OptiSlots

Opti
Slots

System Optymalizacji Ewolucyjnej Harmonogramów

Problem?

Problem?

obecna rekrutacja nie jest idealna

- brak możliwości ułożenia spójnego planu
- “kto pierwszy ten lepszy” (niesprawiedliwe)
- sztywne godziny rekrutacji
- dużo pracy sekretariatu
- duże obciążenie serwerów
- nikt nie jest zadowolony (plany często nie odpowiadają właścicielom)

Nasze rozwiązanie

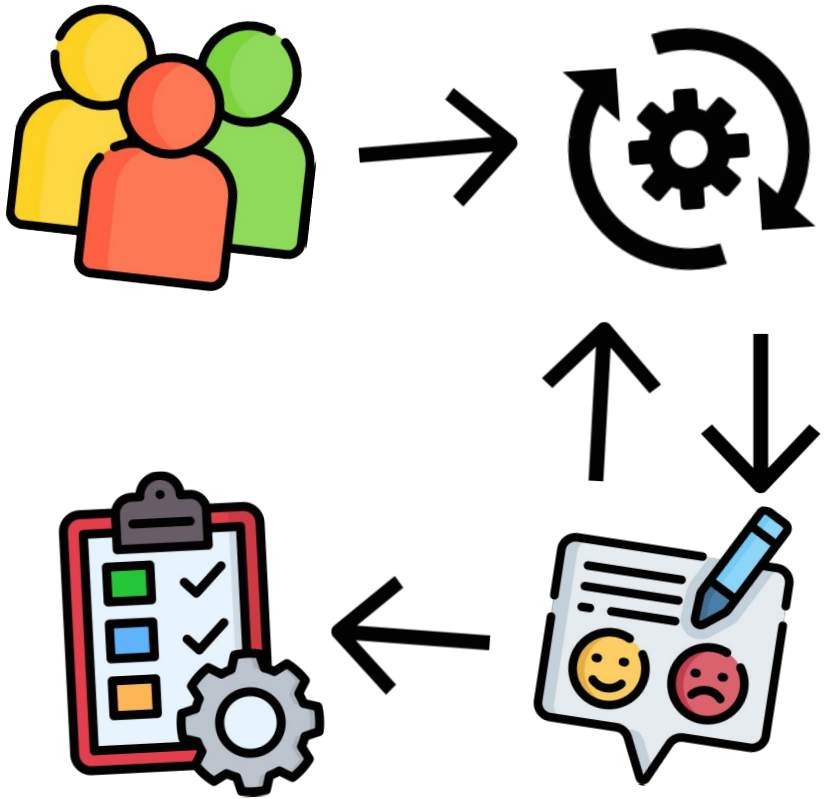
Nasze rozwiązanie:

Jak działa OptiSlots?

1. Zbieranie preferencji
2. Optymalizacja harmonogramu
3. Feedback fine-tuning
4. Gotowy plan

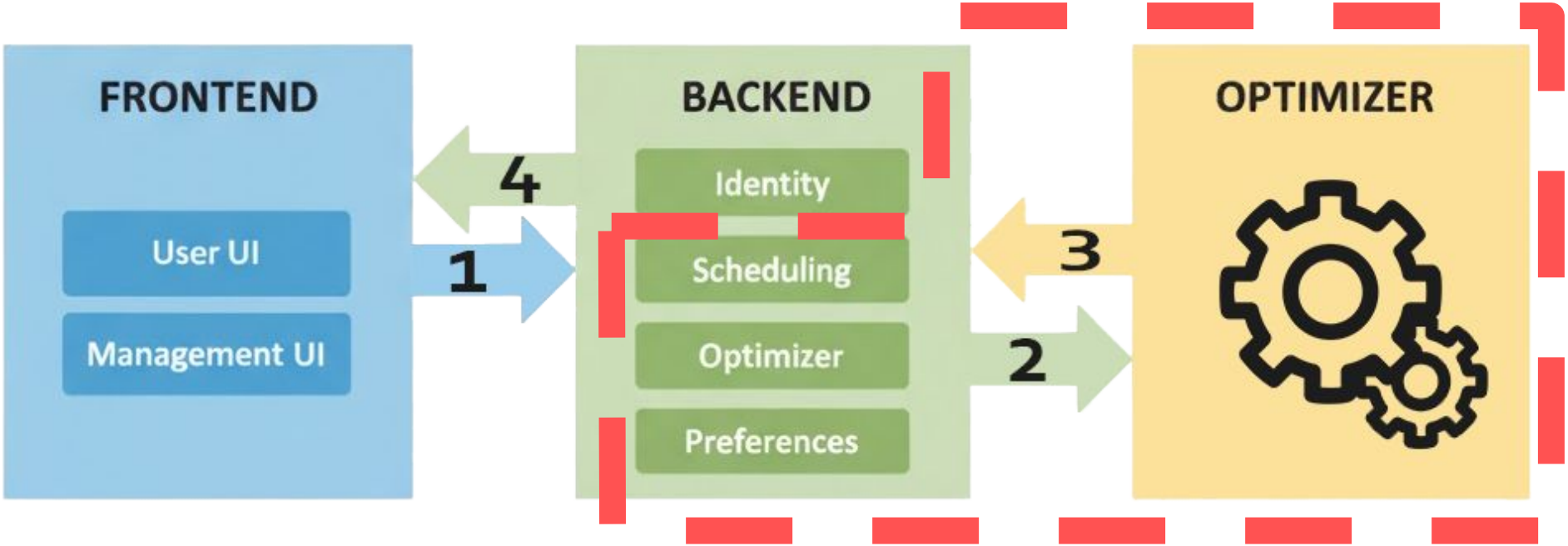
Zapewnia to:

1. Personalizację priorytetów
2. Ciągłą rekrutację
3. Dostęp do informacji zwrotnej
4. Optymalizację indywidualnych preferencji

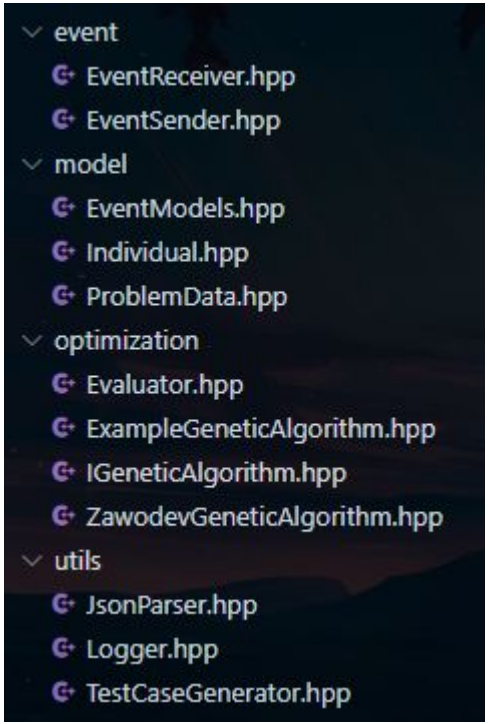


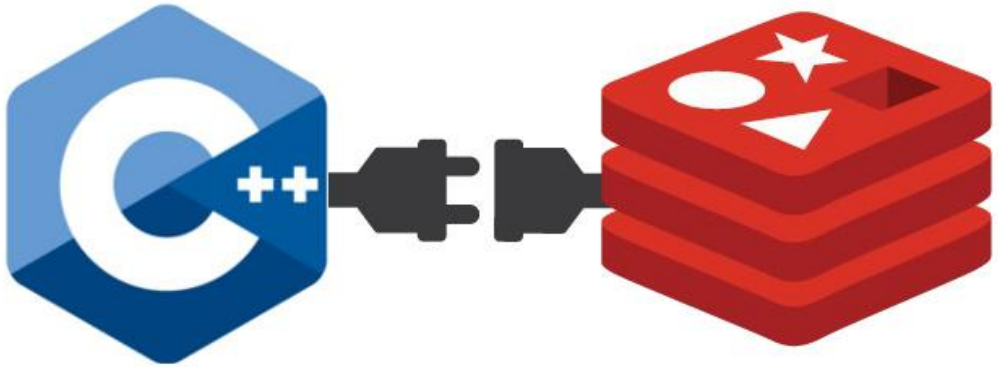
Aleksander Stepaniuk

Zakres obowiązków

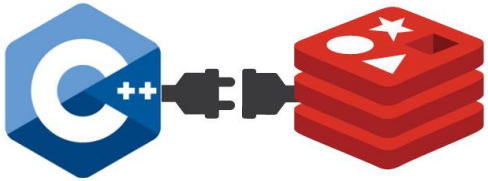


Optymalizator C++, Broker Redis





Dlaczego C++ oraz Redis?



