EuroTravel

Koncepcja wykonania systemu

Hubert Adamiak, Maciej Ciepiela

1. Scenariusze przypadków użycia

Przypadek 1: Logowanie/Rejestracja

1. Użytkownik uruchamia aplikację.
2. Użytkownik ma do wyboru trzy opcje „Zaloguj się”, „Zarejestruj się” lub „Kontynuuj jako gość”
    a. Wybiera „Zaloguj się”:
        i. Użytkownik wypełnia pola „adres e-mail” oraz „hasło”
        ii. System dokonuje weryfikacji:
            - Jeśli dane są poprawne następuje zalogowanie
            - Jeśli dane nie są poprawne wyświetla komunikat „Proszę podać poprawne dane”
    b. Wybiera „Zarejestruj się”:
        i. Otwiera się formularz do wypełnienia:
            - Adres e-mail (maks. 100 znaków, tekst)
            - Imię i nazwisko (maks. 100 znaków, tekst)
            - Data urodzenia (DD-MM-RRRR)
            - Przysługująca ulga (pole do wyboru spośród istniejących ulg)
            - Hasło (maks. 50 znaków, tekst, ukryty)
        ii. System dokonuje weryfikacji:
            - Jeśli dane są poprawne użytkownik dostaje e-mail z potwierdzeniem
            - Jeśli dane nie są poprawne wyświetla komunikat o błędzie
3. Użytkownik zostaje automatycznie zalogowany i przeniesiony do aplikacji.

Scenariusz alternatywny:

2c. Wybiera „Kontynuuj jako gość”:

- użytkownik przechodzi do aplikacji w ograniczonym trybie gościa.

Przypadek 2. Generowanie trasy

1. Użytkownik na górze ekranu głównego wybiera opcję „Wyszukaj trasę”
2. System wyświetla formularz:
    - Punkt początkowy (maks. 100 znaków, wybranie współrzędnych na mapie/pobranie lokalizacji użytkownika/podanie nazwy miejsca)
    - Punkt docelowy (maks. 100 znaków, wybranie współrzędnych na mapie/ podanie nazwy miejsca)
    - Data rozpoczęcia podróży (DD-MM-RRRR HH:MM)
    - Preferencje środków transportu (lista wielokrotnego wyboru: autobus/pociąg/tramwaj)
    - Dodaj przystanek (przycisk akcyjny dodający pole „Przystanek” z właściwościami pola „Punkt docelowy”)
3. System przetwarza dane:
    - Jeśli któraś z informacji jest niepoprawna wyświetla komunikat o błędzie
    - Jeśli są poprawne rozpoczyna wyszukiwanie
4. System przeszukuje dostępne opcje i generuje optymalne trasy
5. System wyświetla proponowane trasy użytkownikowi:
6. Użytkownik wybiera jedną z podanych tras
7. System wyświetla możliwości „Zakup bilet” oraz „Pobierz trasę”:
    a. Użytkownik wybiera opcje „zakup bilet”:
        i. System przekierowuje do płatności
        ii. Zapisuje wygenerowany bilet na koncie w formie .pdf wraz z przebiegiem trasy
    b. Użytkownik wybiera opcje „pobierz trasę”:
        i. System przesyła informacje dotyczące trasy na urządzenie użytkownika i pobiera je do jego pamięci
        ii. System wyświetla podgląd pobranej trasy

Przypadek 3. Dodawanie opinii

1. Użytkownik klika w ikonę konta na ekranie głównym
2. Użytkownik wchodzi w historię konta
3. System wyświetla historie tras wraz z opcją „Dodaj opinie”
4. Użytkownik klika „Dodaj opinie” przy jednej z tras
5. System wyświetla formularz do wypełnienia:
    - Przewoźnik (lista do wyboru spośród wszystkich przewoźników na wybranej trasie)
    - Ocena (skala 1-5)
    - Opinia (maks. 300 znaków, tekst)
6. Użytkownik wypełnia formularz i klika „Dodaj opinie”.
7. System weryfikuje podane informacje, po czym zapisuje opinie w bazie danych i wyświetla ją w informacjach o przewoźniku.

