Bezpieczeństwo usług chmurowych

5 kwietnia 2025

Spis treści

1 Wprowadzenie **3**
- 1.1 Opis aplikacji . . . 3
- 1.2 Wymagania systemowe . . . 3

2 Idea aplikacji i przykłady zastosowania **3**
- 2.1 Idea aplikacji . . . 3
- 2.2 Przykłady zastosowania . . . 3

3 Funkcjonalności **4**
- 3.1 Rejestracja użytkownika . . . 4
- 3.2 Logowanie użytkownika . . . 4
- 3.3 Logowanie administratora . . . 4
- 3.4 Aktywacja konta poprzez e-mail . . . 4
- 3.5 Uwierzytelnianie dwuskładnikowe (2FA) . . . 4

4 Bezpieczeństwo i zabezpieczenia **6**
- 4.1 Logowanie przez Google, Facebook i Microsoft . . . 6
  - 4.1.1 Ograniczenia dostępu . . . 6
  - 4.1.2 Implementacja w aplikacji . . . 6
  - 4.1.3 Integracja z aplikacją . . . 7
- 4.2 OAuth 2.0 . . . 7
  - 4.2.1 Rejestracja aplikacji w dostawcy tożsamości . . . 7
  - 4.2.2 Inicjowanie procesu autoryzacji . . . 8
  - 4.2.3 Uwierzytelnianie użytkownika . . . 8
  - 4.2.4 Generowanie kodu autoryzacyjnego . . . 8
  - 4.2.5 Wymiana kodu autoryzacyjnego na token dostępu . . . 8
  - 4.2.6 Używanie tokenu dostępu . . . 8
  - 4.2.7 Wylogowanie i unieważnienie tokenów . . . 8
- 4.3 Zabezpieczenia wynikające z aktywacji konta poprzez e-mail . . . 9
  - 4.3.1 Rejestracja użytkownika . . . 9
  - 4.3.2 Aktywacja konta . . . 9
  - 4.3.3 Bezpieczeństwo procesu aktywacji . . . 9
- 4.4 Bezpieczne przechowywanie haseł . . . 10
- 4.5 Wykorzystanie pliku .env . . . 10
- 4.6 Tworzenie administratora . . . 11
- 4.7 Zabezpieczenie przed słabymi hasłami . . . 12
  - 4.7.1 Konfiguracja weryfikatorów haseł . . . 12
- 4.8 Logowanie dwuetapowe 2FA . . . 12
  - 4.8.1 Etapy procesu logowania dwuetapowego . . . 12
  - 4.8.2 TOTP (Time-based One-Time Password) . . . 13
  - 4.8.3 Kody zapasowe (Backup Tokens) . . . 13
  - 4.8.4 Zabezpieczenia dodatkowe . . . 13
  - 4.8.5 Zainstalowane aplikacje (INSTALLED_APPS) . . . 13
  - 4.8.6 Middleware . . . 13
- 4.9 Wykorzystanie `social_django` . . . 13

5 Scieżki URL **15**

6 Hosting aplikacji na PythonAnywhere **15**
6.1 Logowanie i monitoring . . . 15
6.2 WSGI . . . 16
6.3 Wdrażanie nowej wersji aplikacji . . . 16

7 Testy **17**
7.1 Testowanie bezpieczeństwa systemu logowania . . . 17
7.1.1 Testy rejestracji użytkownika . . . 17
7.1.2 Testy procesu wylogowania . . . 17
7.1.3 Testy dostępu do panelu użytkownika . . . 17
7.1.4 Testy e-maili aktywacyjnych . . . 17
7.2 Scenariusze testowe . . . 18

8 Diagramy **20**
8.1 Diagram sekwencji . . . 20
8.1.1 Opis funkcjonalności . . . 21
8.2 Diagram przypadków użycia . . . 21
8.2.1 Aktorzy systemu . . . 21
8.3 Diagram klas . . . 22
8.3.1 Opis klas . . . 22
8.3.2 Relacje . . . 22
8.3.3 Metody kluczowe . . . 23

9 Wdrożenie **23**
9.1 Dane: . . . 23
9.1.1 Środowisko i technologie . . . 23
9.1.2 Hostowanie . . . 23
9.1.3 Zarządzanie kodem źródłowym . . . 23
9.1.4 Proces wdrożenia . . . 23
9.1.5 Testowanie . . . 23

10 Podział pracy **24**
10.1 Aleksandra Lewandowska . . . 24
10.2 Kacper Malik . . . 24
10.3 Franciszek Łajszczak . . . 24
10.4 Daniel Pietruczyk-Phan . . . 24

1 Wprowadzenie

1.1 Opis aplikacji

Aplikacja webowa umożliwia użytkownikom rejestrację, logowanie oraz zarządzanie bezpieczeństwem konta. Proces rejestracji wymaga potwierdzenia adresu e-mail, a logowanie jest możliwe zarówno dla zwykłych użytkowników (lokalne konta), jak i administratorów (uwierzytelnianie przez Google, Facebook lub Microsoft). Aplikacja wspiera uwierzytelnianie dwuskładnikowe (2FA), co znacząco podnosi poziom bezpieczeństwa.

1.2 Wymagania systemowe

- **Serwer aplikacji**: Python 3.8+ oraz Django
- **Baza danych**: SQLite
- **Zewnętrzni dostawcy autoryzacji**: Google OAuth 2.0
- **Serwer e-mail**: SMTP do wysyłania e-maili aktywacyjnych
- **Dodatkowe biblioteki**:
  - `django-otp` – obsługa uwierzytelniania dwuskładnikowego
  - `two_factor` – wdrożenie 2FA
  - `django.core.signing` – zabezpieczenie tokenów aktywacyjnych

2 Idea aplikacji i przykłady zastosowania

2.1 Idea aplikacji

Aplikacja została zaprojektowana z myślą o zapewnieniu wysokiego poziomu bezpieczeństwa kont użytkowników. Głównym celem systemu jest umożliwienie bezpiecznej rejestracji, logowania oraz zarządzania kontem, w tym z wykorzystaniem uwierzytelniania dwuskładnikowego (2FA). Dzięki wsparciu dla zewnętrznych dostawców autoryzacji (Google, Facebook, Microsoft), administratorzy mogą logować się wygodnie, jednocześnie zachowując wysoki poziom zabezpieczeń.

2.2 Przykłady zastosowania

- **Systemy korporacyjne** – aplikacja może być używana w firmach do zarządzania dostępem pracowników do wewnętrznych systemów. Integracja z Google, Facebook i Microsoft pozwala na łatwą autoryzację dla administratorów IT.
- **Portale internetowe i społecznościowe** – strony wymagające rejestracji użytkowników mogą skorzystać z tego systemu do zapewnienia bezpiecznego logowania oraz ochrony kont przed przejęciem.
- **Aplikacje bankowe i finansowe** – aplikacja może posłużyć jako system logowania do aplikacji finansowych, wymagających wysokiego poziomu bezpieczeństwa.
- **Systemy rezerwacji i sprzedaży biletów** – platformy obsługujące rezerwacje i sprzedaż biletów online mogą wykorzystać mechanizm aktywacji konta poprzez e-mail oraz dodatkowe zabezpieczenia 2FA do ochrony transakcji użytkowników.
- **Systemy medyczne** – aplikacja może znaleźć zastosowanie w systemach przechowywania danych pacjentów, gdzie bezpieczeństwo konta ma kluczowe znaczenie.

3 Funkcjonalności

3.1 Rejestracja użytkownika

Podczas rejestracji użytkownik wypełnia formularz, podając nazwę użytkownika, adres e-mail oraz hasło. Jeśli dane wprowadzone przez użytkownika są niepoprawne, aplikacja wyświetla odpowiedni komunikat i prosi o ich poprawienie. Po zatwierdzeniu poprawnych danych i ich weryfikacji przez system, na podany adres e-mail zostaje wysłany link aktywacyjny. Aby aktywować konto, użytkownik musi kliknąć w otrzymany link. Po pomyślnej aktywacji użytkownik zostaje przekierowany na stronę logowania.

3.2 Logowanie użytkownika

Podczas logowania użytkownik wprowadza swoje dane uwierzytelniające (nazwę użytkownika i hasło). System sprawdza poprawność wprowadzonych danych oraz weryfikuje, czy konto zostało aktywowane. W przypadku błędnych danych lub nieaktywowanego konta aplikacja wyświetla odpowiedni komunikat. Po pomyślnym zalogowaniu użytkownik zostaje przekierowany do widoku profilu.