Redis:

- **Szybkość (In-Memory):** Redis to błyskawiczny, przechowywany w pamięci broker wiadomości (kolejka zadań).
- **Komunikacja asynchroniczna:** Umożliwia oddzielenie "lekkiego" backendu (Django/API) od "ciężkich" i czasochłonnych obliczeń optymalizatora (C++).
- **Skalowalność i decoupling:** Backend (Django) jedynie zleca zadanie optymalizacji do kolejki. Pozwala to na potencjalne przeniesienie modułu C++ na osobną (mocniejszą) maszynę oraz łatwe uruchomienie wielu równoległych optymalizatorów (workerów), które pobierają zadania z Redis, bez obciążania serwera API.

C++:

- **Wydajność:** C++ jest kompilowany bezpośrednio do natywnego kodu maszynowego, a nie interpretowany (jak Python). Zwiększona szybkość obliczeń jest kluczowa dla algorytmu genetycznego, który musi wykonać miliony operacji w krótkim czasie.
- **Kontrola nad pamięcią:** C++ daje pełną, manualną kontrolę nad alokacją i zwalnianiem pamięci, co zapewnia stabilne zużycie zasobów i zwiększoną szybkość działania.
- **Ekosystem:** Istnieje wiele stabilnych i wysoce wydajnych bibliotek C++ do prostej w obsłudze, bezpośredniej i niskopoziomowej komunikacji z Redis.

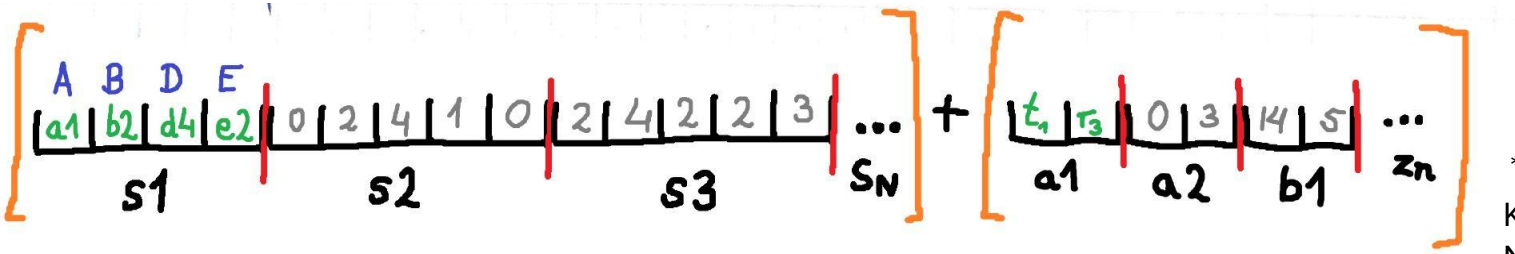
Problem złożoności kombinatorycznej zadania

Liczba wszystkich możliwych harmonogramów jest **astronomiczna** (K! / (K-N)! ~ 10^2500)* - jak znaleźć “wystarczająco dobre” rozwiązanie, bez sprawdzania wszystkich możliwości?

Stosować **metaheurystykę** (algorytm genetyczny), która w inteligentny sposób przeszukuje jedynie obiecujące obszary gigantycznej przestrzeni potencjalnych rozwiązań.

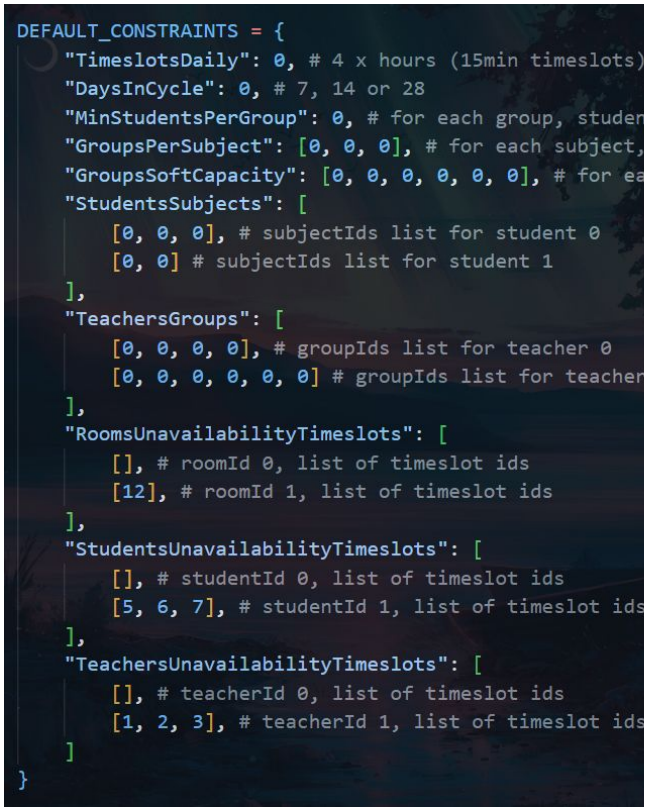
Odpowiednia reprezentacja danych w genotypie:

- **Algorytm genetyczny** musi mieć możliwość płynnie i efektywnie przechodzić po sąsiadujących rozwiązaniach w przestrzeni rozwiązań
- **Operacje genetyczne** takie jak **mutacja** czy **krzyżowanie** muszą działać w sensowny sposób



*gdzie:
K = (sale × timeslot)
N = len(groups)

Spełnienie ograniczeń, ewaluacja rozwiązania



```
DEFAULT_CONSTRAINTS = {
    "TimeslotsDaily": 0, # 4 x hours (15min timeslots)
    "DaysInCycle": 0, # 7, 14 or 28
    "MinStudentsPerGroup": 0, # for each group, studen
    "GroupsPerSubject": [0, 0, 0], # for each subject,
    "GroupsSoftCapacity": [0, 0, 0, 0, 0, 0], # for ea
    "StudentsSubjects": [
        [0, 0, 0], # subjectIds list for student 0
        [0, 0] # subjectIds list for student 1
    ],
    "TeachersGroups": [
        [0, 0, 0, 0], # groupIds list for teacher 0
        [0, 0, 0, 0, 0, 0] # groupIds list for teacher
    ],
    "RoomsUnavailabilityTimeslots": [
        [], # roomId 0, list of timeslot ids
        [12], # roomId 1, list of timeslot ids
    ],
    "StudentsUnavailabilityTimeslots": [
        [], # studentId 0, list of timeslot ids
        [5, 6, 7], # studentId 1, list of timeslot ids
    ],
    "TeachersUnavailabilityTimeslots": [
        [], # teacherId 0, list of timeslot ids
        [1, 2, 3], # teacherId 1, list of timeslot ids
    ]
}
```

Jak zapewnić, że **algorytm** nigdy nie złamie “**twardych**” reguł (np. dwóch grup w tej samej sali w tym samym oknie czasowym), jednocześnie przy tym starając się spełnić “**miękkie**” reguły (preferencje)?

Rozwiązanie:

Implementacja **funkcji celu** (**fitness function**), która nakłada gigantyczne kary za łamanie twardych **ograniczeń**, a nagradza za spełnianie **preferencji**.

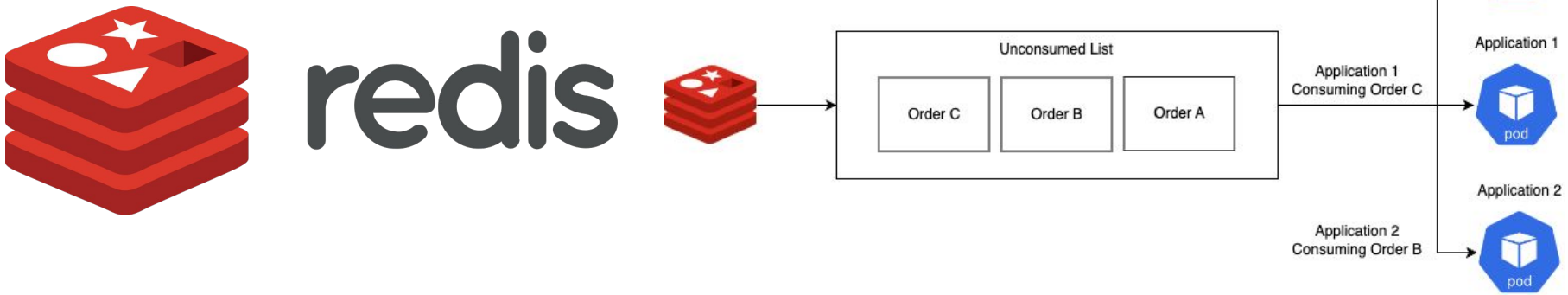
Wynikiem jest **średnia ważona** wszystkich uczestników, na podstawie priorytetu danej osoby (np. średnia ocen) z uwzględnieniem **mediany**, aby nikogo nie skrzywdzić.