Przypadek 4. Dodawanie ciekawego miejsca

1. Użytkownik uruchamia aplikacje i loguje się
2. System wyświetla ekran główny aplikacji z mapą
3. Użytkownik wyszukuje miejsce i znajduje je na mapie
4. Użytkownik klika na wybrane miejsce
5. System wyświetla możliwe opcje:
    a. Ustaw jako punkt startowy
    b. Ustaw jako punkt docelowy
    c. Dodaj ciekawe miejsce
6. Użytkownik wybiera „Dodaj ciekawe miejsce”
7. System wyświetla formularz do wypełnienia:
    - Nazwa miejsca (maks. 100 znaków, tekst)
    - Typ miejsca (lista wyboru, np. Muzeum, Teatr, Restauracja)
    - Opis miejsca (maks. 300 znaków, tekst)
    - Zdjęcia (opcjonalne, możliwość załączenia pliku)
8. Użytkownik wypełnia formularz i klika „Dodaj miejsce”.
9. System zapisuje miejsce w bazie danych i wyświetla je przy wybranej lokalizacji

2. Projekt dialogów i dokumentów

3. Projekt architektury systemu

Rozwiązania sprzętowe:

- Aplikacja będzie hostowana w chmurze
- Serwery będą obsługiwały zapytania i przechowywały dane w bazach danych

Oprogramowanie systemowe:

1. **Baza danych:**
    - **SQL –** główna baza danych przechowująca najważniejsze informacje (np. PostgreSQL)
    - **NoSQL –** Możliwe wykorzystanie dla danych o dużej skali lub luźnej strukturze (np. MongoDB)
2. **Backend:** Python (Flask/Django)
3. **Frontend:** React.js
4. **Usługi chmurowe:** AWS / Google Cloud

Oprogramowanie do testowania:

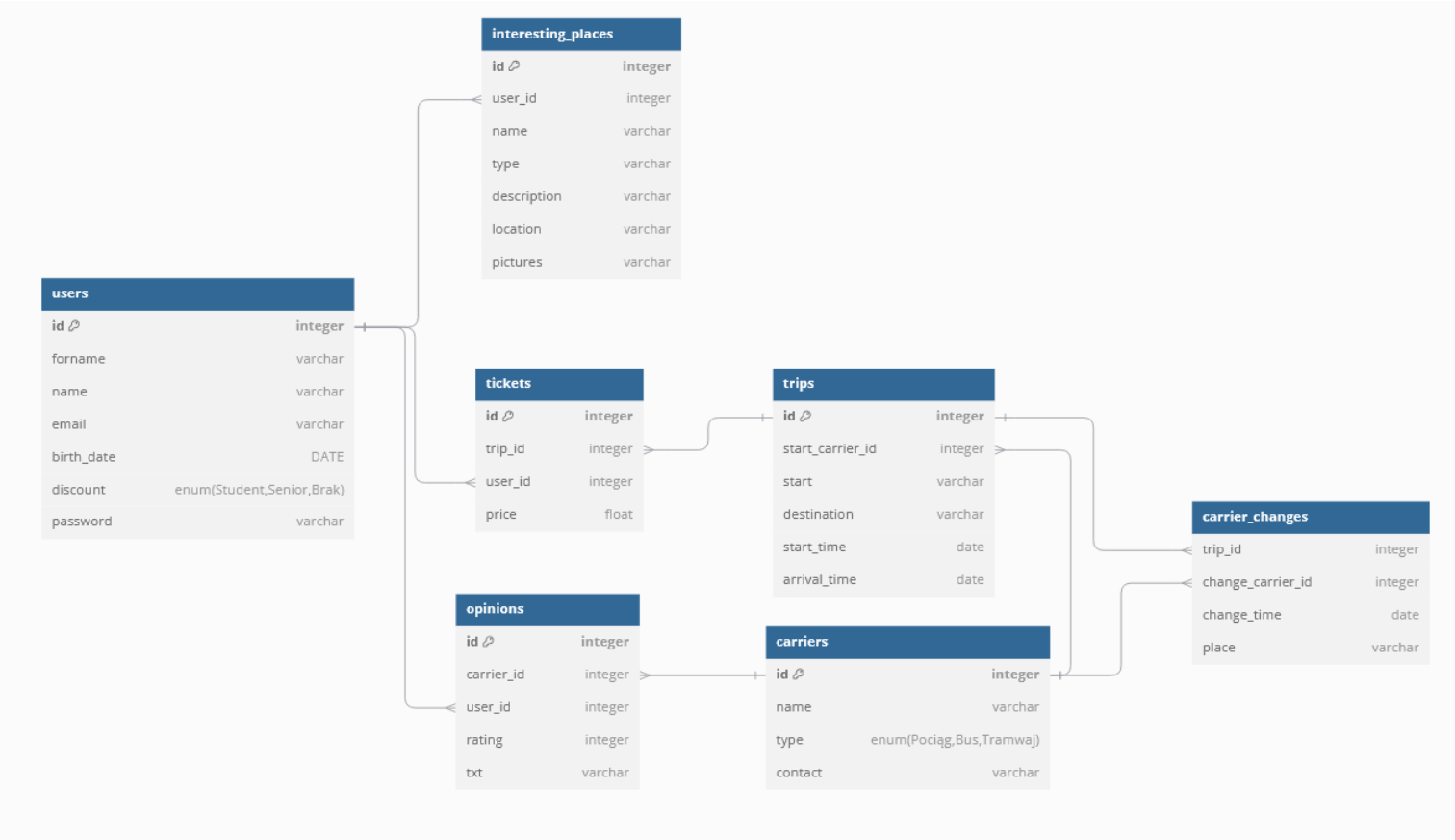
1. **Selenium –** zautomatyzowana platforma testowa dla aplikacji
2. **JUnit –** narzędzie służące do tworzenia powtarzalnych testów jednostkowych oprogramowania pisanego w języku Java.

