

Rezerwacja miejsc hotelowych

-

UAIM 24Z Projekt

Aleksandra Rynkiewicz

Michał Filipkowski

Franciszek Zagól

Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych

12.01.2025

Spis treści

1. Wprowadzenie	3
2. Singe Page Application	4
2.1. Funkcjonalność	4
3. Flask Backend	9
3.1. Ścieżki udostępniane przez aplikacje	12
4. Baza danych	16
5. Aplikacja Android	17
5.1. Drzewo aplikacji i przygotowanie działania w xml	18
5.1.1. AndroidManifest.xml	18
5.1.2. network_security_config.xml	19
5.2. Przygotowanie layoutów	19
5.3. Klasy użytkowe	20
5.3.1. BookingsAdapter	20
5.3.2. NetworkUtils	21
5.3.3. CookieManager	21
5.3.4. Modele danych: Hotel.kt i ProfileResponse.kt	22
5.4. Działanie aplikacji	23
6. Konteneryzacja	29
6.1. Frontend	29
6.2. Backend	29
6.3. Baza danych	29
6.4. docker compose	30
7. Podsumowanie	32

1. Wprowadzenie

Projekt realizuje implementację usługi, która pozwala na internetową rezerwację miejsc. Składa się ona z paru komponentów, a mianowicie:

- SPA (Single Page Application) - aplikacja webowa napisana w frameworku React. Jest ona punktem wejściowym dla klienta, z którego może on przeglądać hotele, filtrować je i wybrać odpowiednie miejsce. Gdy tego dokona może się zalogować i złożyć rezerwację.
- API (Flask Backend) - API stworzone dla aplikacji webowej jak i dla aplikacji Android. Zapewnia połączenie z bazą danych MySQL.
- Baza danych MySQL - baza danych przechowująca dane o klientach, rezerwacjach i hotelach.
- Aplikacja Android - aplikacja pozwalająca na mobilne korzystanie z aplikacji.

Link do repozytorium: [GitHub](#)

2. Single Page Application

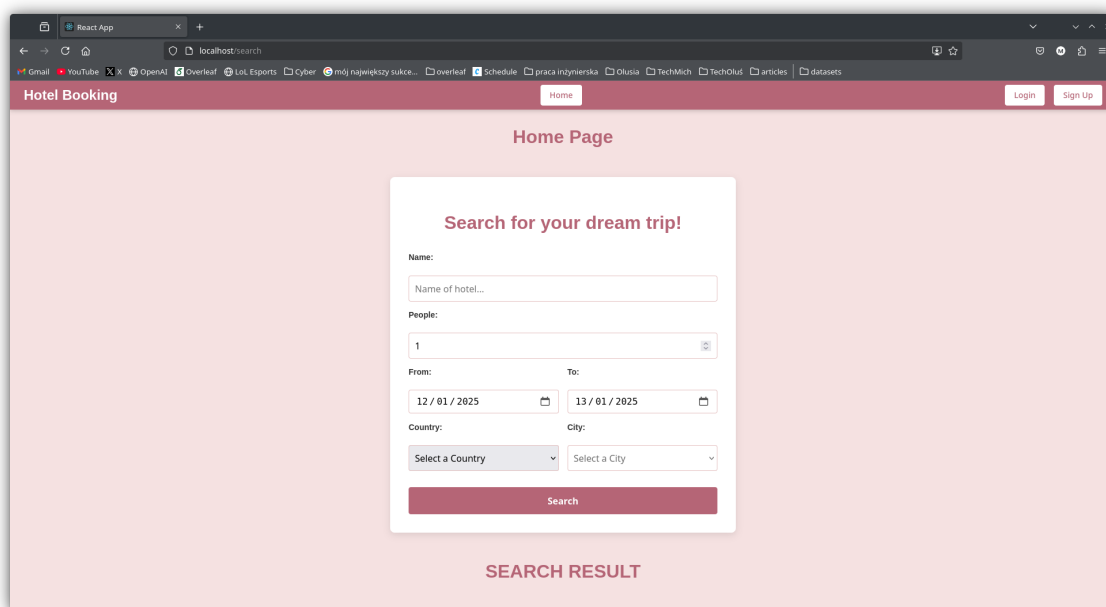
Aplikacja napisana z wykorzystaniem frameworka React i Nodejs.

Jej głównymi funkcjonalnościami są:

- Wyświetlanie odpowiednich stron i podstron w celu zapewnienia usługi
- Obsługa logowania i utrzymania sesji (cookies)
- Komunikacja z API w celu wymiany informacji
- Zapewnienie routingu

2.1. Funkcjonalność

Poniżej zaprezentowane są odpowiednie strony, które służą do interakcji z użytkownikiem. Zaprezentowana zostanie użycie strony oraz najbardziej prawdopodobny scenariusz użycia. Na początku:



Rys. 1. Główna strona internetowa

Po wejściu na stronę użytkownikowi pokazuje się strona główna, która zawiera formularz do wyszukiwania/filtrowania hoteli. Wyżej widać pasek z najważniejszymi skrótami takimi jak **Home**, **Login** i **Sign In**. Po wpisaniu/wybraniu wartości niżej pojawiają się wyniki wyszukiwań z danymi hotelami. W wyszukiwaniu są uwzględnione obecnie złożone rezerwacje więc np. jeżeli hotel jest w pełni zajęty w danym dniu to nie będzie się wyświetlał w wynikach. Jeżeli wartości są nie wypełnione to nie biorą udziału w wyszukiwaniu, więc pusta nazwa nie wyfiltruje żadnych hoteli, zwróci każdy z nich.

Przykładowe wyszukiwanie:

The screenshot shows a web application interface for searching hotels. The search form is titled "Search for your dream trip!" and includes fields for Name, People, From, To, Country, and City. The search results are displayed in a table with columns: Name, City, Country, and Book.

Search Form:

- Name:
- People:
- From:
- To:
- Country:
- City:
-

SEARCH RESULT

Name	City	Country	Book
+ Porto Riverside Inn	Porto	Portugal	<input type="button" value="Book"/>

Rys. 2. Przykładowe wyszukiwanie hoteli

We wyszukiwaniu można rozwinąć detale, żeby można było zobaczyć dostępne pokoje i ich ceny:

The screenshot shows the same web application interface as Rys. 2, but with an expanded search result for "Porto Riverside Inn". The expanded result shows a table with columns: Size and Price Per Night.

Search Form:

- From:
- To:
- Country:
- City:
-

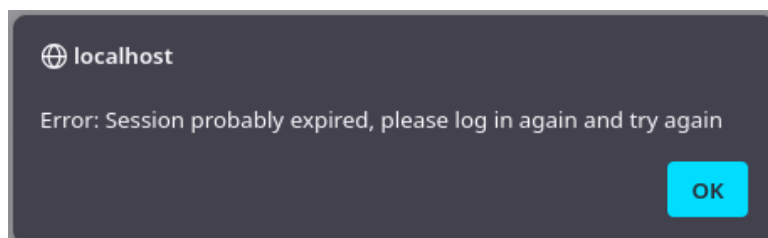
SEARCH RESULT

Name	City	Country	Book
+ Porto Riverside Inn	Porto	Portugal	<input type="button" value="Book"/>

