Dokumentacja techniczna - programowanie zespołowe

czerwiec 2024

1 Opis projektu

1.1 Cel i zakres projektu

Celem projektu jest stworzenie zaawansowanej aplikacji do zarządzania urządzeniami, która spełnia potrzeby w zakresie efektywnego zarządzania bazą danych urządzeń. Aplikacja ma umożliwiać użytkownikom dodawanie, usuwanie i edytowanie wpisów w bazie danych, wyświetlanie urządzeń w postaci tabeli, filtrowanie urządzeń oraz integrację z systemem wgrywania danych. Ponadto, aplikacja ma generować kody QR z podstawowymi informacjami, być zabezpieczona protokołem HTTPS oraz udostępniać interfejs API. Dodatkowo, projekt obejmuje implementację funkcjonalności logowania przez SSO oraz zaawansowane funkcje administracyjne.

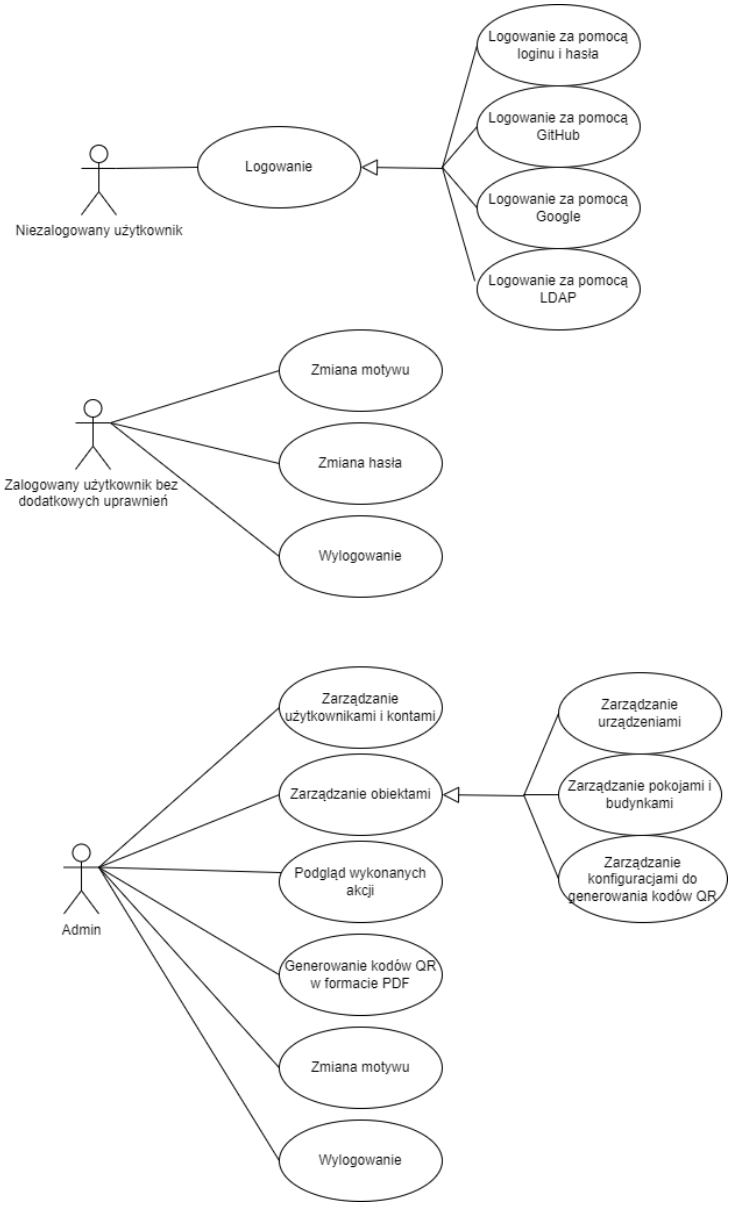
1.2 Kluczowe funkcjonalności

1. Zarządzanie bazą danych:
   - Dodawanie, usuwanie i edytowanie pól w bazie danych.
   - Wyświetlanie urządzeń w formie tabelarycznej.
   - Filtrowanie urządzeń według różnych kryteriów.
   - Integracja z systemem wgrywania danych do nowo utworzonej bazy danych.
   - Generowanie kodów QR zawierających podstawowe informacje o urządzeniach.
2. Bezpieczeństwo i protokoły:
   - Zabezpieczenie aplikacji protokołem HTTPS.
   - Implementacja SSO, w tym logowanie przez Github.
3. Interfejs użytkownika:
   - Parametryzacja wielkości etykiet.
   - Możliwość zmiany wyglądu poprzez podmianę arkuszy stylów CSS.
   - Obsługa trzech typów użytkowników: Viewer, Edytor, Superadmin.
   - Sterowanie aplikacją za pomocą klawiatury.
   - Realizacja operacji poprzez listy rozwijane (select) lub ręczne wprowadzanie danych.
