

Tworzenie aplikacji bazodanowych

Politechnika Śląska

System obsługi przewoźnika komunikacji miejskiej

Sprawozdanie

Autorzy:

Nikodem Wspaniały

Kamil Nabożny

Bartosz Pokorski

Alan Pawleta

Łukasz Wojczuk

1. Spis treści

1.	Spis treści.....	2
2.	Treść zadania.....	2
3.	analiza	3
4.	Opis architektury i użytego stosu technologicznego.....	4
5.	Specyfikacja zewnętrzna	5
6.	Specyfikacja wewnętrzna	13
6.1.	Backend	13
6.2.	Frontend.....	20
7.	Wnioski	22

Tworzenie aplikacji bazodanowych

Politechnika Śląska

2. Treść zadania

Temat:

System obsługi przewoźnika komunikacji miejskiej

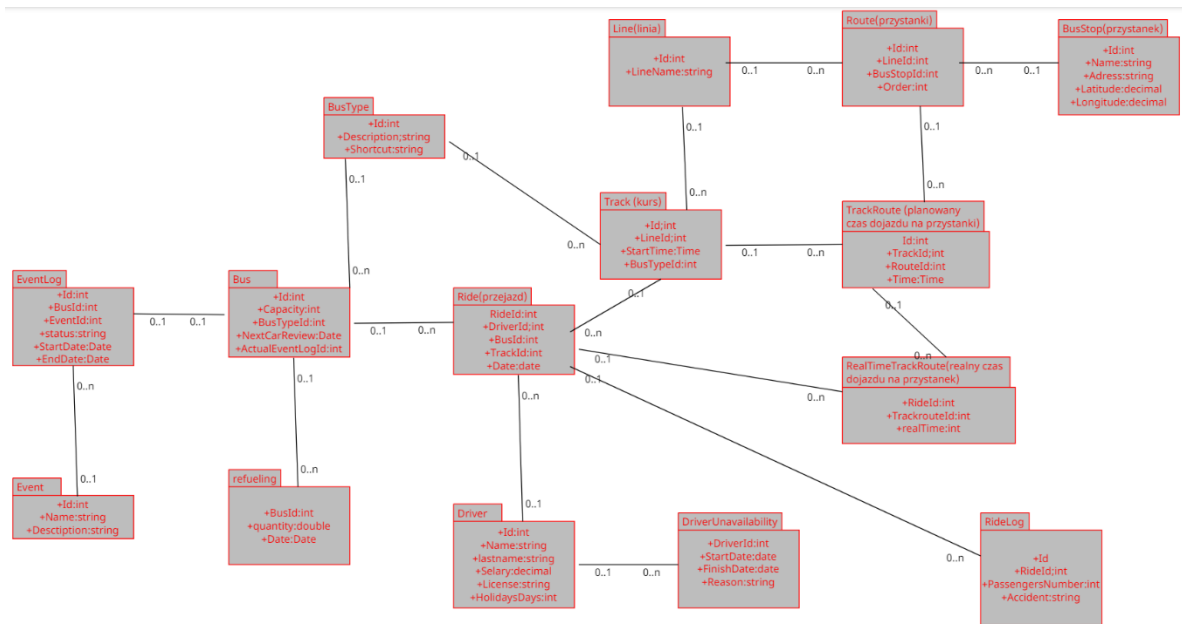
Opis:

System obsługi przewoźnika komunikacji miejskiej; rozkłady jazdy; zarządzanie flotą; raport strategiczny naprawy/przeglądy; ewidencja kursów z przypisaniem do kierowców; raporty operacyjne – spalanie, awaryjność; raporty strategiczne – rentowność poszczególnych linii; spalanie kierowców.

3. analiza

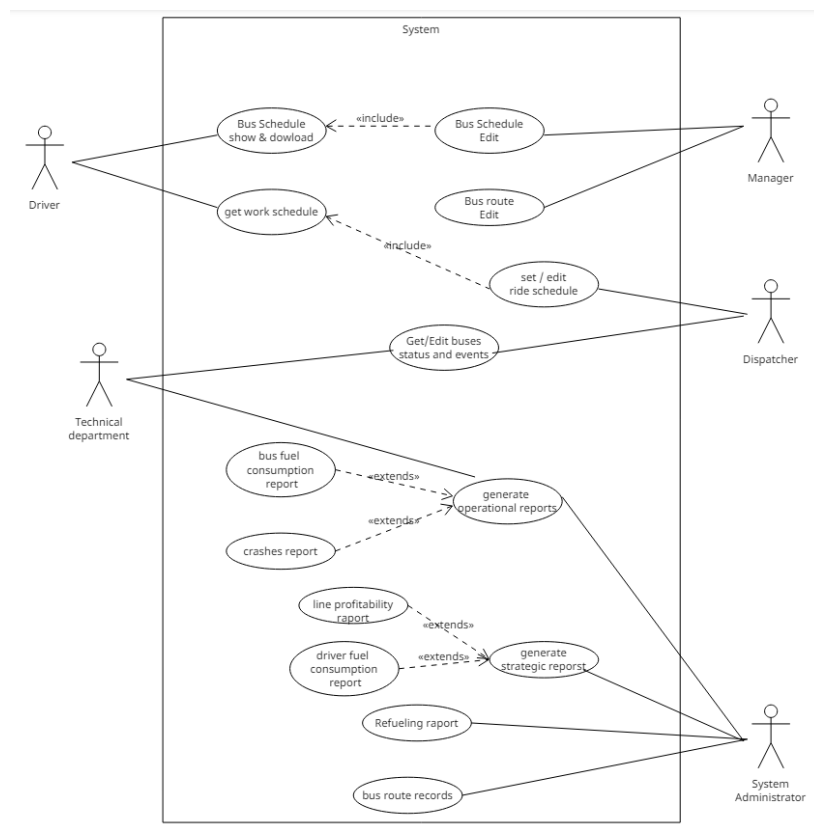
wszystkie modele i diagramy, które zostały stworzone w ramach projektu, diagram klas, przypadki użycia.

Na rysunku 1 został przedstawiony diagram klas oraz pola każdej klasy. Każda klasa zawiera klucz główny, o nazwie 'id'. Na schemacie widać również powiązania między klasami. Diagram został wykonany z pomocą narzędzia UMLetino.



Rysunek 1 diagram klas – UML

Na rysunku 2 widzimy Diagram przypadków użycia aplikacji.