3.3 Logowanie administratora

Administrator może zalogować się, wybierając opcję logowania przez Google, Facebook lub Microsoft. Po kliknięciu w odpowiednią opcję, użytkownik zostaje przekierowany na stronę uwierzytelniania odpowiedniego dostawcy. Jeśli proces logowania przebiegnie pomyślnie, administrator uzyskuje dostęp do panelu administracyjnego aplikacji.

3.4 Aktywacja konta poprzez e-mail

Po rejestracji użytkownika w systemie jego konto pozostaje nieaktywne do momentu potwierdzenia adresu e-mail. System generuje unikalny, jednorazowy token, który jest przypisany do nowo utworzonego konta. Użytkownik otrzymuje wiadomość e-mail z linkiem aktywacyjnym, który przekierowuje go do aplikacji. Po kliknięciu w link aktywacyjny, konto użytkownika zostaje aktywowane. Token, który został użyty, jest następnie usuwany, aby zapobiec jego ponownemu wykorzystaniu.

3.5 Uwierzytelnianie dwuskładnikowe (2FA)

Aplikacja wspiera uwierzytelnianie dwuskładnikowe (2FA). Po pomyślnym wprowadzeniu loginu i hasła użytkownik musi dodatkowo wprowadzić kod wygenerowany przez aplikację mobilną lub przesyłany na adres e-mail. Tylko po poprawnym wprowadzeniu obu składników, użytkownik uzyskuje dostęp do swojego konta.

W przypadku próby wyłączenia 2FA, użytkownik musi potwierdzić tę operację poprzez link aktywacyjny wysyłany na jego e-mail.

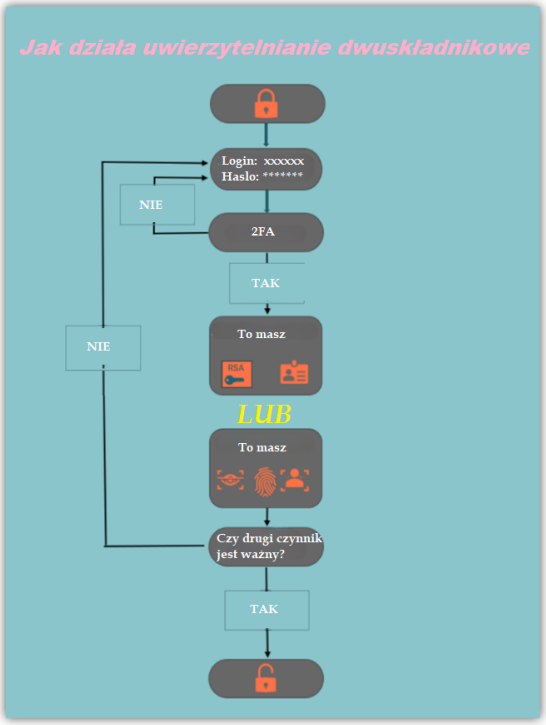
Jeśli użytkownik wyłączył 2FA bądź nie włączył wcześniej tej funkcji, ma możliwość jej aktywowania.

Aplikacja umożliwia także generowanie backup tokenów (tokenów zapasowych). Tokeny zapasowe są jednorazowe i użytkownik może je wykorzystać w sytuacji, gdy urządzenie 2FA jest niedostępne.

Po wygenerowaniu tokenów zapasowych, użytkownik ma możliwość ich wydrukowania lub zapisania w bezpiecznym miejscu. Tokeny zapasowe są widoczne w interfejsie użytkownika, a system zapewnia możliwość ich wydrukowania w formie czytelnej listy, aby użytkownik mógł je przechować na wypadek awarii głównej metody uwierzytelniania.

W celu wydrukowania tokenów zapasowych użytkownik może kliknąć przycisk „Drukuj” obok generowanych tokenów. Wydrukowane tokeny są jednorazowe i powinny być przechowywane w bezpiecznym miejscu, aby zminimalizować ryzyko nieautoryzowanego dostępu.

Jeśli użytkownik wyczerpie wszystkie dostępne tokeny zapasowe, ma możliwość wygenerowania nowych, które będą dostępne po kliknięciu odpowiedniego przycisku na stronie.



Rysunek 1: 2FA

4 Bezpieczeństwo i zabezpieczenia

4.1 Logowanie przez Google, Facebook i Microsoft

Logowanie przez Google, Facebook i Microsoft opiera się na protokole OAuth 2.0, który umożliwia bezpieczną autoryzację użytkowników bez konieczności przechowywania haseł. Użytkownik jest przekierowywany do strony logowania danego dostawcy, gdzie podaje swoje dane, a po pomyślnym zalogowaniu generowany jest token dostępu, który jest przesyłany do aplikacji w celu uwierzytelnienia. Każdy z dostawców zapewnia dodatkowe mechanizmy zabezpieczeń, takie jak uwierzytelnianie dwuskładnikowe (2FA), analiza ryzyka logowania, wykrywanie podejrzanych prób logowania, a także ochrona przed atakami brute force. Dzięki tym środkom proces logowania jest zarówno wygodny, jak i bezpieczny.

4.1.1 Ograniczenia dostępu

W aplikacji z logowania przez Goggle, Facebook i Microsoft mogą korzystać wyłącznie administratorzy. Aplikacja weryfikuje, czy użytkownik próbujący zalogować się jako administrator posiada uprawnienia is_staff i is_superuser . W ten sposób dostęp do panelu administracyjnego jest dodatkowo ograniczony przed nieuprawnionym dostępem.

4.1.2 Implementacja w aplikacji

W aplikacji logowanie za pomocą OAuth 2.0 z wykorzystaniem Google, Facebook i Microsoft zostało zrealizowane przy pomocy pakietu social-auth-app-django (social-auth). System umożliwia łatwą integrację z zewnętrznymi dostawcami tożsamości, eliminując potrzebę przechowywania haseł użytkowników w aplikacji.

Pobieranie identyfikatora użytkownika (uid) jest różne w zależności od dostawcy OAyth 2.0. W aplikacji pobieranie uid zostało ujednolicone:

- Dla Microsoftu (microsoft-graph) zwraca userPrincipalName jako identyfikator użytkownika.
- Dla Facebooka zwraca adres e-mail (email) jako identyfikator.

Główne etapy procesu logowania przez OAuth 2.0 w aplikacji:

- **Przekierowanie do dostawcy tożsamości** - Użytkownik klikając przycisk logowania zostaje przekierowany na stronę logowania wybranego dostawcy (Google, Facebook, Microsoft). Na stronie logowania użytkownik wprowadza swoje dane, takie jak adres e-mail i hasło.
- **Zgoda na dostęp do danych** - Po wprowadzeniu danych logowania, użytkownik musi zaakceptować dostęp do swoich danych, takich jak adres e-mail czy imię, które będą wykorzystywane przez aplikację.
- **Generowanie tokenu dostępu**: Po pomyślnym zalogowaniu, dostawca tożsamości (np. Google) generuje token dostępu, który aplikacja wykorzystuje do uwierzytelnienia użytkownika. Token ten jest bezpiecznie przekazywany do aplikacji i przechowywany w sesji użytkownika.
- **Uwierzytelnienie użytkownika** - Aplikacja wykorzystuje token dostępu, aby autoryzować użytkownika, uzyskać dostęp do jego danych (np. adresu e-mail) oraz zidentyfikować użytkownika w systemie.

Dodatkowo, system wspiera również integrację z mechanizmami bezpieczeństwa dostawców:

- **Uwierzytelnianie dwuskładnikowe (2FA)** -Google, Facebook i Microsoft oferują opcję dwuskładnikowego uwierzytelniania
- **Analiza ryzyka logowania i wykrywanie podejrzanych prób logowania** - Zewnętrzni dostawcy, tacy jak Google, Facebook i Microsoft, analizują ryzyko logowania na podstawie takich czynników, jak lokalizacja użytkownika, adres IP, oraz niecodzienne zachowanie, co pomaga w wykrywaniu i zapobieganiu nieautoryzowanym próbom logowania.
- **Ochrona przed atakami brute force** - Dostawcy tożsamości, tacy jak Google, stosują mechanizmy ochrony przed atakami brute force, blokując dostęp do konta po zbyt wielu nieudanych próbach logowania, co zwiększa bezpieczeństwo aplikacji.