4. Dodatkowe funkcjonalności:
   - Rejestracja i przechowywanie logów operacji.
   - Przygotowanie kontenera oraz plików z volumenem.
   - Dodanie panelu administratora do zarządzania grupami użytkowników i ich uprawnieniami.
   - Ustalenie technologii bazy danych na infrastrukturze MariaDB.

2 Przypadki użycia

Na rysunku 1 przedstawiono diagram przypadków użycia. W zależności od roli użytkownik może wykonać różne akcje:

- Niezalogowany użytkownik: Może tylko się zalogować. Może do tego wykorzystać login i hasło, konto Github, konto Google lub konto na Taurusie.
- Zalogowany użytkownik bez dodatkowych uprawnień: Poza zmianą motywu lub hasła, może się też wylogować.
- Admin posiada wszystkie uprawnienia. Dodatkowo może zarządzać użytkownikami i kontami: dodawać i usuwać użytkowników, nadawać uprawnienia, dodawać do i tworzyć grupy użytkowników, edytować dane. Zarządzanie obiektami (urządzeniami, pokojami, budynkami, wydziałami, konfiguracjami) polega na możliwości dodawania, usuwania, edytowania, przeglądania i filtrowania wyszukiwań obiektów. Podgląd wykonanych akcji umożliwia adminowi sprawdzenie historii wykonanych przez niego zmian w aplikacji. Admin może też wygenerować PDF, zawierający kody QR, reprezentujące wybrane wcześniej urządzenia.



Rysunek 1: Diagram przypadków użycia.

3 Technologie i narzędzia

3.1 Technologie:

- **Język programowania**: Python
- **Framework webowy:** Django
- **Serwer HTTP:** Gunicorn
- **Baza danych:** MariaDB
- **Protokół:** HTTPS
- **SSO (Single Sign-On)**: Implementacja SSO, w tym obsługa logowania przez Github.
- **Technologie frontendowe:** HTML, CSS, JavaScript

