Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Programowanie aplikacji internetowych

Dokumentacja projektu
Aplikacja do planowania podróży

Anna Muszyńska 318547
Jakub Walasek 325330
Michał Sokala 325323
Mikołaj Smolarek 325322
Patryk Jankowski 325280
Zoja Hordyńska 324865

Warszawa, 2025

Spis treści

1. Temat projektu

Tematem projektu jest aplikacja umożliwiająca szczegółowe planowanie oraz organizację podróży. Przeznaczona jest zarówno dla osób podróżujących w celach prywatnych lub zawodowych, jak i dla grup organizujących wspólne wyjazdy. Aplikacja wspomaga również zarządzanie kosztami, umożliwiając podział kosztów w przypadku wielu uczestników podróży, a także dokładne śledzenie wydatków związanych z wyjazdem.

2. Zespół i role

Michał Sokala
Rola: Kierownik projektu, programista backend i bazy danych
Zadania:
— Stworzenie i implementacja modelu bazy danych,
— Implementacja logiki biznesowej,
— Integracja i komunikacja pomiędzy frontendem i backendem,
— Nadzór nad harmonogramem i postępem prac.

Patryk Jankowski
Rola: Programista backend
Zadania:
— Implementacja logiki biznesowej,
— Stworzenie panelu administratora,
— Konteneryzacja aplikacji i jej instalacja.

Mikołaj Smolarek
Rola: Programista frontend i backend
Zadania:
— Stworzenie szablonu projektu,
— Integracja i komunikacja pomiędzy frontendem i backendem,
— Stworzenie komponentów związanych z frontendem.

Zoja Hordyńska
Rola: Programistka backend i bazy danych
Zadania:
— Stworzenie i implementacja modelu bazy danych,
— Implementacja logiki biznesowej i obsługa API,
— Przygotowanie dokumentacji technicznej.

Anna Muszyńska
Rola: Programistka frontend
Zadania:
— Dokumentacja funkcjonalna aplikacji,
— Implementacja interfejsu użytkownika (dodawanie podróży, zarządzanie kosztami).

Jakub Walasek
Rola: Programista frontend
Zadania:
— Projekt i realizacja interfejsu użytkownika do planowania podróży,
— Implementacja funkcji interfejsu, które korzystają z informacji z bazy danych.

3. Założenia funkcjonalne

Przegląd podróży
— Dodawanie podróży: możliwość definiowania nowej podróży poprzez wpisanie nazwy (np. „Wycieczka do Rzymu"), daty, oraz lokalizacji.
— Kalendarz: zaplanowane podróże widoczne będą w dostępnym kalendarzu.

Plan podróży
— Etapy podróży: podział podróży na miejscowości oraz wybór środków transportu na poszczególne etapy.
— Lista miejsc do odwiedzenia: możliwość ręcznego wpisywania planowanych miejsc do odwiedzenia w trakcie każdego etapu podróży.
— Pogoda: prognoza pogody dla poszczególnych lokalizacji w określonych dniach.

Zarządzanie kosztami
— Możliwość ręcznego wpisywania kosztów w różnych kategoriach (np. zakwaterowanie, transport, rozrywka itd.).
— Dodawanie informacji o kwotach związanych z podróżą i umożliwienie podziału kosztów między jej uczestników.
— Możliwość generowania raportu kosztów w postaci wykresu kołowego z podziałem na poszczególne kategorie wydatków.

Grupy wyjazdowe
— Możliwość dodania wielu uczestników do podróży i tworzenia grup wyjazdowych.
— Określenie kosztów ponoszonych przez każdą osobę.

Funkcje dodatkowe
— Lista rzeczy do zabrania: ręczne tworzenie checklisty rzeczy do zabrania.
— Konwertowanie cen: automatyczna konwersja cen na wybraną walutę.

4. Założenia architektoniczne

4.1. Podstawowa architektura aplikacji

Aplikacja będzie działać w modelu klient-serwer z podziałem na dwie główne warstwy:
— Frontend – interfejs użytkownika odpowiedzialny za prezentację danych i interakcję z użytkownikiem.
— Backend – logika aplikacji, zarządzanie danymi oraz integracja z zewnętrznymi usługami, takimi jak prognozy pogody.

— Baza danych – przechowywanie danych aplikacji, w tym informacji o użytkownikach, podróżach, kosztach, etapach podróży, grupach wyjazdowych.

