Opis funkcjonalności i sposobu uruchomienia aplikacji IP Tagger

Wykorzystanie aplikacji.

Aplikacja służy do tworzenia w szybki sposób dużej bazy adresów IP z odpowiednimi tagami, oraz dodawania, usuwania, edytowania adresów IP i tagów. Wykorzystanie aplikacji to między innymi:

1. **Zarządzanie adresacją IP**:

   - **Firmy i korporacje**: W dużych organizacjach, które mają setki lub tysiące urządzeń, aplikacja może pomagać w zarządzaniu przydzielaniem adresów IP, monitorowaniu ich użycia oraz unikaniu konfliktów adresów.
   - **Dostawcy usług internetowych (ISP)**: ISP mogą używać takiej aplikacji do zarządzania blokami adresów IP przypisanych do różnych klientów, lokalizacji lub typów usług.

2. **Bezpieczeństwo sieci**:

   - **Monitoring i analiza ruchu sieciowego**: Przypisywanie tagów do adresów IP może pomóc w szybkiej identyfikacji źródeł ruchu sieciowego, co jest przydatne w przypadku incydentów bezpieczeństwa.
   - **Blokowanie podejrzanych adresów**: Dzięki tagom można łatwo identyfikować i blokować podejrzane adresy IP, np. te związane z aktywnością botnetów lub próbami ataków.

3. **Zarządzanie centrami danych**:

   - **Automatyzacja zadań**: Aplikacja może wspomagać automatyzację przydzielania i zwalniania adresów IP w dynamicznych środowiskach, takich jak centra danych czy chmury obliczeniowe.
   - **Śledzenie zasobów**: Przypisane tagi mogą pomóc w śledzeniu, które zasoby (np. serwery, urządzenia sieciowe) są przypisane do określonych adresów IP.

4. **Zgodność z regulacjami i audyt**:

   - **Raportowanie i audyty**: Możliwość przypisywania tagów do adresów IP ułatwia tworzenie raportów i przeprowadzanie audytów zgodności z regulacjami dotyczącymi zarządzania danymi i bezpieczeństwa.

5. **Analityka i raportowanie**:

   - **Analiza statystyczna**: Tagi mogą być używane do kategoryzacji i analizy statystycznej użycia adresów IP w różnych segmentach sieci.
   - **Śledzenie zmian**: Aplikacja może monitorować zmiany w przypisaniu adresów IP, co jest przydatne do analizowania trendów i planowania przyszłych potrzeb.

6. **Optymalizacja i planowanie sieci**:

- **Planowanie rozbudowy sieci**: Dzięki analizie użycia adresów IP można lepiej planować przyszłą rozbudowę infrastruktury sieciowej.
- **Optymalizacja zasobów**: Przypisane tagi mogą pomóc w identyfikacji niewykorzystanych lub niedostatecznie wykorzystanych adresów IP, co pozwala na bardziej efektywne zarządzanie zasobami.