3.2 Narzędzia do zarządzania projektem:

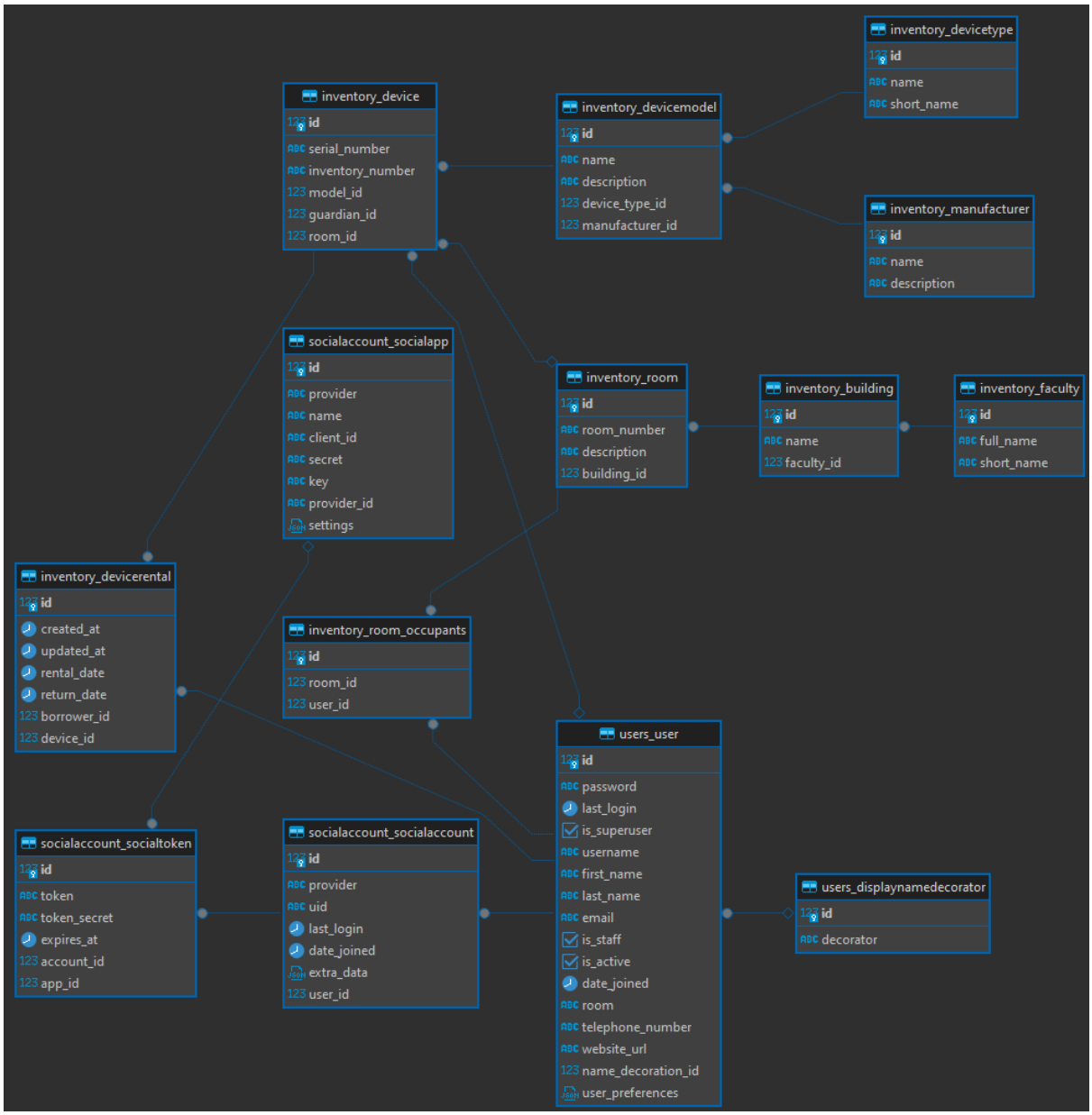
- **System kontroli wersji**: Git z repozytorium na platformie GitHub
- **Konteneryzacja:** Docker (zaimplementowany kontener oraz pliki z volumenem)

