – adres podkonta użytkownika
- * target – adres podkonta lub adres portfela Bitcoin
- * amount
- * title
- * fee
- * sent_time
- * transaction_hash

– **Pobieranie listy transakcji przychodzących zalogowanego Usera**

- URL: /api/trans/in
- Metoda: GET
- Response:
HTTP 200 OK
Lista:
 - * source – adres podkonta użytkownika
 - * target – adres podkonta lub adres portfela Bitcoin
 - * amount
 - * title
 - * fee
 - * sent_time
 - * transaction_hash

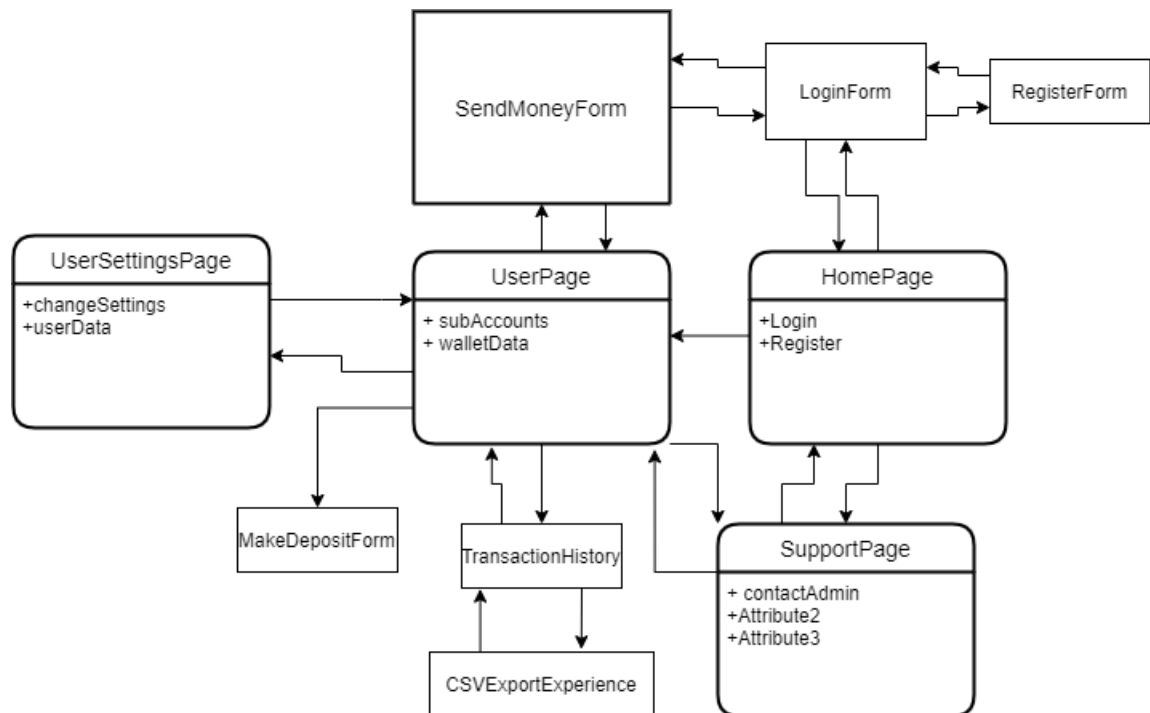
– **Pobieranie danych transakcji o podanym hashu**

- URL: /api/deposit/{transaction_hash}/
- Metoda: GET
- Response:
HTTP 200 OK
 - * source – adres podkonta użytkownika
 - * target – adres podkonta lub adres portfela Bitcoin
 - * amount
 - * title
 - * fee
 - * sent_time
 - * transaction_hash