Size	Price Per Night
1	\$110
2	\$140
3	\$180
4	\$220

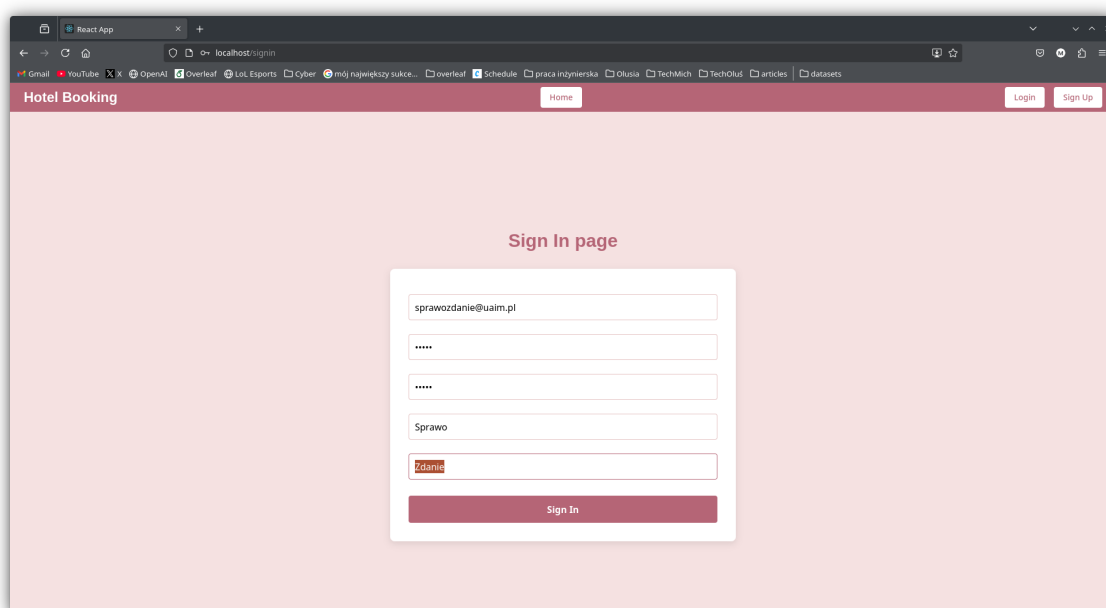
Rys. 3. Przykładowe pokoje w wcześniej znalezionym hotelu

Użytkownik chcąc złożyć rezerwację powinien posiadać konto. Próbuąc złożyć rezerwację zostanie o tym powiadomiony.



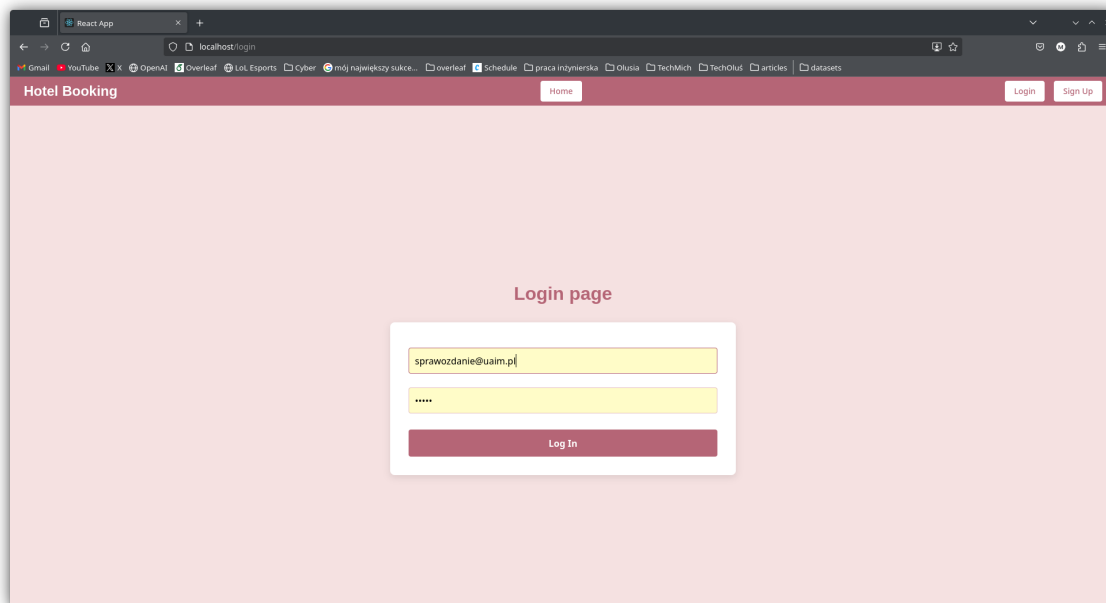
Rys. 4. Ogólny komunikat proszący o zalogowanie się

Przechodzi więc do strony z rejestracją, podaje wszystkie potrzebne dane i się rejestruje:



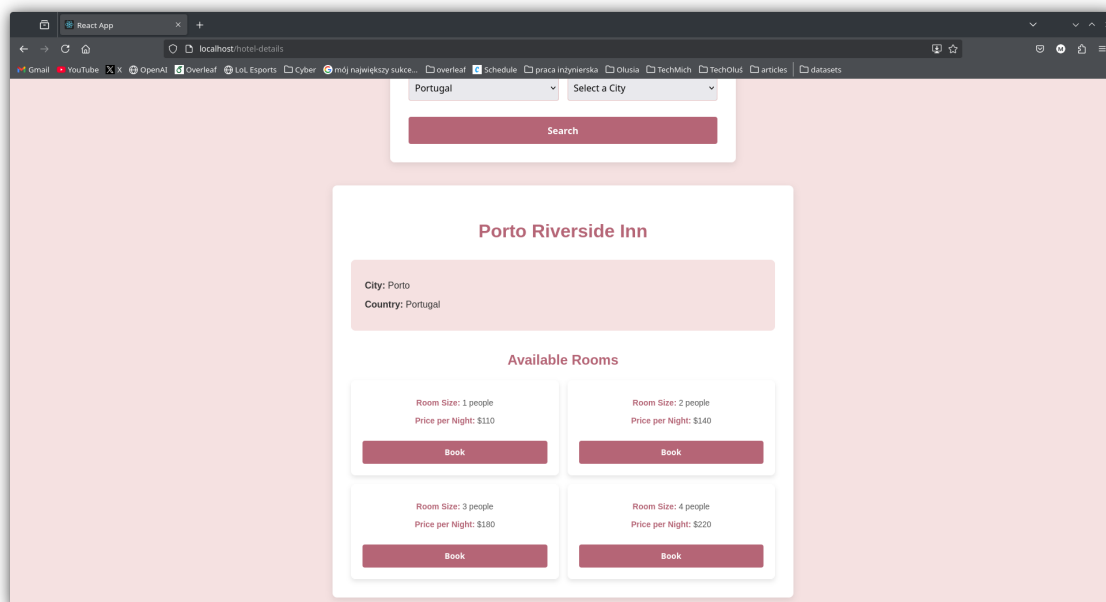
Rys. 5. Strona rejestracji wraz z danymi nowego użytkownika

Następnie przechodzi do strony logowania i loguje się za pomocą utworzonego użytkownika



Rys. 6. Strona logowania wraz z danymi

Teraz użytkownik może dokończyć rezerwację. Po kliknięciu przycisk **Book** obok hotelu przechodzi do wyboru pokoju, a następnie do formularza potwierdzającego rezerwację.



Rys. 7. Wybór pokoju

I na koniec ostateczna weryfikacja danych i potwierdzenie.

The screenshot shows a web browser window with a React App running on localhost:3000. The page displays a booking confirmation form for 'Porto Riverside Inn' in 'Portugal, Porto'. The form includes a 'People' field set to 1, a 'From' date of 12/01/2025, and a 'To' date of 13/01/2025. Below this, there is a section titled 'Available Rooms' with four room options: Room Size: 1 people (Price per Night: \$110), Room Size: 2 people (Price per Night: \$140), Room Size: 3 people (Price per Night: \$180), and Room Size: 4 people (Price per Night: \$220). The first room is marked as 'Selected' with a purple button, while the others have 'Book' buttons. At the bottom of the form is a large 'Book!' button.

Rys. 8. Formularz potwierdzający rezerwację

Po naciśnięciu przycisku **Book!** zostaje on przeniesiony do strony swojego profilu gdzie widzi nowo powstałą rezerwację.