4.1.3 Integracja z aplikacją

Integracja z `social-auth-app-django` jest realizowana poprzez odpowiednią konfigurację w pliku settings.py oraz mapowanie ścieżek URL w pliku urls.py. W pliku settings.py definiujemy klucze API i sekrety dla każdego z dostawców, a także ustawiamy odpowiednie przekierowania po udanym logowaniu. W pliku urls.py natomiast, dodajemy odpowiednie ścieżki, które odpowiadają za obsługę logowania przez OAuth 2.0. Przykład:

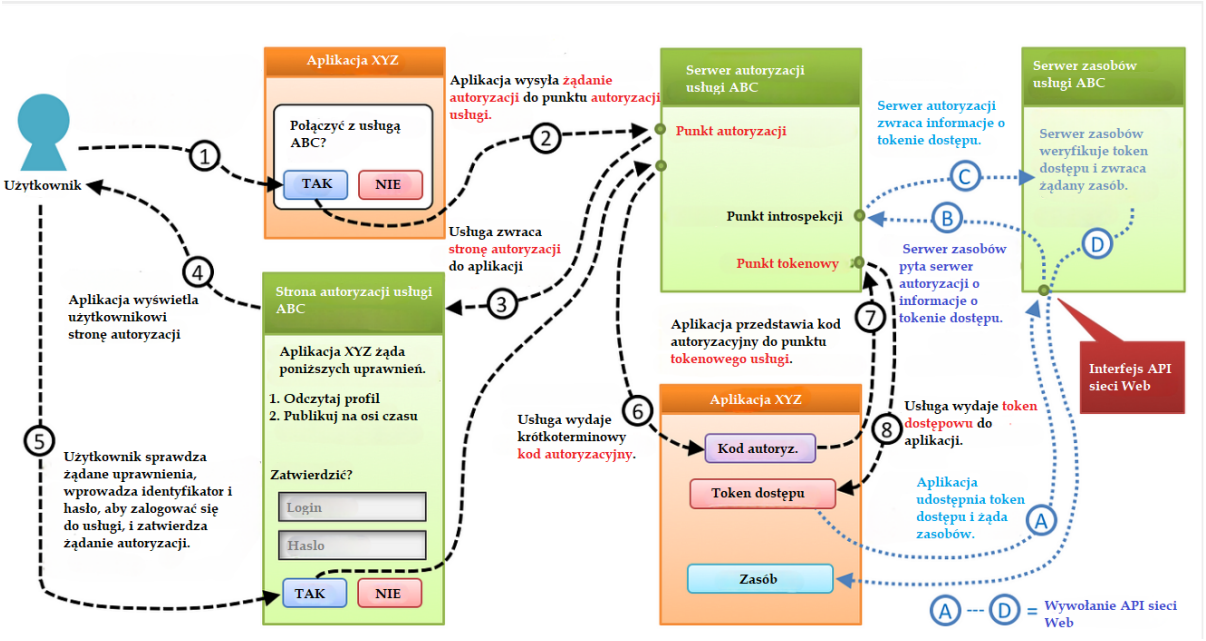
```
path('oauth/', include('social_django.urls', namespace='social'))
```

Dzięki temu aplikacja automatycznie generuje wszystkie niezbędne URL-e do logowania, odbierania odpowiedzi od dostawcy i przekierowywania użytkowników po udanej autoryzacji.

4.2 OAuth 2.0

OAuth 2.0 to protokół autoryzacji, który pozwala użytkownikom na bezpieczne logowanie się do aplikacji lub udostępnianie zasobów bez konieczności podawania swoich haseł. Zamiast tego, OAuth 2.0 wykorzystuje mechanizm tokenów, które umożliwiają aplikacjom dostęp do danych użytkownika na podstawie zgody użytkownika.

OAuth 2.0 działa na zasadzie autoryzacji przez delegację, co oznacza, że aplikacja (klient) uzyskuje dostęp do zasobów użytkownika (np. danych konta Google, Facebooka czy Microsoftu), ale bez potrzeby przechowywania haseł użytkownika. Proces ten odbywa się za pośrednictwem zaufanego dostawcy tożsamości, który obsługuje uwierzytelnianie użytkownika i generowanie tokenów.



Rysunek 2: OAuth2.0

4.2.1 Rejestracja aplikacji w dostawcy tożsamości

Aby aplikacja mogła korzystać z OAuth 2.0, musi zostać zarejestrowana w systemie dostawcy tożsamości. W tym procesie aplikacja otrzymuje dwa kluczowe elementy:

- **Client ID** – unikalny identyfikator aplikacji w systemie dostawcy.
- **Client Secret** – sekret, który jest używany do weryfikacji tożsamości aplikacji.

Aby zarejestrować aplikację, należy podać:

- **Redirect URI** – URL, na który dostawca tożsamości przekieruje użytkownika po zakończeniu procesu autoryzacji.

4.2.2 Inicjowanie procesu autoryzacji

Gdy użytkownik chce zalogować się do aplikacji, aplikacja kieruje go do strony logowania dostawcy tożsamości. W tym celu wysyła żądanie autoryzacji, które zawiera:

- **Client ID** – identyfikator aplikacji.
- **Redirect URI** – adres, na który zostanie przekierowany użytkownik po zakończeniu procesu.
- **Zakresy dostępu (Scopes)** – zakres danych, do których aplikacja chce uzyskać dostęp.
- **State** – parametr, który jest używany do zabezpieczenia przed atakami typu CSRF (Cross-Site Request Forgery).

Dostawca tożsamości sprawdza poprawność żądania i kieruje użytkownika na stronę logowania.

4.2.3 Uwierzytelnianie użytkownika

Użytkownik wprowadza swoje dane logowania na stronie dostawcy tożsamości. W przypadku, gdy użytkownik jest już zalogowany, dostawca może pomijać krok logowania i przejść do kolejnego kroku.

Jeśli użytkownik nie jest jeszcze zalogowany, może zostać poproszony o wykonanie dodatkowych czynności weryfikacyjnych, np. uwierzytelnianie dwuskładnikowe (2FA). Dodatkowo użytkownik będzie miał możliwość wyrażenia zgody na dostęp aplikacji do swoich danych, takich jak adres e-mail, zdjęcie profilowe itp.

4.2.4 Generowanie kodu autoryzacyjnego

Po pomyślnym uwierzytelnieniu użytkownika i zatwierdzeniu dostępu, dostawca tożsamości generuje kod autoryzacyjny. Jest to krótki ciąg znaków, który potwierdza, że użytkownik wyraził zgodę na dostęp aplikacji do swoich zasobów. Kod ten jest następnie przekazywany aplikacji poprzez redirect URI.

4.2.5 Wymiana kodu autoryzacyjnego na token dostępu

Po otrzymaniu kodu autoryzacyjnego, aplikacja wysyła zapytanie do dostawcy tożsamości, aby wymienić kod na token dostępu (Access Token). Zapytanie to zawiera:

- **Client ID** i **Client Secret** (w celu weryfikacji tożsamości aplikacji).
- **Authorization Code** - kod autoryzacyjny otrzymany w poprzednim kroku.
- **Redirect URI** - adres przekierowania, który musi pasować do tego, który został użyty w poprzednich krokach.

Dostawca tożsamości weryfikuje zapytanie i, jeśli jest poprawne, zwraca token dostępu. W odpowiedzi mogą również znajdować się inne tokeny:

- **Access Token** – token dostępu, który pozwala aplikacji na dostęp do zasobów użytkownika.
- **Refresh Token** – token odświeżania, który pozwala aplikacji na uzyskanie nowego tokena dostępu, gdy poprzedni wygaśnie.
- **ID Token** – token zawierający dane użytkownika w formacie JSON Web Token (JWT)

4.2.6 Używanie tokenu dostępu

Po otrzymaniu tokena dostępu aplikacja może go używać, aby uzyskać dostęp do chronionych zasobów użytkownika na serwerze zasobów. Token dostępu jest dołączany do każdego żądania jako nagłówek autoryzacji HTTP.