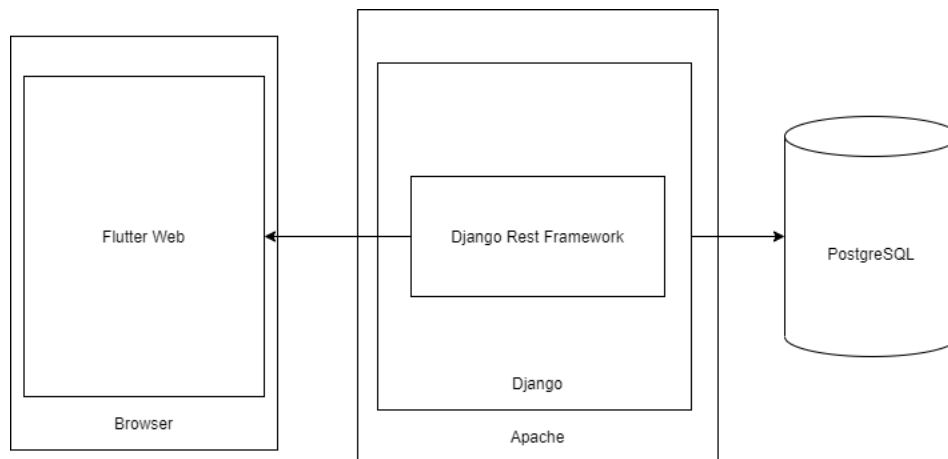
– **Pobieranie historii logowania aktualnego Usera**

- URL: /api/login_history/
- Metoda: GET
- Response:
HTTP 200 OK
 - * user
 - * action
 - * date
 - * ip_address

6.3. Ogólny flow UI



7. Stos technologiczny



Użyte technologie:

- **Apache** - serwer webowy zajmujący się odbieraniem i odpowiadaniem na żądania HTTP
- *mod_wsgi* - interfejs łączący aplikacje backendu Django z serwerem Apache
- **Django** - framework do tworzenia aplikacji webowych w języku Python; zostanie wykorzystany do stworzenia aplikacji backendu; umożliwia komunikację z bazą danych przy pomocy kodu Pythona
- **Django Rest Framework** - framework umożliwiający tworzenie REST API do komunikacji backendu z frontendem; w prosty sposób integruje się z Django i korzysta z jego funkcjonalności
- **bit** - biblioteka Python do obsługi transakcji w bitcoinach
- **PostgreSQL** - relacyjna baza danych
- **Flutter Web** - framework umożliwiający użycie zestawu narzędzi Flutter do tworzenia aplikacji webowych; zostanie wykorzystany do stworzenia klienta przeglądarkowego aplikacji
- **RabbitMQ**
- **Docker**

8. Projekt testów

- **Testy jednostkowe** – testowanie metod we wszystkich modułach z wykorzystaniem wbudowanych bibliotek – unittest dla Django oraz flutter_test dla Flutter.
- **Testy integracyjne** – testowanie komunikacji i odpowiedniej współpracy między modułami, w szczególności między frontendem i backendem oraz backendem i bazą danych. Również zostaną wykorzystane moduły unittest oraz flutter_test.
- **Testy funkcjonalne** – testowanie funkcjonalności aplikacji oraz scenariuszy użycia za pomocą Selenium oraz manualnego testowania.
- **Testy wydajnościowe** – testowanie czasu odpowiedzi oraz obciążenia z użyciem Locust.

9. Analiza ryzyka

9.1. Lista ryzyk

1. Skomplikowana integracja z siecią blockchain, niedostateczna wiedza na temat jej działania.
2. Zachowanie spójności między operacjami wykonywanymi lokalnie, w bazie serwisu, a tymi w sieci blockchain.
3. Brak doświadczenia w wykorzystywanych technologiach - dla większości członków projektu tworzenie aplikacji webowych jest czymś nowym.
4. Wybór złych technologii lub bibliotek, niosący za sobą konieczność przepisywania oprogramowania.
5. Założona funkcjonalność może okazać się niemożliwa do realizacji.
6. Niedotrzymanie terminów - trudność oszacowania nakładu czasowego potrzebnego na projekt.
7. Niewykrycie błędów podczas procesu testowania. Im wcześniej zostanie wykryty błąd, tym łatwiej go usunąć.
8. Trudność zapewnienia bezpieczeństwa danych - odporność na włamania, ataki typu sql injection.
9. Trudności w sprawdzeniu skalowalności.