4.2. Frontend

— Cel: Zapewnienie responsywnego interfejsu użytkownika, który umożliwia dodawanie, edytowanie i przeglądanie danych podróży, zarządzanie kosztami i planowanie etapów podróży.
— Komunikacja z backendem: Frontend będzie komunikował się z backendem za pomocą REST API.

4.3. Backend

— Cel: Backend będzie odpowiedzialny za przetwarzanie logiki biznesowej aplikacji, w tym zarządzanie danymi użytkowników, podróży, kosztów i etapów podróży. Będzie umożliwiał uwierzytelnianie użytkowników i zarządzanie sesjami. Będzie także integrować zewnętrzne usługi, takie jak prognozy pogody.
— API: Backend będzie implementować RESTful API, które będzie służyć do obsługi żądań z frontend.
— Logika biznesowa: Backend zarządza danymi o podróżach, kosztach, etapach podróży, a także grupach wyjazdowych. Będzie wykorzystywał API w celu pobierania informacji o pogodzie, a także będzie zaimplementowana logika związana z podziałem kosztów na kategorie.

4.4. Baza danych

— Cel: Baza danych będzie służyć do przechowywania danych aplikacji, takich jak informacje o użytkownikach, podróżach, kosztach, etapach podróży i grupach wyjazdowych.
— Model danych będzie obejmował również powiązania między uczestnikami, podróżami, grupami wyjazdowymi itd.
— Komunikacja z backendem: wykorzystanie ORM, aby zarządzać danymi w bazie, co zapewni łatwą integrację między modelem danych a aplikacją.

5. Narzędzia programowe

Poniżej przedstawiono opis wykorzystania poszczególnych narzędzi programowych w powiązaniu z założeniami funkcjonalnymi i architektonicznymi.

5.1. Frontend - React

— Interfejs użytkownika i interaktywność: React jest dobrym wyborem do budowy dynamicznych, interaktywnych aplikacji frontendowych. Pozwala na efektywne renderowanie UI w odpowiedzi na zmiany w danych, co jest kluczowe dla aplikacji zarządzającej podróżami.
— Komunikacja z backendem: React integruje się z API backendu opartego na Django. Używając Axios, frontend w React będzie w stanie efektywnie komunikować się z backendem, przesyłając dane o podróżach, kosztach, grupach i etapach podróży.

— Przegląd i dodawanie nowych podróży: Dzięki React, interfejs użytkownika może zapewnić dynamiczną obsługę formularzy do dodawania podróży z jednoczesną walidacją danych.
— Zarządzanie kosztami: Frontend w React może zapewnić interaktywną obsługę formularzy do dodawania i edytowania kosztów w różnych kategoriach. Można użyć konkretnych komponentów do wyświetlania wykresów kołowych, które prezentują podział kosztów.

5.2. Backend - Django

— API i logika biznesowa: Django pozwala na szybkie tworzenie skalowalnych aplikacji backendowych z wbudowanym wsparciem dla REST API (dzięki Django REST Framework). Obsługuje również logiczną strukturę danych, którą łatwo zintegrować z frontendem.
— Zarządzanie podróżami: Dodawanie nowych podróży będzie możliwe dzięki wbudowanej funkcjonalności modelowania danych w Django, a Django ORM (Object-Relational Mapping) pozwala na łatwą manipulację danymi w bazie.
— Integracja z zewnętrznymi usługami: Django jest dobrze przystosowane do integracji z zewnętrznymi usługami, takimi jak prognoza pogody, przez co zapewnia elastyczność w implementacji takich funkcji.

5.3. Baza danych - SQLite

— Prostota i szybkość rozwoju: SQLite jest lekką bazą danych, idealną do zastosowań, gdzie aplikacja ma być rozwijana w sposób szybki i elastyczny, bez konieczności konfiguracji skomplikowanego systemu. Django obsługuje SQLite jako domyślną bazę danych, co pozwala na szybkie uruchomienie projektu i testowanie funkcjonalności.
— Przechowywanie danych o podróżach, etapach, kosztach, grupach: SQLite sprawdza się przy przechowywaniu prostych danych, takich jak informacje o podróżach, etapach, kosztach, czy uczestnikach. Jest to dobre rozwiązanie, biorąc pod uwagę wymogi aplikacji, które nie wymagają zaawansowanych operacji na dużych bazach danych.