Broker wiadomości (Redis)

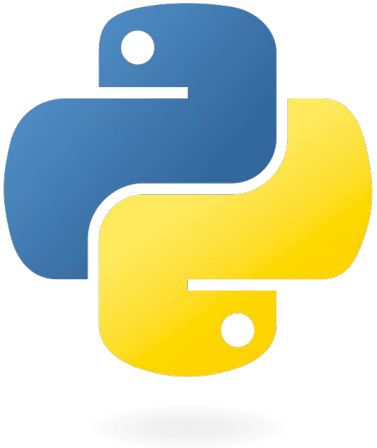
Redis nie wykonuje obliczeń, ale rozwiązuje problemy **komunikacyjne** i **architektoniczne**.

Bez asynchronicznej komunikacji:

- Optymalizacja jest "ciężka" i trwa długo (ponad 15 minut). API musi odpowiedzieć natychmiast.
- Co jeśli kilka podmiotów naraz rozpocznie optymalizację? Jeden worker zablokuje się na wiele godzin.
- Skąd Django ma wiedzieć, czy zadanie C++ się udało, czy zakończyło błędem?

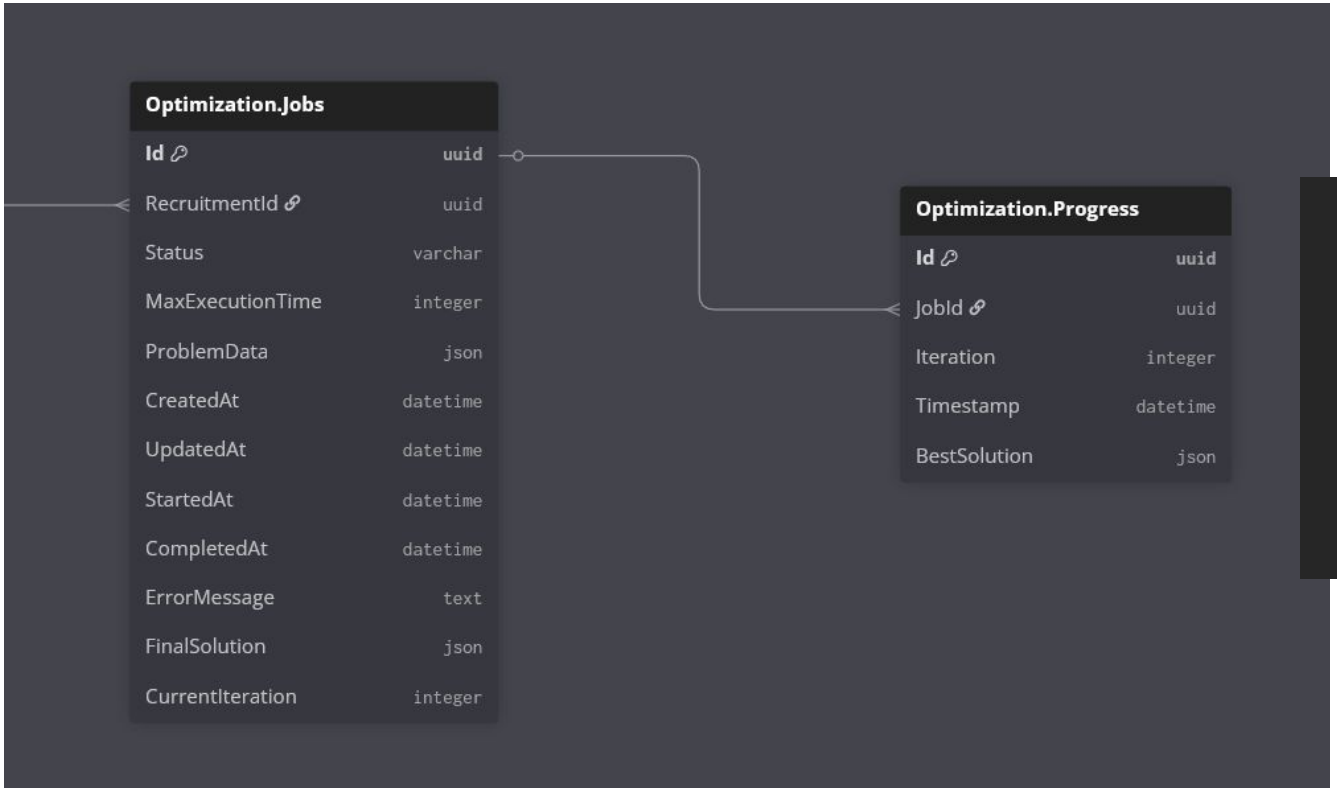


Backend Django



django

Komunikacja Backendu z Optimizer



GET health check

POST create job

GET job status

GET job list

POST job cancel

Konwersja typów do przepływu flow

BACKEND
- Identity
- Scheduling
- Optimizer
- Preferences

json (Preferences) → json (ProblemData)

OPTIMIZER

OPTIMIZER

vector<int> (genotype) → sql (Meetings)

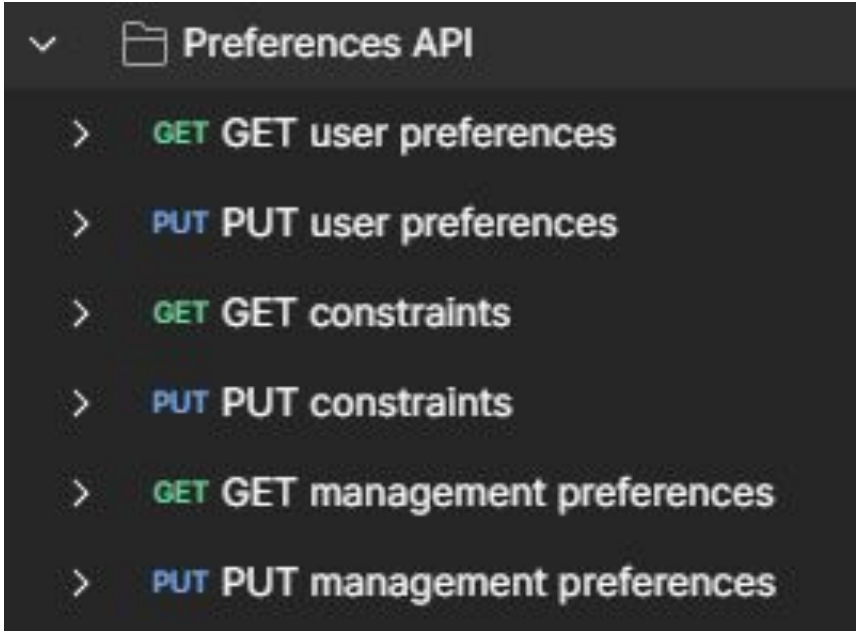
BACKEND
- Identity
- Scheduling
- Optimizer
- Preferences