9.2. Macierz ryzyka

Prawdopodobieństwo \ Konsekwencje	niewielkie	średnie	poważne
niewielkie			-Wybór złych technologii
średnie	-Sprawdzenie skalowalności	-Niedostateczna wiedza na temat blockchain	-Założona funkcjonalność nie będzie możliwa do zrealizowania -Niedotrzymanie terminu
wysokie	-Brak doświadczenia w wybranych technologiach	-Zachowanie spójności między operacjami lokalnymi i na blockchainie -Niewykrycie (niekrytycznych) błędów	-Zapewnienie bezpieczeństwa danych

9.3. Sposoby reagowania na ryzyko

1. **Unikanie ryzyka** - Dokonanie szczegółowej i wszechstronnej analizy dostępnych technologii pozwoli na wybór optymalnej i zapewniającej całkowitą realizację założeń projektu (uwzględniając potencjał rozbudowy). Dzięki planowaniu prac z wyprzedzeniem, nawet przy niedoszacowaniu czasu potrzebnego na implementację, możliwe będzie dotrzymanie terminów. W unikaniu ryzyka pomocne są również analiza statyczna kodu i testowanie, w zakresie bezpieczeństwa - zlecenie audytu zewnętrznego (w „rzeczywistych” projektach).
2. **Przeniesienie ryzyka** - wykorzystując gotowe, dobrze i ustawicznie testowane narzędzia (frameworki) część ryzyka, zwłaszcza w obszarze bezpieczeństwa, przenosi się na podmioty dostarczające dane technologie.
3. **Akceptacja ryzyka** - niektóre zagrożenia nie są możliwe do całkowitej eliminacji.

10. Narzędzia wspomagające realizację projektu

- **GitHub**: wykorzystywany do przechowywania i inspekcji kodu, dyskusji, przydzielania zadań, śledzenia błędów i wkładu każdej z osób.
- **Overleaf**: edytor \LaTeX on-line, umożliwiający współpracę kilku osób jednocześnie.
- **Facebook Messenger**: komunikator. Wykorzystywany zarówno w formie tekstowej, ale także w formie rozmów głosowych podczas intensywnych prac nad projektem.

Ze względu na ograniczoną skalę wykonywanego przez nas projektu, ograniczamy się do wymienionych wyżej rozwiązań. W przypadku znacznie większych projektów, zasadne wydaje się wykorzystanie dodatkowego oprogramowania, na przykład **JIRA** - do śledzenia i planowania pracy czy **Confluence** - do przechowywania treści pojawiających się w trakcie rozwoju projektu w jednym miejscu: członków zespołu i ich ról, haseł, wymagań i innej dokumentacji.

11. Dokumentacja użytkownika

11.1. Instrukcja urochienia

Potrzebne narzędzia:

docker pip virtualenv PostgreSQL

Należy stworzyć wirtualne środowisko dla serwera Django, uruchomić je komendą `source path/to-venv/bin/activate` (Linux) lub `path\to\venv\Scripts\activate.bat` (Windows), a następnie zainstalować konieczne pakiety z pliku `requirements.txt` komendą `pip install -r requirements.txt`.

Uruchomienie obrazu RabbitMQ w dockerze

Przed uruchomieniem serwera aplikacji WBB konieczne jest włączenie pomocniczego programu RabbitMQ. W tym celu wykorzystamy przygotowany przez społeczność RabbitMQ obraz dockera:

```
docker run -d -p 5672:5672 rabbitmq
```

Uruchomienie serwera Django

Uruchom stworzone wcześniej środowisko wirtualne `virtualenv`, a następnie z poziomu folderu `../bank_backend` wykonaj komendy:

1. `python manage.py makemigrations`
2. `python manage.py migrate`
3. `python manage.py runserver`

Od razu potem powinieneś uruchomić programy- planery zadań okresowych

Serwer aplikacji WBB do poprawnego działania polega na zadaniach wykonywanych w tle.

- Aby je uruchomić, po wykonaniu wszystkich poprzednich kroków, włącz nowe okno terminala, przejdź do folderu `../bank_backend`, uruchom stworzone wcześniej środowisko wirtualne `virtualenv`, a następnie wykonaj komendę: `celery -A bank_backend worker`
- Następnie włącz drugie okno terminala, wykonaj te same kroki zaczynając od przejścia do folderu `../bank_backend`, tym razem wykonaj inną komendę: `celery -A bank_backend beat`