Rysunek 2 Diagram przypadków użycia

4. Opis architektury i użytego stosu technologicznego

Architektura projektu dzieli się na 3 części:

- Frontend
- Backend
- Database

Baza danych została zaprojektowana, stworzona oraz zaimplantowana przy pomocy narzędzia pgAdmin w PostgreSQL. Aby połączyć bazę danych z backendem, wykorzystujemy adapter psycopg2. Aby połączyć backend z frontendem, wykorzystujemy technologię Flask, która umożliwia wysyłanie zapytań do serwera w pythonie przez JavaScripta. Po stronie Frontendowej do stworzenia widoków użytkownika użyliśmy biblioteki ReactJS.

Stos techniczny w skrócie:

Frontend:

- JavaScript
- ReactJS
- HTML / CSS
- Figma

Backend:

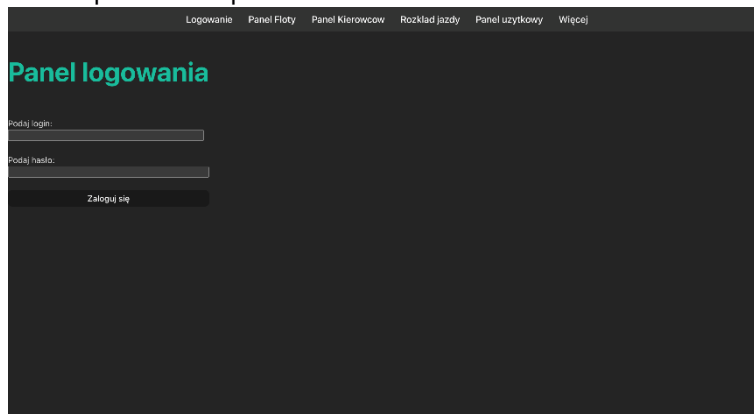
- Python
- Flask
- Swagger
- Psycopg2

Database:

- PostgreSQL
- pgAdmin
- UMLetino

5. Specyfikacja zewnętrzna

Pierwszym panelem jest panel logowania. Użytkownik może zalogować się na swoje konto, do którego są przypisane odpowiednie panele.



Rysunek 3. Panel logowania dla użytkownika.

Panel floty odpowiada za dodawanie, usuwanie oraz wyszukiwanie busów. Wyszukiwanie odbywa się za pomocą pól: Skrót oraz Opis. Jeśli chcemy dodać autobus musimy uzupełnić pola Shortcut, Ilość miejsc, Opis oraz datę przeglądu. Aby usunąć dany rekord musimy kliknąć “usuń” w danym wierszu. Wszystkie dane wyświetlają się w tabeli poniżej.

Logowanie
Panel Floty
Panel Kierowcow
Rozkład jazdy
Panel uzytkowy
Więcej

Panel Floty

Wyszukaj busy
Skrót:
Opis:
Shortcut:
Ilość miejsc:
Opis:
Data przegladu: dd.mm.yyyy
Dodaj autobus

ID	SKROT	ILOSC MIEJSC	OPIS	DATA WAZNOSCI PRZEGLADU	MODYFIKACJA
1	sl	25	pojedynczy, niskopodlogowy	2024-10-16	Usuń
2	sl	25	pojedynczy, niskopodlogowy	2024-10-18	Usuń
3	dl	75	podwójny, niskopodlogowy	2025-01-12	Usuń

Rysunek 4. Panel floty do modyfikacji autobusów.

Logowanie
Panel Floty
Panel Kierowcow
Rozkład jazdy
Panel uzytkowy
Więcej

Panel Floty

Wyszukaj busy
Skrót: sl
Opis: pojedynczy
Shortcut:
Ilość miejsc:
Opis:
Data przegladu: dd.mm.yyyy
Dodaj autobus

ID	SKROT	ILOSC MIEJSC	OPIS	DATA WAZNOSCI PRZEGLADU	MODYFIKACJA
1	sl	25	pojedynczy, niskopodlogowy	2024-10-16	Usuń
2	sl	25	pojedynczy, niskopodlogowy	2024-10-18	Usuń
6	sl	25	pojedynczy, niskopodlogowy	2024-11-01	Usuń

Rysunek 5. Panel floty z wprowadzonymi danymi do wyszukiwania.

Panel kierowców służy do modyfikacji kierowców. Z listy rozwijanej możemy wybrać kierowcę, którego chcemy zobaczyć. Do dodania kierowcy musimy uzupełnić wszystkie pola tekstowe oraz kliknąć przycisk “Dodaj kierowcę”. Aby usunąć kierowcę wybieramy interesujące nas pole z listy rozwijanej pod tekstem “Usuń kierowcę” oraz klikamy przycisk “usuń kierowcę”. Wszystkie dane wyświetlają się w tabeli poniżej.

Logowanie
Panel Floty
Panel Kierowcow
Rozkład jazdy
Panel uzytkowy
Więcej

Panel Kierowcow

Wybierz kierowcę:
Ważność kierowcy
Imię:
Nazwisko:
Licencja:
Wypłata: 0
Dni wolne: 0
Dodaj kierowcę
Usuń kierowcę:
Wybierz kierowcę
Usuń kierowcę

IMIE	NAZWISKO	ZAROBKI	LICENCJA	ILOSC DNI WOLNYCH
------	----------	---------	----------	-------------------

Rysunek 6. Panel kierowców na którym możemy modyfikować kierowców.

Logowanie
Panel Floty
Panel Kierowców
Rozkład jazdy
Panel użytkowy
Więcej

Wybierz kierowcę:

Alan Paweł

Imię:
Nazwisko:
Licencja:
Wypłata:
Dni wolne:

Dodaj kierowcę

Usuń kierowcę:

Wybierz kierowcę

Usuń kierowcę

IME	NAZWISKO	ZAROBKI	LICENCJA	ILUŚĆ DNI WOLNYCH
Alan	Paweł	\$3,000.00	D2	24

Rysunek 7. Panel kierowców, na którym widać wprowadzone dane do wyświetlania.

Panel rozkładu jazdy służy do wyświetlania rozkładu dla danego przystanku. W pole tekstowe wpisujemy nazwę przystanku oraz klikamy na interesujący nas przystanek z listy, która się pokaże. W tabeli poniżej wyświetlają się autobusy wraz z godzinami odjazdu, które kursują przez ten przystanek. Nad tabelą wyświetla się kierunek jazdy autobusu.