Preferencje użytkowników

- Każdy PUT zużywa token
- Każdy użytkownik ma limit np. 3 tokenów

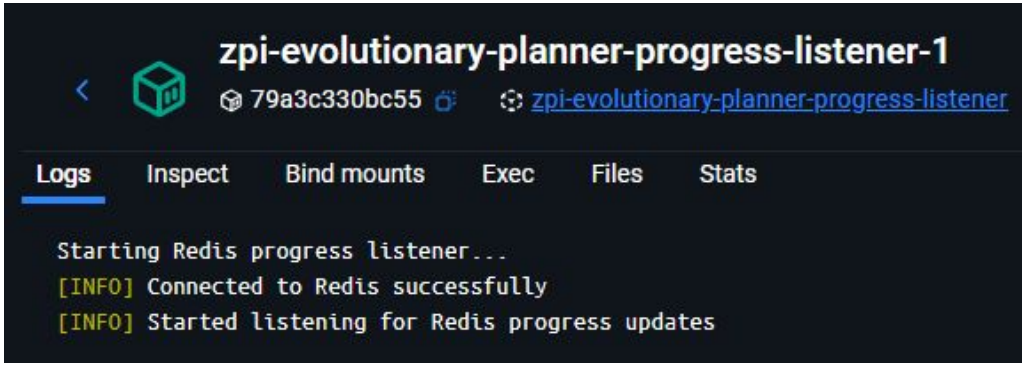
Pozostałe

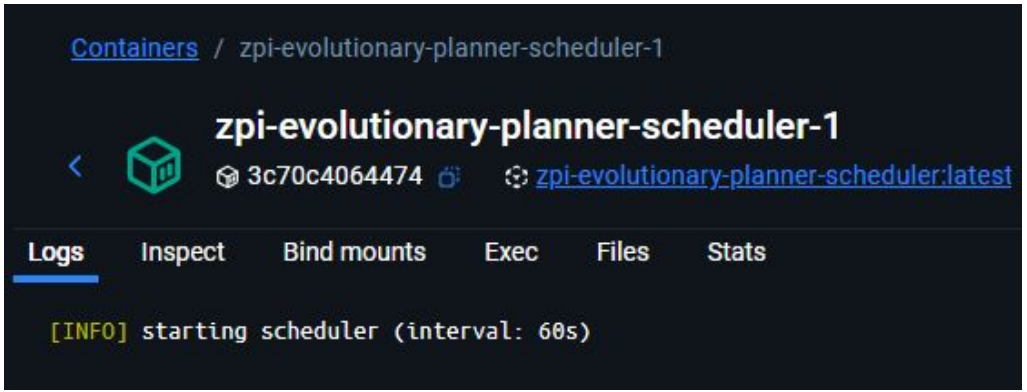
- 1:1 jedna tabela dla jednej rekrutacji
- Dostępne tylko dla managementu




```
DEFAULT_USER_PREFERENCES = {
    "WidthHeightInfo": 0, # weight, positive means prefer wider, negative means prefer taller
    "GapsInfo": [0, 0, 0], # minGaps, maxGaps, weight
    "PreferredTimeslots": [0, 0, 0, 0, 0, 0, 0], # for each timeslot in cycle, weight
    "PreferredGroups": [0, 0, 0, 0, 0] # for each group, weight
}
```

Background Tasks

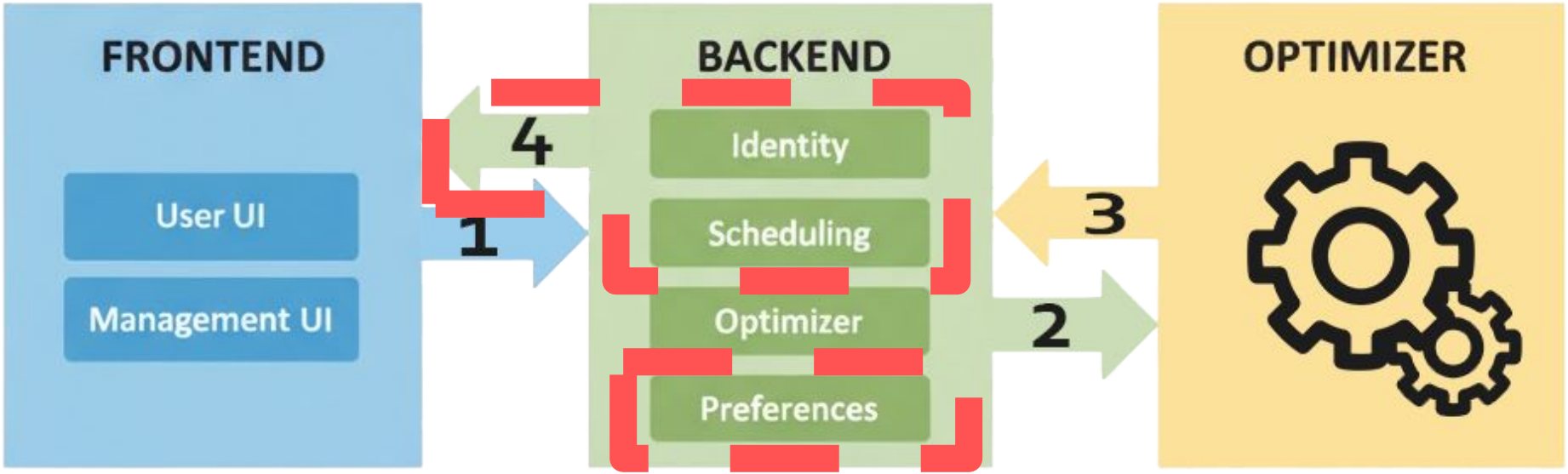




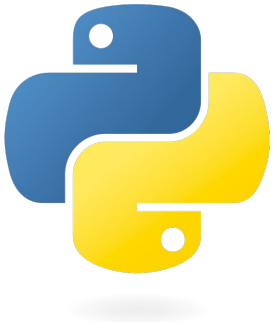
1. **Progress Listener**
   - Obserwuje redis
   - Reaguje na eventy
   - Propaguje flow dalej
2. **Scheduler**
   - Sprawdza datę
   - Rozpoczyna optymalizację
   - Archiwizuje starsze plany

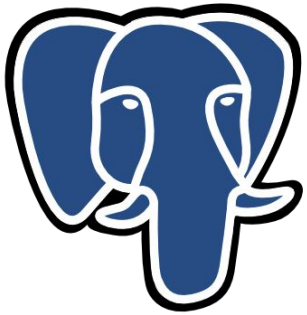
Kacper Zakrzewski

Zakres obowiązków

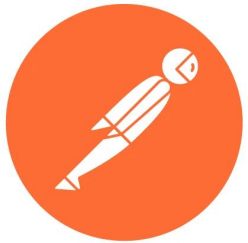


Backend aplikacji





PostgreSQL

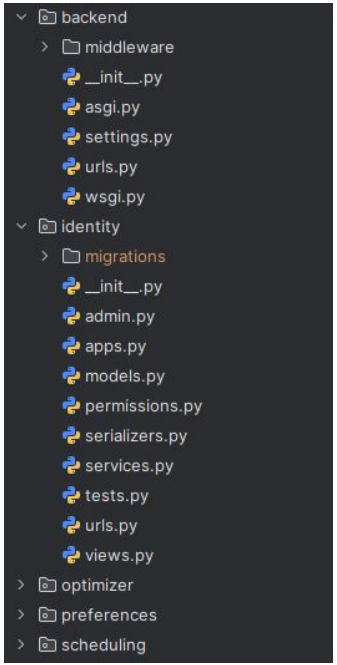


POSTMAN

Dlaczego Django + Django REST Framework?

- Kompletny framework typu “batteries included”: gotowe moduły do uwierzytelniania, panel admina, ORM
- Szybki rozwój: mniej kodu przy zachowaniu wysokiej jakości
- Stabilność i bezpieczeństwo: regularne aktualizacje, sprawdzone w produkcji
- Django REST Framework umożliwia łatwe tworzenie API dla frontendu i aplikacji mobilnych
- Duża społeczność i dokumentacja ułatwiająca rozwiązywanie problemów
- Wysoka skalowalność: od projektów akademickich po duże serwisy globalne