4.2.7 Wylogowanie i unieważnienie tokenów

Jeśli użytkownik zdecyduje się wylogować z aplikacji, aplikacja może wysłać żądanie do dostawcy tożsamości, aby unieważnić token dostępu i refresh token. Dzięki temu dostęp do zasobów użytkownika zostaje zamknięty.

4.3 Zabezpieczenia wynikające z aktywacji konta poprzez e-mail

Proces aktywacji konta użytkownika w aplikacji opiera się na dwóch głównych etapach: rejestracji oraz aktywacji konta za pomocą linku wysyłanego na adres e-mail użytkownika. Poniżej opisano szczegóły tego procesu.

4.3.1 Rejestracja użytkownika

Podczas rejestracji użytkownik wypełnia formularz rejestracyjny. Po przesłaniu formularza, system wykonuje następujące kroki:

1. Utworzony zostaje nowy obiekt użytkownika w bazie danych, jednak konto nie jest jeszcze aktywowane. Atrybut `is_active` użytkownika jest ustawiany na `False`.
2. Hasło użytkownika jest szyfrowane za pomocą metody `set_password`, co zapewnia bezpieczne przechowywanie hasła w bazie danych.
3. Na adres e-mail użytkownika wysyłany jest e-mail z linkiem aktywacyjnym. Link ten zawiera unikalny identyfikator użytkownika (`uid`) oraz token aktywacyjny, który jest generowany przez Django w ramach mechanizmu `default_token_generator`.

Link aktywacyjny wygląda następująco:

`https://example.com/activate/{uid}/{token}/`

Wiadomość e-mail zawiera ten link, a także instrukcje dotyczące aktywacji konta.

4.3.2 Aktywacja konta

Po kliknięciu przez użytkownika w link aktywacyjny, aplikacja przeprowadza proces aktywacji konta. Główne kroki to:

1. **Dekodowanie identyfikatora użytkownika (uid):**
   Link aktywacyjny zawiera zakodowany identyfikator użytkownika w formacie URL-safe base64. Ten identyfikator jest dekodowany w celu uzyskania rzeczywistego `pk` użytkownika.
2. **Weryfikacja tokena:**
   Wartość tokena zawarta w URL-u jest weryfikowana za pomocą `default_token_generator.check_token`, co pozwala upewnić się, że token jest prawidłowy i nie został zmieniony.
3. **Aktywacja konta:**
   Jeśli token jest poprawny, konto użytkownika zostaje aktywowane poprzez ustawienie `user.is_active = True`, a zmiany są zapisywane w bazie danych.
4. **Przekierowanie:**
   Po pomyślnej aktywacji użytkownik zostaje przekierowany na stronę profilu lub logowania.

4.3.3 Bezpieczeństwo procesu aktywacji

Proces aktywacji konta jest zabezpieczony poprzez następujące mechanizmy:

- **Tokeny aktywacyjne:** Każdy token jest unikalny i przypisany do konkretnego użytkownika. Tokeny są generowane przy użyciu `default_token_generator`, co zapewnia ich bezpieczeństwo. Tokeny są ważne tylko przez określony czas, co minimalizuje ryzyko ich przejęcia.
- **Kodowanie identyfikatora użytkownika (uid):** `uid` w URL jest zakodowane w formacie base64, co sprawia, że nie jest łatwe do odgadnięcia. Dodatkowo, aplikacja sprawdza poprawność `uid` oraz tokena, co zabezpiecza przed manipulacją URL.
- **Link aktywacyjny jest jednorazowy:** Link aktywacyjny jest ważny tylko przez krótki okres, co zapobiega jego ponownemu użyciu.

4.4 Bezpieczne przechowywanie haseł

Aplikacja stosuje mechanizm hashowania haseł, wykorzystując wbudowaną metodę `set_password()` dostępną w Django, wykorzystującą algorytmy PBKDF2 i SHA-256. Jest to kluczowy element zabezpieczeń, ponieważ zamiast przechowywać hasła w bazie danych w postaci jawnej, czyli niezaszyfrowanej, są one poddawane procesowi jednostronnego szyfrowania. W trakcie tego procesu dane są przekształcane w unikalną, zaszyfrowaną wartość, tzw. hash. Kluczową cechą tego procesu jest jego nieodwracalność – nie można łatwo odzyskać oryginalnej wartości z wygenerowanego hasha. Dzięki temu, nawet jeśli ktoś przejmie bazę danych z hashowanymi hasłami, nie będzie w stanie ich łatwo odzyskać.

PBKDF2 to algorytm stosowany do bezpiecznego hashowania haseł. Jest zgodny ze standardami NIST (amerykańskiego instytutu zajmującego się opracowywaniem standardów w zakresie technologii) i szeroko stosowany w aplikacjach internetowych, w tym w Django. PBKDF2 działa poprzez zastosowanie funkcji HMAC do wielokrotnego przekształcania hasła, co znacząco utrudnia jego złamanie.

PBKDF2 używa kilku kluczowych mechanizmów:

1. **Sol (salt)** – unikalna wartość dodawana do hasła przed rozpoczęciem procesu hashowania. Zapobiega to atakom rainbow tables, polegającym na wykorzystaniu wcześniej obliczonych wartości hash dla dużej liczby haseł.
2. **Wiele iteracji** – hasło jest przetwarzane przez funkcję HMAC wiele razy (np. 260 000 razy w Django), co zwiększa czas potrzebny na każdą próbę ataku brute-force. Ataki te polegają na systematycznym sprawdzaniu wszystkich możliwych kombinacji haseł, aż do znalezienia właściwego.
3. **Funkcja HMAC** – używana do generowania wartości hash.

HMAC (Hash-Based Message Authentication Code) to mechanizm zapewniający integralność i autentyczność danych przy użyciu funkcji skrótu SHA-256 oraz tajnego klucza. W kontekście PBKDF2 HMAC jest wykorzystywany do wielokrotnego przekształcania hasła, co zwiększa bezpieczeństwo przechowywania danych.

SHA-256 jest funkcją skrótu, która zamienia dowolne dane wejściowe na 256-bitowy (32-bajtowy) ciąg znaków o ustalonej długości.

HMAC działa według schematu:

1. Hasło jest łączone z tajnym kluczem, który został wygenerowany dla naszej aplikacji.
2. Następnie jest przetwarzane przez funkcję hashującą SHA-256.
3. Wynik jest ponownie mieszany z kluczem i ponownie hashowany.
4. Proces ten jest powtarzany wiele razy.

HMAC jest odporny na ataki typu „length extension", polegają na manipulowaniu danymi wejściowymi funkcji hashującej, aby wygenerować poprawny hash dla nowej wiadomości bez znajomości oryginalnego hasła czy tajnego klucza i zapewnia silną ochronę danych, szczególnie w systemach wymagających wysokiego poziomu bezpieczeństwa.

4.5 Wykorzystanie pliku .env

Wszystkie zmienne środowiskowe wykorzystywanie w aplikacji są przechowywane poza plikiem źródłowym w pliku .env. Zapobiega to jego przypadkowemu ujawnieniu w repozytorium projektu. Zmienne przechowywane w pliku .env:

1. `SOCIAL_AUTH_GOOGLE_OAUTH2_KEY`
   Klucz API dla integracji z Google OAuth2, używany do logowania za pomocą Google.
2. `SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET`
   Sekret API dla integracji z Google OAuth2, używany razem z kluczem, aby umożliwić logowanie za pomocą Google.

3. `SOCIAL_AUTH_FACEBOOK_KEY`
   Klucz API dla integracji z Facebook OAuth2, używany do logowania za pomocą Facebooka.

4. `SOCIAL_AUTH_FACEBOOK_SECRET`
   Sekret API dla integracji z Facebook OAuth2, używany razem z kluczem, aby umożliwić logowanie za pomocą Facebooka.

5. `SOCIAL_AUTH_MICROSOFT_GRAPH_KEY`
   Klucz API dla integracji z Microsoft Graph, umożliwiający logowanie za pomocą konta Microsoft.

6. `SOCIAL_AUTH_MICROSOFT_GRAPH_SECRET`
   Sekret API dla integracji z Microsoft Graph, używany razem z kluczem, aby umożliwić logowanie za pomocą Microsofta.

7. `EMAIL_HOST_PASSWORD`
   Hasło do serwera pocztowego (SMTP) dla wysyłania e-maili z aplikacji, chroniące dane logowania przed umieszczaniem ich w kodzie.