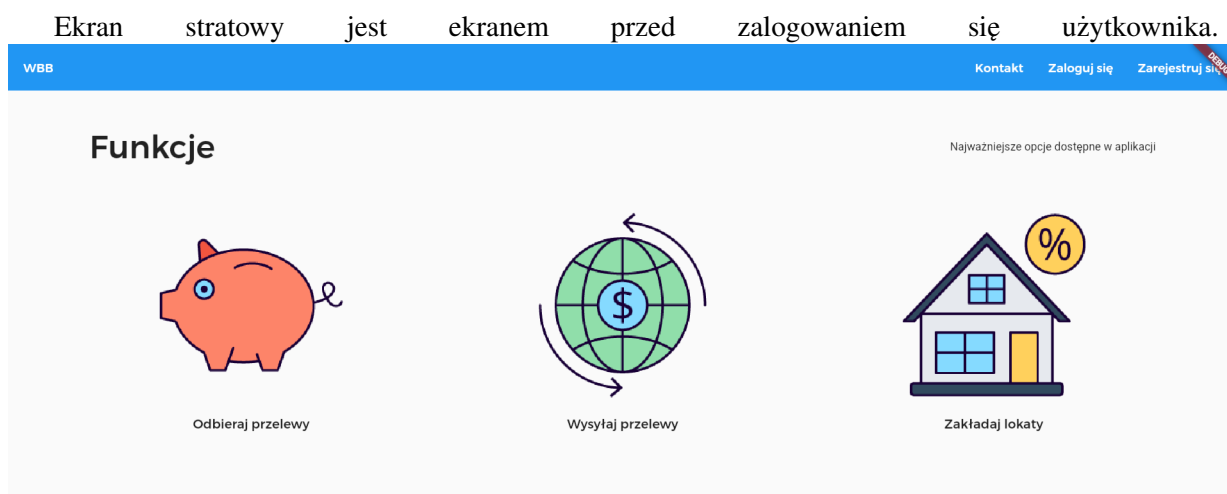
Uruchamianie Fluttera Przejdź do folderu `../bank` i wykonaj:

- `flutter channel beta`

- flutter config --enable-web
- flutter run -d chrome

11.2. Ekrany przed zalogowaniem

11.2.1. Ekran startowy



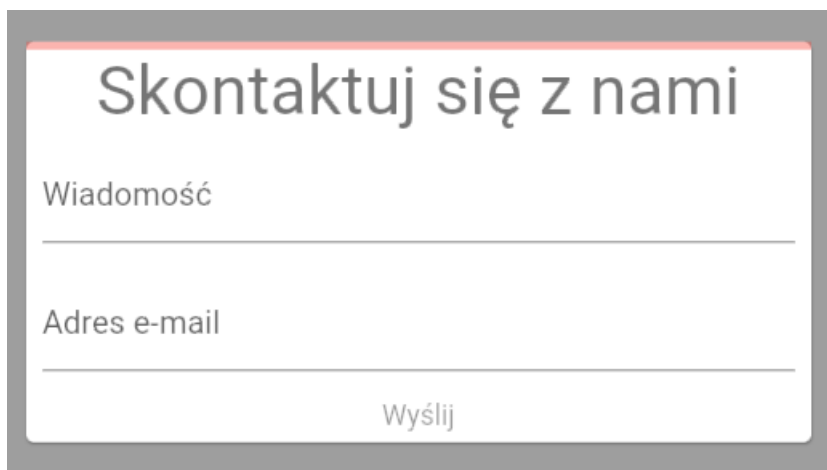
W górnym pasku znajdują się kolejno:

- Logo banku WBB, po wciśnięciu zostajemy przekierowani na stronę główną,
- Kontakt, po wciśnięciu pojawia się formularz kontaktowy,
- Zaloguj się, po wciśnięciu pojawia się formularz zalogowania,
- Zarejestruj się, po wciśnięciu pojawia się formularz utworzenia konta.

Poniżej znajduje się grafika z informacjami o najważniejszych funkcjach Banku, czyli obsługa przelewów i lokat.

11.2.2. Kontakt

Formularz kontaktowy umożliwia wysłanie zapytania. Należy podać adres mailowy oraz treść zapytania. Domyślnie administrator odpowie mailowo.