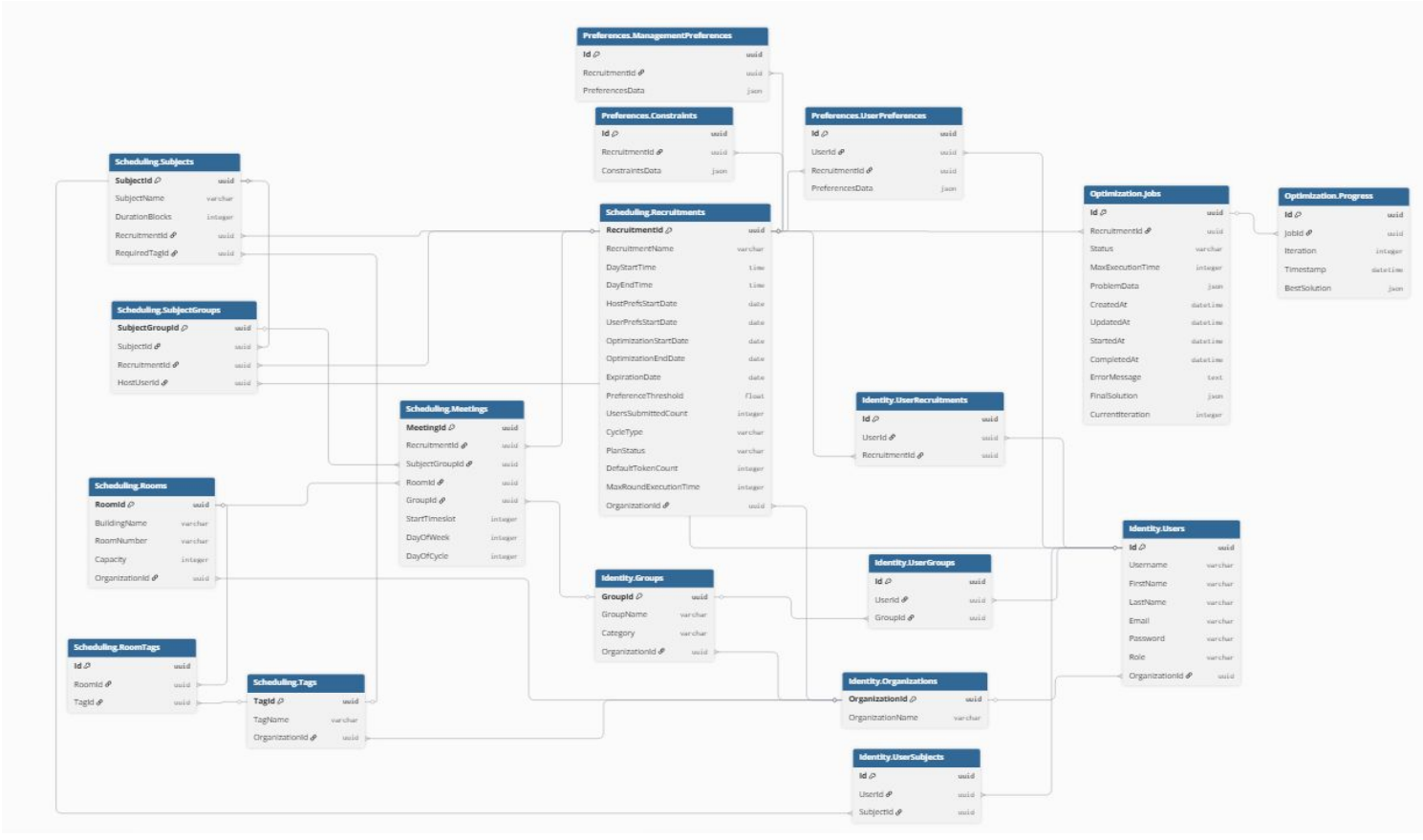
Struktura backendu aplikacji Django



Aplikacja backendowa jest podzielona na 4 główne moduły (aplikacje):

- Identity: Odpowiedzialna za podstawowe modele użytkownika, grup, organizacji.
- Optimizer: Odpowiedzialna za zarządzanie informacjami dotyczącymi jednostek optymalizacji.
- Preferences: Odpowiedzialna za zarządzanie preferencjami użytkowników.
- Scheduling: Odpowiedzialna za zarządzanie informacjami dotyczącymi spotkań, sal, rekrutacji, przedmiotów itd.

Struktura bazy danych

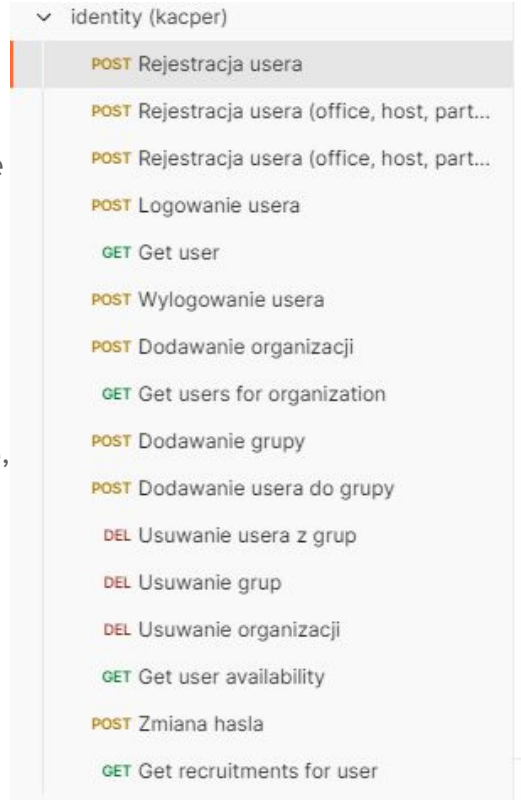


Struktura bazy danych

- Podział na logiczne domeny: planowanie zajęć, preferencje, optymalizacja i zarządzanie użytkownikami
- Dane dotyczące zajęć powiązane z pomieszczeniami, prowadzącymi i rekrutacjami (edycjami planowania)
- Możliwość definiowania preferencji i ograniczeń przez użytkowników oraz system
- Przechowywanie parametrów i wyników procesu optymalizacji (pełna historyczność iteracji)
- Obsługa wielu organizacji (np. wydziały, uczelnie) dzięki multi-tenant strukturze
- Użycie UUID dla skalowalności i łatwej integracji z API
- Elastyczne dane konfiguracyjne w formacie JSON umożliwiają szybkie zmiany logiki biznesowej

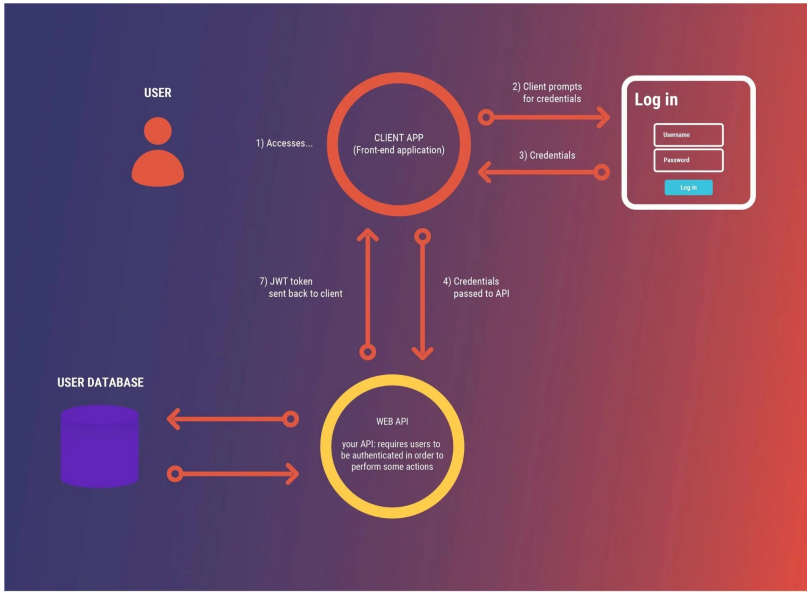
Identity

- Odpowiada za bezpieczne tworzenie i uwierzytelnianie użytkowników, zarządzanie rolami i uprawnieniami oraz dostarczanie tokenów oraz mechanizmów kontroli dostępu wykorzystywanych przez cały backend.
- Modele danych: User, Organization, Group, UserGroup — relacje między użytkownikiem a organizacją i grupami, pola profilu i uprawnień.
- Kontrola dostępu: specjalne klasy permisji stosowane w widokach.
- Bezpieczeństwo: hashowanie haseł, walidacja siły hasła, ochrona przed CSRF/XSS, ograniczenia na endpointy administracyjne.



