Dokumentacja projektu wykonywanego w ramach zajęć BAZY DANYCH I

Konrad Ciężadło

I. Projekt koncepcji, założenia

1. Zdefiniowanie tematu projektu

Temat:

TrainCheck - wyszukiwarka połączeń kolejowych oparta o dane GTFS.

Opis zadania:

Celem projektu jest stworzenie aplikacji webowej, która umożliwia import podstawowych danych GTFS do relacyjnej bazy danych (MySQL), a następnie przetworzenie ich na czytelne dane. Pozwala wyszukiwać bezpośrednie połączenia, harmonogram stacyjny oraz pewne statystyki.

Cele projektu:

- umożliwienie automatycznego importu danych GTFS do bazy (tabele: agency, routes, trips, stops, stop_times, calendar_dates, calendar),
- utworzenie wyszukiwarki bezpośrednich połączeń między stacjami (z wyborem daty i godziny), określa czas przejazdu,
- harmonogram stacyjny: najbliższe odjazdy i przyjazdy z wybranej stacji z destynacją/początkiem trasy i stacjami pośrednimi,
- udostępnienie statystyk z danych GTFS (ilość występowania danych stacji, ilość przejazdów danej agencji, liczba kursów na liniach),
- dodania własnego bezpośredniego przejazdu między dwoma stacjami (custom trip).

2. Analiza wymagań użytkownika

Baza danych powinna zapewniać:

- Import i przechowywanie danych GTFS
- możliwość wczytania danych z plików .txt (GTFS) do tabel,
- ignorowanie niepotrzebnych pól w feedzie,
- obsługę calendar i calendar_dates.
- wyszukiwanie bezpośrednich połączeń dla wybranej daty i godziny odjazdu,

- zwracanie: daty, godziny odjazdu/przyjazdu, identyfikatora/nazwy pociągu (linia), czasu przejazdu,
- pobieranie 10 najbliższych odjazdów oraz 10 najbliższych przyjazdów dla danej stacji i czasu,
- zwracanie informacji: godzina odjazdu/przyjazdu, linia/pociąg (route), stacja początkowa/docelowa, stacje pośrednie.
- pobieranie zestawień do statystyk (liczniki kursów itp.).
- dostęp do importowania plików GTFS i dodawania custom przejazdów (tylko dla kont z rolą ADMIN),
- kontrola danych przy dopisywaniu kursów (referencje do istniejących stacji, kontrolna czasu).

3. Zaprojektowanie funkcji

Views:

- v_departures - ujednolicone dane odjazdów (stop_name, trip_id, stop_sequence, departure_time, route info, service_id),
- v_arrivals - jak v_departures, ale dla przyjazdów (arrival_time),
- v_trip_stops - lista przystanków dla kursu uporządkowana po stop_sequence,
- v_agency_activity - liczba kursów przypisanych do agencji,
- v_route_trip_count - liczba kursów na każdej linii (route).

Funkcje/operacje:

- pobieranie listy stacji: SELECT DISTINCT stops.stop_name ... (datalist dla UI),
- 10 najbliższych odjazdów z danej stacji (od wybranego czasu),
- 10 najbliższych przyjazdów na daną stację (od wybranego czasu),
- wyznaczanie stacji początkowej/docelowej oraz przystanków pośrednich na podstawie stop_sequence,
- dopasowanie odcinka dla trip_id (jego stop_sequence),
- obliczenie czasu przejazdu jako różnicy czasów przyjazdu i odjazdu,
- agregacje oparte o widoki v_agency_activity, v_route_trip_count oraz licznik użyć stacji,
- kurs uznaje się za aktywny w danym dniu, jeśli wynika to z calendar (zakres dat + dzień tygodnia) oraz jest modyfikowany przez calendar_dates (exception_type=1 dodaje, exception_type=2 usuwa).

