Opis systemu MANAGE-TASKS-APP

Olha i Yakymenko

Spis treści

1 Wprowadzenie

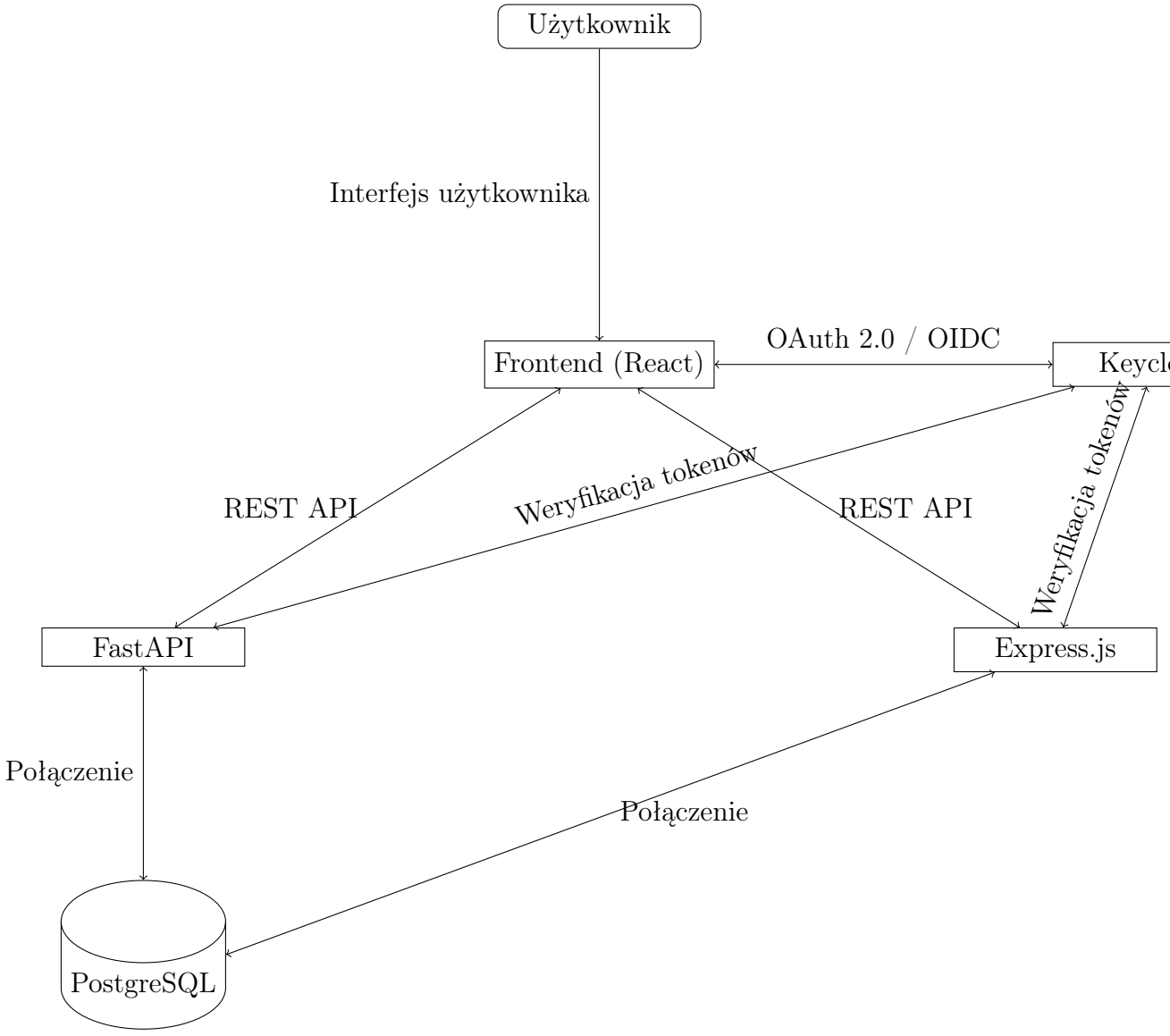
Dokumentacja opisuje system MANAGE-TASKS-APP - aplikację do zarządzania zadaniami z kontrolą dostępu opartą na rolach, wykorzystującą Keycloak do uwierzytelniania i autoryzacji.

2 Architektura systemu

2.1 Komponenty

- Frontend: Aplikacja React.js
- Backend:
  - FastAPI (Python) - główna logika aplikacji
  - Express.js - obsługa próśb o uprawnienia admina
- Dostawca tożsamości: Keycloak
- Baza danych: PostgreSQL

2.2 Diagram architektury



3 Logowanie z frontendu (public client)

1. Użytkownik chce się zalogować w aplikacji frontendowej (public client).
2. Frontend generuje żądanie autoryzacji do serwera Keycloak:
   - `client_id` – identyfikator klienta frontendu,
   - `response_type=code` – chcemy uzyskać kod autoryzacyjny,
   - `redirect_uri` – adres, na który Keycloak ma przekierować po logowaniu,

- `scope=openid` – wymagane do uzyskania ID Token (OpenID Connect),
- `code_challenge` – zakodowany ciąg (PKCE, np. SHA256 z losowego ciągu),
- `code_challenge_method=S256` – metoda kodowania (SHA256).