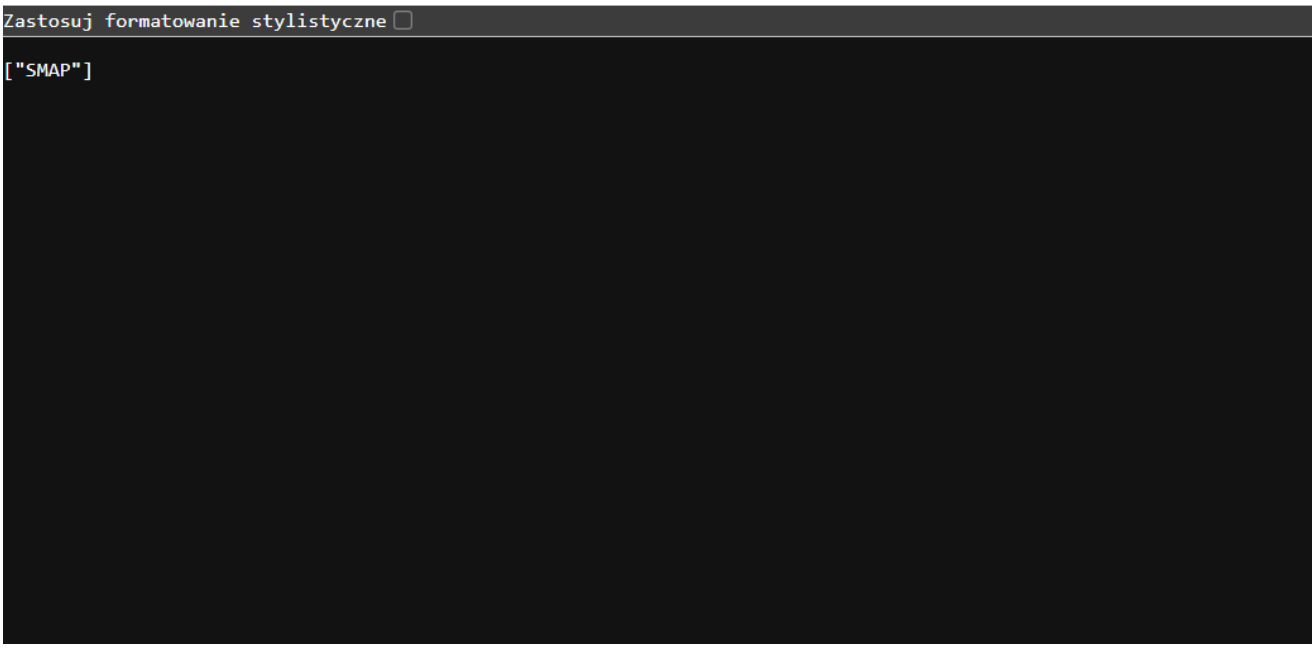
Główne wymagania funkcjonalne.

Utworzenie API dla dwóch endpointów:

1. GET /ip-tags-json/{ip}

Ten endpoint, w odpowiedzi na żądanie klienta, powinien zwracać dokument w formacie JSON, zawierający

listę tagów, które odpowiadają danemu adresowi IP (lub pustą listę, jeśli adresowi nie odpowiada żaden tag;



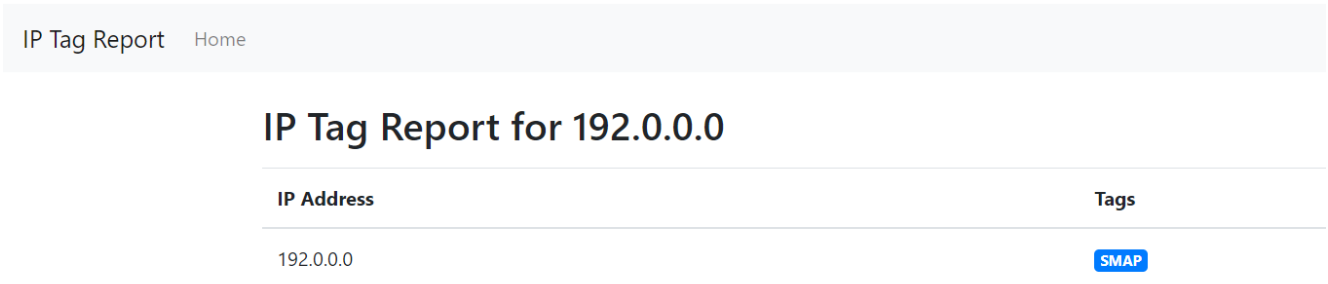
2. GET /ip-tags-report/{ip}

Ten endpoint, w odpowiedzi na żądanie klienta, powinien zwracać dokument w formacie HTML,