II. Projekt diagramów (konceptualny)

4. Budowa i analiza diagramu przepływu danych (DFD)

Przepływy danych:

- Autoryzacja: login/register → generowanie JWT → zapis/odczyt użytkownika.
- Import GTFS: pobranie → rozpakowanie → LOAD DATA LOCAL INFILE → dodanie do tabel.
- Wyszukaj połączenie bezpośrednie: start, cel, data, godzina → kwerenda → wyniki.
- Harmonogram stacyjny: stacja, data, godzina → odjazdy/przyjazdy → wyniki.
- Statystyki: agregacje z widoków → prezentacja danych.
- Dodaj trasę: formularz → insert do tabel GTFS → walidacja spójności (triggery).

5. Zdefiniowanie encji (obiektów) oraz ich atrybutów

Encje domenowe/GTFS

- agency(agency_id, agency_name, agency_url, agency_timezone, ...)
- routes(route_id, agency_id, route_short_name, route_long_name, route_type, ...)
- trips(trip_id, route_id, service_id, ...)
- stops(stop_id, stop_name, ...)
- stop_times(trip_id, stop_sequence, arrival_time, departure_time, stop_id, ...)
- calendar(service_id, monday..sunday, start_date, end_date)
- calendar_dates(service_id, date, exception_type)

Encje aplikacyjne

- users(username, password, role, ...)

6. Zaprojektowanie relacji pomiędzy encjami

Relacje 1-N:

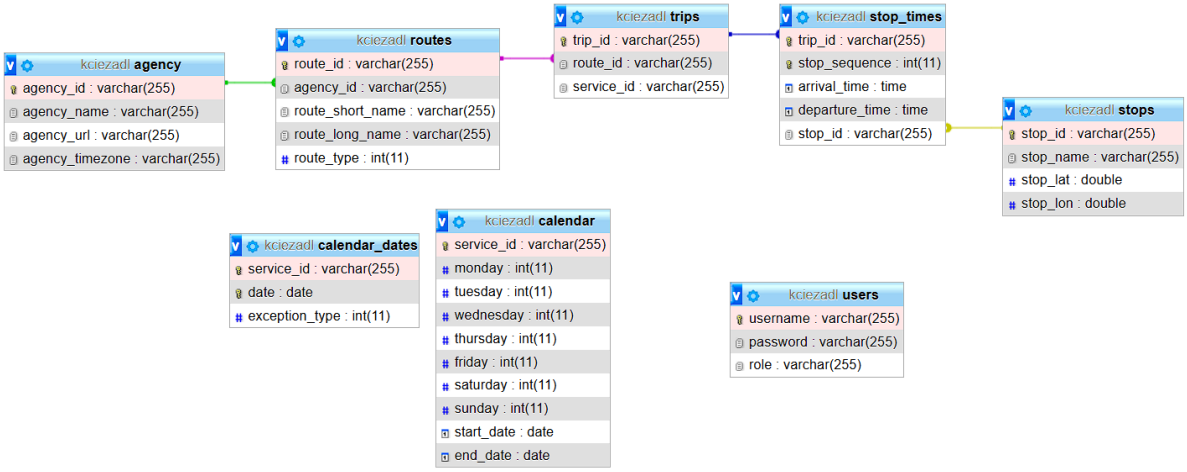
- agency (1) - (N) routes przez routes.agency_id
- routes (1) - (N) trips przez trips.route_id
- trips (1) - (N) stop_times przez stop_times.trip_id
- stops (1) - (N) stop_times przez stop_times.stop_id

Relacja N-M:

- trips (N) - (M) stops

calendar (0/1) - (1) trips.service_id

calendar_dates (0/1) - (1) trips.service_id



Obraz 1.: ERD bazy danych w projekcie.