8. `SECRET_KEY`
   Klucz sekretu używany przez Django do ochrony sesji i innych danych w aplikacji. Jest to jeden z najważniejszych elementów bezpieczeństwa, więc jego przechowywanie w zmiennych środowiskowych jest dobrym rozwiązaniem.

9. `DEBUG`
   Określa, czy aplikacja jest w trybie debugowania. Wartość `True` oznacza, że włączony jest tryb debugowania, który ujawnia szczegóły błędów w aplikacji. W środowisku produkcyjnym ta zmienna powinna być ustawiona na `False`.

10. `ALLOWED_HOSTS`
    Lista dozwolonych hostów, które mogą łączyć się z aplikacją Django. Zawiera adresy IP i nazwy domen, takie jak `127.0.0.1`, `localhost`, czy `localhost:8000`.

11. `BASE_URL`
    Adres URL podstawowy, który jest używany do konfiguracji aplikacji, na przykład do ustawienia pełnego adresu aplikacji, który może być przydatny w e-mailach, linkach, czy w konfiguracji API.

4.6 Tworzenie administratora

Aplikacja umożliwia tworzenie użytkownika administratora, który może logować się wyłącznie za pomocą zewnętrznych dostawców OAuth 2.0, takich jak Google, Microsoft oraz Facebook. Taki użytkownik nie ma lokalnego hasła, co minimalizuje ryzyko nieautoryzowanego dostępu do konta przy użyciu słabego hasła.

Aplikacja pozwala na stworzenie użytkownika o uprawnieniach administratora, który nie korzysta z tradycyjnego hasła, ale loguje się za pomocą OAuth2. Proces ten składa się z kilku kroków:

1. **Podanie e-maila:** Aplikacja wymaga podania adresu e-mail dla nowego użytkownika. Jeśli e-mail nie jest podany jako argument, użytkownik zostaje poproszony o jego wprowadzenie.

2. **Tworzenie użytkownika:** Aplikacja tworzy konto użytkownika w systemie na podstawie podanego adresu e-mail. Użytkownik jest automatycznie oznaczany jako administrator (`is_staff=True`, `is_superuser=True`) oraz aktywowany (`is_active=True`).

3. **Brak lokalnego hasła:** Zamiast tradycyjnego hasła, aplikacja ustawia `set_unusable_password()`, co oznacza, że użytkownik nie ma lokalnego hasła do logowania. Zamiast tego może logować się wyłącznie przez dostawców OAuth.

4. **Powiązanie z dostawcami OAuth:** użytkowanik jest łączony z następującymi dostawcami OAuth:

   - Google OAuth2
   - Microsoft Graph
   - Facebook OAuth2

Każdy z tych dostawców jest powiązany z użytkownikiem przy pomocy `UserSocialAuth`, który zawiera identyfikator użytkownika (w tym przypadku e-mail) jako `uid`.

5. **Obsługa błędów:** Jeśli wystąpi problem podczas tworzenia użytkownika (np. e-mail jest już zajęty), aplikacja zgłasza odpowiedni błąd.

Przykładowe uruchomienie skryptu: Aby stworzyć użytkownika administratora, należy uruchomić skrypt za pomocą poniższego polecenia:

```
python manage.py createsuperuser emailadmin@example.com
```

Jeśli adres e-mail nie zostanie podany jako argument, aplikacja poprosi o jego wprowadzenie.

4.7 Zabezpieczenie przed słabymi hasłami

Django oferuje mechanizm walidacji haseł, który zapewnia, że użytkownicy tworzą silne hasła. Funkcjonalność ta jest konfigurowana poprzez ustawienie weryfikatorów haseł w pliku `settings.py`. Poniższe weryfikatory pomagają zapobiec stosowaniu haseł, które mogą być łatwe do odgadnięcia lub złamania.

4.7.1 Konfiguracja weryfikatorów haseł

W pliku `settings.py` w sekcji `AUTH_PASSWORD_VALIDATORS` znajdują się domyślnie cztery weryfikatory, które dbają o bezpieczeństwo haseł użytkowników.

- **UserAttributeSimilarityValidator:** Sprawdza, czy hasło użytkownika nie jest zbyt podobne do innych atrybutów użytkownika, takich jak imię, nazwisko czy login. Zapobiega to używaniu haseł, które mogą być łatwe do odgadnięcia.
- **MinimumLengthValidator:** Weryfikuje, czy hasło użytkownika ma minimalną długość (domyślnie 8 znaków). Dzięki temu użytkownicy nie mogą ustawić haseł zbyt krótkich, które są łatwiejsze do złamania.
- **CommonPasswordValidator:** Sprawdza, czy hasło użytkownika nie jest popularnym, powszechnie używanym hasłem. Takie hasła, jak `"123456"` czy `"password"`, są szczególnie podatne na ataki.
- **NumericPasswordValidator:** Weryfikuje, czy hasło użytkownika nie składa się wyłącznie z cyfr, które są łatwiejsze do odgadnięcia, zwłaszcza w przypadku ataków typu brute-force.

4.8 Logowanie dwuetapowe 2FA

Uwierzytelnianie dwuetapowe (2FA) jest mechanizmem zabezpieczeń, który zwiększa bezpieczeństwo logowania, wymagając dwóch różnych składników (etapów) w celu potwierdzenia tożsamości użytkownika. Zwykle obejmuje coś, co użytkownik wie (np. hasło), oraz coś, co użytkownik posiada (np. telefon lub aplikacja generująca kody). Dzięki temu nawet jeśli hasło użytkownika zostanie skradzione, osoba trzecia nie będzie w stanie uzyskać dostępu do konta bez drugiego etapu uwierzytelniania.

4.8.1 Etapy procesu logowania dwuetapowego

1. **Wprowadzenie hasła:** Użytkownik wprowadza swoje hasło w tradycyjny sposób (pierwszy etap). Jeśli wprowadzone hasło jest poprawne, system przechodzi do kolejnego etapu.
2. **Generowanie kodu weryfikacyjnego:** Po wprowadzeniu hasła użytkownik musi podać drugi składnik, którym zazwyczaj jest kod weryfikacyjny generowany na urządzeniu użytkownika (np. telefonie komórkowym) lub wysyłany na adres e-mail.
3. **Weryfikacja drugiego składnika:** Użytkownik wprowadza kod lub potwierdza powiadomienie na swoim urządzeniu (np. w aplikacji mobilnej). Jeśli kod jest poprawny, dostęp do konta jest przyznany.
4. **Opcjonalne metody backupowe:** W przypadku utraty dostępu do urządzenia 2FA, użytkownik może skorzystać z kodów zapasowych, które zostały wygenerowane podczas początkowej konfiguracji 2FA.

4.8.2 TOTP (Time-based One-Time Password)

TOTP to algorytm, który generuje jednorazowe hasła oparte na czasie. Kody są generowane na podstawie wspólnego klucza tajnego oraz bieżącego czasu. Wzór na wygenerowanie hasła OTP jest następujący:

$$OTP = HMAC - SHA1(secret, current\_time)$$

gdzie:

- `secret` to klucz tajny, wymieniany między użytkownikiem a systemem podczas konfiguracji,
- `current_time` to bieżący czas (np. liczba sekund, które upłynęły od określonego punktu początkowego, np. epoki UNIX).

Kody OTP są ważne przez określony czas (zwykle 30 sekund) i po tym czasie wygasają, co zapewnia dodatkową ochronę. TOTP jest powszechnie wykorzystywane w aplikacjach takich jak Google Authenticator (z której korzysta Google i Facebook) i Microsoft Authenticator.

4.8.3 Kody zapasowe (Backup Tokens)

Kody zapasowe są generowane podczas początkowej konfiguracji 2FA i użytkownik powinien je zapisać w bezpiecznym miejscu. W przypadku, gdy użytkownik nie ma dostępu do głównego urządzenia (np. telefonu), kody zapasowe mogą być wykorzystane do odzyskania dostępu do konta.

4.8.4 Zabezpieczenia dodatkowe

- **Zabezpieczenie przed atakami brute force:** Każdy kod 2FA jest jednorazowy i ma krótki czas życia, co praktycznie uniemożliwia przeprowadzenie ataku brute force.
- **Szyfrowanie kluczy tajnych:** Klucz tajny używany do generowania kodów 2FA jest przechowywany w sposób bezpieczny, aby zapobiec jego przechwyceniu przez osoby trzecie.
- **Weryfikacja urządzeń:** Systemy 2FA mogą umożliwiać rejestrowanie zaufanych urządzeń, co upraszcza proces logowania na tych urządzeniach.