Logowanie
Panel Floty
Panel Kierowców
Rozkład jazdy
Panel użytkowy
Więcej

Rozkład jazdy dla przystanku o danym przystanku

• malina2

Kierunek jazdy: dworzec

LINIA	CZAS ODJAZDU
1	10:00:00, 11:00:00, 12:00:00, 13:00:00, 14:00:00, 15:00:00, 16:00:00, 17:00:00, 18:00:00, 19:00:00, 20:00:00
3	19:00:00, 20:00:00, 21:00:00, 22:00:00, 23:00:00
A	09:30:00, 09:30:00, 12:00:00, 12:30:00, 15:00:00, 15:30:00, 20:00:00
C	06:15:00, 06:45:00, 07:15:00, 07:45:00, 08:15:00, 08:45:00

Rysunek 8. Panel rozkładu jazdy dla danego przystanku

Logowanie
Panel Floty
Panel Kierowców
Rozkład jazdy
Panel użytkowy
Więcej

Rozkład jazdy dla przystanku o danym przystanku

• dworzec

Kierunek jazdy: petla

LINIA	CZAS ODJAZDU
1	10:00:00, 11:00:00, 12:00:00, 13:00:00, 14:00:00, 15:00:00, 16:00:00, 17:00:00, 18:00:00, 19:00:00, 20:00:00
3	19:00:00, 20:00:00, 21:00:00, 22:00:00, 23:00:00
A	09:30:00, 09:30:00, 12:00:00, 12:30:00, 15:00:00, 15:30:00, 20:00:00
C	06:15:00, 06:45:00, 07:15:00, 07:45:00, 08:15:00, 08:45:00

Rysunek 9. Panel rozkładu jazdy dla danego przystanku.

Panel użytkowy służy do dodawania rekordów. Możemy dodać w nim linię autobusu, zdarzenie oraz przystanek. Wystarczy wpisać wartości w odpowiednie pola tekstowe oraz wcisnąć odpowiedni przycisk “Dodaj”.

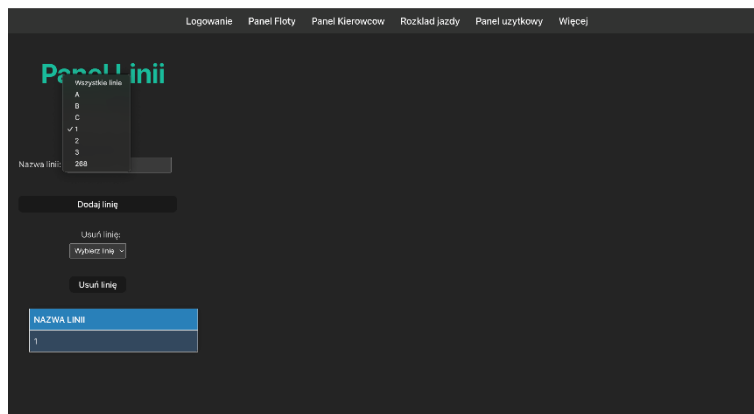
Rysunek 10. Panel użytkowy, w którym możemy dodać rekordy do bazy danych.

Panel zdarzeń pozwala nam dodać, usuwać oraz przeglądać zdarzenia. Za pomocą listy rozwijanej możemy wyszukiwać oraz wyświetlać zdarzenia. Po wpisaniu odpowiednich wartości w pola Nazwa i Opis możemy dodać zdarzenie przyciskiem “Dodaj zdarzenie”. Usunąć zdarzenie możemy wybierając je z listy rozwijanej pod napisem “usuń zdarzenie” oraz klikając przycisk “Usuń zdarzenie”. W tabeli poniżej wyświetlają się zdarzenia.

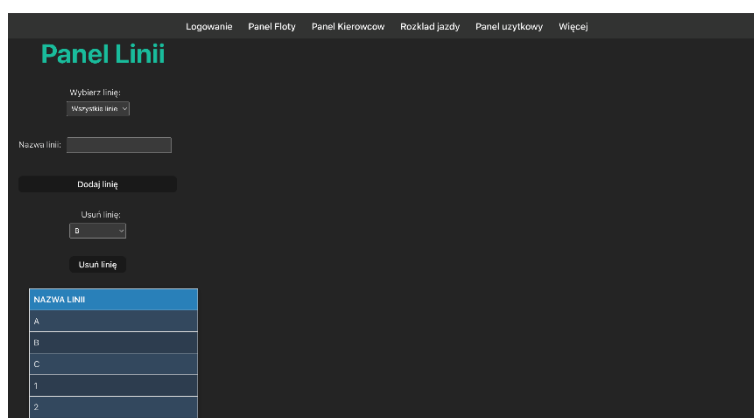
NAZWA	OPIS
dłużawa opora	pojazd uszkodzony

Rysunek 11. Panel zdarzeń pozwala na dodawanie, usuwanie oraz przeglądanie zdarzeń.

Panel linii pozwala na dodawanie, usuwanie oraz przeglądanie linii w bazie danych. Z listy rozwijanej możemy wybrać daną linię i sprawdzić czy istnieje ona w bazie. Wpisując nazwę linii w pole “nazwa” i klikając przycisk “dodaj” możemy dodać linię do bazy. Usunąć linię możemy poprzez wybranie jej z listy rozwijanej i kliknięcie przycisku “usuń linię”. W tabeli poniżej wyświetlą się wszystkie linie lub konkretna linia, która nas interesuje.

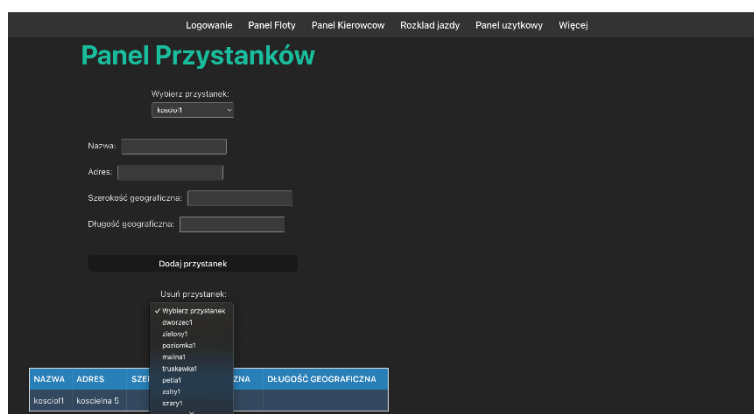


Rysunek 12. Panel linii na którym widzimy zaznaczoną linię “1”.