6. Realizacja projektu

6.1. Proces powstawania aplikacji

Na początku stworzono szablon projektu, który obejmował połączenie frontendu z backendem (konfiguracja bazy danych była domyślnie zaimplementowana w Django), a także konfigurację i uzwględnienie wszystkich potrzebnych do dalszej pracy pakietów. Szablon powstał po to, aby wszyscy uczestnicy projektu posiadali jednakową "bazę" i mogli rozpocząć pracę w przeznaczonej dla siebie przestrzeni, to znaczy albo w plikach związanych z backendem albo z frontendem. Zdecydowanie usprawniło to pracę nad projektem i ograniczyło potencjalne konflikty w konfiguracji programu przez uczestników. W międzyczasie stworzono model bazy danych, który ukazany będzie w dalszej części dokumentacji, w którym uwzględniono podział na tabele, potrzebne do realizacji logiki biznesowej, a także relacje między tymi tabelami. Prace nad backendem rozpoczęły się od implementacji modelu bazy danych w Django, a więc odpowiedniej modyfikacji plików models, views oraz serializers. Spowodowało to jednoczesną implementację metod CRUD, które są obsługiwane przez Django bez potrzeby ich jednoznacznej implementacji.

Następnie stworzono i udoskonalono panel admina. Na zakończenie prac związanych z backendem, zaimplementowano dalszą część logiki biznesowej, czyli umożliwienie podziału kosztów między uczestników podróży, podział kosztów na kategorie czy pobieranie informacji o pogodzie. Po zaimplementowaniu wszystkich funkcjonalności po stronie backendu, rozpoczęto pracę nad frontendem. Najpierw stworzono widok strony głównej z konkretnymi zakładkami, związanymi z komponentami, których implementację później rozszerzano. Dodano możliwość dodawania danych w każdej zakładce (np. dane związane z podróżami, uczestnikami itd.). Następnie korzystano z gotowych komponentów MUI, aby ulepszyć wygląd poszczególnych pól. Dodano logo i przedstawiono także podział kosztów w postaci wykresu kołowego w zakładce "Koszty".

6.2. Problemy

Zdecydowanie największym problemem było zrozumienie działania Django oraz Reacta. Ponieważ nikt z uczestników nie miał wcześniej do czynienia z tymi frameworkami, dużo czasu oraz pracy należało włożyć w naukę ich, a także dowiedzenie się jakie możliwości zapewniają "z góry" - na przykład to, że nie jest konieczna bezpośrednia implementacja metod CRUD, ponieważ po odpowiedniej modyfikacji konkretnych plików, metody te obsługiwane są przez framework. Wyzwaniem było również połączenie frontendu z backendem, a także zrozumienie działania migracji w Django i rozwiązywanie problemów z nimi związanych, które pojawiały się przy modyfikacji bazy danych. Zauważono również pewne nieścisłości w dokumentacji podczas pracy nad panelem admina, jednak wszystkie wymienione problemy rozwiązano, czy to metodami prób i błędów czy też odpowiednio długim czasem poświęconym na szukaniu rozwiązania.

6.3. Modyfikacje

Na początku pracy nad projektem zdecydowano się zmienić używaną bazę danych z zakładanego PostgreSQL na SQLite z uwagi na domyślne jej użycie w Django, co ułatwiało implementację modelu, a jednocześnie całkowicie spełniało wymogi tej aplikacji. Zmianie uległo również logowanie. Uproszczono je do wpisywania imienia i nazwiska użytkownika, zakładając, że z aplikacji korzystać będzie jedna osoba, która tworzyć będzie swoje własne podróże. Może ona nadal dodawać inne osoby do podróży, jednak będą one widoczne tylko dla głównego użytkownika. Logika logowania i walidacja użytkowników nie zostały jednak zaimplementowane, a odpowiednie pola w interfejsie graficznym są miejscem, w którym w przyszłości takie logowanie mogłoby się odbywać. Dalszy rozwój aplikacji mógłby opierać się na korzystaniu z narzędzi Django do uwierzytelniania i logowania, co pozwoliłoby na dostęp do tych samych informacji dla wielu użytkowników, jednak wymagałoby to również wykorzystania publicznej bazy danych. Z uwagi na trudność w implementacji zdecydowano się także nie umieszczać kalendarza, związanego z terminami podróży.

7. Opis techniczny

Naszym założeniem było korzystanie z dwóch frameworków Django oraz React. Django został użyty do obsługi backendu, natomiast React został użyty do obsługi frontendu.