4.8.5 Zainstalowane aplikacje (INSTALLED_APPS)

- `'django_otp'` – aplikacja odpowiedzialna za obsługę 2FA, która oferuje wsparcie dla jednorazowych haseł opartych na czasie (TOTP) oraz statycznych kodów uwierzytelniających.
- `'django_otp.plugins.otp_totp'` – wtyczka do `django_otp`, która umożliwia generowanie jednorazowych haseł opartych na czasie (TOTP)
- `'django_otp.plugins.otp_static'` – wtyczka, która pozwala na używanie statycznych kodów uwierzytelniających
- `'two_factor'` – aplikacja zapewniająca wygodne zarządzanie ustawieniami oraz interfejsem użytkownika dla logowania dwuskładnikowego (2FA). Umożliwia konfigurację i weryfikację metod 2FA.

4.8.6 Middleware

- `'django_otp.middleware.OTPMiddleware'` – middleware jest odpowiedzialne za zarządzanie sesjami użytkowników oraz weryfikację kodów OTP (One Time Password) w trakcie trwania sesji użytkownika. Middleware sprawdza, czy użytkownik przeszedł przez proces weryfikacji 2FA przed przyznaniem dostępu do chronionych zasobów.

4.9 Wykorzystanie `social_django`

W przypadku korzystania z logowania przez zewnętrzne usługi, takie jak `Google`, `Facebook` czy `Microsoft` (social-auth), może zaistnieć potrzeba zaimplementowania dodatkowego procesu 2FA. Choć `social_django` nie obsługuje bezpośrednio logowania dwuskładnikowego, aplikacje takie jak `django_otp` i `two_factor` mogą być używane w połączeniu z tymi metodami logowania zewnętrznego, aby dodać dodatkową warstwę bezpieczeństwa.

Integracja logowania 2FA z `social_django` umożliwia użytkownikom korzystanie z metod uwierzytelniania, takich jak Google OAuth, Facebook OAuth oraz Microsoft OAuth, w połączeniu z zabezpieczeniem opartym na dodatkowym kodzie uwierzytelniającym (np. TOTP), co zapewnia wyższy poziom ochrony przed nieautoryzowanym dostępem.

5 Scieżki URL

Ścieżka	Nazwa	Opis
/login/	two_factor:login	Logowanie użytkownika z obsługą 2FA.
/register/	register	Rejestracja nowego użytkownika.
/logout/	logout	Wylogowanie użytkownika.
/activate/<uid>/<token>/	activate	Aktywacja konta po rejestracji przy użyciu tokena.
/two_factor/disable/	two_factor:disable	Wyłączenie uwierzytelniania dwuskładnikowego (2FA).
/two_factor/disable-2fa-confirm/<uid>/<token>/	disable_confirm	Potwierdzenie dezaktywacji 2FA poprzez unikalny token.

Tabela 1: Dostępne ścieżki w aplikacji 'accounts'

6 Hosting aplikacji na PythonAnywhere

PythonAnywhere to popularna platforma hostingowa umożliwiająca łatwe wdrażanie aplikacji napisanych w Pythonie.

Zalety PythonAnywhere:

- **Łatwa konfiguracja** – PythonAnywhere oferuje gotowe środowisko do uruchamiania aplikacji Django, eliminując konieczność ręcznej konfiguracji serwera.
- **Zintegrowana baza danych** – platforma obsługuje popularne systemy bazodanowe, takie jak SQLite.
- **Zarządzanie przez przeglądarkę** – wszystkie operacje można wykonywać przez interfejs webowy, bez konieczności używania terminala na lokalnej maszynie.
- **Hosting aplikacji zgodnych z WSGI (Web Server Gateway Interface)** – umożliwia uruchamianie aplikacji napisanych w Django na serwerach obsługujących standard WSGI.
  Na platformie PythonAnywhere aplikacja działa to poprzez serwer uWSGI.

6.1 Logowanie i monitoring

PythonAnywhere udostępnia trzy rodzaje logów:

- **Log serwera** – zawiera informacje o działaniu serwera aplikacyjnego.
- **Log zapytań** – zapisuje przychodzące żądania HTTP.
- **Log błędów** – rejestruje błędy aplikacji i serwera.

Aktualnie nie stosujemy żadnych dodatkowych systemów monitorowania poza ręcznym przeglądaniem logów.

6.2 WSGI

WSGI (Web Server Gateway Interface) to standardowy interfejs komunikacji między serwerem HTTP a aplikacją napisaną w Pythonie. Dzięki WSGI aplikacje webowe mogą działać na różnych serwerach bez konieczności dostosowywania kodu do konkretnej implementacji serwera.

Działanie WSGI:

1. Użytkownik wysyła żądanie HTTP do aplikacji.
2. Serwer HTTP odbiera żądanie i przekazuje je do serwera WSGI.
3. Serwer WSGI przekazuje żądanie do aplikacji.
4. Aplikacja przetwarza żądanie i generuje odpowiedź HTTP.
5. Serwer WSGI odbiera odpowiedź aplikacji i przekazuje ją do serwera HTTP, który odsyła ją użytkownikowi.

6.3 Wdrażanie nowej wersji aplikacji

Zawsze po wdrożeniu nowej wersji aplikacji na produkcje wykonywane są scenariusze testowe w celu sprawdzenia, czy występują ewentualne błędy w nowej wersji.

7 Testy

7.1 Testowanie bezpieczeństwa systemu logowania

Aplikacja zawiera testy jednostkowe sprawdzające poprawność procesu uwierzytelniania i rejestracji użytkowników. Testy te pomagają wykryć potencjalne luki w zabezpieczeniach oraz zapewniają zgodność aplikacji z zasadami bezpieczeństwa.

7.1.1 Testy rejestracji użytkownika

Testy sprawdzają, czy użytkownik może poprawnie się zarejestrować oraz czy aplikacja odrzuca niepoprawne dane, np.:

- Brak nazwy użytkownika.
- Niepasujące hasła.
- Słabe hasła (np. "123456", "password").

7.1.2 Testy procesu wylogowania

Test sprawdza, czy użytkownik po wylogowaniu jest poprawnie przekierowany do ekranu logowania. Jest to istotne dla ochrony sesji użytkownika oraz zapobiegania nieautoryzowanemu dostępowi po zakończeniu sesji.

7.1.3 Testy dostępu do panelu użytkownika

Testy weryfikują, czy użytkownicy niezalogowani nie mają dostępu do zasobów wymagających uwierzytelnienia.

- Jeśli użytkownik nie jest zalogowany, aplikacja przekierowuje go do ekranu logowania.
- Jeśli użytkownik jest zalogowany, zostaje przeniesiony do swojego profilu.

Dzięki temu aplikacja jest odporna na próby ominięcia procesu logowania.

7.1.4 Testy e-maili aktywacyjnych

Po poprawnej rejestracji system wysyła e-mail aktywacyjny do użytkownika. Testy sprawdzają, czy wiadomość zawiera poprawny link aktywacyjny oraz czy konto nowego użytkownika pozostaje nieaktywne do czasu jego potwierdzenia. Zapobiega to tworzeniu fałszywych kont i automatycznym rejestracjom przez boty.