The screenshot shows the 'User Profile' page of the 'Hotel Booking' application. The page displays the user's email as 'sprawozdanie@uaim.pl'. Below the email, there is a section titled 'Bookings' which contains a table with the following data:

Hotel Name	City	Country	From	To	People	Action
Porto Riverside Inn	Porto	Portugal	12/01/2025	13/01/2025	1	Cancel

Rys. 9. Strona profilu wraz z rezerwacją

Widoczny jest też tutaj przycisk **Cancel**, który pozwala anulować rezerwację.

3. Flask Backend

Frontend komunikuje się z Backendem za pomocą zapytań HTTP w celu wymiany informacji o aktualnym stanie hoteli i ich dostępności, a także po to, żeby dodawać nowych użytkowników i rezerwacje.

Na początku omówimy plik konfiguracyjny, który odpowiada za Flaska

```
# config.py
import os

class Config:
    """
    Podstawowa konfiguracja aplikacji.
    """
    SECRET_KEY = "bf81378boqswf9813fb8as129" # Klucz tajny dla sesji
    JWT_SECRET_KEY = "bf3829gfbqobufw13ibf9139o2"
    SQLALCHEMY_DATABASE_URI = "mysql+pymysql://uaimUser:uaimPassword@db:3306/hotel" # URI
    ↪ bazy danych
    SQLALCHEMY_TRACK_MODIFICATIONS = True # Wyłączenie ostrzeżenia o zmianach
    DEBUG = True # Debug domyślnie wyłączony
    TESTING = True # Tryb testowy domyślnie wyłączony
    JWT_TOKEN_LOCATION = ['cookies'] # JWT tylko w ciasteczkach
    JWT_COOKIE_SECURE = False # Włącz HTTPS w środowisku produkcyjnym
    JWT_ACCESS_COOKIE_PATH = '/' # Ścieżka dostępu do ciasteczka
    JWT_COOKIE_CSRF_PROTECT = False
```

Są tutaj zmienne wskazujące na bazę danych oraz konfiguracja obsługi ciasteczek z wykorzystaniem tokenów JWT.

Ten plik wykorzystywany jest podczas inicjalizacji modułu app.

```
# app/__init__.py
from flask import Flask
from flask_migrate import Migrate
from app.routes import init_routes
from app.db import db
from flask_cors import CORS
from flask_jwt_extended import JWTManager
# Inicjalizacja rozszerzeń (globalna, ale bez przypisania do aplikacji)

migrate = Migrate()
jwt = JWTManager()

def create_app(config_class="config.Config"):
    """
    Funkcja fabryczna do tworzenia instancji aplikacji Flask.
    """
    # Tworzenie instancji aplikacji
    app = Flask(__name__)
    app.config.from_object(config_class)

    # Inicjalizacja rozszerzeń z aplikacją
    db.init_app(app)
    migrate.init_app(app, db)
    jwt.init_app(app)
    CORS(app, supports_credentials=True)
    init_routes(app)
    # Dodanie obsługi błędów (opcjonalnie)
    register_error_handlers(app)
```

```

    return app

def register_error_handlers(app):
    """Funkcja rejestrująca obsługę błędów."""
    @app.errorhandler(404)
    def not_found_error():
        return "404: Not Found", 404

    @app.errorhandler(500)
    def internal_error():
        db.session.rollback() # Przywrócenie sesji w przypadku błędu bazy danych
        return "500: Internal Server Error", 500

```

Plik tworzy instancję aplikacji Flask oraz przypisuje jej konfigurację wraz z połączeniem do bazy danych i obsługą ciasteczek.

Sama aplikacja startowana jest w pliku `app/run.py`

```

# app/run.py
from app import create_app

app = create_app()

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=False)

```

Cała aplikacja obsługuje bazę danych za pomocą ORM, a modele są zdefiniowane w `app/models.py`

```

# app/models.py
# coding: utf-8
from dataclasses import dataclass
from datetime import datetime

from app.db import db

@dataclass
class Booker(db.Model):
    __tablename__ = 'Booker'

    ID: int = db.Column(db.Integer, primary_key=True, nullable=False, unique=True,
        ↳ autoincrement=True)
    Email: str = db.Column(db.String(50), primary_key=True, nullable=False, unique=True)
    Password: str = db.Column(db.String(72))
    FirstName: str = db.Column(db.String(20), nullable=False)
    SecondName: str = db.Column(db.String(30), nullable=False)

@dataclass
class Booking(db.Model):
    __tablename__ = 'Booking'

    ID: int = db.Column(db.Integer, primary_key=True, unique=True, autoincrement=True,
        ↳ nullable=False)
    HotelID: int = db.Column(db.ForeignKey('Hotel.ID'), nullable=False, index=True)
    RoomID: int = db.Column(db.ForeignKey('Room.ID'), nullable=False, index=True)
    BookerEmail: str = db.Column(db.ForeignKey('Booker.Email'), nullable=False, index=True)
    From: datetime = db.Column(db.Date, nullable=False)
    To: datetime = db.Column(db.Date, nullable=False)

```

```

Booker = db.relationship('Booker', primaryjoin='Booking.BookerEmail == Booker.Email',
    ↳ backref='bookings')
Hotel = db.relationship('Hotel', primaryjoin='Booking.HotelID == Hotel.ID',
    ↳ backref='bookings')
Room = db.relationship('Room', primaryjoin='Booking.RoomID == Room.ID',
    ↳ backref='bookings')

@dataclass()
class City(db.Model):
    __tablename__ = 'City'

    Name: str = db.Column(db.String(100), primary_key=True, nullable=False)
    CountryName: str = db.Column(db.ForeignKey('Country.Name'), primary_key=True,
    ↳ nullable=False, index=True)

    Country = db.relationship('Country', primaryjoin='City.CountryName == Country.Name',
    ↳ backref='cities')

@dataclass()
class Country(db.Model):
    __tablename__ = 'Country'

    Name: str = db.Column(db.String(100), primary_key=True, unique=True)
    Code: str = db.Column(db.String(2), nullable=False)

@dataclass()
class Hotel(db.Model):
    __tablename__ = 'Hotel'
    __table_args__ = (
        db.ForeignKeyConstraint(['CityName', 'CountryName'], ['City.Name',
        ↳ 'City.CountryName']),
        db.Index('Hotel_fk2', 'CityName', 'CountryName')
    )

    ID: int = db.Column(db.Integer, primary_key=True, unique=True, autoincrement=True,
    ↳ nullable=False)
    Name: str = db.Column(db.String(50), nullable=False)
    CountryName: str = db.Column(db.String(100), nullable=False)
    CityName: str = db.Column(db.String(100), nullable=False)

    City = db.relationship('City', primaryjoin='and_(Hotel.CityName == City.Name,
    ↳ Hotel.CountryName == City.CountryName)', backref='hotels')

@dataclass()
class Room(db.Model):
    __tablename__ = 'Room'

    ID: int = db.Column(db.Integer, primary_key=True, unique=True, autoincrement=True,
    ↳ nullable=False)
    Size: int = db.Column(db.Integer, nullable=False)
    HotelID: int = db.Column(db.ForeignKey('Hotel.ID'), nullable=False, index=True)

```

```

PricePerNight: int = db.Column(db.Integer, nullable=False)

Hotel = db.relationship('Hotel', primaryjoin='Room.HotelID == Hotel.ID',
↳ backref='rooms')

```

3.1. Ścieżki udostępniane przez aplikację

Ścieżki, które są wykorzystywane przez Frontend znajdują się w *app/routes.py*. Omówimy teraz każdą z metod.