Bezpieczeństwo i kontrola dostępu

- Autoryzacja oparta o JWT zapewnia bezstanową obsługę sesji i łatwą integrację z frontendem
- Szyfrowane tokeny przekazywane w nagłówkach chronią przed ujawnieniem danych wrażliwych
- Każdy endpoint zabezpieczony dedykowanymi uprawnieniami opartymi o role użytkowników
- Separacja danych między organizacjami dzięki mechanizmom ról i powiązań z jednostką organizacyjną
- Automatyczna weryfikacja tokena i uprawnień po stronie backendu ogranicza ryzyko nieautoryzowanego dostępu
- Zgodność z dobrymi praktykami bezpieczeństwa REST API w Django i DRF

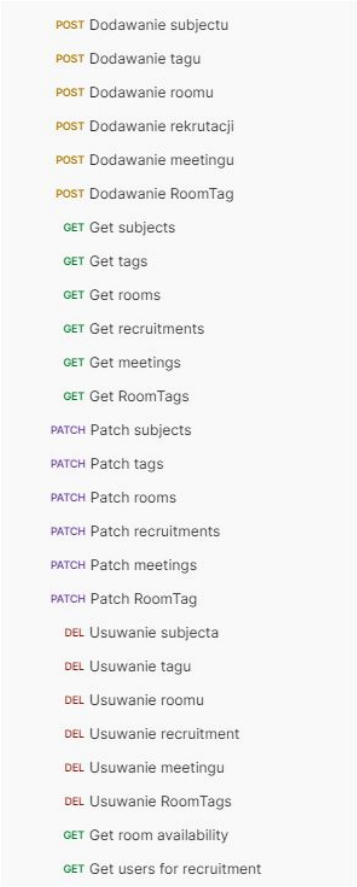


Scheduling

- Odpowiada za zarządzanie przedmiotami, pokojami, rekrutacjami, spotkaniami itd.
- Podstawowe operacje CRUD dla każdego modelu.
- Query do pozyskania potrzebnych informacji w zależności od poszczególnych relacji (bez używania sql).

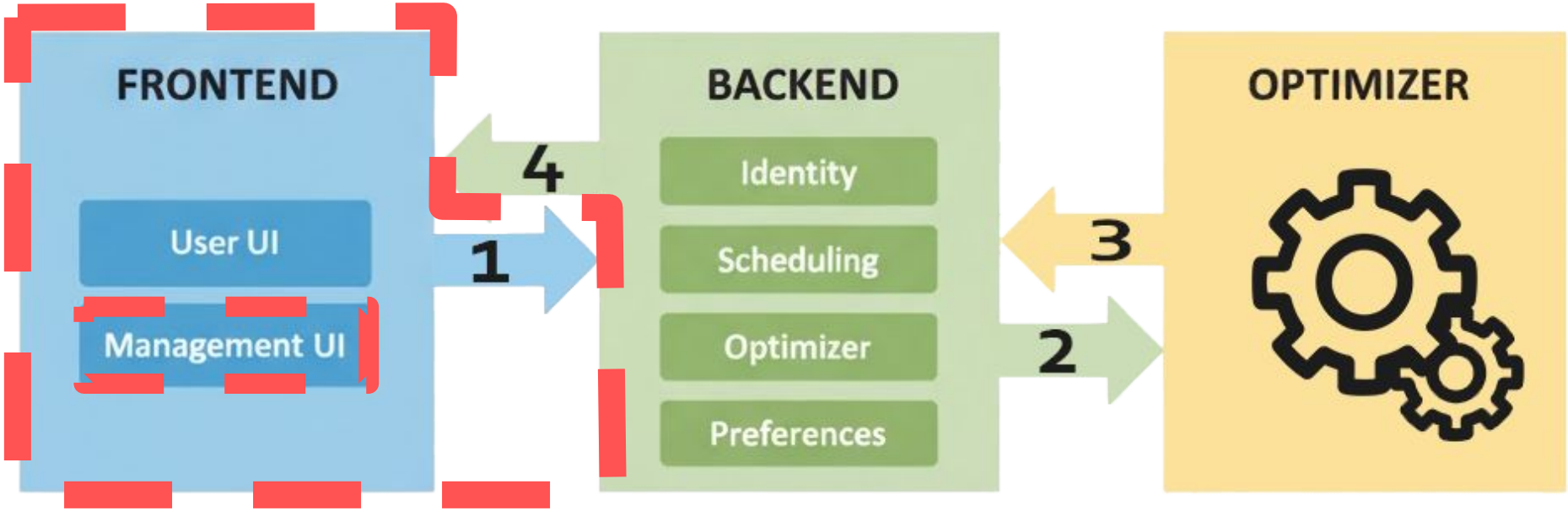
```python
def get_active_meetings_for_room(room_or_id: Union[Room, str, int]) -> QuerySet:
    """
    Returns a QuerySet of all Meeting objects related to the given Room (instance or PK),
    where recruitment.plan_status == 'active'.
    """
    room_id = room_or_id.pk if hasattr(room_or_id, 'pk') else room_or_id

    qs = (
        Meeting.objects
        .filter(room_id=room_id, recruitment__plan_status='active')
        .select_related('recruitment', 'subject_group__subject', 'subject_group__group', 'subject_group__host_user', 'room', 'required_tag')
        .order_by('day_of_week', 'start_hour')
    )
    return qs
```



Jakub Borsuk

Zakres Obowiązków



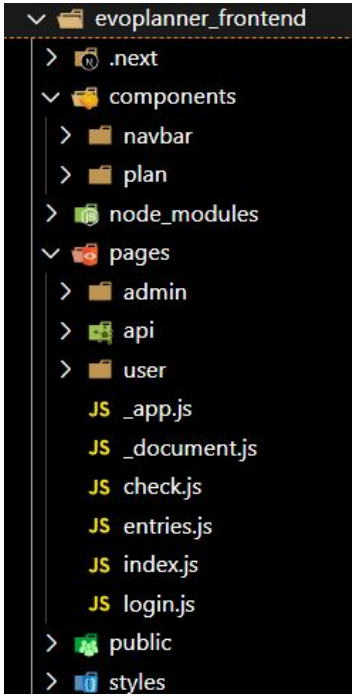
Front-end: wybór frameworku

Dlaczego akurat Next.js?

- **Wsparcie:** React jest wspierany przez Facebook. Dostępne są materiały i rozwiązania na dowolne problemy.
- **Virtual DOM** i **Wysoka reaktywność**
- **Wydajność**

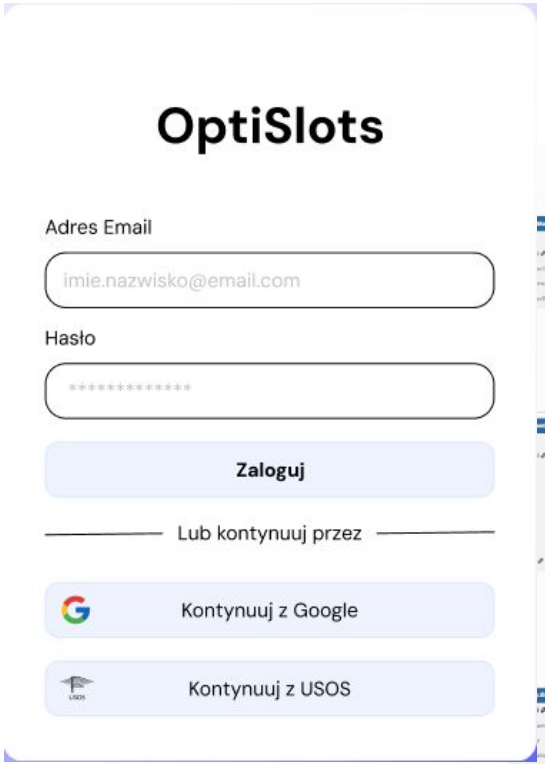
Inne rozważane opcje:

Struktura frontendu



- **Components:** Zawiera wszystkie ‘klasy’ obiektów które mogą być ponownie używane między ekranami
- **Admin:** Zbiór stron dla administracji firmy (zarządzanie harmonogramami).
- **User:** Zbiór stron dla użytkowników
- Reszta stron jest ogólnie dostępna niezależnie od roli (strona główna, kontakt, logowanie itp.)
- **Public:** Publiczne obrazy (np. logo)
- **Styles:** kolekcja modularnych plików css używanych przez strony.

Zapewnienie bezpieczeństwa



- **XSS (Cross-Site Scripting):** React robi to za nas.
- **Przetrzymywanie wrażliwych danych:** Między innymi tokeny
- **Odświeżanie sesji** i **wylogowywanie**
- **Walidacja użytkowników:** Jeden adres obsługuje administracje i klientów brących udział w spotkaniach.