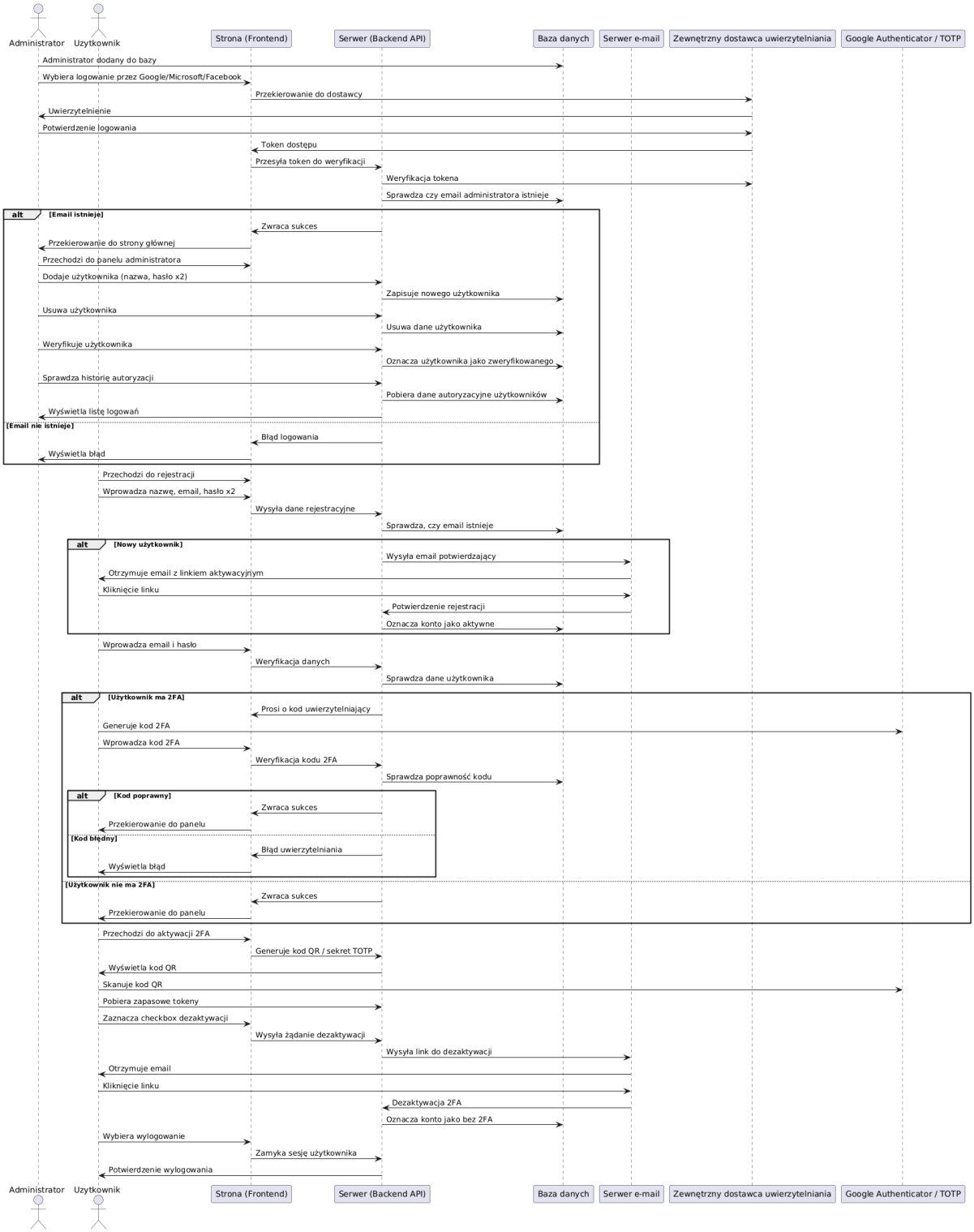
7.2 Scenariusze testowe

Scenariusz	Warunki wstępne	Kroki testowe	Oczekiwany wynik	Nieporządany wynik
Rejestracja nowego użytkownika	Użytkownik nie ma konta, posiada e-mail.	• Użytkownik Otwiera stronę rejestracji. • Wprowadza dane. • Klika „Zarejestruj się". • Aktywuje konto przez e-mail.	Konto zostaje utworzone.	Brak e-maila, link aktywacyjny nie działa, konto tworzy się bez aktywacji.
Próba logowania przed aktywacją konta	Konto utworzone, ale nieaktywne.	• Użytkownik próbuje się zalogować.	System odrzuca logowanie i prosi o aktywację.	Logowanie udane mimo braku aktywacji, brak komunikatu o aktywacji.
Logowanie po aktywacji	Konto aktywowane.	• Użytkownik Otwiera stronę logowania. • Wpisuje poprawne dane. • Klika „Zaloguj się".	Użytkownik zostaje zalogowany.	System odrzuca poprawne dane, brak przekierowania po zalogowaniu.
Nieudane logowanie (błędne dane)	Konto istnieje i jest aktywowane.	• Użytkownik wpisuje błędne dane i klika „Zaloguj się".	System odrzuca logowanie, komunikat o błędzie.	System pozwala na logowanie mimo błędnych danych, brak komunikatu o błędzie.
Włączenie 2FA	Użytkownik jest zalogowany.	• Użytkownik wybiera „Włącz Uwierzytelnianie dwuskładnikowe" po zalogowaniu na swoje konto. • Użytkownik skanuje kod QR w aplikacji Google Authenticator. • Użytkownik wpisuje kod jednorazowy.	2FA aktywne, przy logowaniu wymagany kod.	Kod QR nie generuje się, system nie akceptuje poprawnego kodu.
Logowanie z aktywnym Uwierzytelnianiem dwuskładnikowym	Użytkownik ma aktywne 2FA.	• Użytkownik wpisuje poprawne dane logowania. • System żąda kodu. • Użytkownik wpisuje kod z aplikacji Google Authenticator.	Użytkownik zostaje zalogowany.	System akceptuje błędny kod, brak prośby o kod 2FA.
Wyłączenie 2FA	Użytkownik ma aktywne 2FA.	• Użytkownik loguje się na swoje konto. • Klika „Wyłącz 2FA". • Potwierdza wyłączenie Uwierzytelniania dwuskładnikowego. • Otrzymuje e-mail i klika link znajdujący się w nim.	2FA wyłączone, logowanie nie wymaga kodu.	2FA pozostaje aktywne mimo wyłączenia, brak e-maila potwierdzającego wyłączenie.

Scenariusz	**Warunki wstępne**	**Kroki testowe**	**Oczekiwany wynik**	**Niepożądany wynik**
Logowanie administratora	Konto administratora istnieje	• Użytkownik wybiera opcję "Zaloguj jako administrator". • Wybiera metodę logowania. • Użytkownik zostaje przekierowany do dostawcy. • Użytkownik potwierdza logowanie.	Administrator zostaje zalogowany do panelu.	Logowanie nie działa, błąd autoryzacji.
Nieudane logowanie administratora (niepoprawne dane)	Konto administratora istnieje.	• Użytkownik otwiera stronę logowania administratora. • Wybiera metodę logowania. • Wpisuje błędne dane.	System odrzuca logowanie.	System loguje mimo błędnych danych.
Generowanie tokenów zapasowych	Użytkownik jest zalogowany	• Użytkownik przechodzi do ustawień. • Wybiera „Generuj tokeny zapasowe”. • Otrzymuje listę kodów.	Tokeny zapasowe są generowane.	Tokeny nie generują się, brak opcji wydruku.
Użycie tokenu zapasowego do logowania	Użytkownik wygenerował wcześniej tokeny zapasowe	• Użytkownik loguje się podając email i jeden z tokenów zapasowych.	System akceptuje token.	Tokeny nie działają, użytkownik nie został zalogowany
Rejestracja konta z istniejącym e-mailem	Konto z danym e-mailem już istnieje.	• Użytkownik otwiera stronę rejestracji. • Wprowadza email i dane. • Próbuje zakończyć rejestrację.	System odrzuca rejestrację.	System pozwala na duplikat konta.
Wykorzystanie wszystkich tokenów zapasowych	Użytkownik posiada konto z włączoną weryfikacją dwuetapową	• Użytkownik generuje tokeny zapasowe • Wykorzystuje każdy token zapasowy przy logowaniu	Po wykorzystaniu wszystkich tokenów użytkownik musi wygenerować kolejne, aby móc się za ich pomocą zalogować	System pozwala na logowanie przy pomocy tokenów zapasowych mimo tego, że żadne nie są wygenerowane

8 Diagramy

8.1 Diagram sekwencji



Rysunek 3: Diagram sekwencji przedstawiający interakcje w systemie

Diagram sekwencji przedstawia interakcje pomiędzy użytkownikiem, administratorem oraz systemem podczas kluczowych operacji w systemie.