7.1. Backend

Backend został podzielony na dwa obszary o nazwach folderów 'Django' oraz 'api'. W folderze 'Django' znajdują się takie pliki jak:

- __init__.py - Plik oznaczający folder jako moduł Python.
- asgi.py - Służy do obsługi protokołu ASGI (Asynchronous Server Gateway Interface), umożliwiającego obsługę zapytań asynchronicznych.
- settings - Główna konfiguracja projektu Django. Znajdują się tutaj ustawienia, takie jak połączenie z bazą danych, zainstalowane aplikacje, middleware itp.
- urls.py - Główne ścieżki URL projektu. Łączy różne aplikacje, takie jak api, z projektem głównym.
- wsgi.py - Służy do obsługi protokołu WSGI (Web Server Gateway Interface), który pozwala na uruchamianie aplikacji na serwerach takich jak Gunicorn.

Są to pliki, które są automatycznie genereowane przy tworzeniu projektu Django, które w większości nie wymagają zmian. Jedyne zmiany jaki zastosowaliśmy to dodanie konfiguracji dla CORS oraz REST Framework w pliku settings.py oraz dodanie ścieżek URL w pliku urls.py.

Natomiast w drugiej części znajdują się takie pliki jak:

- __init__.py - Plik oznaczający folder jako moduł Python.
- admin.py - Plik do konfiguracji panelu administracyjnego Django. Można tutaj zarejestrować modele, aby były widoczne w panelu admina.
- apps.py - Definiuje konfigurację aplikacji Django, np. nazwę aplikacji.
- models.py - Zawiera definicje modeli, które reprezentują dane przechowywane w bazie danych.
- serializers.py - Umożliwia konwersję danych między formatem JSON (używanym w API) a obiektami modeli Django. Dzięki temu dane z bazy są łatwo dostępne dla frontendu.
- tests.py - Miejsce na testy aplikacji. Można tutaj pisać testy, aby sprawdzić, czy aplikacja działa poprawnie.
- urls.py - Definiuje ścieżki URL dla tej aplikacji. Łączy endpointy API (np. /api/tasks/) z widokami.
- views.py - Zawiera logikę widoków API, czyli kod obsługujący żądania HTTP (GET, POST, PUT, DELETE) i odpowiadający danymi.
- Folder 'migrations' - Zawiera pliki, które śledzą zmiany w bazie danych. Są generowane automatycznie przez Django podczas migracji.

Wszystkie te pliki zostały wygenerowane automatycznie, przy tworzeniu projektu, ale w tym przypadku zmiany jakie zaszły w części z nich są znaczne. W pliku 'admin.py' mamy zdefiniowane modele w witrynie administracyjnej Django i dostosowuje ich wyświetlanie w interfejsie administracyjnym. Przykładowo dla Travel mamy zdefiniowane, jakie pola są wyświetlone oraz jakie filtry można zastosować w przeszukiwaniu danych. Z kolei w pliku 'models.py', mamy zdefiniowane wszystkie tabele oraz rodzaje danych jakie są przechowywane w poszczególnych polach tabeli np: char lub boolean itp. Natomiast w pliku 'serializers.py' konwertujemy dane dla każdego modelu znajdującego się w pliku 'models.py' na format JSON. W pliku 'urls.py' znajdują się ścieżki URL pod jakimi są zdefiniowane konkretne widoki. W naszym przypadku każdy model ma osobny widok i URL. W pliku 'views.py' mamy zdefiniowane widoki dla modeli pobranych z 'serializers.py' po konwersji na format JSON. Mamy tam również zdefiniowane funkcje wyświetlającą pogodę pobraną z innej strony dla danego miejsca podróży oraz funkcje kalkulujące koszta podróży.

7.2. Frontend

- Folder node_modules - Zawiera wszystkie paczki zainstalowane za pomocą npm (Node Package Manager).
- Folder public - Zawiera statyczne pliki frontendowe, np. index.html, który jest bazą dla aplikacji React.
- Folder src - Główna logika aplikacji React.
  - components
  - App.css - Stylizacja dla aplikacji React, np. kolory, czcionki i układ strony.
  - App.js - Główny plik aplikacji React. Łączy różne komponenty i wyświetla ich zawartość.
  - App.test.js - Zawiera testy jednostkowe dla aplikacji React. Służy do sprawdzania poprawności działania komponentów.
  - index.css - Główne style dla całej aplikacji React.
  - index.js - Punkt startowy aplikacji React. Renderuje aplikację w elemencie HTML
  - logo.svg - Zawiera logo React
  - reportWebVitals.js - Umożliwia mierzenie wydajności aplikacji, np. czasu ładowania.
  - setupTests.js - Plik konfiguracyjny dla testów aplikacji React.
- package.json - Zawiera informacje o aplikacji React i listę zależności (paczki npm).
- package-lock.json - Automatycznie generowany plik, który zapisuje dokładne wersje wszystkich zainstalowanych paczek npm