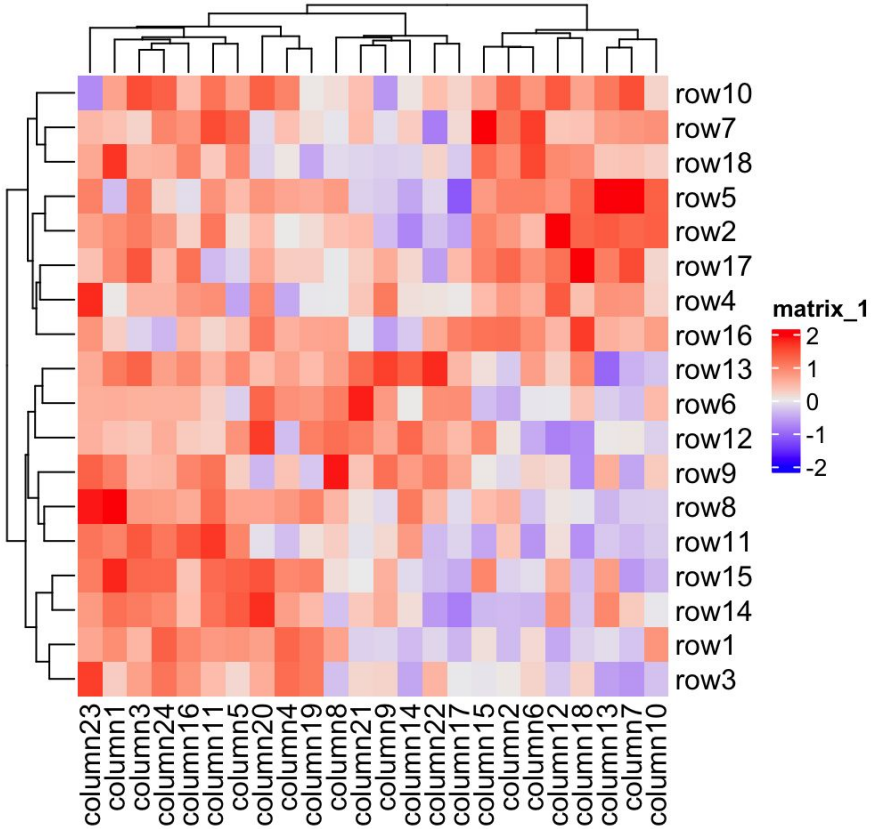
Współpraca użytkowników

Ważnym problemem w administracyjnej części systemu jest obsługiwanie zmian tych samych elementów.

Administracja może edytować dane użytkowników. Problem istnieje przy edycji danych usuniętych przez innego administratora firmy.

Ten sam problem może istnieć przy edytowaniu planów. Ważne jest zapewnienie żeby pokoje otrzymywały prawidłowe dane przy zmianach

Elementy ciągłe vs projekt aplikacji



- Heatmapa
- Preferencje użytkowników/ aktualnie generowany plan z perspektywy administracji
- Opcje:

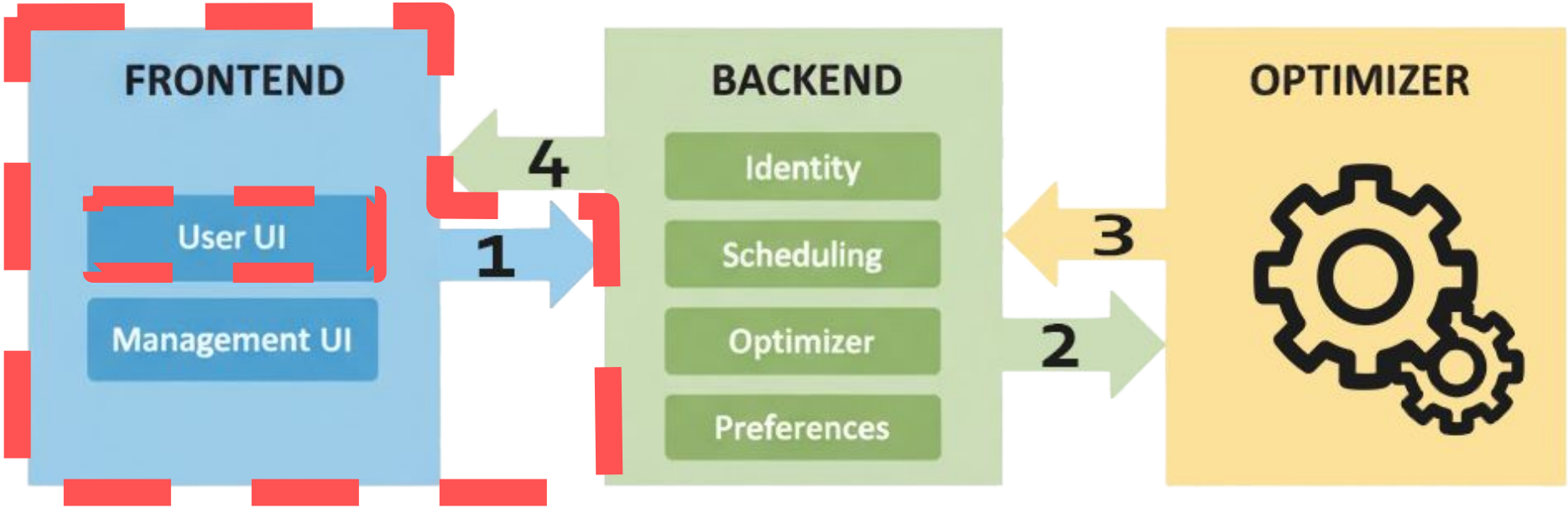
  Websocket

  SSE

  Pobieranie z API co x czasu

Piotr Bonar

Zakres Obowiązków



UI - Wybieranie Priorytetów - Co trzeba zawrzeć?

- Terminy preferencji czasowych - dzień tygodnia, godzina, “chcę/nie chcę”

  np. Bardzo nie chcę mieć zajęć od 7:30-13:45 w środę

- Odpowiedzi na pytania dodatkowe

  np. Czy wolisz mieć zajęcia rano, czy wieczorem? Rano!

- Hierarchia wartości preferencji

  Np. Zdecydowanie bardziej mi zależy na preferencji nr. 1

UI - Wybieranie Priorytetów - Co jest problematyczne?

- Duża ilość informacji do pokazania
- Duża ilość preferencji do wyboru
- Trudność w zarządzaniu złożonym zgłoszeniem
- Problem z ustalaniem terminu
- Kontrolowanie aktualnych preferencji
- Użytkownik potencjalnie może chcieć tworzyć wersje tymczasowe

UI - Wybieranie Priorytetów - Jak rozwiązać?