Poniższa metoda służy do rejestracji użytkownika. Przyjmuje ona zapytania POST. Hashuje też hasło za pomocą bcrypta i dodaje użytkownika do bazy danych.

```

@app.route('/api/register-user', methods=['POST'])
def register_user():
    body = request.get_json()
    email = body['email']
    password = body['password'].encode()
    first_name = body['firstName']
    second_name = body['secondName']

    password = bcrypt.hashpw(password, bcrypt.gensalt())

    new_booker = Booker(Email=email, Password=password, FirstName=first_name,
↳ SecondName=second_name)

    db.session.add(new_booker)
    db.session.commit()

    return jsonify({"Success": "New Booker registered successfully!"}), 200

```

Następna metoda służy do logowania użytkownika. Email jest kluczem głównym więc najpierw sprawdza czy użytkownik istnieje, a potem porównuje hashe hasła.

```

@app.route('/api/login', methods=['POST'])
def login():
    body = request.get_json()
    email = body['email']
    password = body['password'].encode()

    booker = Booker.query.filter_by(Email=email).first()
    if not booker:
        return jsonify({"Error": "Email or password not correct!"}), 400

    if bcrypt.checkpw(password, booker.Password.encode()):
        token = create_access_token(identity=email)
        response = jsonify({"Success": "Logged in successfully!"})
        set_access_cookies(response, token)
        return response, 200

    return jsonify({"Error": "Email or password not correct!"}), 400

```

Następna metoda zwraca kraje i miasta używane przez formularz do wyszukiwania hoteli. Zwracane są tylko zwracane te kraje i miasta, w których posiadany jest hotel.

```

@app.route("/api/get-cities-and-countries", methods=['GET'])
def get_cities():
    hotels = Hotel.query.all()

    country_cities = {}

```

```

for hotel in hotels:
    city = hotel.CityName
    country = hotel.CountryName
    if country not in country_cities:
        country_cities[country] = []
        country_cities[country].append(city)
    else:
        country_cities[country].append(city)

return jsonify(country_cities)

```

Metoda służy do pobierania hoteli z bazy danych. Są obsługiwane dwa scenariusze:

1. GET: zwraca wszystkie hotele wraz z hotelami.
2. POST: zwraca hotele na podstawie danych przekazanych w ciele zapytania wraz z dostępnymi pokojami, które odpowiadają filtrom.

```

@app.route("/api/get-hotels", methods=['GET', 'POST'])
def get_hotels():

    if request.method == "GET":
        hotels = Hotel.query.all()
        hotels_rooms = []
        for hotel in hotels:
            rooms = hotel.rooms
            available_rooms = []
            for room in rooms:
                available_rooms.append(room)
            if len(available_rooms) != 0:
                hotel = asdict(hotel)
                hotel['available_rooms'] = available_rooms
                hotels_rooms.append(hotel)
        return jsonify(hotels_rooms)

    elif request.method == "POST" and request.get_json() != {}:
        filters = request.get_json()

        query = Hotel.query

        if "name" in filters and filters["name"] != '':
            query = query.filter(Hotel.Name.ilike(f"%{filters['name']}%"))
        if "city" in filters and filters["city"] != '':
            query = query.filter(Hotel.CityName == filters["city"])
        if "country" in filters and filters["country"] != '':
            query = query.filter(Hotel.CountryName == filters["country"])
        if "id" in filters and filters["id"] != '':
            query = query.filter(Hotel.ID == filters["id"])

        hotels = query.all()
        hotels_rooms = []
        for hotel in hotels:
            rooms = hotel.rooms
            available_rooms = []
            for room in rooms:
                bookings = room.bookings
                add = True
                for booking in bookings:

```

```

        if booking.From <= datetime.strptime(filters["to"],
        ↪ '%Y-%m-%d').date() or booking.To >=
        ↪ datetime.strptime(filters["from"], '%Y-%m-%d').date():
            add = False
            break
        if room.Size < int(filters["size"]):
            add = False
        if add:
            available_rooms.append(room)
    if len(available_rooms) != 0:
        hotel = asdict(hotel)
        hotel['available_rooms'] = available_rooms
        hotels_rooms.append(hotel)
    else:
        hotel = asdict(hotel)
        hotel['available_rooms'] = []
        hotels_rooms.append(hotel)

    return jsonify(hotels_rooms)

elif request.get_json() == {}:
    return jsonify({"Reload": "paged refreshed"}), 489

```

Ścieżka służąca do tworzenia rezerwacji wymaga aktualnej sesji, na co wskazuje @jwt_required(). Dodaje nową rezerwację do bazy danych, po wcześniejszym zweryfikowaniu informacji.

```

@app.route("/api/make-reservation", methods=['POST'])
@jwt_required()
def make_reservation():
    data = request.get_json()
    email = get_jwt_identity()
    room_id = data.get("room_id")
    hotel_id = data.get("hotel_id")
    from_date = datetime.strptime(data.get("from"), "%Y-%m-%d")
    to_date = datetime.strptime(data.get("to"), "%Y-%m-%d")
    if not all([room_id, hotel_id, from_date, to_date]):
        return jsonify({"Error": "Missing data!"}), 400
    if from_date >= to_date:
        return jsonify({"Error": "From date cannot be greater than to date!"}), 400

    reservation = Booking(HotelID=hotel_id, RoomID=room_id, From=from_date, To=to_date,
    ↪ BookerEmail=email)
    db.session.add(reservation)
    db.session.commit()

    return jsonify({"Success": "Booking successfully added"}), 200

```

Funkcja pomocnicza, sprawdza, czy token jest aktualny.

```

@app.route("/api/check-token", methods=['GET'])
@jwt_required()
def check_token():
    return jsonify({"isLoggedIn": True}), 200

```

Funkcja do wylogowywania użytkownika - unieważnia token sesji.

```

@app.route("/api/logout", methods=['POST'])
def logout():
    resp = jsonify({'logout': True})

```

```
unset_jwt_cookies(resp)
return resp, 200
```

Funkcja zbiera dane o zalogowanym użytkowniku - wykorzystywane przez stronę profilu. Zwraca też rezerwacje danego użytkownika.

```
@app.route("/api/get-profile-data", methods=['GET'])
@jwt_required()
def get_profile():
    email = get_jwt_identity()
    bookings = Booking.query.filter_by(BookerEmail=email).all()

    data = {"email": email, "bookings": []}
    for booking in bookings:
        hotel_name = booking.Hotel.Name
        country = booking.Hotel.CountryName
        city = booking.Hotel.CityName
        from_date = booking.From
        to_date = booking.To
        people = booking.Room.Size
        booking_id = booking.ID
        to_add = {
            "bookingID": booking_id,
            "hotelName": hotel_name,
            "country": country,
            "city": city,
            "fromDate": from_date,
            "toDate": to_date,
            "people": people
        }
        data["bookings"].append(to_add)

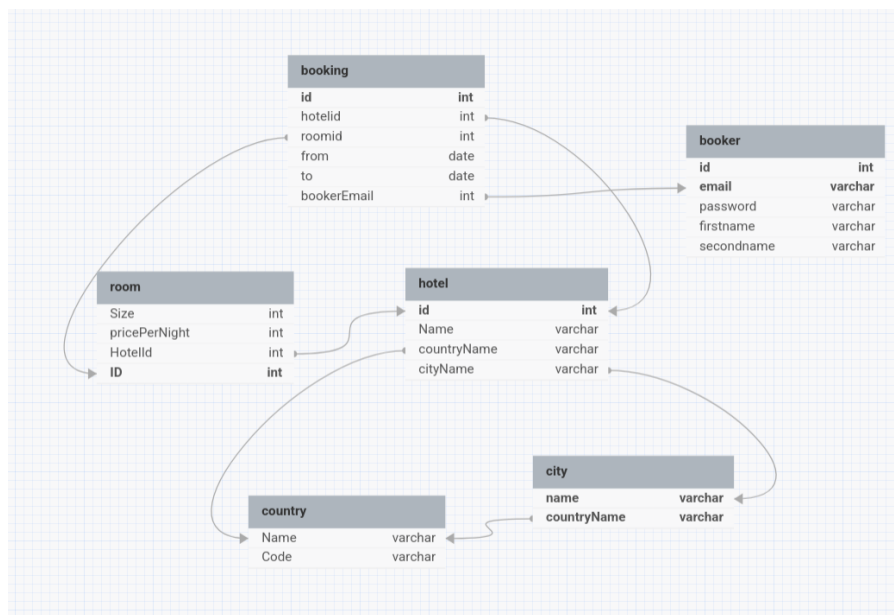
    return jsonify(data)
```