Folder 'node_modules' zawiera wszystkie zależności i podzależności wymagane przez projekt. Zależności te są określone w pliku package.json. Znajdują się tam kody bibliotek, pliki konfigurujące, mapy źródłowe i definicje typów. Natomiast w folderze 'public' znajdują się zdjęcia logo automatycznie tworzone przy tworzeniu projektu React. Z kolei w folderze 'src' znajdują się pliki zawierające kod źródłowy dla frontendu. Dalej zostaną opisane pliki znajdujące się w folderze 'scr'. W folderze 'components' znajdują się pliki .json, które odpowiadają za wyglądy i funkcjonalności poszczególnych stron. Dane pobierane są przy pomocy funkcji 'axios' z plików Django. 'App.js' importuje widoki oraz tworzy ścieżki pod którymi dane widoki się znajdują. 'App.css' zawiera wygląd stron. Reszta plików nie była edytowana.

7.3. Inne

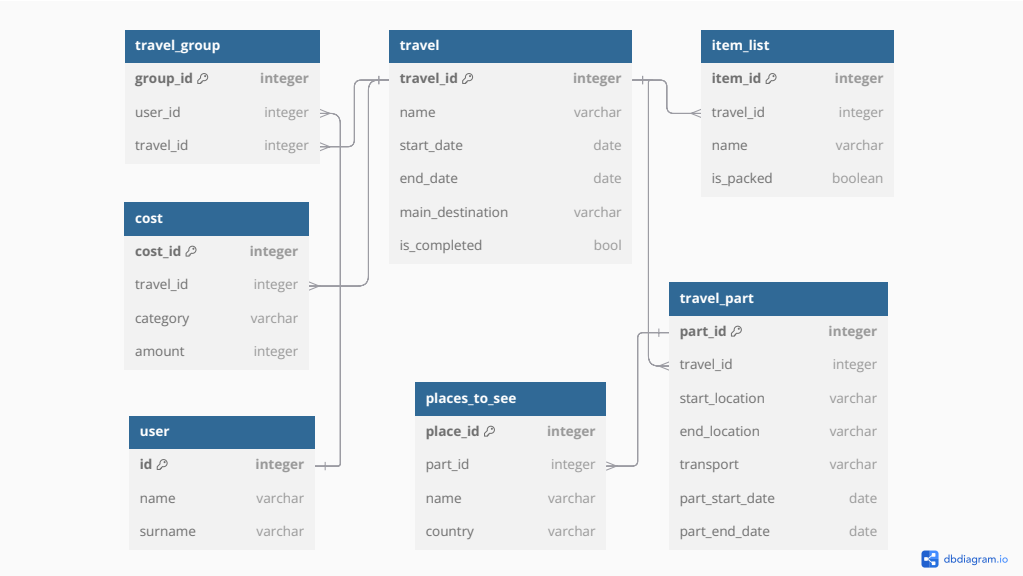
- Folder node_modules
- Folder templates
- db.sqlite3 - Plik db.sqlite3 to domyślny plik bazy danych SQLite używany przez Django do przechowywania wszystkich danych dla aplikacji.
- identifier.sqlite
- manage.py - Narzędzie Django do zarządzania projektem. Używane do uruchamiania serwera, tworzenia migracji i zarządzania bazą danych.
- package.json
- package-lock.json -Automatycznie generowany plik, który zapisuje dokładne wersje wszystkich zainstalowanych paczek npm

Pliki 'package.json' oraz 'package-lock.json' działają na podobnej zasadzie, co wcześniej. Plik 'db.splite3' zawiera wszystkie tabele i rekordy zdefiniowane przez modele Django. Ten plik jest automatycznie tworzony i zarządzany przez Django podczas uruchamiania migracji. Służy do utrwalania danych, takich jak informacje o użytkownikach, szczegóły podróży, koszty i inne dane specyficzne dla aplikacji. Plik 'manage.py' to narzędzie wiersza poleceń, które umożliwia interakcję z projektem Django na różne sposoby. Jest on generowany automatycznie podczas tworzenia nowego projektu Django. Oto niektóre z kluczowych funkcji, które zapewnia: uruchamianie serwera deweloperskiego, migracje baz danych. W projekcie dokonano modyfikacji

domyślnego pliku base_site.html znajdującego sięę w folderze templates, który odpowiada za dostosowanie wyglądu panelu administracyjnego Django. Plik ten pozwala na zmianę lokalizacji, w której framework Django poszukuje źródeł informacji o wyglądzie panelu administracyjnego.