4 Struktura bazy danych

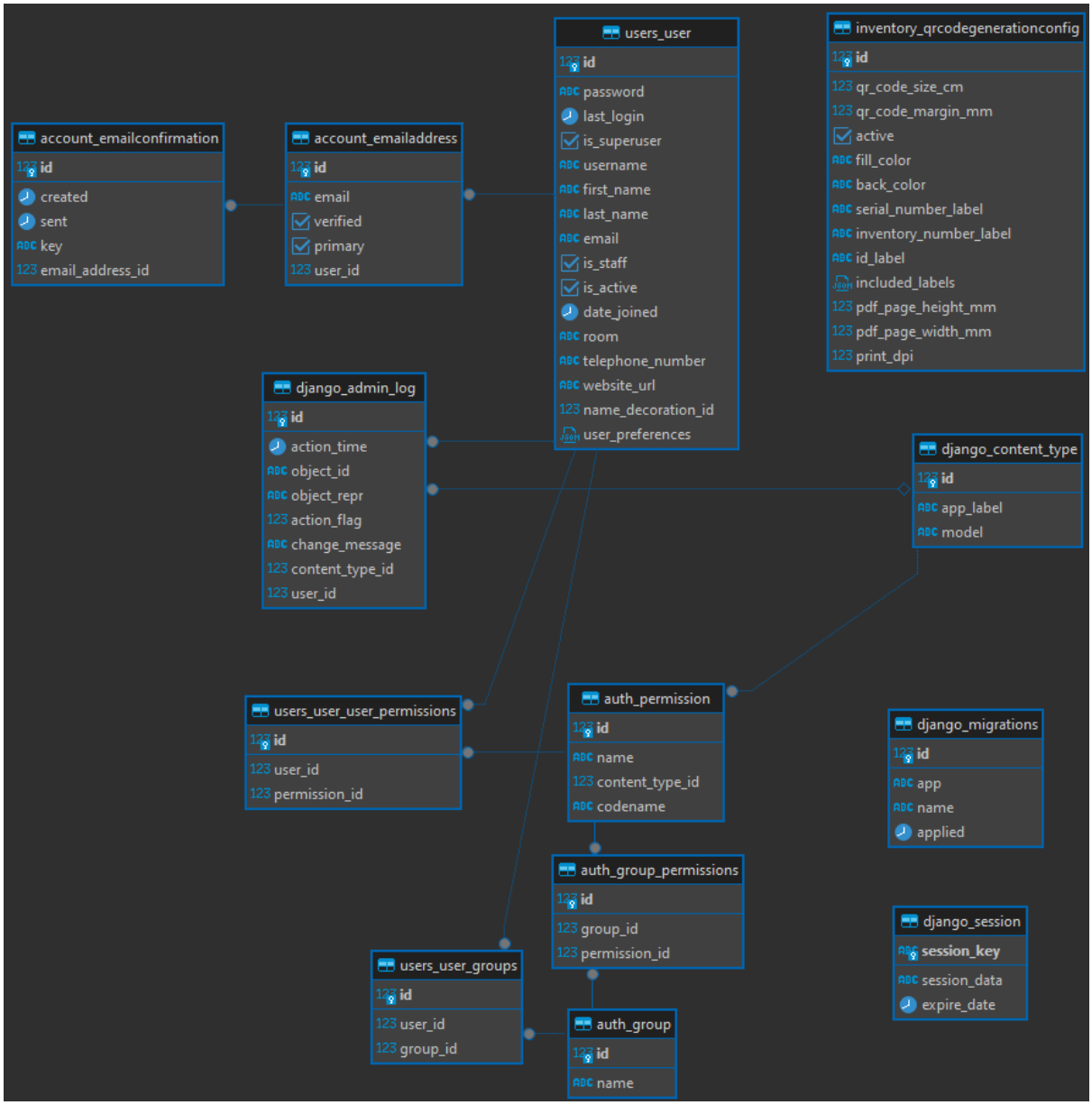
Na rysunku 2 i 3 przedstawiono schemat bazy danych, wykorzystywanej w aplikacji. Składa się ona z następujących tabeli:

- inventory_device – zawiera urządzenia
- inventory_device_model – zawiera modele urządzenia
- inventory_devicetype – zawiera typy urządzenia
- inventory_manufacturer – zawiera producentów urządzeń
- socialaccount_socialapp – zawiera sposoby uwierzytelniania za pomocą innych aplikacji
- inventory_room – zawiera pokoje
- inventory_building – zawiera budynki
- inventory_faculty – zawiera wydziały
- inventory_faculty – zawiera wypożyczenia urządzeń
- inventory_room_occupants – zawiera osoby, zajmujące konkretny pokój
- socialaccount_socialtoken – zawiera token stworzony podczas uwierzytelniania przez inną aplikację
- socialaccount_socialaccount – zawiera konta utworzone podczas uwierzytelniania przez inną aplikację
- users_user – zawiera użytkowników
- users_displanamedecorator – zawiera tytuły naukowe
- account_emailconfirmation – zawiera potwierdzenia posiadania podanego adresu email
- account_emailaddress – zawiera adresy emial
- inventory_qrcodegenerationconfig – zawiera konfiguracje wykorzystywane podczas generowania kodu QR
- django_admin_log – zawiera zapisy akcji wykonywanych przez użytkowników
- django_content_type – zawiera modele wykorzystywane w aplikacji

- users_user_user_permissions – zawiera uprawnienia konkretnego użytkownika
- auth_permissions – zawiera uprawnienia
- django_migrations – zawiera migracje
- auth_group_permissions – zawiera uprawnienia grupy użytkowników
- django_session – zawiera sesje
- user_user_groups – zawiera użytkowników należących do konkretnej grupy
- auth_group – zawiera grupy użytkowników



Rysunek 2: Schemat bazy danych cz.1.



Rysunek 3: Schemat bazy danych cz. 2.

5 Endpointy

W tej sekcji zostaną opisane endpointy aplikacji.