III. Projekt logiczny

7. Projektowanie tabel, kluczy, indeksów

```
CREATE TABLE agency (
  agency_id       VARCHAR(255) PRIMARY KEY,
  agency_name     VARCHAR(255),
  agency_url      VARCHAR(255),
  agency_timezone  VARCHAR(255)
);

CREATE TABLE routes (
  route_id        VARCHAR(255) PRIMARY KEY,
  agency_id       VARCHAR(255),
  route_short_name  VARCHAR(255),
  route_long_name   VARCHAR(255),
  route_type      INT,

  FOREIGN KEY (agency_id) REFERENCES agency(agency_id)
);

CREATE TABLE calendar (
  service_id  VARCHAR(255) PRIMARY KEY,
  monday     INT,
  tuesday    INT,
  wednesday   INT,
  thursday    INT,
  friday     INT,
```

```
  saturday   INT,
  sunday     INT,
  start_date  DATE,
  end_date   DATE
);

CREATE TABLE calendar_dates (
  service_id     VARCHAR(255),
  date         DATE,
  exception_type  INT,

  PRIMARY KEY (service_id, date),
);

CREATE TABLE trips (
  trip_id    VARCHAR(255) PRIMARY KEY,
  route_id   VARCHAR(255),
  service_id VARCHAR(255),

  FOREIGN KEY (route_id) REFERENCES routes(route_id),
);

CREATE TABLE stops (
  stop_id    VARCHAR(255) PRIMARY KEY,
  stop_name  VARCHAR(255),
  stop_lat   DOUBLE,
  stop_lon   DOUBLE
);

CREATE TABLE stop_times (
  trip_id       VARCHAR(255),
  stop_sequence   INT,
  arrival_time    TIME,
  departure_time  TIME,
  stop_id       VARCHAR(255),

  PRIMARY KEY (trip_id, stop_sequence),

  FOREIGN KEY (trip_id) REFERENCES trips(trip_id),
  FOREIGN KEY (stop_id) REFERENCES stops(stop_id)
);

CREATE TABLE users (
  username VARCHAR(255) PRIMARY KEY,
  password VARCHAR(255),
  role VARCHAR(255)
);
```

8. Słowniki danych

agency:

Kolumna	**Dziedzina**	**Ograniczenia**	**Uwagi**
agency_id	VARCHAR(255)	**PK**, unikalne, implicit NOT NULL	
agency_name	VARCHAR(255)	NULL	
agency_url	VARCHAR(255)	NULL	
agency_timezone	VARCHAR(255)	NULL	nazwa strefy IANA

routes:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
route_id	VARCHAR(255)	**PK**, implicit NOT NULL	
agency_id	VARCHAR(255)	**FK → agency(agency_id)**, NULL	
route_short_name	VARCHAR(255)	NULL	
route_long_name	VARCHAR(255)	NULL	
route_type	INT	NULL	typ GTFS (np. 2 dla pociągów)

calendar:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
service_id	VARCHAR(255)	**PK**, implicit NOT NULL	

monday...sunday	INT	NULL	flaga kursowania (0/1)
start_date	DATE	NULL	
end_date	DATE	NULL	

calendar_dates:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
service_id	VARCHAR(255)	część **PK**, implicit NOT NULL	
date	DATE	część **PK**, implicit NOT NULL	
exception_type	INT	NULL	1 = dodaj, 2 = usuń

trips:

Kolumna	**Typ**	**Ograniczenia**
trip_id	VARCHAR(255)	**PK**, implicit NOT NULL
route_id	VARCHAR(255)	**FK → routes(route_id)**, NULL
service_id	VARCHAR(255)	NULL

stops:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
stop_id	VARCHAR(255)	**PK**, implicit NOT NULL	
stop_name	VARCHAR(255)	NULL	
stop_lat	DOUBLE	NULL	stop_lat ∈ [-90,90]

stop_lon	DOUBLE	NULL	stop_lon ∈ [-180,180]

stop_times:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
trip_id	VARCHAR(255)	część **PK**, implicit NOT NULL, **FK → trips(trip_id)**	
stop_sequence	INT	część **PK**, implicit NOT NULL	stop_sequence > 0, kolejno
arrival_time	TIME	NULL	
departure_time	TIME	NULL	
stop_id	VARCHAR(255)	NULL, **FK → stops(stop_id)**	

users:

Kolumna	**Typ**	**Ograniczenia**	**Uwagi**
username	VARCHAR(255)	**PK**, implicit NOT NULL	
password	VARCHAR(255)	NULL	
role	VARCHAR(255)	NULL	rola (USER/ADMIN)

9. Analiza zależności funkcyjnych i normalizacja tabel (dekompozycja do 3NF)

agency – 3NF - agency_id (PK) jednoznacznie wyznacza pozostałe atrybuty (agency_name, agency_url, agency_timezone) - brak zależności przechodnich.

routes – 3NF - route_id (PK) wyznacza agency_id, nazwy i route_type. Dane agencji nie są powielane w routes (są w osobnej tabeli), więc brak zależności przechodnich.

trips – 3NF - trip_id (PK) wyznacza route_id i service_id - brak atrybutów zależnych od innych atrybutów niekluczowych.

stops – 3NF - stop_id (PK) wyznacza stop_name, stop_lat, stop_lon - atrybuty są atomowe.

stop_times – 3NF - (trip_id, stop_sequence) (PK złożony) wyznacza arrival_time, departure_time, stop_id. Nie ma zależności częściowych (2NF) ani przechodnich (3NF), bo wartości zależą od całego klucza (kurs + pozycja).

calendar – 3NF - service_id (PK) wyznacza flagi dni tygodnia oraz zakres dat (start_date, end_date) - brak przechodnich zależności.

calendar_dates – 3NF - (service_id, date) (PK złożony) wyznacza exception_type - brak zależności częściowych i przechodnich.

users – 3NF - username (PK) wyznacza password i role - brak zależności przechodnich.

10. Zaprojektowanie operacji na danych

Widoki ułatwiające odczyt:

```
CREATE OR REPLACE VIEW v_departures AS
SELECT
  s.stop_name,
  st.trip_id,
  st.stop_sequence,
  st.departure_time,
  t.route_id,
  r.route_long_name,
  r.route_short_name,
  t.service_id
FROM stop_times st
JOIN stops s  ON s.stop_id = st.stop_id
JOIN trips t  ON t.trip_id = st.trip_id
JOIN routes r ON r.route_id = t.route_id;

CREATE OR REPLACE VIEW v_arrivals AS
SELECT
  s.stop_name,
  st.trip_id,
  st.stop_sequence,
  st.arrival_time,
  t.route_id,
  r.route_long_name,
  r.route_short_name,
  t.service_id
FROM stop_times st
```

```
JOIN stops s  ON s.stop_id = st.stop_id
JOIN trips t  ON t.trip_id = st.trip_id
JOIN routes r ON r.route_id = t.route_id;

CREATE OR REPLACE VIEW v_trip_stops AS
SELECT
  st.trip_id,
  st.stop_sequence,
  s.stop_name
FROM stop_times st
JOIN stops s ON s.stop_id = st.stop_id;
```

Wyszukiwanie połączenia bezpośredniego (10 wyników):

```
SELECT
  d.departure_time,
  a.arrival_time,
  d.route_id,
  d.route_long_name,
  d.trip_id,
  SEC_TO_TIME(TIME_TO_SEC(a.arrival_time) - TIME_TO_SEC(d.departure_time)) AS
travel_time
FROM v_departures d
JOIN v_arrivals a
  ON d.trip_id = a.trip_id
WHERE
  d.stop_name = :start_station
  AND a.stop_name = :end_station
  AND d.stop_sequence < a.stop_sequence
  AND TIME_TO_SEC(d.departure_time) >= :time_from_sec
  AND (
    -- GTFS semantics: calendar + calendar_dates(1 add, 2 remove)
    (
      EXISTS (
        SELECT 1
        FROM calendar c
        WHERE c.service_id = d.service_id
          AND :travel_date BETWEEN c.start_date AND c.end_date
          AND CASE DAYOFWEEK(:travel_date)
              WHEN 1 THEN c.sunday
              WHEN 2 THEN c.monday
              WHEN 3 THEN c.tuesday
              WHEN 4 THEN c.wednesday
              WHEN 5 THEN c.thursday
              WHEN 6 THEN c.friday
              WHEN 7 THEN c.saturday
```

```
        END = 1
    )
    OR EXISTS (
     SELECT 1
     FROM calendar_dates cd
     WHERE cd.service_id = d.service_id
       AND cd.date = :travel_date
       AND cd.exception_type = 1
    )
  )
  AND NOT EXISTS (
    SELECT 1
    FROM calendar_dates cd2
    WHERE cd2.service_id = d.service_id
      AND cd2.date = :travel_date
      AND cd2.exception_type = 2
  )
 )
ORDER BY TIME_TO_SEC(d.departure_time)
LIMIT 10;
```