Funkcja, która usuwa rezerwacje danego użytkownika - wymaga zalogowania.

```
@app.route("/api/cancel-booking", methods=['POST'])
@jwt_required()
def cancel_booking():
    data = request.get_json()
    booking = data.get("booking")
    booking_id = booking.get("bookingID")
    print(data)
    email = get_jwt_identity()
    booking_cancel = Booking.query.filter_by(BookerEmail=email, ID=booking_id).first()
    if booking_cancel:
        db.session.delete(booking_cancel)
        db.session.commit()
        return jsonify({"Success": "Booking cancelled"}), 200
    return jsonify({"Error": "Booking not found"}), 404
```

4. Baza danych

Baza danych wykorzystuje MySQL, a jej schemat i relacje są przedstawione poniżej:



Rys. 10. Schemat bazy danych

Baza danych zawiera:

- Country - tabela zawierająca kraje. Ważna jest tylko nazwa.
- City - tabela miast, posiada nawiązanie do kraju, w którym się znajduje
- Hotel - tabela zawierająca informacje o hotelach. Są informacje o miejscu, w którym się znajduje
- Room - tabela opisująca pokój posiada informacje o cenie, wielkości i nawiązuje do hotelu, w którym się znajduje
- Booker - tabela użytkowników, posiada podstawowe informacje takie jak email, imię i nazwisko. Posiada też zahashowane hasła
- Booking - tabela łącząca informacje rozpoczęciu i zakończeniu, a także jaki pokój, w jakim hotelu oraz przez kogo została ona złożona

W bazie danych został też utworzony użytkownik uaimUser, za pomocą którego komunikuje się Flask Backend z bazą danych.

5. Aplikacja Android

Aplikacja została stworzona jako projekt w Android Studio, bazując na wcześniejszym projekcie frontendowym opartym na npm. Struktura aplikacji obejmuje sześć aktywności, które zapewniają użytkownikowi możliwość rejestracji, logowania, wyszukiwania hoteli oraz zarządzania rezerwacjami.

Po uruchomieniu aplikacji użytkownik trafia na ekran główny *HomeActivity*, który umożliwia przeszukiwanie dostępnych hoteli. Z tego miejsca użytkownik może zostać przekierowany do dwóch innych aktywności: ekranu logowania *LoginActivity* lub rejestracji *SignUpActivity*. Aktywność *SignUpActivity* pozwala na utworzenie nowego konta użytkownika, natomiast *LoginActivity* umożliwia zalogowanie się do aplikacji.

Po zalogowaniu użytkownik zostaje przeniesiony na aktywność *LoggedInActivity*, która jest funkcjonalnie zbliżona do *HomeActivity*, ale dodatkowo pozwala na dokonywanie rezerwacji hoteli. W *LoggedInActivity* dostępne są również dwa kluczowe przekierowania: *Logout*, który wylogowuje użytkownika i przenosi go z powrotem na ekran główny, oraz *ProfileActivity*. Aktywność *ProfileActivity* umożliwia użytkownikowi przeglądanie jego wcześniejszych rezerwacji oraz ich anulowanie, jeśli zajdzie taka potrzeba.

Aplikacja została zaprojektowana w sposób intuicyjny i zapewnia płynne przejścia między ekranami, wykorzystując podejście modularne w implementacji funkcji. Dzięki integracji z wcześniejszym projektem frontendowym aplikacja jest spójna pod względem wyglądu i funkcjonalności, co znacząco poprawia doświadczenie użytkownika.

5.1. Drzewo aplikacji i przygotowanie działania w xml

5.1.1. AndroidManifest.xml

Plik **AndroidManifest.xml** pełni kluczową rolę w konfiguracji aplikacji. Zawiera deklaracje wszystkich aktywności aplikacji, a także wymagane uprawnienia, takie jak dostęp do internetu. Dodatkowo, wprowadzone zostały zmiany związane z powiązaniem aplikacji z plikiem *network_security_config.xml*, co pozwala na obsługę połączeń sieciowych.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:networkSecurityConfig="@xml/network_security_config"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="BookingHotel"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar"
        tools:targetApi="31">
        <activity android:name=".HomeActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".LoginActivity" />
        <activity android:name=".SignUpActivity" />
        <activity android:name=".LoggedinActivity" />
        <activity android:name=".BookingActivity" />
        <activity android:name=".ProfileActivity" />
    </application>
</manifest>
```

Rys. 11. Zawartość pliku AndroidManifest.xml

5.1.2. network_security_config.xml

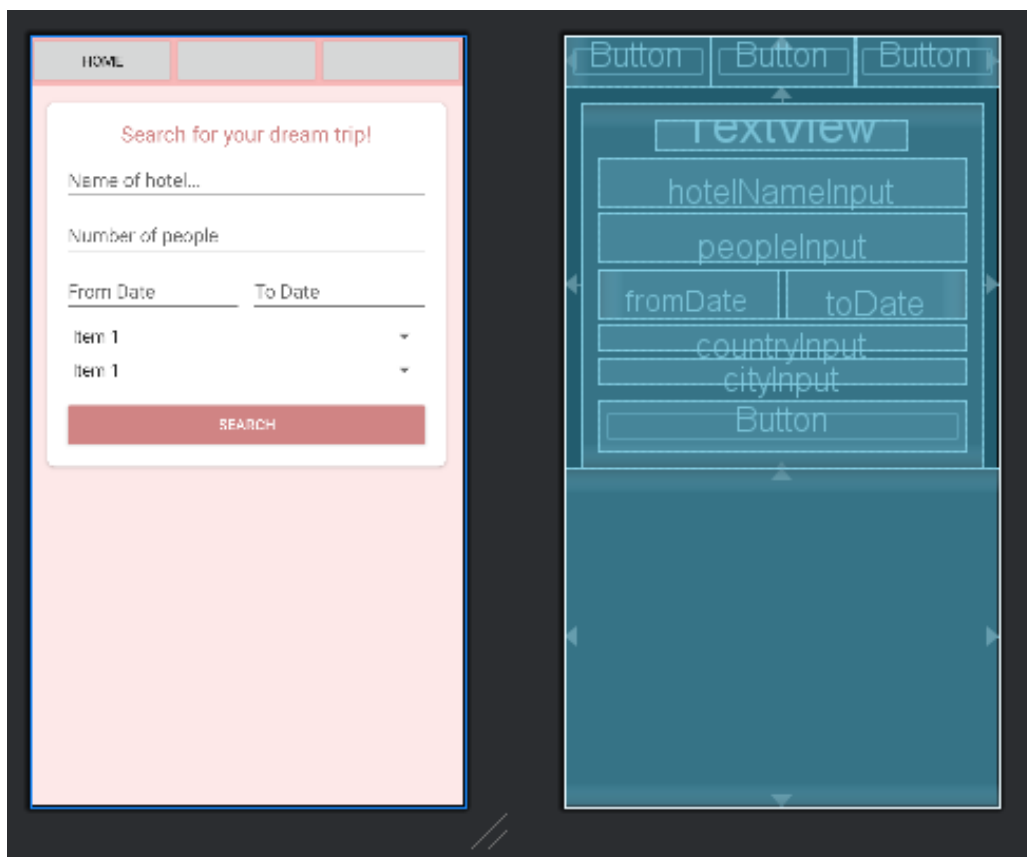
Ten plik został utworzony w celu zdefiniowania niestandardowych zasad bezpieczeństwa sieciowego. Pozwala on na połączenia z lokalnymi serwerami w konfiguracji CLEARTEXT, co jest istotne w przypadku korzystania z testowych środowisk serwerowych.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain includeSubdomains="true">192.168.8.100</domain>
    <domain includeSubdomains="true">10.0.2.2</domain>
  </domain-config>
</network-security-config>
```

Rys. 12. Zawartość pliku network_security_config.xml

5.2. Przygotowanie layoutów

Aplikacja korzysta z kilku plików układu XML, które definiują wygląd interfejsu użytkownika dla poszczególnych ekranów. Przykładem jest plik **activity_home.xml**, który określa strukturę i elementy widoczne na ekranie głównym aplikacji. Zawiera on między innymi pole wyszukiwania hoteli, listę wyników oraz przyciski nawigacyjne.