Rysunek 13. Panel linii na której widać wszystkie linie.

Panel przystanków pozwala na przeglądanie, dodawanie oraz usuwanie przystanków. W tabeli poniżej wyświetlają się wszystkie przystanki wraz z nazwą, adresem i współzrędnymi geograficznymi. Pod tekstem “Wybierz przystanek” z listy rozwijanej możemy wybrać przystanek, który jako jedyny pojawi nam się w tabeli. Wypietniając wszystkie pola tekstowe i klikając przycisk “Dodaj przystanek” dodamy przystanek do bazy danych. Wybierając przystanek z listy rozwijanej pod tekstem “Usuń przystanek” możemy go usunąć klikając przycisk “Usuń przystanek”, który znajdują się pod listą rozwijaną.



Rysunek 14. Panel przystanków, służy do dodawania, usuwania oraz wyświetlania przystanków.

Panel typów autobusów pozwala na przeglądanie, dodawanie oraz usuwanie typów autobusów. Z listy rozwijanej możemy wybrać typ autobusu, który ma się wyświetlić w tabeli (w przeciwnym razie w tabeli wyświetlą się wszystkie autobusy). Wypełniając pola tekstowe oraz klikając przycisk “Dodaj typ autobusu” dodamy autobus do bazy danych. Wybierając typ autobusu z listy rozwijanej pod etykietą “Usuń typ autobusu” oraz klikając przycisk “Usuń typ autobusu” możemy usunąć typ autobusu z bazy.

Rysunek 15. Panel typów autobusów, w którym możemy dodawać, usuwać oraz przeglądać dane.

Panel Event-Log służy do dodawanie, usuwania oraz przeglądania zdarzeń. Poprzez listę rozwijaną pod etykietą “Wybierz event” możemy przeglądać zdarzenia. Uzupełniając wszystkie pola oraz klikając “Dodaj event” możemy dodać zdarzenie do bazy danych, aby usunąć zdarzenie musimy je wybrać z listy rozwijanej pod etykietą “Usuń zdarzenie” i nacisnąć przycisk “Usuń event-log”.

Rysunek 16. Panel Event-Log.

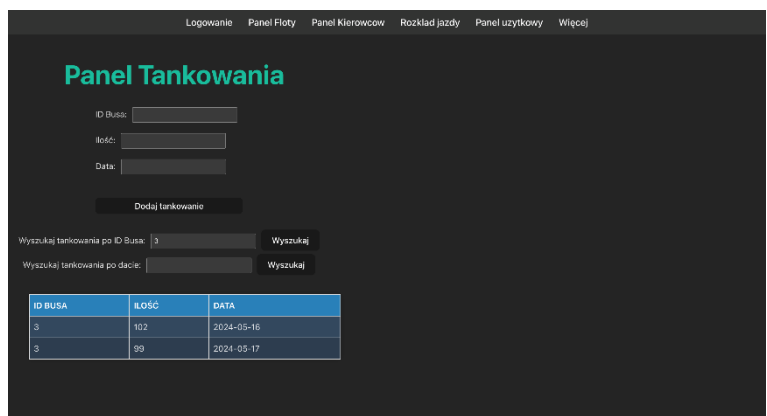
ID	BUS ID	EVENT ID	STATUS	DATA ROZPOCZĘCIA	DATA ZAKOŃCZENIA
1	1	1	status	2024-06-12	2024-06-13

Rysunek 17. Panel Event-Log, w którym widzimy wybrane dane zdarzenie.

Panel spalania pozwala pobrać dane spalania dla danego busa albo kierowcy. Podając ID busa lub kierowcy oraz datę możemy zobaczyć jakie jest spalanie. Spalanie wyświetli się odpowiednio pod etykietami: “Spalanie busa” i “Spalanie Kierowcy”.

Rysunek 18. Panel spalania dla busów i kierowców.

Panel tankowania służy do dodawania oraz przeglądania ilości jaką dany autobus został zatankowany. Wypełniając pola ID busa, Ilość oraz Data, a następnie klikając przycisk “Dodaj tankowanie” możemy dodać informacje o zatankowaniu autobusu. Uzupełniając pola wyszukaj i klikając przycisk “wyszukaj” odpowiednio możemy wyszukiwać informacje o tankowaniu danego busa lub tankowania wszystkich busów w danym dniu.



Rysunek 19 Panel tankowania, który pozwala dodawać oraz przeglądać informacje o tankowaniu.

Panel real-time służy do obliczania i wyświetlania spóźnień autobusów. Poprzez wpisaniu RIDE ID oraz Trackroute ID możemy wyświetlić planowany czas przyjazdu, rzeczywisty czas przyjazdu oraz informację ile minut autobus się spóźnił.

RIDE ID	TRACKROUTE ID	SCHEDULED TIME	REAL TIME	DATE	DIFFERENCE
1	1	10:00:00	10:00:00	2024-05-16	0 min
1	2	10:06:00	10:07:00	2024-05-16	1 min
1	3	10:16:00	10:18:00	2024-05-16	2 min
1	4	10:29:00	10:30:00	2024-05-16	1 min
1	5	10:38:00	10:39:00	2024-05-16	1 min
1	6	10:46:00	10:47:00	2024-05-16	1 min
2	7	11:00:00	11:00:00	2024-05-16	0 min
2	8	11:12:00	11:13:00	2024-05-16	1 min
2	9	11:25:00	11:27:00	2024-05-16	2 min
2	10	11:37:00	11:38:00	2024-05-16	1 min

Rysunek 20. Panel real-time.

RIDE ID	TRACKROUTE ID	SCHEDULED TIME	REAL TIME	DATE	DIFFERENCE
5	27	14:17:00	14:20:00	2024-05-16	3 min

Rysunek 21. Panel real-time wyświetlający spóźnienie dla danego przejazdu na danych odcinku.