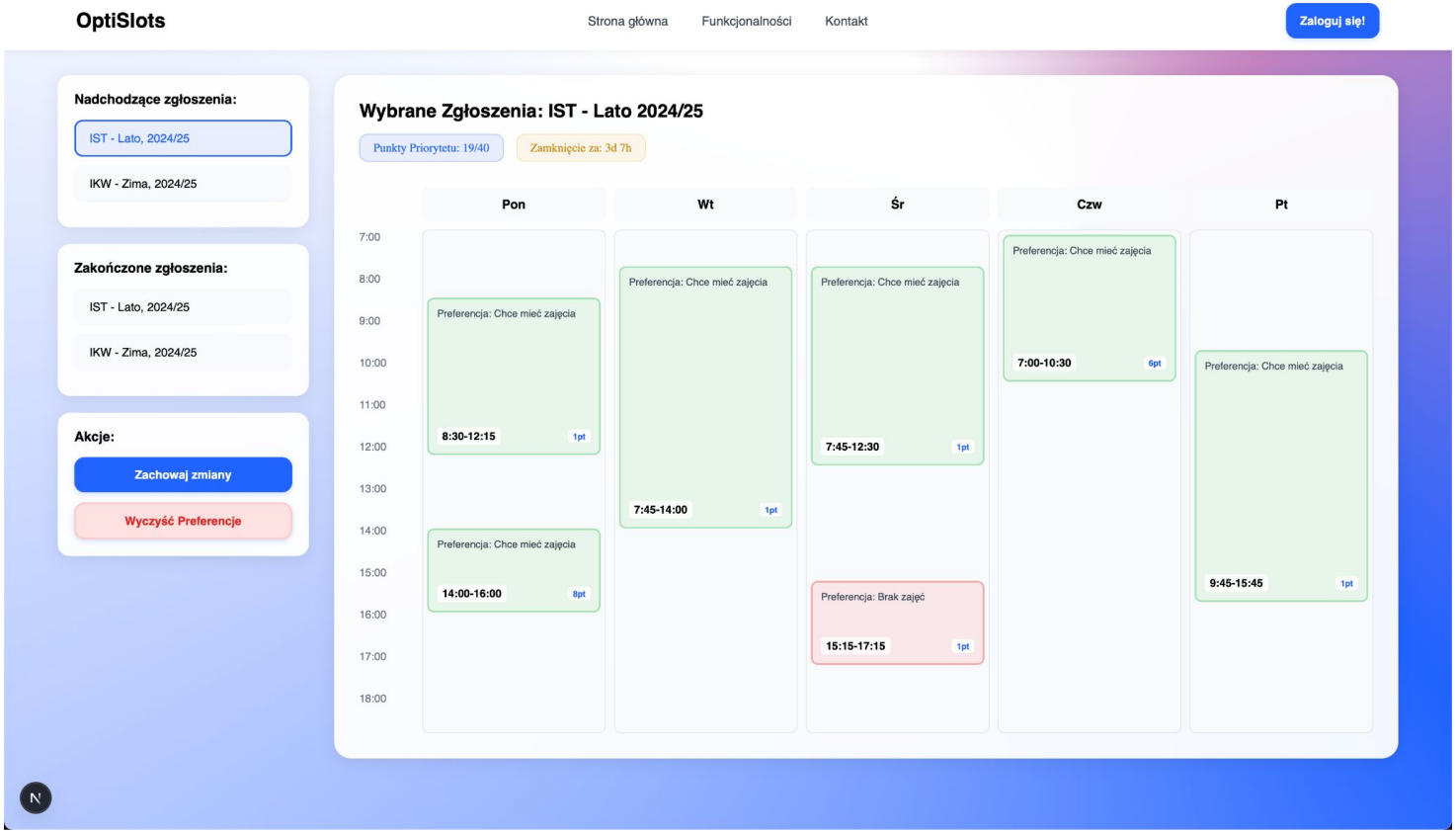
Rozwiązanie preferencji tygodniowych:

- Stworzenie kalendarza tygodniowego
- Stworzenie osobnych pop-up’ów do kontrolowania każdego okna czasowego
- Optymalne zaplanowanie wyglądu okienek, żeby zawrzeć te wszystkie informacje

Rozwiązanie preferencji ogólnych:

- Każdy z elementów musi mieć osobny licznik punktów priorytetu
- Osobne podsumowanie rozłożenia punktów priorytetu

UI - System Priorytetów - Tymczasowe Rozwiązanie



UI/Frontend - Plan Spotkań - Czemu jest potrzebny?

Po tym, jak użytkownicy:

1. określą swoje preferencje (np. dni, godziny)
2. optymalizator wygeneruje najbardziej dopasowany plan,

System musi jasno zakomunikować wynik każdemu z nich.

UI/Frontend - Plan Spotkań - Co jest problematyczne?

UI:

- Zarządzanie, który z planów chcemy wyświetlić
- Przedstawieniu wielu spotkań w różnych godzinach
- Różne długości zajęć

Frontend:

- Przetworzenie danych z API
- Stworzenie komponentu, który będzie to niezawodnie wyświetlał

I zrobienie tego wszystkiego w sposób czytelny!

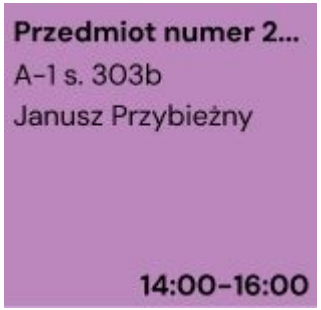
UI - Wybieranie Priorytetów - Czemu jest potrzebny?

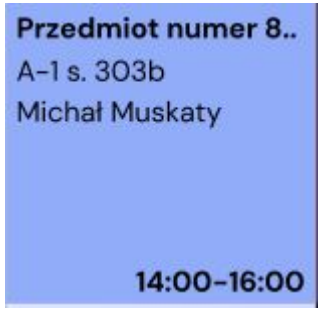
Aby użytkownikom plan się podobał, musi być zgodny z ich preferencjami.

Z tego właśnie powodu jest nam potrzebny system, który pozwoli im je sprecyzować

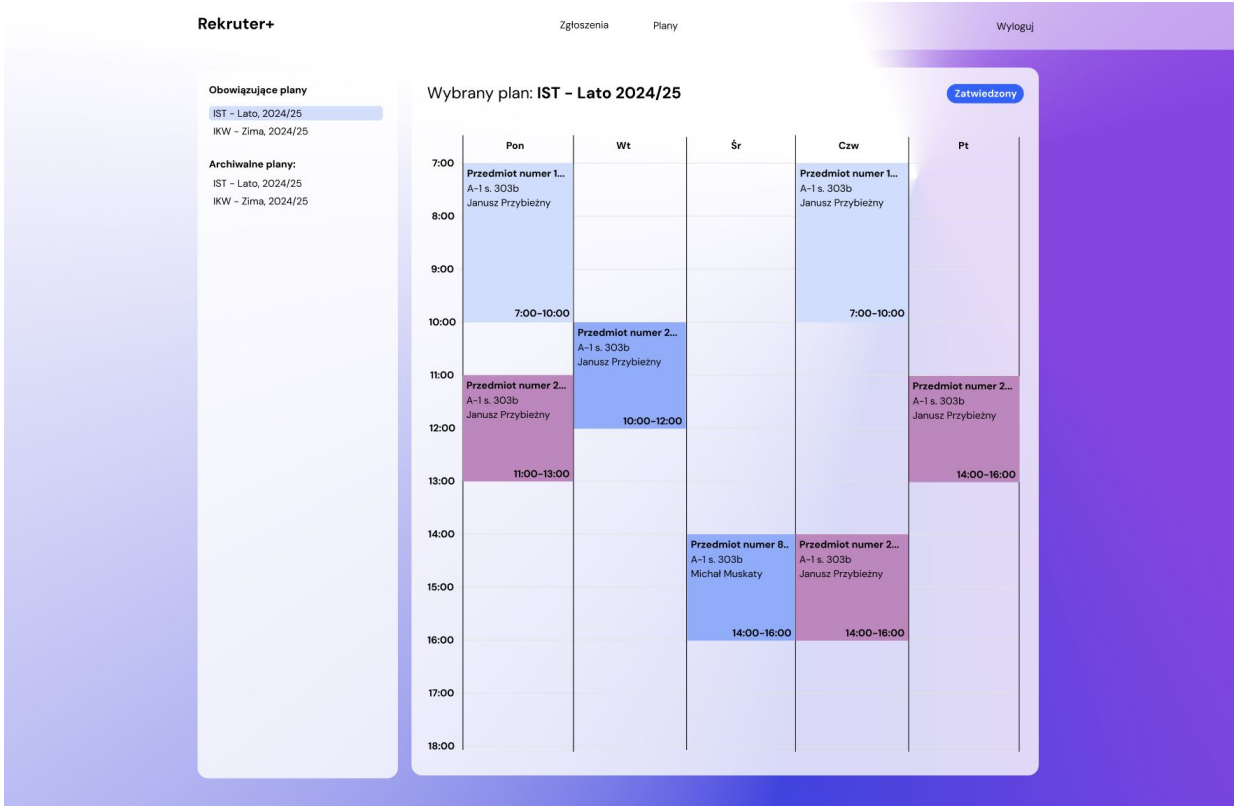
UI/Frontend - Plan Spotkań - Jak to rozwiązać?

- Podział na tygodnie
- Osobne miejsce do wybierania, który z planów chcemy wyświetlić
- Łączenie spotkań, które mają ze sobą coś wspólnego, w kolorystyczne grupy
- Optymalne zaprojektowanie “Kart Spotkań”





UI/Frontend - Plan Spotkań - Tymczasowe Rozwiązanie



Źródła

-https://www.appknox.com/blog/how-jwt-helps-in-securing-your-api

-https://nextjs.org

-https://vuejs.org

- https://www.djangoproject.com

–https://angular.dev

-https://redis.io

-https://www.postman.com

-https://www.postgresql.org.pl

-https://sqlite.org

Dziękujemy za uwagę

Aleksander Stepaniuk, Jakub Borsuk, Piotr Bonar, Kacper Zakrzewski