5.1 /inventory/buildings/

Zapytanie GET zwraca listę wszystkich budynków.
Zapytanie POST pozwala na dodanie budynku do bazy danych. Wymagane jest podanie w body obiektu typu Building.

5.2 /inventory/buildings/{id}/

Zapytanie GET pobiera dane budynku o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane budynku o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu Building.
Zapytanie PATCH służy do częściowej edycji danych budynku o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.

Zapytanie DELETE usuwa budynek o id wskazanym w adresie URL.

5.3 /inventory/device-models/

Zapytanie GET zwraca listę wszystkich modeli urządzeń.
Zapytanie POST pozwala na dodanie modelu urządzenia do bazy danych. Wymagane jest podanie w body obiektu typu DeviceModel.

5.4 /inventory/device-models/{id}/

Zapytanie GET pobiera dane modelu urządzenia o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane modelu urządzenia o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu DeviceModel.
Zapytanie PATCH służy do częściowej edycji danych modelu urządzenia o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa model urządzenie o id wskazanym w adresie URL.

5.5 /inventory/device-types/

Zapytanie GET zwraca listę wszystkich typów urządzeń.
Zapytanie POST pozwala na dodanie typu urządzenia do bazy danych. Wymagane jest podanie w body obiektu typu DeviceType.

5.6 /inventory/device-types/{id}/

Zapytanie GET pobiera dane typu urządzenia o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane typu urządzenia o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu DeviceType.
Zapytanie PATCH służy do częściowej edycji danych typu urządzenia o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa typ urządzenia o id wskazanym w adresie URL.

5.7 /inventory/devices/

Zapytanie GET zwraca listę wszystkich urządzeń.
Zapytanie POST pozwala na dodanie urządzenia do bazy danych. Wymagane jest podanie w body obiektu typu QRCodeData.

5.8 /inventory/devices/{id}/

Zapytanie GET pobiera dane urządzenia o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane urządzenia o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu QRCodeData.
Zapytanie PATCH służy do częściowej edycji danych urządzenia o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa urządzenie o id wskazanym w adresie URL.

5.9 /inventory/faculties/

Zapytanie GET zwraca listę wszystkich wydziałów.
Zapytanie POST pozwala na dodanie wydziału do bazy danych. Wymagane jest podanie w body obiektu typu Faculty.

5.10 /inventory/faculties/{id}/

Zapytanie GET pobiera dane wydziału o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane wydziału o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu Faculty.
Zapytanie PATCH służy do częściowej edycji danych wydziału o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa wydział o id wskazanym w adresie URL.

5.11 /inventory/manufacturers/

Zapytanie GET zwraca listę wszystkich producentów urządzeń.
Zapytanie POST pozwala na dodanie producenta urządzeń do bazy danych. Wymagane jest podanie w body obiektu typu Manufacturer.

5.12 /inventory/manufacturers/{id}/

Zapytanie GET pobiera dane producenta urządzeń o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane producenta urządzeń o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu Manufacturer.
Zapytanie PATCH służy do częściowej edycji danych producenta urządzeń o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa producenta urządzeń o id wskazanym w adresie URL.

5.13 /inventory/qr-generate/?id=

Zapytanie GET zwraca dane urządzeń o id oddzielonych przecinkiem, podanych jako parametr. Opcjonalne parametry zapytania:

- back_color – kolor tła, domyślnie biały
- fill_color – kolor kodu QR, domyślnie czarny
- pdf_page_height_mm – wysokość wygenerowanej strony pdf w mm, domyślnie 297
- pdf_page_width_mm – szerokość wygenerowanej strony pdf w mm, domyślnie 210
- print_dpi – rozdzielczość generowanych kodów QR w dpi, domyślnie 72
- qr_code_margin_mm – rozmiar marginesu wokół kodu QR w mm, domyślnie 3
- qr_code_size_cm – rozmiar kodu QR w cm, domyślnie 5

5.14 /inventory/rooms/

Zapytanie GET zwraca listę wszystkich pokoi.
Zapytanie POST pozwala na dodanie pokoju do bazy danych. Wymagane jest podanie w body obiektu typu Room.