7.4. Baza danych

Na rysunku 7.1 widoczny jest schemat bazy danych, wykorzystywanej w aplikacji, która przechowuje informacje odnośnie podróży, kosztów, uczestników itd. Przedstawione są również relacje między poszczególnymi tabelami. Relacja wiele do wielu pomiędzy tabelami user i travel na schemacie przedstawiona jest za pomocą tabeli pośredniej travel_group, jednak w projekcie taka tabela nie musi być tworzona, ponieważ po zamodelowaniu danych i określeniu takiej relacji Django automatycznie tworzy tabelę pośrednią, która upraszcza korzystanie z metod CRUD przez użytkownika, ponieważ nie musi on tworzyć zapytań bezpośrednio do tej tabeli, a może odnosić się jedynie do tabel user i travel. Baza danych jest domyślnie zaimplementowana w Django.



Rys. 7.1: Schemat bazy danych

8. Instalacja aplikacji

8.1. Standardowa

Aby uruchomić naszą aplikację, najpierw musimy przygotować odpowiednie środowisko. Pierwszym sposobem jest wykorzystanie edytora kodu (zalecany Pycharm). W tym przypadku będziemy musieli zainstalować kilka rozszerzeń aby móc zacząć działać.

Najpierw tworzymy nowy projek na podstawie repozytorium z kodem źródłowym wykorzystując na przykład git clone. Następnie instalujemy wszystkie zalecane biblioteki wykorzystując polecenie:

```
pip install -r requirements.txt
```

Po zainstalowaniu wymaganych pakietów, możemy uruchomić backend oraz frontend używając odpowiednio:

```
python manage.py runserver
```

oraz dwóch poleceń npm:

```
cd myfrontend
npm install
npm start
```

W ten sposób nasza aplikacja zostaje uruchomiona na localhost, gdzie możemy zacząć z niej korzystać.

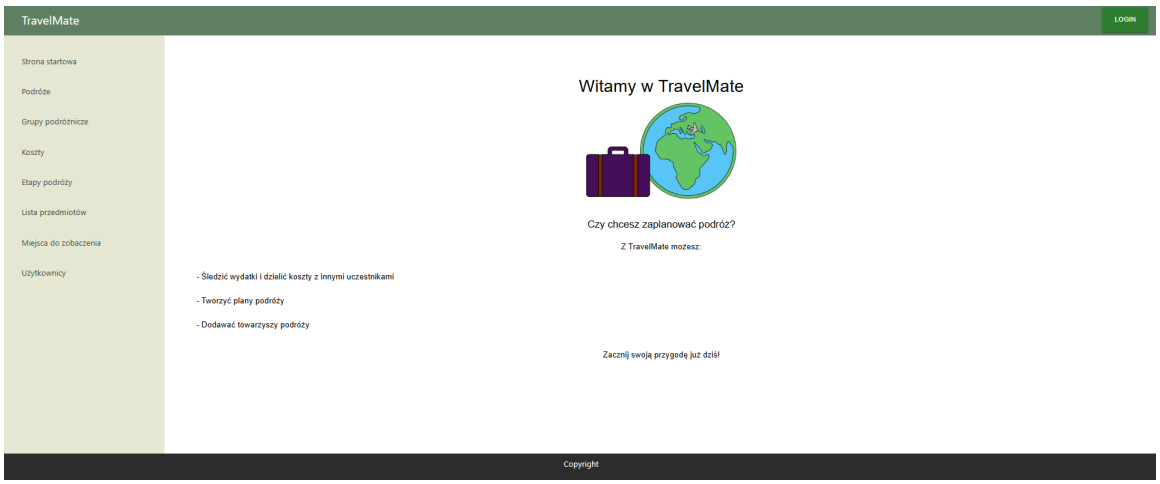
8.2. Docker

Instalacja naszej aplikacji używając oprogramowania docker desktop jest znacząco prostsza. Wystarczy pobrać repozytorium wspomniane wcześniej i użyć komendy aby aplikacja uruchomiła się na localhost:

```
docker compose up --build
```