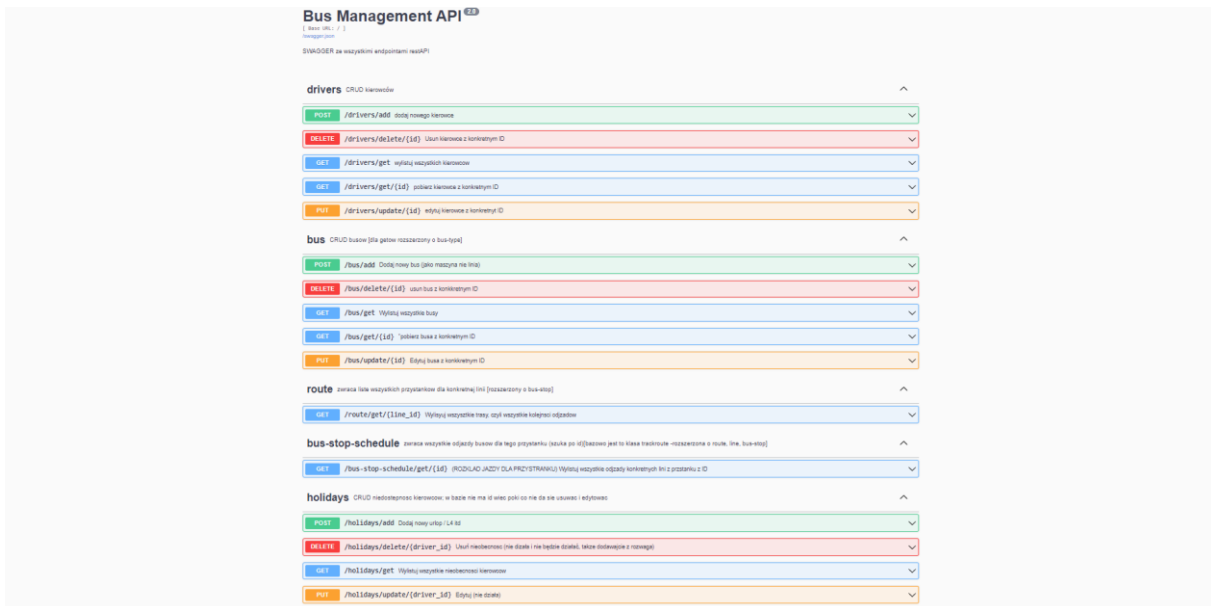
6. Specyfikacja wewnętrzna

6.1. Backend

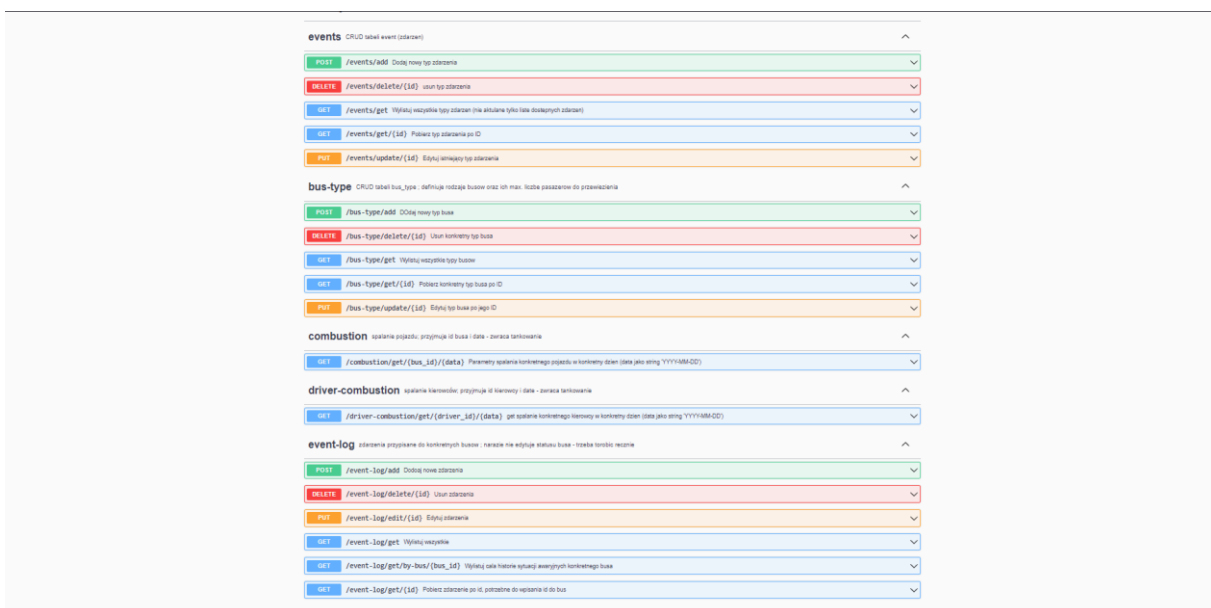
Do projektu został dodany Swagger, aby w celu wizualizacji wszystkich stworzonych endpointów oraz modeli potrzebnych do wysyłania zapytania POST, CREATE.

Listę znajdziemy, przy włączonym serwerze, pod adresem <http://127.0.0.1:5000/swagger/> (port 5000 jest portem domyślnym, może się różnić).

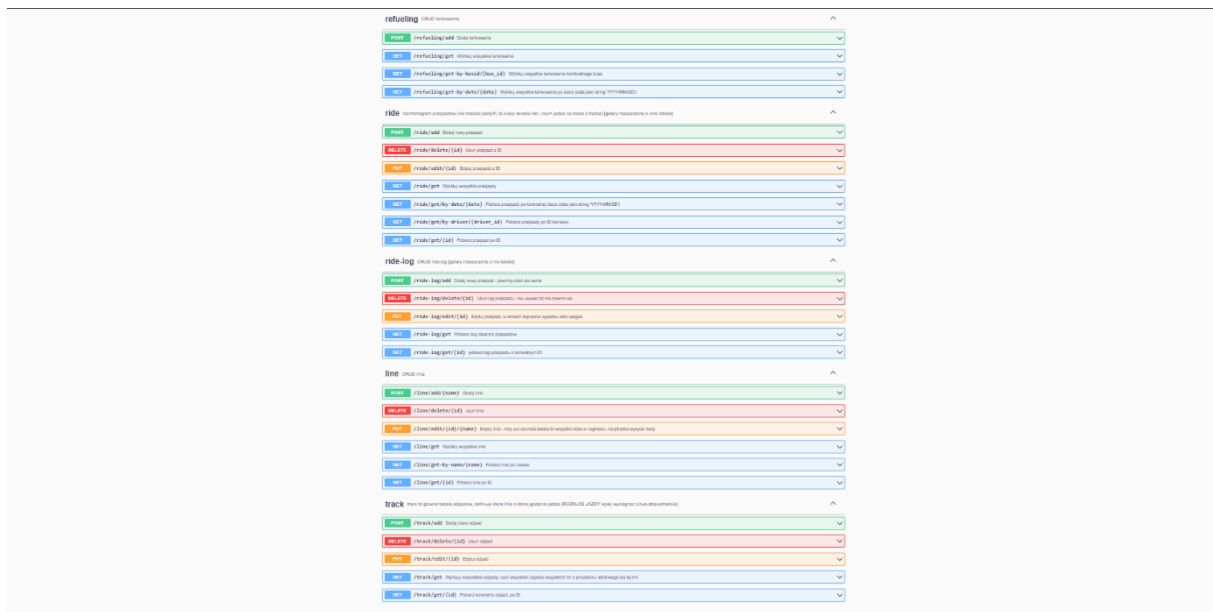
Dokumentacja Swagger:



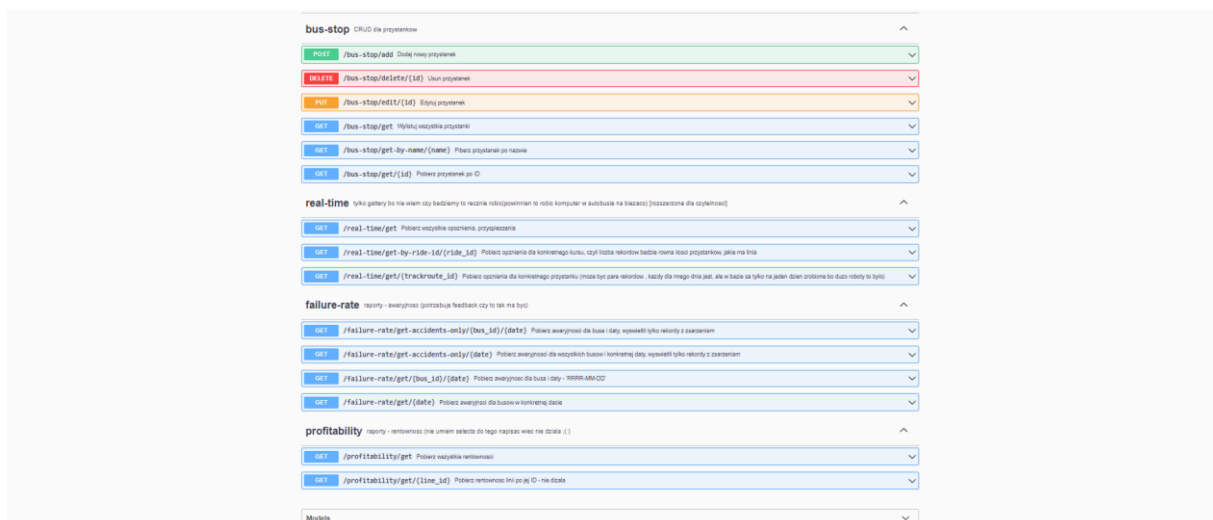
Rysunek 22 Lista endpointów - 1



Rysunek 23 Lista endpointów - 2



Rysunek 24 Lista endpointów - 3



Rysunek 25 Lista endpointów - 4

Lista modeli wygenerowana przez Swagger:

Models	
DriverModel	{ ... }
BusModel	{ ... }
HolidayModel	{ ... }
EventModel	{ ... }
BusTypeModel	{ ... }
eventLogModel	{ ... }

Rysunek 26 Lista modeli - 1

eventLogModel	{ ... }
refuelingModel	{ ... }
rideModel	{ ... }
rideLogModel	{ ... }
trackModel	{ ... }
BusStopModel	{ ... }

Rysunek 27 Lista modeli – 2

Każdy endpoint jest osobną klasą w projekcie w pythonie, który jest tworzony przy pomocy technologii Flask. Te metody definiują argumenty, które przyjmują oraz wywołują konkretne, dedykowane metody aby oddzielić logikę. Wszystkie klasy są zdefiniowane w pliku app.py. Oraz ich główne metody znajdują się w folderze Queries.

```

@ns_driver.route('/get')
class DriverList(Resource):
    def get(self):
        """wylistuj wszystkich kierowcow"""
        try:
            return driverGetAll()
        except Exception as e:
            return error(e)

@ns_driver.route('/add')
class DriverAdd(Resource):
    @ns_driver.expect(driver_model)
    def post(self):
        """dodaj nowego kierowce"""
        try:
            return driverCreate(request.json)
        except Exception as e:
            return error(e)

```

Rysunek 28 Rysunek przedstawia przykład 2 klas po stronie serwerowej, gdzie metoda post dla klasy dodającej nowego kierowcę w klasie DriverAdd oczekuje dostać w zapytaniu body, z konkretnym modelem. Metoda get dla klasy DriverList nie przyjmuje parametrów, zwraca listę wszystkich kierowców w przypadku powodzenia.

```

from database import create_connection
from Queries.Extends.responseExtend import concatNameValue

def driverGetAll():
    connection = create_connection()
    if connection is None:
        return {"error": "Nie udało się połączyć z bazą danych"}, 500
    cursor = connection.cursor()
    query = 'SELECT * FROM public."driver"'
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    drivers = cursor.fetchall()
    cursor.close()
    connection.close()
    response = concatNameValue(columns, drivers)
    return {"drivers": response}

```

Rysunek 29 Przykład metody z folderu Queries łączącej się z bazą danych i wysyłającej zapytanie przy użyciu dodatkowego importu 'create_connection' pokazanego na rysunku 30

Na rysunku 30 widzimy dwie metody.

Create_connection() to bezparametrowa metoda, która przy użyciu psycopg2 łączy serwer z bazą danych, jest to metoda wywoływana przez każdą metodę z zapytaniem (przykładowa metoda na rysunku 29). Ta metoda wykorzystuje statyczne parametry niezbędne do podłączenia się do serwera bazy z pliku config.py (rysunek 31).