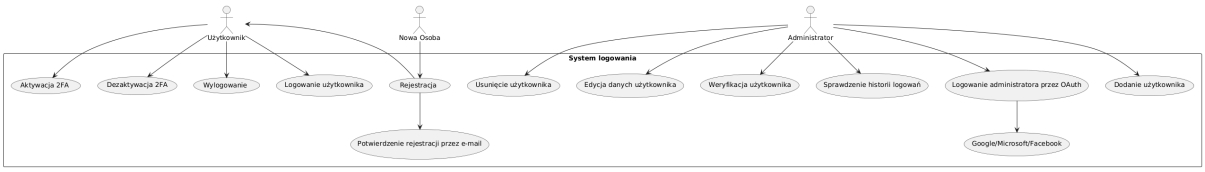
8.1.1 Opis funkcjonalności

- **Administrator:**
  - Może zalogować się przez Google, Microsoft lub Facebook (wymagane istniejące konto w bazie danych)
  - Po zalogowaniu uzyskuje dostęp do panelu administracyjnego
  - Może dodawać, usuwać oraz weryfikować użytkowników
  - Ma możliwość przeglądania historii logowań
- **Użytkownik:**
  - Proces rejestracji wymaga podania nazwy, e-maila oraz hasła
  - Po rejestracji otrzymuje link aktywacyjny na podany adres e-mail
  - Po potwierdzeniu aktywacji może zalogować się do systemu
  - W przypadku aktywnego uwierzytelniania dwuskładnikowego (2FA), wymagane jest dodatkowe potwierdzenie przez kod z aplikacji uwierzytelniającej
- **Uwierzytelnianie dwuskładnikowe (2FA):**
  - Aktywacja wymaga zeskanowania kodu QR i zapisania kodów zapasowych
  - Dezaktywacja wymaga potwierdzenia poprzez link przesłany na e-mail

Diagram szczegółowo ilustruje przebieg każdej z tych operacji, uwzględniając interakcje między:

- Użytkownikiem końcowym
- Systemem głównym
- Zewnętrznymi usługami uwierzytelniania (Google, Microsoft, Facebook)
- Serwisem e-mailowym wysyłającym wiadomości aktywacyjne

8.2 Diagram przypadków użycia



Rysunek 4: Diagram przypadków użycia systemu

Diagram przypadków użycia przedstawia funkcjonalności systemu dostępne dla dwóch głównych aktorów:

8.2.1 Aktorzy systemu

- **Użytkownik** - osoba korzystająca z podstawowych funkcji systemu
- **Administrator** - osoba zarządzająca systemem z uprawnieniami administracyjnymi

Funkcjonalności użytkownika

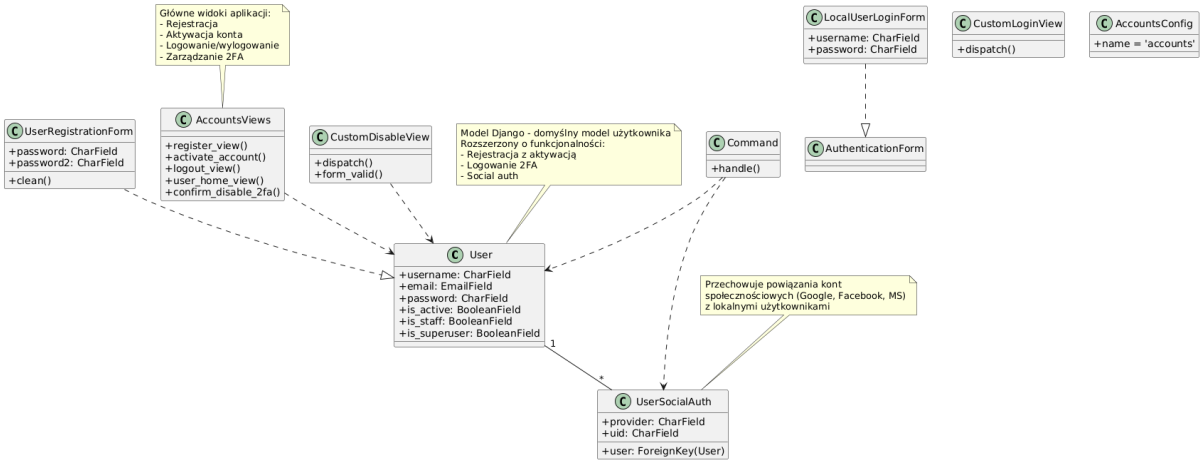
- **Rejestracja** - założenie nowego konta w systemie
- **Logowanie/Wylogowanie** - uwierzytelnianie w systemie
- **Zarządzanie 2FA** - konfiguracja uwierzytelniania dwuetapowego

Funkcjonalności administratora

- **Logowanie przez OAuth** - uwierzytelnianie przez zewnętrznych dostawców
- **Zarządzanie użytkownikami** - dodawanie, usuwanie i edycja kont
- **Weryfikacja kont** - potwierdzanie statusu kont użytkowników
- **Przegląd historii logowań** - monitorowanie aktywności w systemie

Diagram wizualizuje wszystkie możliwe akcje wykonywane przez poszczególne role w systemie, uwzględniając interakcje między aktorami a funkcjonalnościami systemu.

8.3 Diagram klas



Rysunek 5: Diagram klas

Diagram przedstawia główne komponenty systemu uwierzytelniania oraz ich wzajemne relacje. Poniżej znajduje się szczegółowy opis elementów widocznych na diagramie.

8.3.1 Opis klas

- **User** - podstawowy model użytkownika Django rozszerzony o:
  - Mechanizm aktywacji przez e-mail
  - Wsparcie dla uwierzytelniania dwuskładnikowego
  - Integrację z logowaniem społecznościowym
- **UserSocialAuth** - przechowuje powiązania między kontem lokalnym a:
  - Google OAuth2 (pole `provider='google-oauth2'`)
  - Facebook (pole `provider='facebook'`)
  - Microsoft Graph (pole `provider='microsoft-graph'`)

8.3.2 Relacje

Na diagramie szczególnie istotne są następujące zależności:

- Asocjacja **1-*** między `User` a `UserSocialAuth` oznaczająca, że:
  - Jeden użytkownik może mieć wiele powiązanych kont społecznościowych
  - Każde konto społecznościowe jest przypisane do dokładnie jednego użytkownika
- Dziedziczenie formularzy:

- `UserRegistrationForm` dziedziczy z `ModelForm`
- `LocalUserLoginForm` dziedziczy z `AuthenticationForm`

- Zależności widoków:
  - `CustomLoginView` rozszerza `LoginView`
  - `CustomDisableView` dziedziczy z `FormView`

8.3.3 Metody kluczowe

- W klasie `UserRegistrationForm`:
  - `clean()` - walidacja zgodności haseł
- W klasie `CustomDisableView`:
  - `form_valid()` - wysyłka maila potwierdzającego wyłączenie 2FA
- W komendzie `create_oauth_admin`:
  - `handle()` - główna logika tworzenia administratora

9 Wdrożenie

9.1 Dane:

9.1.1 Środowisko i technologie

Aplikacja działa w środowisku **Linux** (dystrybucja **Ubuntu**) i wykorzystuje **serwer aplikacyjny uWSGI**. Główną bazą danych jest **SQLite**, a kod napisany jest w **Pythonie 3.13**.

9.1.2 Hostowanie

Aplikacja hostowana jest na **eu.pythonanywhere**

9.1.3 Zarządzanie kodem źródłowym

Kod źródłowy przechowywany jest na **GitHub**, a testy jednostkowe są automatycznie uruchamiane za pomocą **GitHub Actions**.

9.1.4 Proces wdrożenia

Proces wdrożenia odbywa się **manualnie** poprzez połączenie **SSH**, które pobiera najnowszą wersję repozytorium z GitHub.

- GitHub SSH
  - Bezpieczeństwo - klucze prywatne nigdy nie opuszczają maszyny
  - Wygoda - nie trzeba za każdym razem podadać danych uwieerzytelniających

9.1.5 Testowanie

Po wdrożeniu każdej nowej wersji na produkcję wykonywane są **scenariusze testowe**, aby zweryfikować poprawność działania aplikacji.

10 Podział pracy

10.1 Aleksandra Lewandowska

- Dokumentacja aplikacji

10.2 Kacper Malik

- Testowanie aplikacji
- Scenariusze testowe
- Pomoc w dokumentacji

10.3 Franciszek Łajszczak

- **zarządzanie zespołem**
- Backend aplikacji
- Deployment aplikacji

10.4 Daniel Pietruczyk-Phan

- Diagramy
- Pomoc w dokumentacji