9. Instrukcja użytkownika

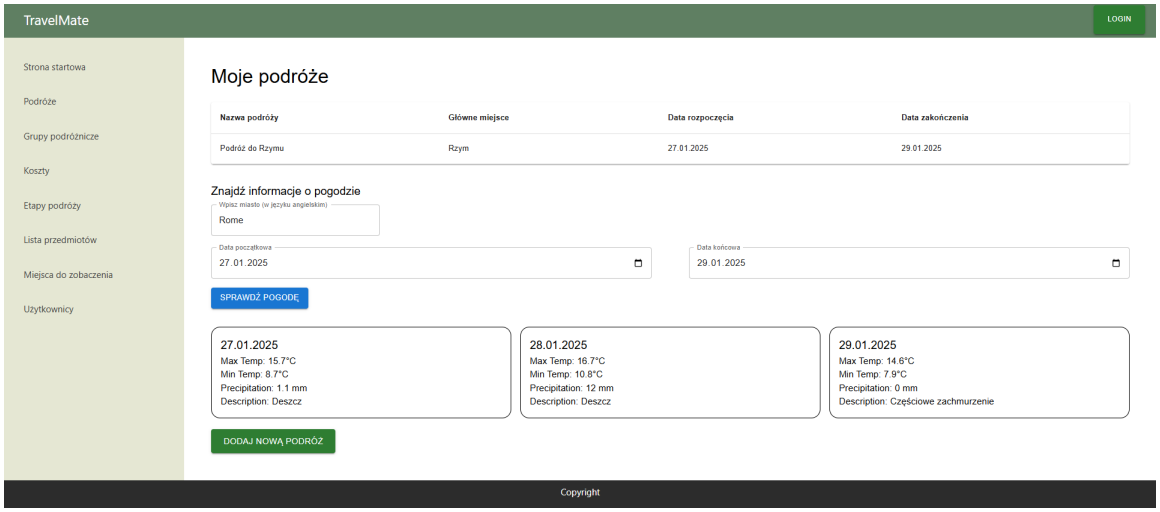
Aplikacja otwiera się na stronie głównej, widocznej na rysunku 9.1, w której do wyboru, po lewej stronie, znajdują się poszczególne zakładki: "Podróże", "Grupy podróżnicze", "Koszty", "Etapy podróży", "Lista przedmiotów", "Miejsca do zobaczenia" oraz "Użytkownicy".



Rys. 9.1: Strona główna

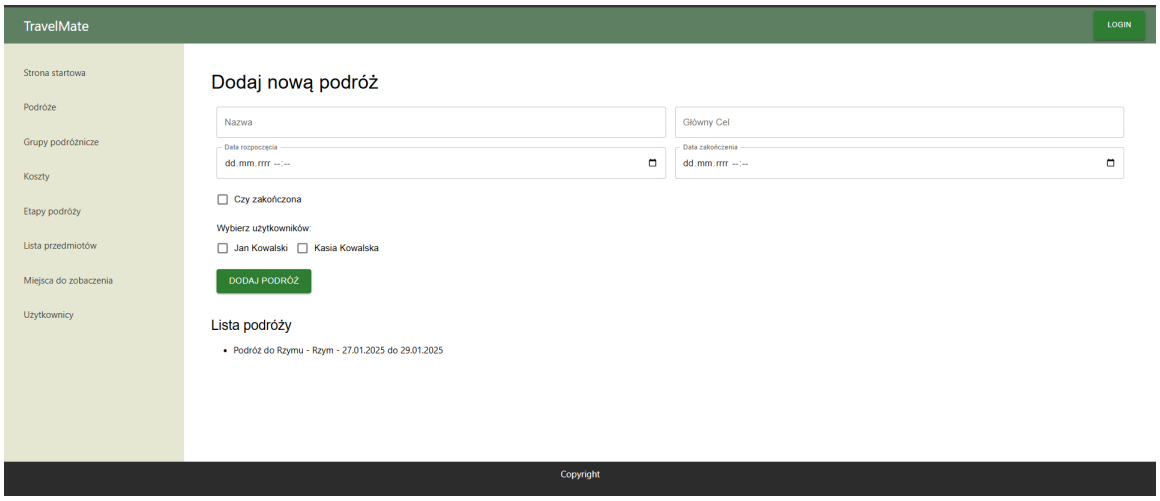
W prawym górnym rogu znajduje się przycisk do logowania, po jego kliknięciu możliwe jest zarejestrowanie nowego użytkownika, bądź też zalogowanie, jednak jest to jedynie komponent interfejsu graficznego i o ile rejestracja nowego użytkownika jest możliwa, to (jak opisano wcześniej) sama logika logowania nie została zaimplementowana.

Po kliknięciu w zakładkę "Podróże" (rys. 9.2) widoczne są informacje o dodanych podróżach: nazwa, destynacja podróży, data jej rozpoczęcia i zakończenia. Możliwe jest również sprawdzenie pogody w konkretnym mieście w wybranym terminie po wpisaniu nazwy miasta w języku angielskim.