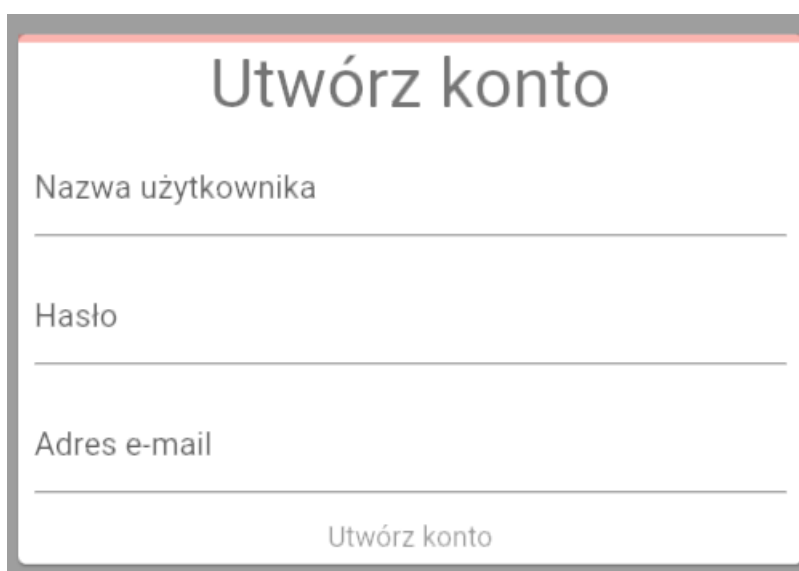
Skontaktuj się z nami

Wiadomość

Adres e-mail

Wyślij

11.2.3. Formularz tworzenia konta



Utwórz konto

Nazwa użytkownika

Hasło

Adres e-mail

Utwórz konto

Formularz przyjmuje nazwę użytkownika, hasło oraz adres e-mail. Wymagania co do hasła to standardowe wymagania Django, tzn. nie może być podobne do nazwy użytkownika lub maila, itp. Przykładowo wypełniony formularz:

- Przelew, przekierowuje do formularza przelewu,
- Lokata, przekierowuje do formularza lokaty,
- Rachunki, przekierowuje do widoku podrachunków oraz umożliwia otwarcie bądź zamknięcia obecnych rachunków,
- Odbiorcy, przekierowuje do książki kontaktowej,
- Historia, przekierowuje do widoku historii transakcji,
- Kontakt, przekierowuje do formularza kontaktowego, identycznego jak ten przed zalogowaniem,
- Zdarzenia, przekierowuje do widoku zdarzeń typu ostatnie logowanie itp,
- Ustawienia, przekierowuje do widoku ustawień i umożliwia zmianę podstawowych informacji,
- Wyloguj, wylogowuje z konta.

11.3.2. Przelew

Widok przelew umożliwia wykonanie przelewu na e-mail lub adres portfela.

Typ przelewu

Na adres BTC Na adres e-mail

Dane odbiorcy

Adres e-mail
mwieclawek@student.agh.edu.pl

Tytuł
tt

Parametry

Kwota
1

Konto
Główne

Czas przelewu

Data
2021-01-17

Godzina
20:58

Regulowanie fee

Należy podać adres, tytuł, kwotę, konto z którego mają zostać pobrane środki, datę i godzinę przelewu oraz ustawić fee.

11.3.3. Lokata

Widok umożliwia utworzenie lokaty.

Typ lokaty

3,2% (czas: 3 dni) (kapitalizacja: 1 dzień)

3,8% (czas: 6 dni) (kapitalizacja: 1 dzień)

4,2% (czas: 14 dni) (kapitalizacja: 1 dzień)

Parametry

Nazwa

wpisz nazwę lokaty

Kwota

wpisz kwotę

Z Konta

Główne

Czas rozpoczęcia lokaty

Data

wybierz datę operacji

Godzina

wybierz godzinę operacji

Widok umożliwiający założenie lokaty. Można wybrać jedną z trzech dostępnych opcji, oraz wskazać rachunek i kwotę przeznaczoną do zainwestowania.

11.3.4. Rachunki

Zarządzanie rachunkami

Zarządzanie rachunkami

Adres

Widok umożliwia otwarcie lub zamknięcie podrachunku oraz importować pliki CSV lub PDF.

11.3.5. Odbiorcy

Lista odbiorców

Lista odbiorców				
<input type="checkbox"/>	Tytuł	Adres		
<input type="checkbox"/>	Kontrahent 1	kolega@wp.pl		
<input type="checkbox"/>	Kontrahent 2	1836NGficqz5cSHWdd1qcEuqaEmFAHJRwC		

Widok umożliwia podgląd zdefiniowanych odbiorców.

