ChadGuide

Planowanie podróży po Europie

Paweł Kauf, Filip Rabiega

3 lutego 2026

Wstęp

Problem: Planowanie podróży wielomiastowej wymaga sprawdzenia wykładniczej liczby kombinacji kolejności miast. Ręczne porównywanie jest niepraktyczne.
Cel: Znalezienie optymalnych (najtańszych/najkrótszych) tras lotniczych łączących wiele miast europejskich.

Wstęp

ChadGuide jest *ChadRozwiązaniem*.

- Oprócz optymalizacji wykonuje jeszcze walidacje lotów
- Wszystko dzieje się przez wygodną oraz przejrzystą aplikację webową

Użyte technologie

- Python: `poetry`, `pandas`, `streamlit`, `pandera`, `fastapi`, `uvicorn`, `pydantic`, `sse-starlette`, `playwright`, `pytest`
- JavaScript: `vanilla js`, `tailwind css`, `leaflet.js`
- SQLite
- nginx
- API: Duffel, Gepapify (`httpx async`)

Demo

- Nasz projekt można obejrzeć na https://github.com/fiadra/chadguide.
- Działającą publiczną instancją jest https://www.chadguide.site.

Zbieranie danych o lotach

- Zbieranie danych o lotach do bazy jest realizowane przez paczkę `flight scanner`, która łączy się z DuffelAPI (148 lotnisk, 89k lotów)
- Dane w naszej bazie zawsze będą trochę do tyłu względem prawdziwych, ponieważ koszty biletów oraz ich dostępność ciągle się zmieniają. Dane dla tygodnia 01.07.2026–07.07.2026 są ekstrapolowane na cały miesiąc.

Zbieranie danych o atrakcjach

- Zbieranie danych o atrakcjach dzieje się asynchronicznie podczas działania algorytmu. Dane są zczytywane z bazy statycznej, gdzie mamy ich dane oraz zdjęcia.
- Zdjęcia atrakcji zostały zebrane metodami webscrapingu.

Algorytm

- Sercem projektu jest zmodyfikowany algorytm Dijkstry, który wyznacza drogi Pareto-optymalne względem ceny oraz czasu trwania lotu. Jednocześnie radzi sobie on z grafami, które są zmienne w czasie.
- Droga p *dominuje* drogę q, gdy istnieje kryterium, w którym p jest lepsze od q oraz nie istnieje kryterium, w którym p jest gorsze niż q.
- Droga jest *Pareto optymalna*, gdy nie jest dominowana przez żadną inną drogę.

Algorytm

- Interesują nas tylko te drogi, które są Pareto optymalne, oraz które wychodzą i wracają do wyznaczonego miasta, przelatując przy tym przez wszystkie miasta, które użytkownik chce odwiedzić.
- Inne drogi są automatycznie odrzucane.
- Przed uruchomieniem algorytmu wykonujemy prosty *pruning*, tzn. pozbywamy się tych połączeń w grafie, które prawie na pewno nie będą częścią optymalnego rozwiązania.