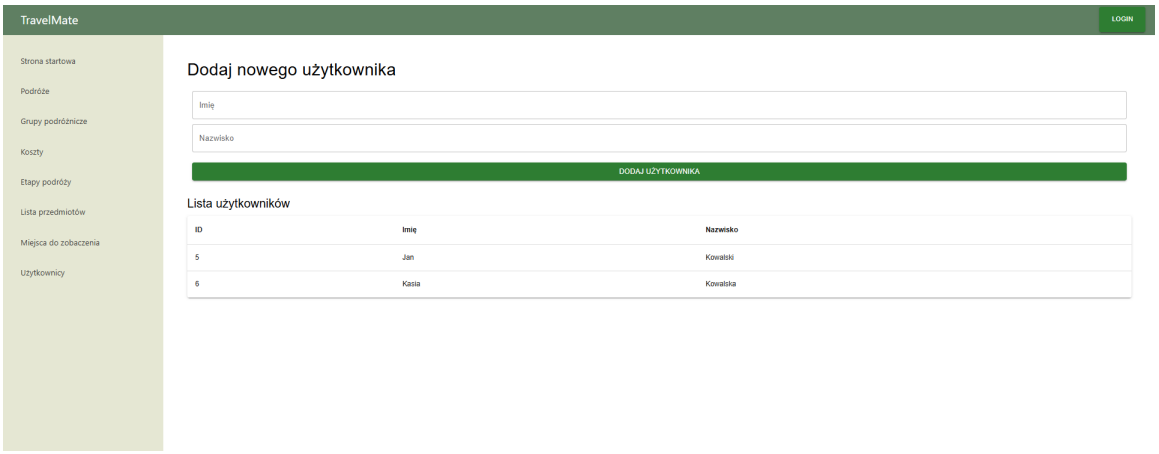
Rys. 9.2: Widok podróży wraz z pogodą

Aby dodać nową podróż należy kliknąć przycisk "Dodaj nową podróż" i wypełnić wszystkie wymagane pola (rys. 9.3).

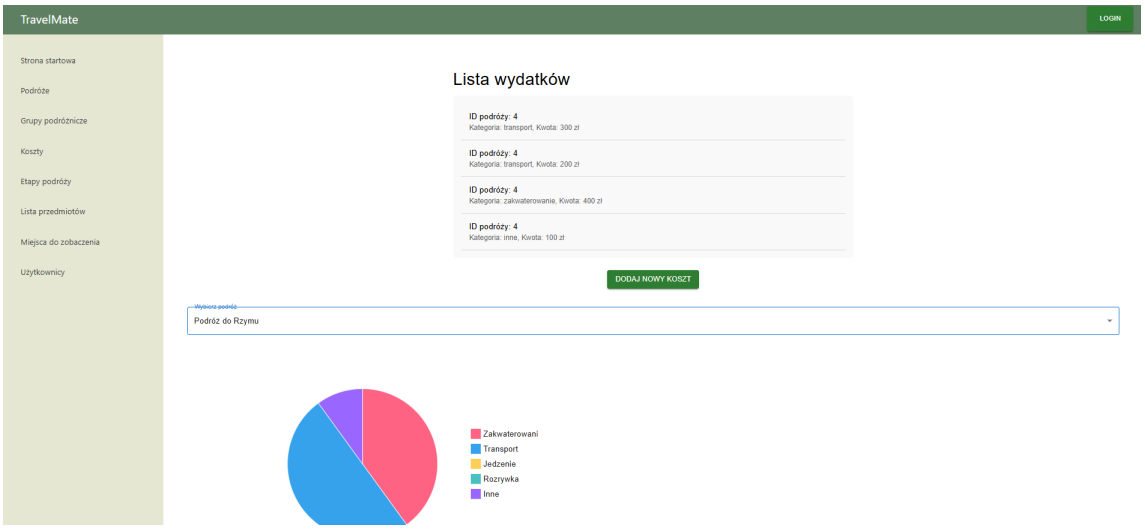


Rys. 9.3: Dodawanie podróży

Działanie pozostałych zakładek jest analogiczne - po ich wybraniu pojawiają się szczegółowe informacje, a także możliwość dodania nowych danych. Na rysunku 9.4 widoczny jest widok związany z użytkownikami. W przypadku zakładki "Koszty" widoczny jest także wykres kołowy z podziałem na kategorie wydatków dla konkretnej podróży (rys. 9.5).



Rys. 9.4: Widok użytkowników



Rys. 9.5: Koszty i wykres kołowy