11.3.6. Historia

Historia przelewów

Historia przelewów										
<input type="checkbox"/>	Tytuł	Kwota	Saldo po operacji	Data zlecenia	ID	Tracking	Adres			
<input type="checkbox"/>	Pizza z szynką i pieczarkami	-7	118.5	2021-14-11	4		1836NGficqz5cSHWdd1qcEuqaEmFAHJRwC			
<input type="checkbox"/>	Transfer	-30	125.5	2021-14-09	3		kolega@wp.pl			
<input type="checkbox"/>	Pizza z szynką i pieczarkami	-7	155.5	2021-14-05	2		1836NGficqz5cSHWdd1qcEuqaEmFAHJRwC			
<input type="checkbox"/>	Transfer środków	3.5	162.5	2021-14-02	1		1352NGficqz5cSHWdd1qcEuqaEmFAHJRwC			
<input type="checkbox"/>	Zasilanie	159	159	2021-14-01	0		1352NGficqz5cSHWdd1qcEuqaEmFAHJRwC			

Widok umożliwia sprawdzenie historii transakcji oraz eksport wybranych rekordów do plików CSV lub PDF.

Historia transakcji



Tytuł	Kwota	Saldo po operacji	Data zlecenia	ID	Tracking	Adres
Pizza z szynką i pieczarkami	-7	118.5	2021-14-11	4	https://www.blockchain.com/btc/tx/172a8d141cb0861166b2eb766e6444544c267797b71939226802721ab2684df6	1836NGficqz5cSHWdd1qcEuqaEmFAHJrWc
Transfer	-30	125.5	2021-14-09	3	https://www.blockchain.com/btc/tx/172a8d141cb0861166b2eb766e6444544c267797b71939226802721ab2684df6	kolega@wp.pl
Pizza z szynką i pieczarkami	-7	155.5	2021-14-05	2	https://www.blockchain.com/btc/tx/172a8d141cb0861166b2eb766e6444544c267797b71939226802721ab2684df6	1836NGficqz5cSHWdd1qcEuqaEmFAHJrWc

11.3.7. Zdarzenia

Widok umożliwia podgląd zdarzeń na koncie, takich jak np ostatnie logowanie.

<input type="checkbox"/>	Zdarzenie	Data	Adres IP
<input type="checkbox"/>	user_logged_in	2021-01-16T18:07:35.450Z	127.0.0.1

11.3.8. Ustawienia

Widok umożliwia zmianę podstawowych ustawień, takich e-mail itp.

E-mail
przykladowyemail@gmail.com

Hasło
Pozostaw puste, jeśli chcesz zostać przy aktualnym

Personalizacja

Jasny motyw



Powiadomienia e-mail

Przelew przychodzący



Nieudane logowanie



Bezpieczeństwo

Zatwierdzanie przelewów e-mailem



Przesyłanie wyciągów e-mailem



Zapisz

12. Najważniejsze moduły

12.1. Django

Do back-endu została użyta technologia Django (BSD), która jest napisana w Pythonie. Wybraliśmy ją dlatego, ponieważ

1. Realizuje ona wzorzec model-template-view,
2. Współpracuje z Apache poprzez WSGI,
3. Posiada mapowanie obiektowo-relacyjne wysokiego poziomu.

12.1.1. Modele

Modele w Django odpowiadają za ORM. W naszej aplikacji są to:

Wallet

```
class Wallet(models.Model):
    owner = models.OneToOneField(
        User, on_delete=models.CASCADE, primary_key=True)
    private_key = models.CharField(max_length=52) # WIF format
    wallet_address = models.CharField(max_length=34)
```

Klasa odpowiedzialna za portfel. Posiada pole właściciel, klucz prywatny w formacie WIF oraz adres portfela.

SubAccount

```
class SubAccount(models.Model):
    sub_address = models.SlugField(max_length=DEFAULT_SLUG_LENGTH, default='')
    # Having a Wallet object you can access its SubAccounts with Wallet.subaccount_set.all()
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    balance = models.DecimalField(max_digits=22, decimal_places=9, blank=True, default=0)
    def save(self, *args, **kwargs):
        """ Add Slug creating/checking to save method. """
        slug_save(self) # call slug_save, listed below
        super(SubAccount, self).save(*args, **kwargs)
```

Klasa odpowiedzialna za podrachunki. Posiada pola właściciel, adres oraz saldo.