Algorytm

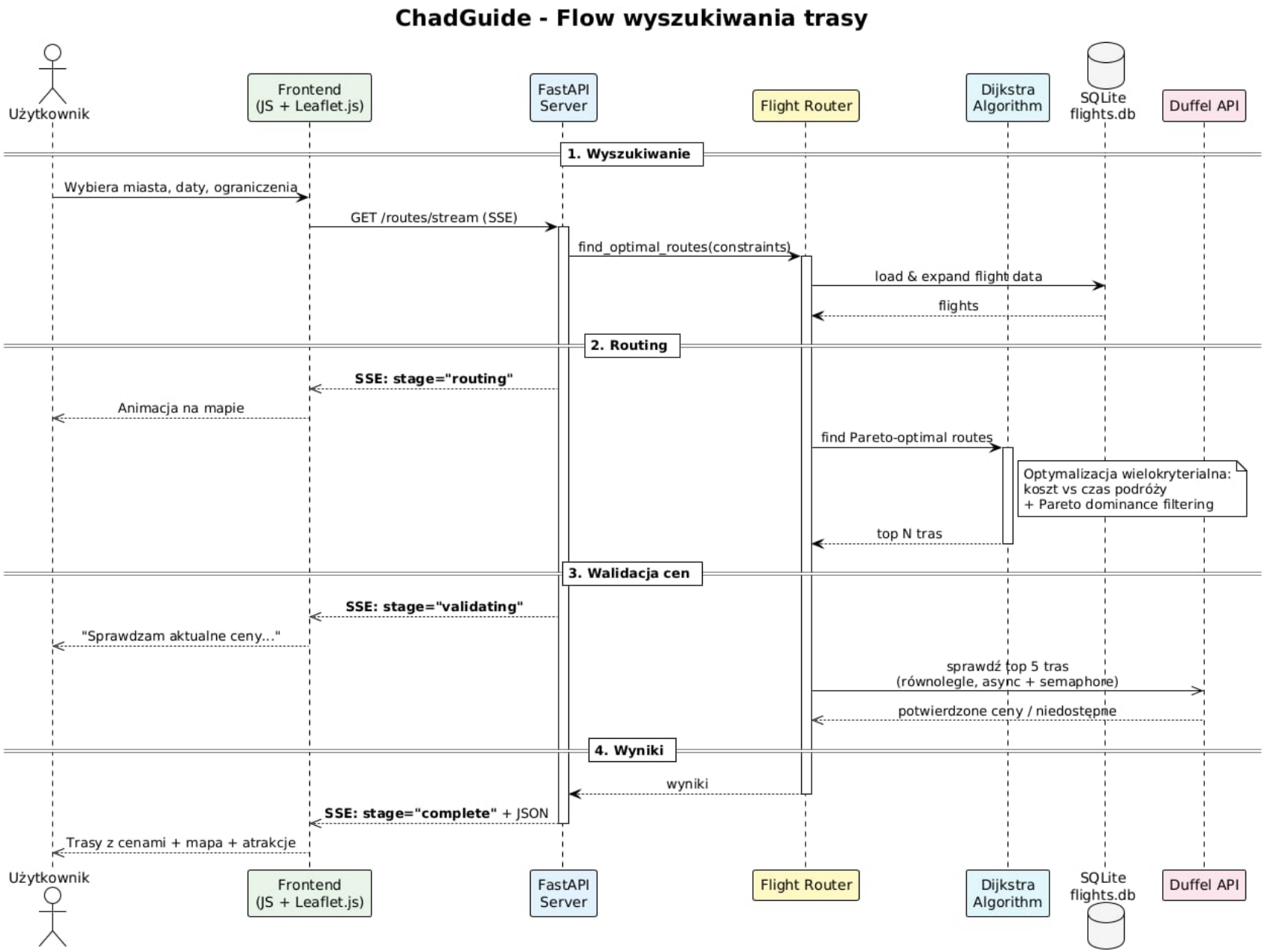
```
   Input: a timetable graph and a query
   Output: a set of Pareto-optimal labels at the terminal
 1 foreach node v do
 2     list<Label> labelListAt(v) := ∅;
 3 PriorityQueue pq := ∅;
 4 foreach node v in start interval do
 5     Label startLabel := createStartLabel(v);
 6     pq.insert(startLabel);
 7 while ! pq.isEmpty() do
 8     Label label := pq.extractLabel();
 9     foreach outgoing edge e=(v,w) of v=label.getNode() do
10         if isInfeasible(e) then continue; // ignore this edge
11         Label newLabel := createLabel(label, e);
12         if newLabel is dominated then continue;
13         // newLabel is not dominated
14         pq.insert(newLabel);
15         labelListAt(w).insert(newLabel);
16         labelListAt(w).removeLabelsDominatedBy(newLabel);
```

Algorithm 1. Pseudocode for the generalized Dijkstra algorithm

Walidacja

- Znalezione drogi są *walidowane*, tzn. sprawdzamy, czy sprawdzamy ich dostępność oraz sprawdzamy ceny na aktualne.
- Top 5 tras jest weryfikowanych asynchronicznie z API w czasie rzeczywistym.

Diagram


ChadGuide - Flow wyszukiwania trasy
Użytkownik
Frontend (JS + Leaflet.js)
FastAPI Server
Flight Router
Dijkstra Algorithm
SQLite flights.db
Duffel API
1. Wyszukiwanie
Wybiera miasta, daty, ograniczenia
GET /routes/stream (SSE)
find_optimal_routes(constraints)
load & expand flight data
flights
2. Routing
SSE: stage="routing"
Animacja na mapie
find Pareto-optimal routes
Optymalizacja wielokryterialna:
koszt vs czas podróży
+ Pareto dominance filtering
top N tras
3. Walidacja cen
SSE: stage="validating"
"Sprawdzam aktualne ceny..."
sprawdź top 5 tras
(równolegle, async + semaphore)
potwierdzone ceny / niedostępne
4. Wyniki
wyniki
SSE: stage="complete" + JSON
Trasy z cenami + mapa + atrakcje

Git, dokumentacja, testy, ...

- Całość powstała dzięki systemowi kontroli wersji Git.
- Staraliśmy się prowadzić dokładną dokumentację poszczególnych modułów oraz paczek.
- Do wszystkich istotnych części projektu napisaliśmy testy, również te wydajnościowe. Do testów została użyta paczka `pytest`.

Co mogliśmy zrobić lepiej?

- Za dużo czasu zajęło nam znalezienie odpowiedniego dostawcy danych lotów.
- Niektóre częsci kodu są nadmiernie złożone.
- Złożoność zasadniczej części alogytmu jest duża, jeżeli miast które użytkownik chce odwiedzić jest dużo. Padł pomysł, aby tę część kodu przepisać na C, ale nie było na to czasu.

Podsumowanie

Czy wypełniliśmy cele? *Tak.*
Jest to nasz pierwszy *akademicki* projekt, którego, naszym zdaniem, nie powinnyśmy się wstydzić.

Pytania?

Dziękujemy za uwagę.