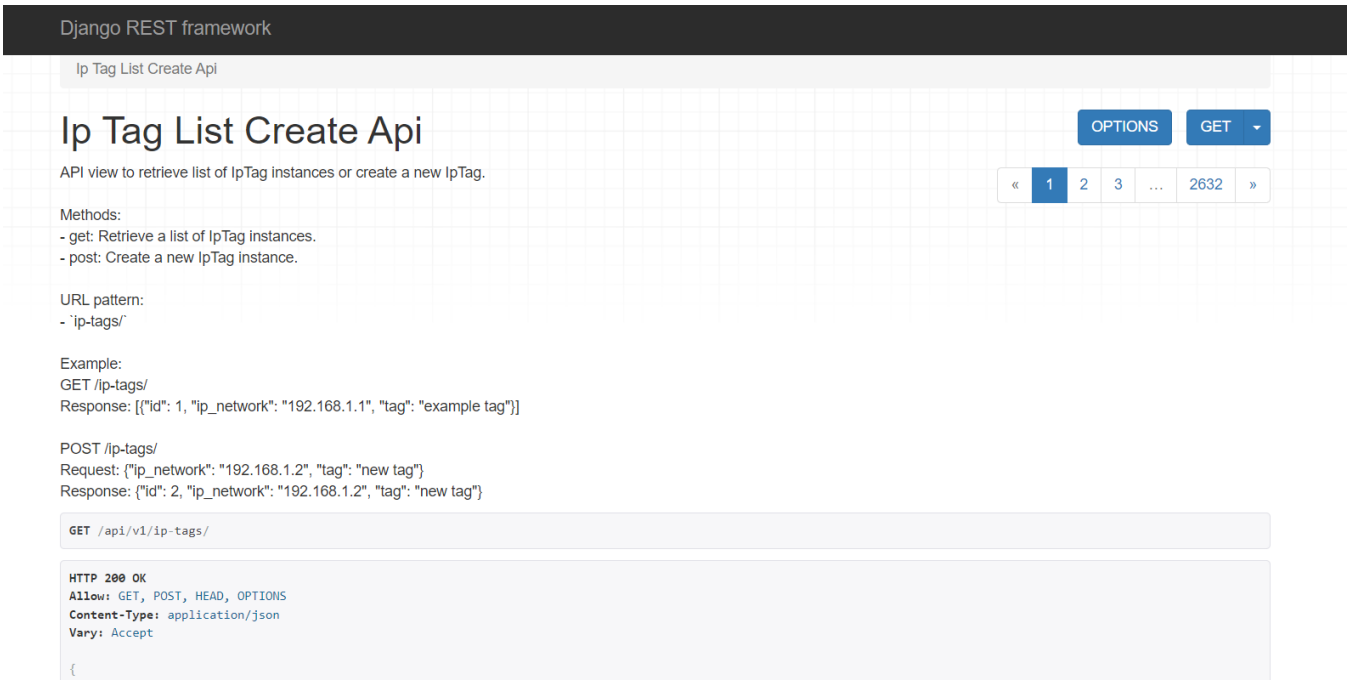
zawierający tabelę prezentującą tagi, które odpowiadają danemu adresowi IP – np.:

Adres IP Pasujące tagi

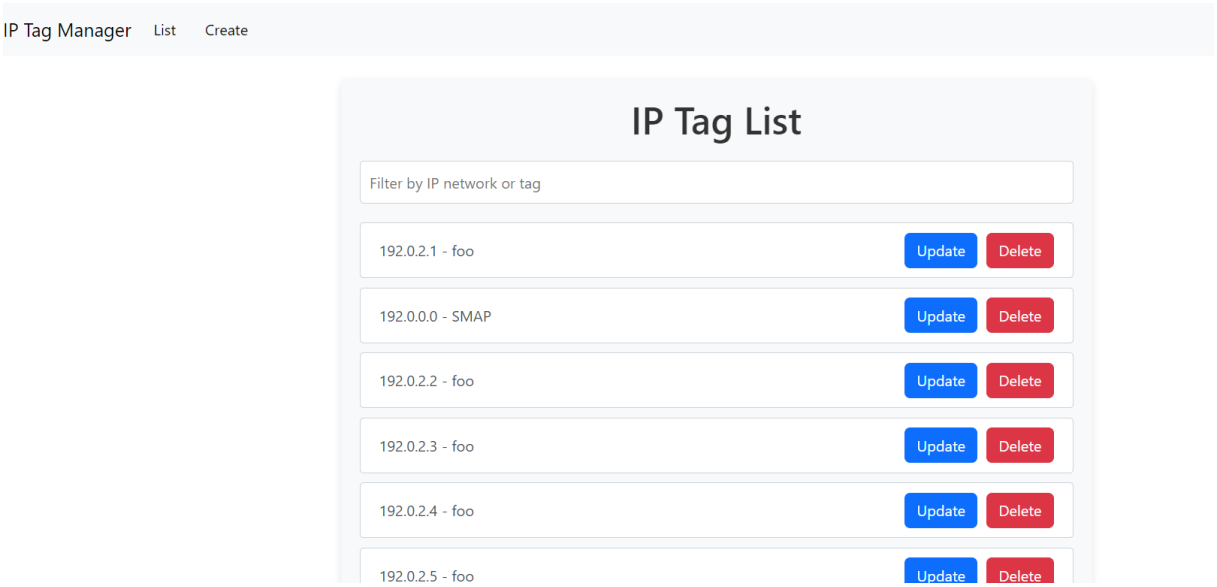
198.51.100.227 just a TAG

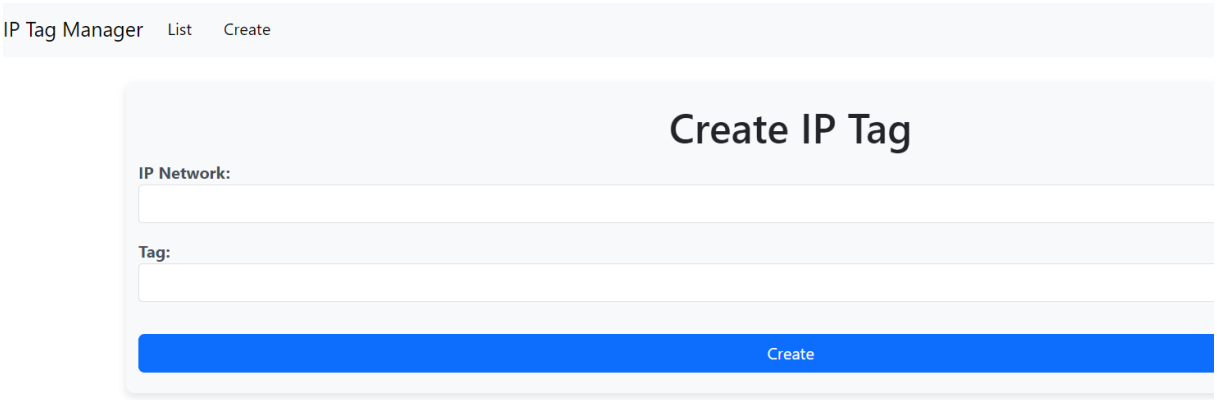


3. Utworzenie API do obsługi Create, Read, Update, Destroy.



4. Utworzenie warstwy frontend do obsługi API





Baza wiedzy i logika dopasowywania tagów do adresów IP.

Baza wiedzy powinna zostać odczytana podczas inicjalizacji usługi z pliku w formacie JSON. Ścieżkę do tego pliku powinna określać konfiguracja programu (sposób konfiguracji jest dowolny, ale musi być opisany w README.md). Należy założyć, że dane w pliku JSON są zawsze poprawne i kompletne.

Baza wiedzy – po odczytaniu i deserializacji – jest listą (obiektem typu list), której elementami są słowniki (obiekty typu dict). Każdy z tych słowników ma następującą strukturę: {"tag": WARTOŚĆ, "ip_network": WARTOŚĆ}