Rys. 13. Render pliku activity_home.xml

5.3. Klasy użytkowe

Aplikacja korzysta z kilku klas narzędziowych, które wspierają jej funkcjonowanie:

5.3.1. BookingsAdapter

Klasa **BookingsAdapter.kt** odpowiada za zarządzanie listą rezerwacji użytkownika, zapewniając odpowiednie dopasowanie danych do elementów interfejsu użytkownika. Implementuje wzorzec ViewHolder w celu optymalizacji wydajności wyświetlania listy.

```
override fun onBindViewHolder(holder: BookingsViewHolder, position: Int) {
    val booking = bookings[position]
    holder.hotelNameText.text = booking.hotelName
    holder.cityCountryText.text = String.format("${booking.city}, ${booking.country}")
    holder.dateRangeText.text = String.format("Dates: ${booking.fromDate} - ${booking.toDate}")
    holder.peopleText.text = String.format("People: ${booking.people}")

    holder.cancelButton.setOnClickListener {
        AlertDialog.Builder(holder.itemView.context)
            .setTitle("Cancel Booking")
            .setMessage("Are you sure you want to cancel this booking?")
            .setPositiveButton(text: "Yes") { dialog, _ ->
                dialog.dismiss()

                val url = "http://$ip:5000/api/cancel-booking"

                val rawData = "{\"booking\": {\"bookingID\":${booking.bookingID}\n" +
                    "}"

                NetworkUtils.makeCancelPostRequest(url, rawData) { response ->
                    if (response != null && response.isSuccessful) {
                        val responseBody = response.body?.string()
                        if (responseBody?.contains(other: "\"Success\"") == true) {
                            Handler(Looper.getMainLooper()).post {
                                bookings.removeAt(position)
                                notifyItemRemoved(position)
                                notifyItemRangeChanged(position, bookings.size)
                            }
                        } else {
                            Log.e(tag: "BookingsAdapter", msg: "Failed to cancel booking: $responseBody")
                        }
                    } else {
                        Log.e(tag: "BookingsAdapter", msg: "Failed to cancel booking: $response")
                    }
                }
            }
    }
}
```

Rys. 14. Fragment kodu klasy BookingsAdapter

5.3.2. NetworkUtils

Klasa **NetworkUtils.kt** dostarcza narzędzi do wykonywania operacji sieciowych, takich jak wysyłanie żądań HTTP różnego rodzaju. Dzięki niej aplikacja może komunikować się z serwerem w sposób bezpieczny i niezawodny.

```
fun makeGetRequest(url: String, callback: (String?) -> Unit) {
    val request = Request.Builder().url(url).build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            Log.e(tag: "NetworkUtils", msg: "GET request failed", e)
            callback(null)
        }

        override fun onResponse(call: Call, response: Response) {
            if (response.isSuccessful) {
                callback(response.body?.string())
            } else {
                callback(null)
            }
        }
    })
}
```

Rys. 15. Przykładowa funkcja z klasy NetworkUtils

5.3.3. CookieManager

Klasa **CookieManager.kt** zarządza ciasteczkami w aplikacji, umożliwiając obsługę sesji użytkownika. Dzięki niej aplikacja może przechowywać i odczytywać dane sesyjne, co pozwala na zachowanie stanu zalogowania użytkownika.

```
class CookieManager : CookieJar {
    private val cookieStore = mutableMapOf<String, MutableList<Cookie>>()

    override fun saveFromResponse(url: HttpUrl, cookies: List<Cookie>) {
        cookieStore[url.host] = cookies.toMutableList()
    }

    override fun loadForRequest(url: HttpUrl): List<Cookie> {
        return cookieStore[url.host] ?: emptyList()
    }
}
```

Rys. 16. Fragment kodu klasy CookieManager

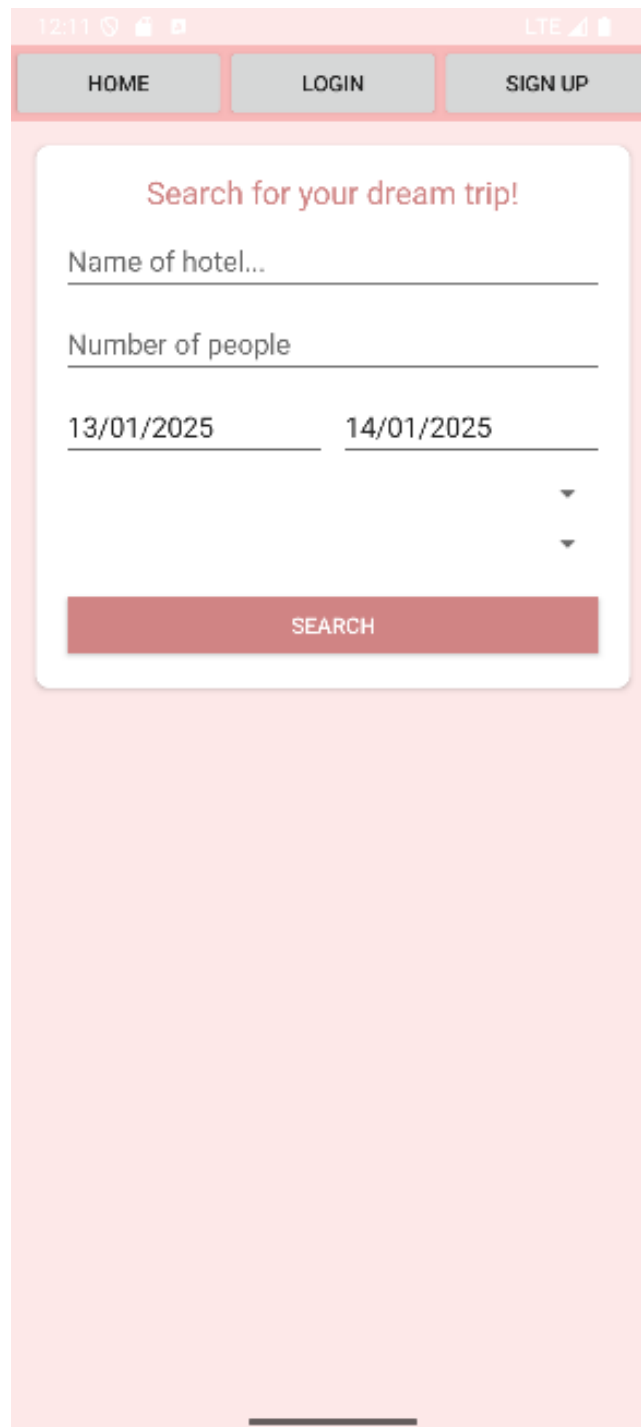
5.3.4. Modele danych: **Hotel.kt** i **ProfileResponse.kt**