BankDeposit

```
class BankDeposit(models.Model):
    # Having a SubAccount object you can access its BankDeposit info with SubAccount.bankdeposit;
    # CAREFUL! Accessing BankDeposit of a SubAccount that is not a deposit will throw ObjectDoesNotExist! This is intended behaviour!
    # Accessing SubAccount.bankdeposit throws ObjectDoesNotExist, which tells us this is not a Deposit account
    account = models.OneToOneField(
        SubAccount, on_delete=models.CASCADE, primary_key=True)
    interest_rate = models.DecimalField(default=0.314, max_digits=5, decimal_places=3) # Always present, set to default
    start_date = models.DateTimeField(auto_now=True) # Always present, set to default
    deposit_period = models.DurationField(default=timedelta(seconds=35)) # Always present, set to default
    capitalization_period = models.DurationField(blank=True, null=True) # Capitalization is optional
    last_capitalization = models.DateTimeField(blank=True, null=True) # Capitalization is optional
    title = models.CharField(max_length=256)
```

Klasa BankDeposit odpowiada za lokaty. Posiada pola konto, procent, dzień rozpoczęcia, czas trwania, okres kapitalizacji, ostatnią kapitalizację oraz tytuł.

Transaction

```
class Transaction(models.Model):
    # Having a SubAccount objects you can access its related transactions with SubAccount.(outgoing/incoming)_transactions.all()
    source_id = models.ForeignKey(SubAccount, on_delete=models.PROTECT, related_name='outgoing_transactions', blank=True, null=True)
    target_id = models.ForeignKey(SubAccount, on_delete=models.PROTECT, related_name='incoming_transactions', blank=True, null=True)
    source = models.CharField(max_length=34, blank=True, null=True)
    target = models.CharField(max_length=34, blank=True, null=True)
    amount = models.DecimalField(max_digits=22, decimal_places=9)
    send_time = models.DateTimeField(auto_now=True)
    confirmation_time = models.DateTimeField(blank=True, null=True)
    title = models.CharField(max_length=256)
    fee = models.DecimalField(max_digits=22, decimal_places=15, blank=True, null=True) # Nullable, miner/bank fee for the transaction
    transaction_hash = models.CharField(max_length=64, blank=True, null=True) # Nullable, blockchain ID of the transaction
```

Klasa Transaction odpowiada za transakcje. Posiada pola identyfikator nadawcy oraz odbiorcy, kwotę, czas transakcji, czas potwierdzenia, tytuł i opłatę.

LoginRecord

```
class LoginRecord(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    action = models.CharField(max_length=64)
    date = models.DateTimeField(auto_now=True)
    ip_address = models.GenericIPAddressField(null=True)
```

Klasa LoginRecord odpowiada jednemu zalogowaniu użytkownika do serwisu. Posiada pola użytkownik, akcja, data i adres IP (w przypadku lokalnego serwera adres 127.0.0.1).

12.1.2. Widoki

Widoki odpowiadają za logikę aplikacji. Są to odpowiednio:

- ListView
- DetailView
- WalletListView
- SubAccountListView
- SubAccountDetailView

- BankDepositListView
- BankDepositDetailView
- TransactionListView
- TransactionOutgoingListView
- TransactionIncomingListView
- TransactionDetailView
- LoginRecordListView

Każda z klas ma zdefiniowaną metodę `get` i inne metody w zależności od potrzeb. Służą one do prezentowania użytkownikowi informacji. Pobierają je z modelu, a następnie przekazują do front-endu.

12.2. Flutter

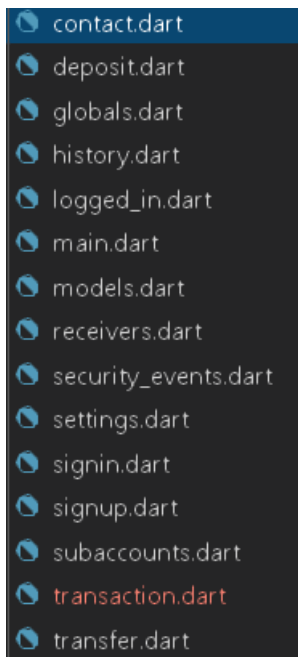
Do stworzenia programu od strony wizualnej, została użyta technologia Flutter od Google, która zapewnia renderowanie na niskim poziomie za pomocą silnika graficznego Skia.