3. Użytkownik zostaje przekierowany do formularza logowania Keycloak i podaje swoje dane uwierzytelniające (login/hasło).
4. Po poprawnym uwierzytelnieniu, Keycloak przekierowuje użytkownika z powrotem na `redirect_uri`, przekazując w URL parametr `code` – kod autoryzacyjny.
5. Frontend odbiera ten `authorization code`.
6. Frontend wykonuje zapytanie POST do Keycloak (`/token`), przekazując:
   - `client_id`,
   - `code` – uzyskany kod autoryzacyjny,
   - `redirect_uri` – ten sam co wcześniej,
   - `code_verifier` – pierwotny ciąg, z którego wygenerowano `code_challenge`.
7. Keycloak weryfikuje `code_verifier` i jeśli wszystko się zgadza, zwraca:
   - `access_token` – JWT, używany do autoryzacji (np. przy wywołaniach backendu),
   - `id_token` – JWT zawierający dane o użytkowniku (np. login, email),
   - (opcjonalnie) `refresh_token` – służący do odnowienia `access_token`.
8. Frontend przechowuje otrzymane tokeny (np. w pamięci przeglądarki lub bezpiecznym magazynie).

Weryfikacja tokena przez backend (Keycloak, introspekcja)

4 Weryfikacja tokenu przez backend (private client)

1. Backend otrzymuje zapytanie HTTP z nagłówkiem:

```
Authorization: Bearer <access_token>
```

2. Token jest przesyłany do Keycloak w celu walidacji (introspekcji).
3. Backend wysyła żądanie POST do Keycloak na endpoint:

   `/protocol/openid-connect/token/introspect`

4. W treści żądania przesyła:
   - `token` – access token otrzymany od klienta
   - `client_id` – identyfikator backendu jako klienta w Keycloak
   - `client_secret` – (jeśli backend to confidential client)
5. Keycloak zwraca odpowiedź JSON zawierającą:
   - `active` – wartość `true`, jeśli token jest ważny
   - dane o użytkowniku (np. `sub`, `username`, `scope`, `exp`, `aud`)
6. Backend na podstawie odpowiedzi podejmuje decyzję, czy zaakceptować żądanie.

5 Zalety i wady introspekcji

- **Zalety:**
  - Możliwość natychmiastowej weryfikacji ważności tokena (np. po wylogowaniu).
  - Brak konieczności lokalnej walidacji podpisu JWT.
- **Wady:**
  - Wymaga połączenia z serwerem Keycloak.
  - Może powodować opóźnienia i dodatkowe obciążenie sieciowe.

5.1 Role

- **admin** - Może tworzyć zadania, przeglądać wszystkie zadania, zarządzać prośbami o uprawnienia admina

6 Endpointy API

6.1 Endpointy FastAPI

Endpoint	Metoda	Opis
/api/task/assign	POST	Tworzenie nowego zadania (tylko admin)
/api/task	DELETE	Usuwanie zadania (tylko admin)
/api/admin/tasks	GET	Pobieranie wszystkich zadań ze statusami (tylko admin)
/api/task	PUT	Aktualizacja statusu zadania
/api/my-tasks	GET	Pobieranie zadań użytkownika
/api/summary	GET	Podsumowanie (inne dla admina i użytkownika)

6.2 Endpointy Express.js

Endpoint	Metoda	Opis
/api-ex/request-admin	POST	Prośba o uprawnienia admina
/api-ex/admin/requests	GET	Pobieranie próśb (tylko admin)
/api-ex/admin/approve	POST	Akceptacja prośby
/api-ex/admin/reject	POST	Odrzucenie prośby

7 Implementacja zabezpieczeń

7.1 Weryfikacja tokenów

System używa introspekcji tokenów Keycloak do weryfikacji:

```
def verify_token(authorization: Optional[str] = Header(None)) -> dict:
    token = authorization.replace("Bearer ", "")
    introspection_url = f"{keycloak_url}/realms/{realm}/protocol/openid

    response = requests.post(
        introspection_url,
        data={'token': token},
        auth=(client_id, client_secret)
    )

    if not response.json().get("active", False):
        raise HTTPException(status_code=401, detail="Nieprawid owy to
```

7.2 Sprawdzanie ról

```
// Przykład komponentu React
const { keycloak } = useKeycloak();
const isAdmin = keycloak.hasRealmRole('admin');
```

8 Przykłady użycia

8.1 Tworzenie zadania (Admin)

```
const handleSubmit = async () => {
  const response = await axios.post('/api/task/assign', {
    title: 'Nowe zadanie',
    date: '2025-06-30',
    employees: ['user1@example.com', 'user2@example.com']
  }, {
    headers: { Authorization: `Bearer ${keycloak.token}` }
  });
};
```

8.2 Aktualizacja statusu zadania (Użytkownik)

```
const handleComplete = async (taskId) => {
  await fetch('/api/task', {
    method: 'PUT',
    headers: {
      Authorization: `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ task_id: taskId, completed: true })
  });
};
```

9 Podsumowanie

System MANAGE-TASKS-APP zapewnia bezpieczne zarządzanie zadaniami z właściwą kontrolą dostępu opartą na rolach. System demonstruje implementację OAuth 2.0 i OpenID Connect z Keycloak, chroniąc zarówno komponenty frontendu jak i backendu.