Drugą metodą jest error, która wypisuje na konsoli przekazany parametr (opis błędu), oraz zwraca error, który jest przekazywany do klienta serwera. Przykład użycia na rysunku 28.


```

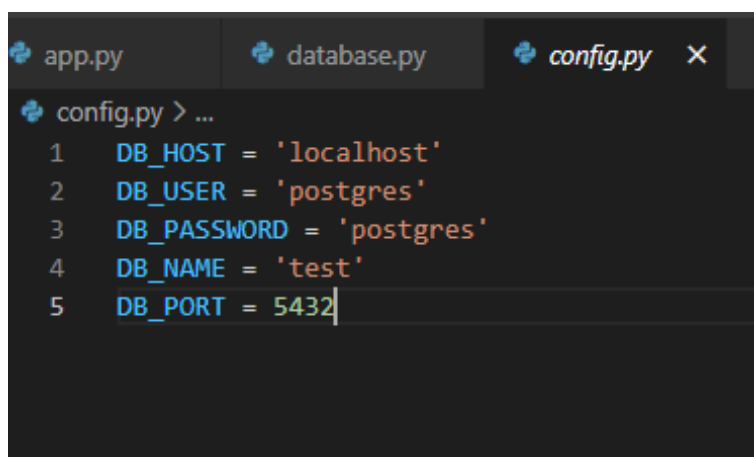
import psycopg2
from config import DB_HOST, DB_USER, DB_PASSWORD, DB_NAME, DB_PORT

def create_connection():
    conn = None
    try:
        conn = psycopg2.connect(
            database = DB_NAME,
            user = DB_USER,
            host = DB_HOST,
            password = DB_PASSWORD,
            port = DB_PORT
        )
        print("Connected to the PostgreSQL database successfully")
        return conn
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
        return None

def error(e):
    print("Wystąpił błąd podczas pobierania danych:", e)
    return {"error": e}, 500

```

Rysunek 30 rysunek przedstawia 2 metody powiązane z łączeniem się z bazą danych znajdujące się w pliku *database.py*.



```

app.py  database.py  config.py X
config.py > ...
1  DB_HOST = 'localhost'
2  DB_USER = 'postgres'
3  DB_PASSWORD = 'postgres'
4  DB_NAME = 'test'
5  DB_PORT = 5432

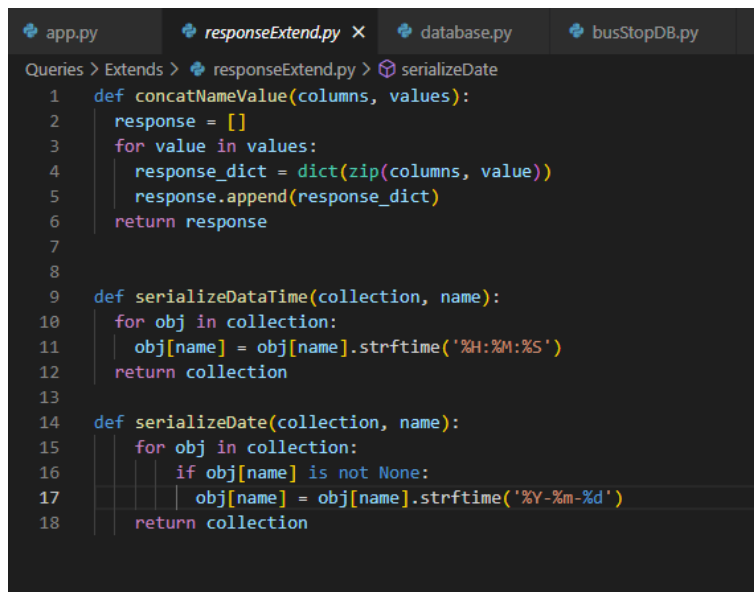
```

Rysunek 31 Plik konfiguracyjny połączenie z bazą danych

Dodatkowo plik *responseExtend.py* zawiera dodatkowe metody, pomocne przy generowaniu response dla klienta (rysunek 32).

Metody:

- `concatNameValue(columns, value)` – łączy opisy pól w obiektach wraz z ich wartością
- `serializeDateTime(collection, name)` – zmienia format wyświetlanych godzin, aby można je było skompresować
- `serializeDate(collection, name)` – zmienia format wyświetlanych dat, aby można je było skompresować

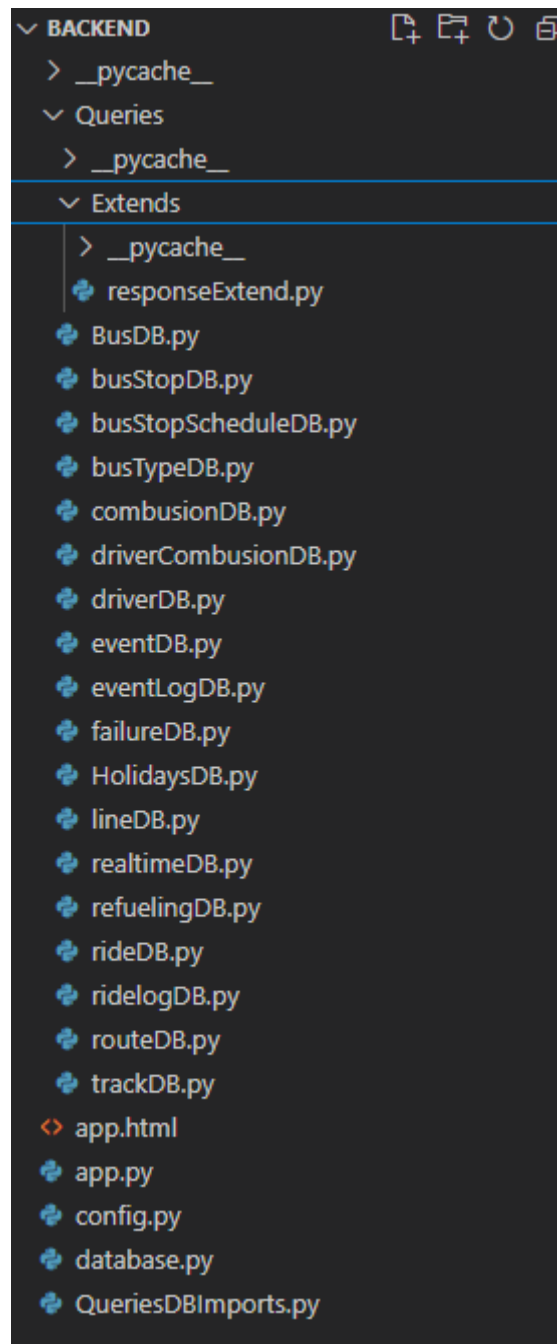


```
app.py  responseExtend.py X  database.py  busStopDB.py

Queries > Extends > responseExtend.py > serializeDate

1  def concatNameValue(columns, values):
2      response = []
3      for value in values:
4          response_dict = dict(zip(columns, value))
5          response.append(response_dict)
6      return response
7
8
9  def serializeDateTime(collection, name):
10     for obj in collection:
11         obj[name] = obj[name].strftime('%H:%M:%S')
12     return collection
13
14  def serializeDate(collection, name):
15     for obj in collection:
16         if obj[name] is not None:
17             obj[name] = obj[name].strftime('%Y-%m-%d')
18     return collection
```