Klasy **Hotel.kt** oraz **ProfileResponse.kt** reprezentują obiekty modelu danych używane w aplikacji:

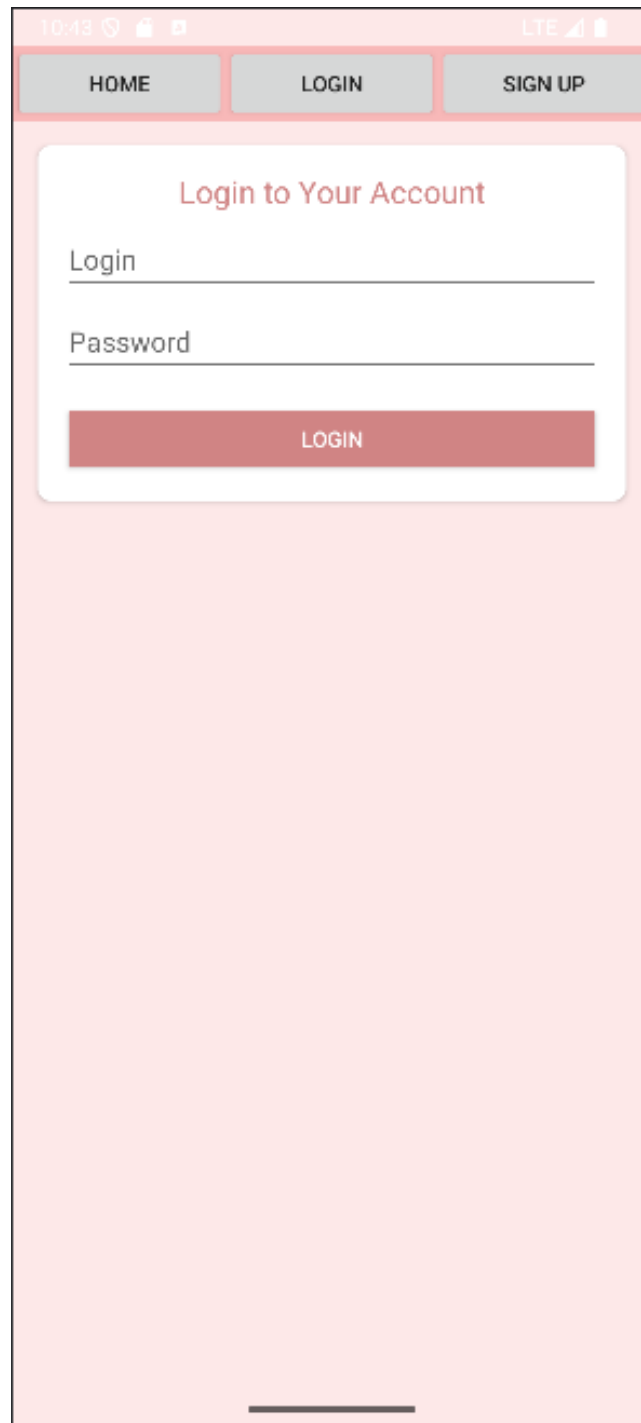
- **Hotel.kt**
Klasa zawiera informacje o hotelu, takie jak jego nazwa, lokalizacja oraz dostępność.
- **ProfileResponse.kt**
Klasa reprezentuje odpowiedź serwera dotyczącą danych użytkownika, takich jak szczegóły profilu oraz lista dokonanych rezerwacji.

5.4. Działanie aplikacji

Po starcie aplikacji użytkownik zostaje od razu przeniesiony do ekranu głównego - HomeActivity. W tym widoku dostępne jest wyszukiwanie dostępnych hoteli, możliwość wprowadzania parametrów jest analogiczna do aplikacji przeglądarkowej. Możemy więc przeglądać hotele, ale przycisk rezerwacji przekieruje użytkownika do ekranu logowania. Przejście do ekranu logowania jest również dostępne z górnego panelu nawigacji, który umożliwia także przejście do ekranu rejestracji. To jedyne czynności dostępne dla niezalogowanych użytkowników.



Rys. 17. Wygląd ekranu HomeActivity



The image shows a mobile application interface for a login screen. At the top, there is a status bar with the time 10:43, a heart icon, a battery icon, and the text 'LTE'. Below the status bar is a navigation bar with three buttons: 'HOME', 'LOGIN', and 'SIGN UP'. The main content area features a white card with the title 'Login to Your Account' in red. Inside the card, there are two input fields: 'Login' and 'Password', both with red text and red borders. Below the input fields is a red button with the text 'LOGIN' in white. The background of the screen is a light pink color. At the bottom of the screen, there is a thin horizontal line representing the home indicator bar.