Warstwa reprezentacyjna

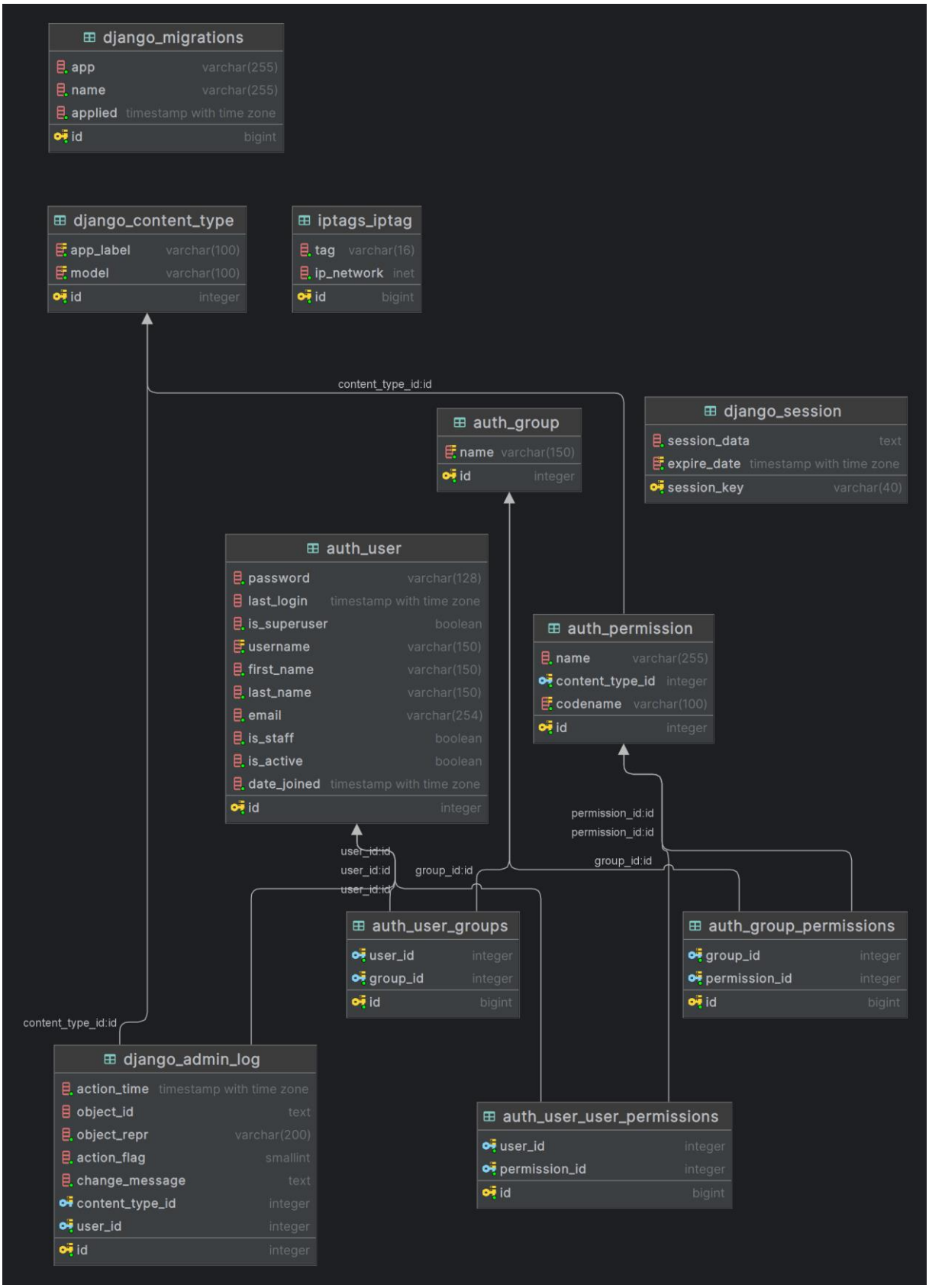
Warstwa prezentacyjna projektu jest stworzona na bazie:

- Django Template
- HTML
- Bootstrap
- React

Użyty stack technologiczny.

- Python
- Django
- DRF
- pytest
- Docker
- React
- Postgresql

Diagram klas



Uruchomienie projektu

1. Klonujemy projekt:

```bash
git@github.com:BlazejBielski/ip_management.git
```

2. Przechodzimy do katalogu z projektem.

3. Kopiujemy pliki z katalogu envs nadając im odpowiednio nazwy:

- backend.env
- postgres.env

4. Przykładowe uzupełnienie plików env:

```
DJ_SECRET_KEY="django-insecure-ih$-8kb2(_49%8t=+o^s$n=0rvpqci0d-8hgui&-phzslfi4y5"
DJ_DEBUG=1
DJ_ALLOWED_HOSTS=localhost 0.0.0.0 127.0.0.1
DJANGO_LOGLEVEL='info'
```

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=postgres
POSTGRES_HOST=postgres
POSTGRES_PORT=5432

DB_CONNECTION_STRING=postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${POSTGRES_HOST}:${POSTGRES_PORT}/${POSTGRES_DB}
```

5. Uruchamiamy projekt.

```bash
```

docker compose up

```

Uwaga — serwis przy starcie zasila bazę danych z bazy wiedzy, może to potrwać do kilku minut.

Serwis API i dwa pierwsze punkty z funkcjonalności wystartują pod adresem 127.0.0.1:8000.

Uruchomienie frontendu.

Upewniamy się, że jesteśmy w katalogu frontend.

Uruchamiamy komendą npm start

## Testy jednostkowe

Testy jednostkowe obejmują podstawowe testy modeli, serializerów i widoków.
Uruchomienie testów jest możliwe tylko po wystartowaniu projektu.

1. Wewnątrz kontenera:

- uruchamiamy powłokę kontenera komendą

```bash

docker compose exec api bash

```

wewnątrz kontenera odpalamy testy komendą

```bash

pytest

```

2. Spoza kontenera:

```bash

docker compose exec api pytest

```
```