Struktura logiczna oprogramowania:

System oparty jest na architekturze trójwarstwowej, która obejmuje następujące komponenty:

1. **Warstwa prezentacji:** Frontend stworzony z wykorzystaniem React.js, który odpowiada za interfejs użytkownika oraz komunikację z warstwą logiki biznesowej poprzez API REST.
2. **Warstwa logiki biznesowej:** Backend napisany w Pythonie z wykorzystaniem frameworka Flask/Django. Warstwa ta obsługuje logikę biznesową, komunikację z bazą danych oraz autoryzację użytkowników.
3. **Warstwa danych:** Relacyjna baza danych PostgreSQL przechowuje najważniejsze informacje, takie jak dane użytkowników, trasy oraz bilety. Dla nieustrukturyzowanych danych, takich jak opinie czy multimedia, może zostać wykorzystana baza NoSQL MongoDB.

4. Projekt bazy danych



5. Zasady kodowania

1. **Czytelność:** Kod powinien być pisany zgodnie ze standardami PEP8 w przypadku Pythona oraz ESLint dla JavaScript. Wszelkie nazwy zmiennych, funkcji i klas powinny być intuicyjne i zgodne z konwencjami.

2. **Modularność:** Aplikacja podzielona jest na moduły, co umożliwia łatwą rozbudowę i utrzymanie. Każdy moduł powinien mieć jednoznaczną odpowiedzialność zgodnie z zasadą Single Responsibility Principle.

3. **Dokumentacja**: Kod powinien zawierać komentarze oraz dokumentację w formacie docstrings (Python) i JSDoc (JavaScript).

4. **Testy:** Kod musi być pokryty testami jednostkowymi i integracyjnymi, które zapewniają niezawodność systemu.

6. Zarządzanie ryzykiem

1. **Niska wydajność systemu** - Testy obciążeniowe, skalowanie poziome infrastruktury.
2. **Niekompatybilność z urządzeniami** - Testy kompatybilności, aktualizacje aplikacji.
3. **Awaria serwera chmurowego** - Regularne kopie zapasowe, wybór stabilnych usług chmurowych.
4. **Brak dostępu do danych** - Korzystanie z zewnętrznych łatwo dostępnych API.
5. **Nieuprawniony dostęp do danych** - Szyfrowanie danych, silne uwierzytelnianie.
6. **Wycieki danych** - Audyty bezpieczeństwa, ograniczenie dostępu do danych.
7. **Utrata danych** - Polityka backupu.
8. **Zła jakość kodu** - Stosowanie standardów kodowania, code review, automatyczne formatowanie.

7. Ocena zgodności

System został zaprojektowany zgodnie z wizją przedstawioną w dokumentach „Specyfikacja wymagań” i „Koncepcja wykonania”. Główne założenia, takie jak optymalizacja tras, zakup biletów i możliwość dodawania ciekawych miejsc, zostały wdrożone zgodnie z wymaganiami funkcjonalnymi i niefunkcjonalnymi. Dalsza praca będzie regularnie weryfikowana pod kątem zgodności z wizją i specyfikacją wymagań poprzez przeglądy dokumentacji i implementacji.