5.15 /inventory/rooms/{id}/

Zapytanie GET pobiera dane pokoju o id wskazanym w adresie URL.
Zapytanie PUT edytuje dane pokoju o id wskazanym w adresie URL. Wymagane jest podanie w body obiektu typu Room.
Zapytanie PATCH służy do częściowej edycji danych pokoju o id wskazanym w adresie URL. Edytowane są tylko dane wysłane w body.
Zapytanie DELETE usuwa pokój o id wskazanym w adresie URL.

5.16 /users/api-token/

Zapytanie GET przyjmuje w body nazwę użytkownika oraz hasło. Zwraca token, który w celach uwierzytelniania będzie zawarty w nagłówku innych zapytań. Nagłówek przyjmuje postać *Authorization: Token XXX*, gdzie XXX to zwrócony przez endpoint token.

6 Środowiska uruchomieniowe

6.1 Opis środowisk

- **Środowisko deweloperskie:**
  - Używane do tworzenia i testowania nowych funkcjonalności.
  - Wykorzystuje lokalną instalację wszystkich zależności i narzędzi.
  - Umożliwia szybkie iteracje i testowanie zmian w kodzie.
- **Środowisko produkcyjne:**
  - Ostateczne środowisko, w którym aplikacja jest dostępna dla użytkowników końcowych.
  - Skonfigurowane z naciskiem na wydajność, bezpieczeństwo i skalowalność.

6.2 Instrukcje dotyczące konfiguracji i wdrożenia

6.2.1 Środowisko deweloperskie

Kroki instalacji środowiska programistycznego

1. **Zainstaluj Mise:**

   ```
   $ curl https://mise.run | sh
   $ echo 'export PATH="$HOME/.mise/bin:$PATH"' >> ~/.bashrc;
   echo 'export PATH="$HOME/.mise/bin:$PATH"' >> ~/.zshrc
   ```

   Restartuj terminal:

   ```
   $ exec bash  # dla Bash
   $ exec zsh   # dla Zsh
   ```

   Zweryfikuj instalację:

   ```
   $ mise version
   2024.4.5 linux-arm64 (d60d850 2024-04-15)
   ```

2. **Zainstaluj zależności systemowe (Debian/Ubuntu):**

   ```
   $ sudo apt install build-essential gdb lcov pkg-config \
         libbz2-dev libffi-dev libgdbm-dev libgdbm-compat-dev liblzma-dev \
         libncurses5-dev libreadline6-dev libsqlite3-dev libssl-dev \
         lzma lzma-dev tk-dev uuid-dev zlib1g-dev libmariadb-dev
   ```

3. **Skonfiguruj projekt:**

   (a) Sklonuj repozytorium:

   ```
   $ git clone https://github.com/Teamdur/DeviceManager.git
   ```

   (b) Przejdź do katalogu projektu:

   ```
   $ cd DeviceManager
   ```

   (c) Zainicjuj Mise:

   ```
   $ mise trust
   $ mise settings set experimental true
   $ mise install
   ```

   (d) Uruchom zadanie konfiguracji:

   ```
   $ mise run setup-dev
   ```

Instrukcje uruchamiania aplikacji lokalnie

```
$ mise run dev
```

6.2.2 Środowisko produkcyjne

Instrukcje instalacji i uruchomienia aplikacji na serwerze

1. **Ustawienie zmiennych środowiskowych:**

```
export SECRET_KEY='your_secret_key'
export MARIADB_USER='your_mariadb_user'
export MARIADB_PASSWORD='your_mariadb_password'
export MARIADB_DATABASE='your_mariadb_database'
export MARIADB_HOST='your_mariadb_host'
export MARIADB_PORT='your_mariadb_port'
export EMAIL_BACKEND='django.core.mail.backends.smtp.EmailBackend'
export EMAIL_HOST='your_email_host'
export EMAIL_PORT='your_email_port'
export EMAIL_HOST_USER='your_email_user'
export EMAIL_HOST_PASSWORD='your_email_password'
export EMAIL_USE_SSL=True
export EMAIL_USE_TLS=False
export GOOGLE_CLIENT_ID='your_google_client_id'
export GOOGLE_CLIENT_SECRET='your_google_client_secret'
export GITHUB_CLIENT_ID='your_github_client_id'
export GITHUB_CLIENT_SECRET='your_github_client_secret'
export AUTHENTIK_CLIENT_ID='your_authentik_client_id'
export AUTHENTIK_CLIENT_SECRET='your_authentik_client_secret'
export API_PORT=api_port
export STATIC_PORT=static_port
```