Harmonogram stacyjny (odjazdy/przyjazdy):
Analogicznie - filtr po stop_name, >= time, aktywność serwisu jak wyżej, ORDER BY time, LIMIT 10, a następnie dociągnięcie listy przystanków dla trip_id z v_trip_stops (do wyliczenia pośrednich).

Statystyki:

```
CREATE OR REPLACE VIEW v_agency_activity AS
SELECT a.agency_id, a.agency_name, COUNT(t.trip_id) AS trip_count
FROM agency a
JOIN routes r ON a.agency_id = r.agency_id
JOIN trips t  ON r.route_id = t.route_id
GROUP BY a.agency_id, a.agency_name
HAVING COUNT(t.trip_id) > 0;

CREATE OR REPLACE VIEW v_route_trip_count AS
SELECT r.route_id, r.route_short_name, r.route_long_name, COUNT(t.trip_id) AS
trip_count
FROM routes r
JOIN trips t ON r.route_id = t.route_id
GROUP BY r.route_id, r.route_short_name, r.route_long_name
HAVING COUNT(t.trip_id) > 10;
```

IV. Projekt funkcjonalny

12. Interfejsy do prezentacji, edycji i obsługi danych

Aplikacja jest podzielona na zakładki (navbar). Każda zakładka renderuje osobny formularz lub raport, a przełączanie zakładek czyści widok.

A. Login / Rejestracja

Formularz logowania: username, password, przycisk Log in.
Formularz rejestracji: username, password, przycisk Register.
Po zalogowaniu panel logowania zmienia się w przycisk Log out i pojawia się navbar.

B. Wyszukaj połączenie*

Pola:
Stacja początkowa (input + datalist z bazy)
Stacja końcowa (input + datalist z bazy)
Data (input type="date")
Godzina (input type="time")
Przycisk: Wyszukaj
Wynik: tabela 10 najbliższych połączeń (odjazd/przyjazd, linia/pociąg, czas przejazdu).

C. Harmonogram stacyjny*

Pola:
Nazwa stacji (input + datalist)
Data (date)
Godzina (time)
Przyciski: Odjazdy, Przyjazdy
Wynik: tabela (godzina, linia/pociąg, kierunek, stacje pośrednie).

D. Statystyki

Brak danych wejściowych lub tylko opcjonalne parametry (np. limit).
Wynik: zestaw raportów w formie tabel.

E. Profil

Formularz zmiany hasła: stare hasło, nowe hasło, przycisk Zmień hasło.

F. Dodaj trasę*

Pola:
Route name
Data
Stacja początkowa (datalist)
Godzina odjazdu
Stacja docelowa (datalist)
Godzina przyjazdu
Przycisk Dodaj

Po wysłaniu pokazuje status (ok/błąd).

G. Admin

Przyciski makro-akcji GTFS:
pobierz i rozpakuj feed (np. KML/PR),
wyczyść tabele,
załaduj wszystkie dane GTFS do bazy.
Wyniki akcji w postaci komunikatów tekstowych (status).
Powiązania między formularzami
Auth po zalogowaniu odblokowuje dostęp do aplikacji, admin-only zakładki zależą od role=ADMIN.