12.2.1. Ekran startowy

```
void main() => runApp(MainApp());

class MainApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Wirtualny Bank Bitcoinów',
      theme: ThemeData(
        visualDensity: VisualDensity.adaptivePlatformDensity,
        brightness: light_theme ? Brightness.light : Brightness.dark,
      ), // ThemeData
      routes: {
        '/': (context) => MainPage(),
        '/signin': (context) => SignInApp(),
        '/signup': (context) => SignUpApp(),
        '/main': (context) => MainApp(),
        '/contact': (context) => ContactApp(),
      },
    ); // MaterialApp
  }
}
```

Domyślnie zostaje uruchomiona `MainApp()`. Parametr `routes` definiuje jaką aplikacja powinna zostać uruchomiona w zależności od adresu, np adres kończący się na `"/signin"` uruchamia `SignInApp()`. Następnie strona zbudowana jest z Widgetów, czyli klas dziedziczących po `StatelessWidget`.



Zasadniczo, górny pasek jest taki sam niezależnie od aktualnego widoku, jednakże różni się w zależności od tego czy użytkownik jest zalogowany czy nie. Paski są zdefiniowane w pliku `globals.dart`

```
class MenuNotLogged extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var screenSize = MediaQuery.of(context).size;

    return Container(
      color: Colors.blue,
      child: Padding(
        padding: EdgeInsets.all(20),
        child: Row(
          children: [
            InkWell(
              onTap: () {
                Navigator.of(context).pushNamed('/main');
              },
              child: Text('WB8',
                style: GoogleFonts.montserrat(
                  fontSize: 15,
                  fontWeight: FontWeight.w500,
                  color: Colors.white)), // Text
            ), // InkWell
            // SizedBox(width: screenSize.width / 50),
            Expanded(
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [],
              ), // Row
            ), // Expanded
            InkWell(
              onTap: () {
                Navigator.of(context).pushNamed('/contact');
              },
            ), // InkWell
          ],
        ),
      ),
    );
  }
}
```

Każdy kolejny przycisk jest tworzony analogicznie jak wyżej. Podajemy jako argument "po wciśnięciu" funkcję, która dokleja do adresu postfix. Jak zostało wcześniej wspomniane, taki zabieg uruchamia aplikację zdefiniowaną w `zoutes` i tym sposobem uzyskujemy przejście do kolejnego widoku. Następnie zostaje podana zawartość przycisku, najczęściej jest to tekst. Analogicznie z każdym kolejnym przyciskiem.

12.3. Komunikacja front-endu z back-endem

Aby serwis był w pełni funkcjonalny konieczna jest komunikacja back-endu z front-endem. W tym przypadku należało kierować zapytania z Fluttera do Django na adres serwera Django. Adres serwera lokalnego to `http://127.0.0.1` oraz numer portu. U nas był to port 8000 lub 8080. Przed uruchomieniem front-endu należy odpowiednio skonfigurować plik `api_requests`.

```
import 'package:http/http.dart' as http;
import 'dart:async';
import 'dart:convert';
import 'package:bank/models.dart';

const String server_port = '8080';
const String server_address = 'http://127.0.0.1';
```

12.3.1. Klasa requestor

Jest to najważniejsza klasa służąca do wysyłania zapytań do serwera oraz otrzymywania odpowiedzi.

Dokładny opis zakończeń API w dziale **6.2. Opis API**.

Przykładowa metoda wysyłająca zapytanie konieczne do zalogowania.

```
Requestor(
  {this.tokenData,
   this.serverAddress = server_address,
   this.serverPort = server_port});

Future<TokenData> login(String username, String password) async {
  final http.Response response = await http.post(
    '${this.serverAddress}:${this.serverPort}/auth/jwt/create',
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8'
    },
    body: jsonEncode(<String, String>{
      'username': username,
      'password': password,
    })),
  );
```

Na tym etapie są łapane błędy związane z siecią. Szczegóły w rozdziale 6.

Bibliografia

- [1] Jimmy Song. *Programming Bitcoin: Learn How to Program Bitcoin from Scratch*. O'Reilly Media, 2019.