2. **Używanie pliku .env do ustawiania zmiennych środowiskowych:**
   Zamiast eksportowania zmiennych środowiskowych, można umieścić je w pliku '.env':

```
SECRET_KEY=your_secret_key
MARIADB_USER=your_mariadb_user
MARIADB_PASSWORD=your_mariadb_password
MARIADB_DATABASE=your_mariadb_database
MARIADB_HOST=your_mariadb_host
MARIADB_PORT=your_mariadb_port
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=your_email_host
EMAIL_PORT=your_email_port
EMAIL_HOST_USER=your_email_user
EMAIL_HOST_PASSWORD=your_email_password
EMAIL_USE_SSL=True
EMAIL_USE_TLS=False
GOOGLE_CLIENT_ID=your_google_client_id
GOOGLE_CLIENT_SECRET=your_google_client_secret
GITHUB_CLIENT_ID=your_github_client_id
GITHUB_CLIENT_SECRET=your_github_client_secret
AUTHENTIK_CLIENT_ID=your_authentik_client_id
AUTHENTIK_CLIENT_SECRET=your_authentik_client_secret
API_PORT=api_port
STATIC_PORT=static_port
```

   Następnie, aby uruchomić kontenery Docker, użyj następującej komendy:

```
docker compose up -d
```

   Kontenery Docker automatycznie załadują zmienne środowiskowe z pliku '.env'.

3. **Uruchomienie aplikacji na serwerze:**
   Uruchomienie projektu następuje poprzez wykonanie polecenia:

```
docker compose -f production.compose.yml up -d
```

6.2.3 Migracja danych:

W celu migracji danych z zrzutu bazy danych do aplikacji należy użyć skryptu data_migration.py. W linii komend należy podać plik, z którego zczytywane będą dane:

```
python data_migration.py FILE
```

Skrypt wygeneruje plik fixtures.yaml w katalogu /devicemanager/fixtures z danymi z tabel: grupy, inv_budynki, inv_modele, inv_pokoje, inv_producenci, inv_typy_urz, inv_urzadzenia, pracownicy, stopnie.

6.2.4 Backup danych:

1. **Tworzenie kopii zapasowej danych:**

   ```
   ./manage.py dumpdata [app_label[.ModelName]} > backup.json
   ```

   Za pomocą `app_label` można wybrać aplikację, z której dane zostaną zapisane w pliku. `ModelName` pozwala na wybór konkretnego modelu z danej aplikacji. Jeśli nie zostaną one podane, to wszystkie dane zostaną zapisane w pliku. Dodatkowo można skorzystać z flagi `--format` i podać w format, w którym zostaną zapisane dane. Domyślny to json.

2. **Ładowanie danych:**

   ```
   ./manage.py loaddata backup.json
   ```

6.2.5 Ceryfikaty domeny:

Użytkownik musi zapewnić sobie certyfikaty domeny we własnym zakresie.

6.2.6 Dane OAuth2

W poniższym podrozdziale zostanie opisane, skąd wziąć dane, aby umożliwić aplikacji uwierzytelnianie użytkowników za pomocą konta Google i konta Github.

- **Google:**
  1. Należy odwiedzić Google Cloud (https://console.cloud.google.com/) i zalogować się.
  2. W lewym górnym rogu należy kliknąć *Select a project*, a następnie *New project*.
  3. Następnie należy podać nazwę projektu i kliknąć *Create*.
  4. Z listy w lewym górnym rogu należy wybrać utworzony projekt, a następnie kliknąć *APIs & Services*.
  5. Z menu po lewej stronie należy wybrać *OAuth consent screen*.
  6. Należy kliknąć *External* i *Create*, a następnie uzupełnić formularz.
  7. W zakładce *Scopes* należy wybrać *email* i *profile*.
  8. Następne dwie zakładki można pominąć.
  9. Z menu po lewej stronie należy wybrać *Credentials*.
  10. Należy kliknąć *Create credentials*, a następnie *OAuth client ID*.
  11. Należy wybrać *Web application*, podać nazwę aplikacji i dodać do *Authorized redirect URIs* następujące URI `http:/<address>:<port>/oauth2/accounts/google/login/callback/`. Oczywiście, address i port muszą odpowiadać tym, na których pracuje aplikacja.
  12. Po wciśnięciu *Create* wyświetlone zostaną *Client ID* i *Client secret*
- **Github:**
  1. Należy wejść na stronę Githuba (https://github.com/), zalogować się, kliknąć w swoje zdjęcie profilowe i wybrać *Settings*.