*Formularze „Wyszukaj", „Harmonogram stacyjny", „Dodaj trasę" współdzielą listę stacji (datalist) pobieraną z jednego endpointu.

13. Wizualizacja danych

Raporty:

- wyniki wyszukiwania połączeń bezpośrednich: data, godzina odjazdu, godzina przyjazdu, linia/pociąg (np. route_long_name), czas przejazdu.
- tablica stacyjna: odjazdy/przyjazdy z informacją o kierunku i przystankach pośrednich.
- najczęściej używane stacje,
- liczba kursów przypisanych do agencji,
- liczba kursów na liniach.

14. Zdefiniowanie panelu sterowania aplikacji

Panel sterowania to poziomy navbar, który przełącza widoki w zależności od interesującej nas funkcjonalności aplikacji (Wyszukaj trasę / Harmonogram stacyjny / Statystyki / Profil), pokazuje tylko dostępne funkcje zależnie od roli, tj. „Dodaj trasę" i „Admin" widoczne są tylko dla ADMIN. Przy przełączaniu zakładek uruchamia funkcję czyszczącą (wszystkie sekcje UI → innerHTML = '').

15. Makropolecenia

Makropolecenia istnieją w panelu Admin, automatyzują obsługę bazy:

- Pobierz i rozpakuj GTFS (dla wybranego feedu) - pobranie zip i rozpakowanie do katalogu roboczego.
- Wyczyść tabele GTFS - zresetowanie danych w bazie.

- Załaduj GTFS do bazy - automatyczne wykonanie serii importów (tabela po tabeli).

V. Dokumentacja

16. Wprowadzanie danych

Automatyczny import GTFS:

pobranie i rozpakowanie plików .txt, import do MySQL (np. LOAD DATA LOCAL INFILE) z automatycznym ignorowaniem niepotrzebnych kolumn.

Ręczne wprowadzanie danych przez ADMIN:

dodanie własnego custom przejazdu bezpośredniego (route/trip/service/stop_times) przez formularz „Dodaj trasę”.

Dane użytkowników aplikacji:

rejestracja/login, zmiana hasła w profilu.

17. Dokumentacja użytkownika

1. Wejdź na stronę aplikacji.
2. Zaloguj się (lub zarejestruj).
   Użyj nawigacji:
3. Wyszukaj połączenie: wybierz stacje z listy, ustaw datę i godzinę, kliknij „Wyszukaj”.
4. Harmonogram stacyjny: wybierz stację, datę i godzinę, wybierz „Odjazdy” lub „Przyjazdy”.
5. Statystyki: przejdź do zakładki, aby zobaczyć raporty.
6. Profil: zmień hasło.
7. Admin: pobierz/rozpakuj GTFS, wyczyść tabele, wykonaj import.
8. Dodaj trasę: dodaj własny przejazd między dwoma istniejącymi stacjami.

18. Opracowanie dokumentacji technicznej

Plik TECHNICAL.md.

Architektura:

- Frontend (JS + HTML/CSS) komunikuje się z backendem przez REST API.
- Backend (Node.js + Express) realizuje autoryzację (JWT), walidację oraz zapytania SQL.

- MySQL przechowuje dane GTFS; część logiki raportowej realizowana jest przez widoki.

API:

- Endpointy wyszukiwania i rozkładów zwracają JSON.
- Endpointy adminowe wymagają roli ADMIN (middleware).

SQL:

- views.sql: definicje widoków,
- imports.js: import pobranych GTFS,
- sql.js: funkcje dostępu do DB używane przez routes.

Nową funkcję najlepiej realizować przez: (a) widok SQL , (b) funkcja w sql.js, (c) endpoint w routes.js, (d) UI w client/api.js.

19) Wykaz literatury

Dokumentacja GTFS:
https://gtfs.org/documentation/schedule/reference/

Obsługa MySQL z node.js:
https://www.w3schools.com/nodejs/

3NF:
https://www.geeksforgeeks.org/dbms/third-normal-form-3nf/