Rysunek 32 Plik responseExtend z pokazanymi metodami



Rysunek 33 Ogólna struktura projektu

Na rysunku 33, została przedstawiona ogólna struktura projektu. W głównym pliku jest plik startowy, plik umożliwiający połączenie z bazą danych oraz plik konfiguracyjny. Dodatkowo Plik QueriesDBImports.py zawiera wszystkie importy ze wszystkich plików w folderze Queries, który następnie jest importowany przez app.py aby ten miał dostęp zapytań. Zostało to wprowadzone w celu poprawienia czytelności kodu. W folderze Queries znajdują się podfolder Extends, który zawiera plik responseExtend.py (rysunek 32) oraz wszystkie pliki z zapytaniami dla każdej klasy, będącej endpointem (rysunek 29).

6.2. Frontend

Frontend został wykonany z pomocą języka JavaScript oraz biblioteki ReactJS. Układ strony oraz jej stylizacja została zrobiona przy użyciu HTML oraz CSS. W projekcie na górze strony został zaimplementowany statyczny pasek nawigacyjny za pomocą, którego możemy przechodzić pomiędzy podstronami. Do przełączania pomiędzy podstronami wykorzystaliśmy routing dla reacta importując bibliotekę react-router-dom.

Każda podstrona jest osobnym komponentem, który jest importowany przez główny plik “App.jsx”. Połączenie z serwerem jest realizowane przy pomocy funkcji fetch() oraz useEffect() dostępnej w react, przez co wszystkie żądania oraz odpowiedzi są konwertowane do formatu JSON. Na każdej podstronie łączymy się z odpowiednimi endpointami do odczytywania danych oraz wysyłania danych do serwera. Wszystkie dane przechowujemy za pomocą funkcji useState() oraz znajdują się one w tablicy.

Przy pomocy prostych metod “handle” obsługujemy formularze oraz zapisujemy wartości w odpowiednie miejsca, żeby móc potem wysłać je na serwer metodą taką jak POST. Przyciski formularzy nie odświeżają stron, przez co wszystko działa płynnie.

```
useEffect(() => {  
  fetch('http://127.0.0.1:5000/bus/get')  
    .then(response => response.json())  
    .then((data) => {  
      console.log(data);  
      setPosts(data.buses);  
      console.log(posts);  
    })  
    .catch((err) => {  
      console.log(err.message);  
    });  
}, []);
```

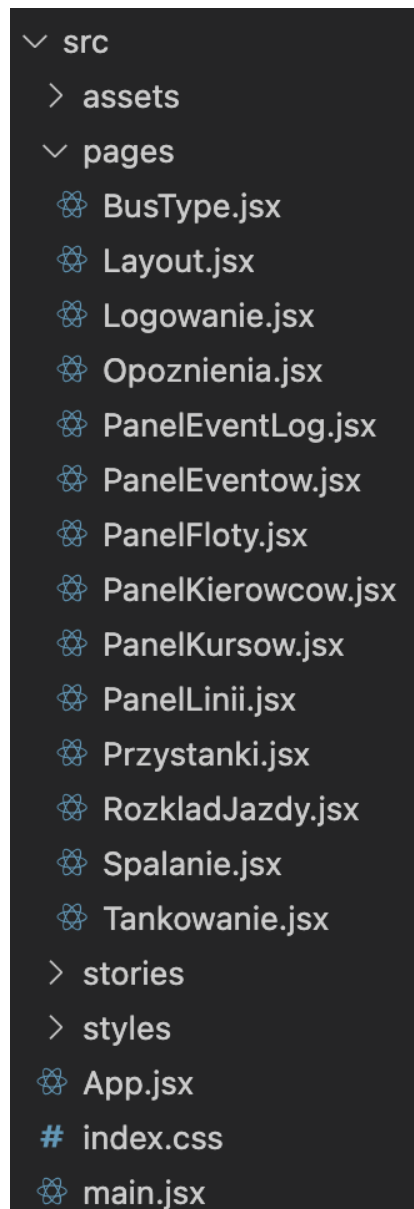
Rysunek 34. Rysunek przedstawia przykładowe połączenie z serwerem, które pobiera wszystkie autobusy z bazy danych oraz zapisuje je do zmiennej “posts”. W razie błędów funkcja zwraca wiadomość co poszło nie tak.

```
const handleNewBusChange = (e) => {  
  const {name, value} = e.target;  
  setNewBus((prevNewBus) => ({  
    ...prevNewBus, [name]: value  
  }));  
}
```

Rysunek 35. Przykładowa funkcja do obsługi formularzy, która umożliwi wpisywanie i odświeżanie danych na bieżąco bez konieczności przeładowywania strony.

```
const handleAddBus = async (e) => {  
  console.log("Sending new bus data:", newBus);  
  e.preventDefault();  
  try {  
    const response = await fetch("http://127.0.0.1:5000/bus/add", {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'application/json',  
      },  
      body: JSON.stringify(newBus),  
    });  
    if(!response.ok) {  
      throw new Error("Network response was not ok");  
    }  
    const data = await response.json();  
  }  
}
```

Rysunek 36. Przykładowa funkcja do wysyłania danych na serwer. Połączenie realizowane za pomocą funkcji asynchronicznej z wykorzystaniem bloku try i catch w razie problemów z połączeniem z serwerem.



Rysunek 37. Ogólna struktura plików na frontendzie. Każda podstrona jest reprezentowana przez osobny plik. Routing jest stworzony w pliku App.jsx.

7. Wnioski

Dzięki projektowi z przedmiotu Tworzenie Aplikacji Bazodanowych mogliśmy wykorzystać wiedzę teoretyczną zdobytą na poprzednich semestrach w sposób praktyczny.

Zajęcia były prowadzone w sposób systematyczny, dzięki czemu projekt był stopniowo rozwijany oraz na bieżąco monitorowany przez prowadzącego, którego uwagi były na bieżąco wdrażane.

Napotkane przez nas przeszkody to problem z prawidłowym zaprojektowaniem modelu bazy danych od podstaw oraz połączenie frontend'u z backend'em (CORS), przez który nasza praca miała lekkie opóźnienie.