2. Z lewej strony, na dole należy wybrać *Developer Settings*, a następnie *OAuth Apps* i *New OAuth App*.
3. Należy wypełnić formularz, do pola *Authorization callback URL* należy wpisać `http:/<address>:<port>/oauth2/accounts/github/login/callback/` z odpowiednim adresem i portem.
4. Po wypełnieniu formularza i wciśnięciu *Register application* na ekranie pojawią się *Client ID* i *Client secrets*, które należy zapisać.

6.2.7 Konfiguracja Nginx:

Poniżej znajduje się przykładowy fragment konfiguracji serwera Nginx:

```
server {
    server_name device-manager.critteros.dev;

    access_log /var/log/nginx/devicemanager.log;
    error_log /var/log/nginx/devicemanager-error.log;

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/device-manager.critteros.dev/fullchain.pem; # managed by C
    ssl_certificate_key /etc/letsencrypt/live/device-manager.critteros.dev/privkey.pem; # managed by
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto https;
    location / {
        proxy_pass http://localhost:11011;
    }
    location ~ ^/(static|media)/ {
        proxy_pass http://localhost:11012;
    }

}
```

Opis konfiguracji:

- **Certyfikaty SSL:** Certyfikaty SSL są zarządzane przez Certbot, a Nginx obsługuje ruch na porcie 443.
- **Serwer aplikacji:** Ruch na głównym adresie URL (`/`) jest przekazywany do serwera aplikacji działającego na porcie 11011, gdzie pracuje Gunicorn.
- **Serwer plików statycznych:** Żądania dotyczące plików statycznych i zasobów (`/static` lub `/media`) są przekazywane do serwera działającego na porcie 11012, który obsługuje pliki statyczne, JavaScript i CSS.

7 Wymagania niefunkcjonalne

7.1 Wydajność

Aplikacja do zarządzania urządzeniami została zaprojektowana i wdrożona z uwzględnieniem wysokiej wydajności. Spełnia ona następujące kryteria:

- **Szybkość działania:** Aplikacja reaguje na działania użytkowników bez zauważalnych opóźnień. Czas odpowiedzi na podstawowe operacje, takie jak dodawanie, usuwanie i edytowanie danych, jest minimalny.
- **Optymalizacja kodu:** Kod aplikacji został zoptymalizowany pod kątem szybkości działania, co minimalizuje zużycie zasobów systemowych.

7.2 gUnicorn

Serwer HTTP gUnicorn (Green Unicorn) został użyty do uruchomienia aplikacji Django. gUnicorn zapewnia:

- **Wysoką wydajność:** gUnicorn obsługuje wielowątkowość i wieloprocesowość, co pozwala na efektywne przetwarzanie dużej liczby równoczesnych żądań.
- **Stabilność i niezawodność:** gUnicorn jest dobrze znanym i szeroko stosowanym serwerem w środowiskach produkcyjnych, zapewniając stabilność i niezawodność działania aplikacji.

Konfiguracja gUnicorn obejmowała:

- **Optymalizację serwera:** gUnicorn został skonfigurowany zgodnie z dobrymi praktykami, aby zapewnić optymalną wydajność i niezawodność.
- **Integrację z Docker:** gUnicorn jest uruchamiany w kontenerach Docker, co ułatwia zarządzanie środowiskami i skalowanie aplikacji.

Realizacja tych wymagań niefunkcjonalnych zapewnia, że aplikacja działa sprawnie, jest skalowalna i łatwa do utrzymania w dłuższej perspektywie czasu.

8 Struktura projektu

Aplikacja składa się z kilku mniejszych aplikacji:

- admin – odpowiedzialna za stronę główną aplikacji.
- inventory – pozwala na zarządzanie urządzeniami, pokojami, budynkami, wydziałami, konfiguracjami.
- users – pozwala na zarządzanie użytkownikami.
- utils – zawiera klasy, które są wykorzystywane w powyższych aplikacjach.