Rys. 18. Wygląd ekranu LoginActivity

10:44 LTE

HOME LOGIN SIGN UP

Create Your Account

Email

Password

Repeat Password

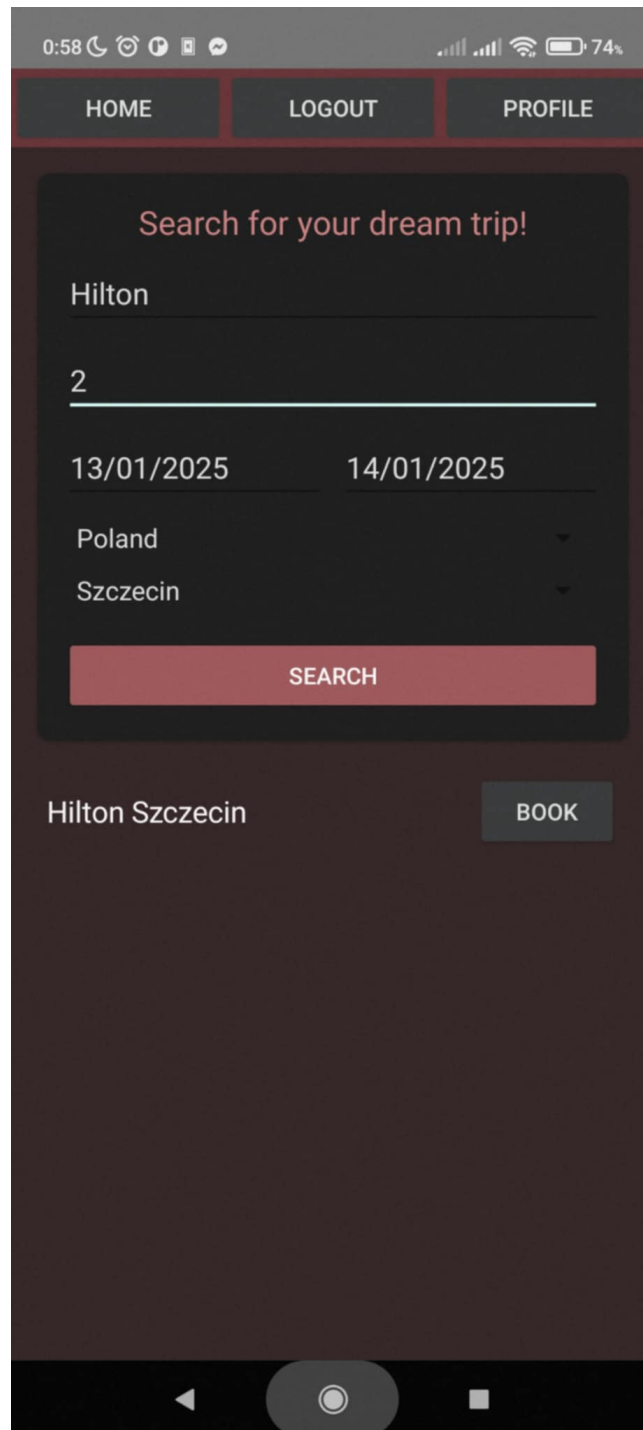
First Name

Second Name

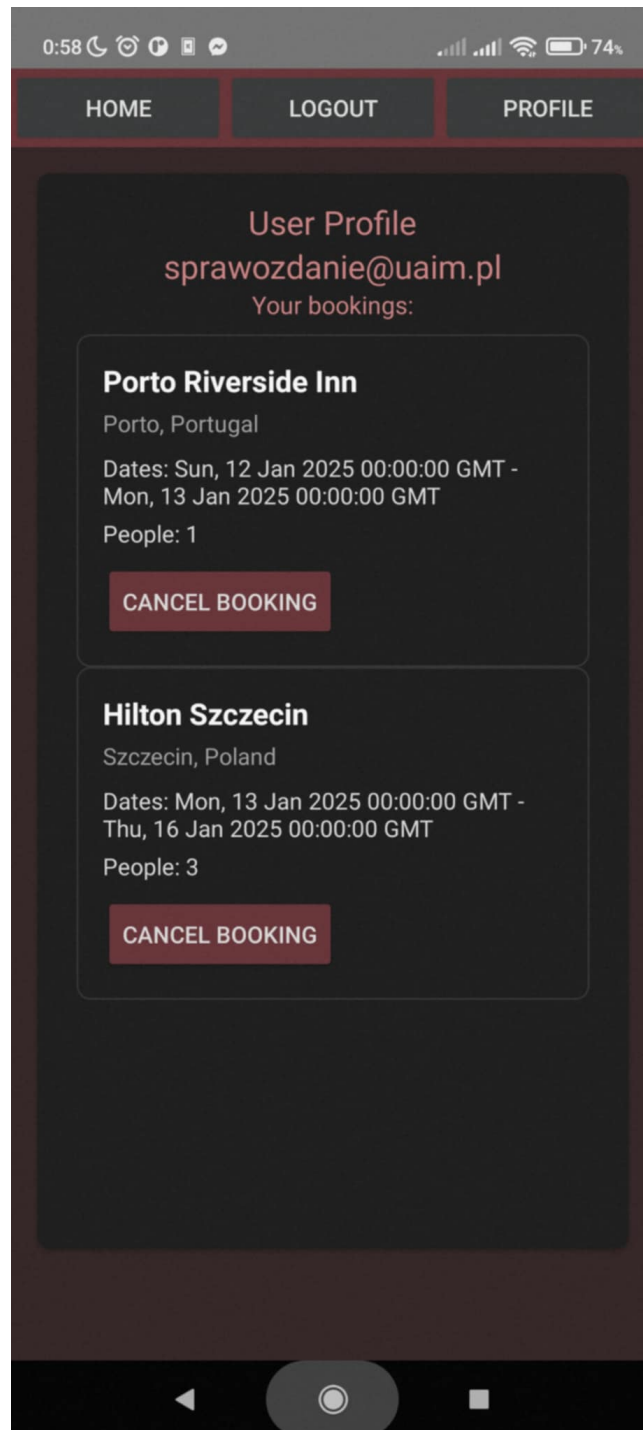
SIGN UP

Rys. 19. Wygląd ekranu SignUpActivity





Zalogowany użytkownik zostaje przekierowany do LoggedinActivity, tutaj analogicznie do HomeActivity może przeglądać hotele. Przyciski Book umożliwiają jednak przekierowanie do rezerwacji pokoju, w tym widoku przedstawiane są informacje o wybranym hotelu oraz możliwość wyboru pokoju i potwierdzenia rejestracji. Oprócz tworzenia nowych rezerwacji użytkownik ma także możliwość anulowania rezerwacji. Jest to dostępne w zakładce Profile, w której wyświetlane są wszystkie dokonane rezerwacje danego użytkownika. Do każdej z nich dołączony jest przycisk anulowania, który po wciśnięciu wyświetla pop-up proszący o potwierdzenie. Jeżeli użytkownik się na to zdecyduje, to wysyłany jest request do backendu, a widok odświeżany. Użytkownik ma także możliwość wylogowania w dowolnym momencie.



Rys. 20. Wygląd ekranu LoggedInActivity



Rys. 21. Wygląd ekranu ProfileActivity

0:58    

HOME LOGOUT PROFILE

Fill the form to complete the booking
Hilton Szczecin
Szczecin, Poland

3

13/01/2025 16/01/2025

Available Rooms

Room size: 3	Price per night: \$150	<input checked="" type="checkbox"/>
Room size: 4	Price per night: \$200	<input type="checkbox"/>

BOOK!

Rys. 22. Wygląd ekranu BookingActivity

6. Konteneryzacja

Całe rozwiązanie (poza aplikacją Android) zostało skonteneryzowane za pomocą docker i docker compose.

6.1. Frontend

Frontend wykorzystuje dwa obrazy node i nginx w celu hostowania zbudowanej aplikacji React.

```
# Budowanie React-a
FROM node:16 AS builder

WORKDIR /app

COPY package.json package-lock.json ./
RUN npm install

COPY . .
RUN npm run build

# Serwowanie statycznych plików za pomocą Nginx
FROM nginx:alpine

COPY --from=builder /app/build /usr/share/nginx/html

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Aplikacja jest budowana, a następnie przekazywana do nginxa, który udostępnia pliki do zewnątrz przez port 80.

6.2. Backend

Plik Dockerfile backendu pobiera wszystkie potrzebne zależności oraz startuje aplikację Flask

```
FROM python:3.9-slim

# Install dependencies
COPY requirements.txt /app/
WORKDIR /app
RUN pip install -r requirements.txt

# Skopiowanie plików aplikacji
COPY . /app

# Ustawienie zmiennej środowiskowej PYTHONPATH
ENV PYTHONPATH=/app

EXPOSE 5000

# Uruchomienie aplikacji
CMD ["python", "app/run.py"]
```

6.3. Baza danych

Baza danych wykorzystuje gotowy obraz i nie posiada pliku Dockerfile, konfiguracja przebiega podczas wywołania docker compose.

6.4. docker compose

Plik docker compose zawiera instrukcje, które stawiają wszystkie kontenery, dołącza je do sieci, udostępnia porty, konfiguruje zmienne środowiskowe oraz nadaje adresy ip w sieci.

```
services:
  db:
    image: mysql:latest
    container_name: hotel_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: hotel
      MYSQL_USER: uaimUser
      MYSQL_PASSWORD: uaimPassword
    volumes:
      - db_data:/var/lib/mysql
      - ./database/db.sql:/docker-entrypoint-initdb.d/db.sql
    networks:
      hotel_network:
        ipv4_address: 172.20.0.60

  backend:
    build:
      context: ./backend
    container_name: hotel_backend
    environment:
      FLASK_APP: app.run:app
      FLASK_ENV: development
      DB_HOST: db
      DB_NAME: hotel
      DB_USER: uaimUser
      DB_PASS: uaimPassword
    depends_on:
      - db
    ports:
      - "5000:5000"
    networks:
      hotel_network:
        ipv4_address: 172.20.0.40

  frontend:
    build:
      context: ./spa
    container_name: hotel_frontend
    ports:
      - "80:80"
    networks:
      hotel_network:
        ipv4_address: 172.20.0.20
    depends_on:
      - backend

networks:
  hotel_network:
    ipam:
      config:
```

- `subnet`: 172.20.0.0/24

```
volumes:  
  db_data:  
    driver: local
```

Szczególną uwagę zwróćmy na część odpowiedzialną za bazę danych. Wypełniana jest ona początkowym stanem bazy danych, oraz ustawia użytkownika wykorzystywanego przez Flask.

7. Podsumowanie

Projekt nauczył nas bardzo wiele. Jego obszar zahaczał o wiele dziedzin i wymagał wszechstronnej wiedzy. Pochłoniął dużo czasu, a i tak nie udało się osiągnąć wszystkich wyznaczonych celów. Jednak nie był to czas