AKADEMIA
EKONOMICZNO-HUMANISTYCZNA
W WARSZAWIE

Patrycja Nowak-Bielecka
58849
Monika Szczepanik
58981

Dokumentacja systemu rezerwacji miejsc w autobusie

Prowadzący: mgr Łukasz Krawczuk

Aplikacyjny projekt zespołowy

1 Cel dokumentu

Celem było utworzenie interaktywnej aplikacji webowej, opartej o architekturę klient-serwer, do której dostęp będzie wymagał użycia przeglądarki internetowej. Dla części klienckiej zastosowano technologię Angular, wykorzystująca nadzbiór języka JavaScript, czyli TypeScript. Do utworzenia responsywnego interfejsu użyto bibliotekę Angular Material, opracowaną przez Google. Do części serwerowej wykorzystano framework NestJS oparty o platformę uruchomieniową Node.JS, który również korzysta z języka TypeScript.

Opracowaliśmy projekt aplikacji biznesowej, mając na uwadze potrzeby osób, które w codziennym życiu mają ograniczony czas na podstawowe działania. Aplikacja ta ułatwia proces rezerwacji miejsc, organizację dojazdów do pracy czy domu, oraz planowanie podróży, co czyni ją nieocenionym narzędziem w optymalizacji codziennych obowiązków zawodowych i prywatnych.

2 Wymagania funkcjonalne i niefunkcjonalne

2.1 Wymagania funkcjonalne

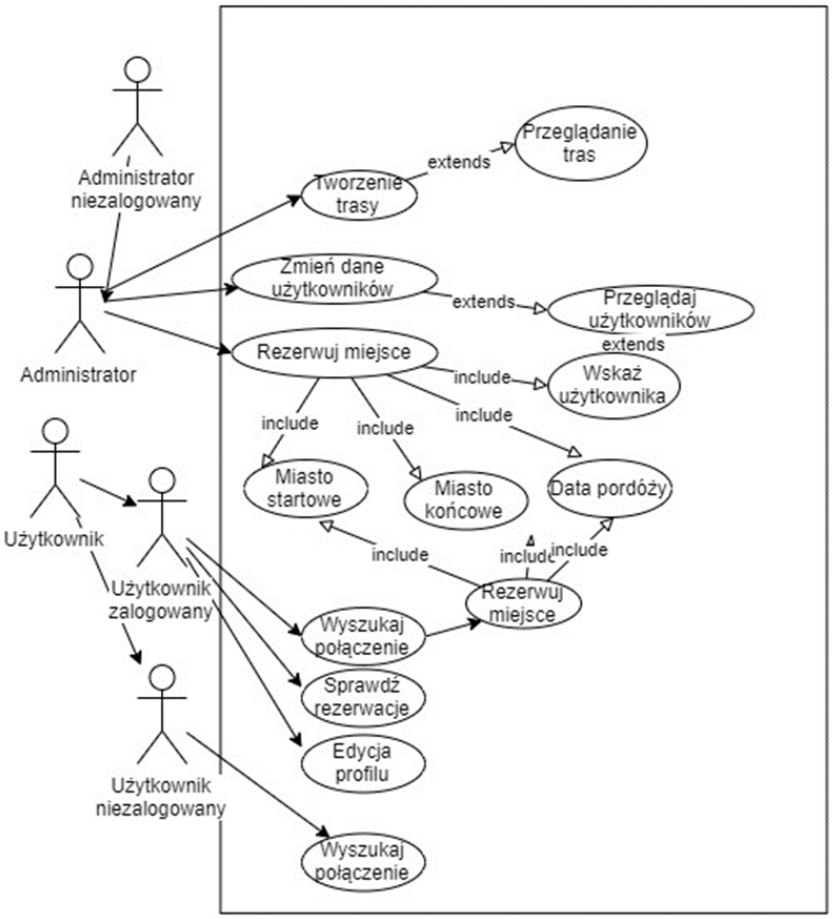
- Rejestracja użytkowników do systemu
- Logowanie do systemu
- Wylogowanie z systemu
- Uaktualnianie danych osobowych danego użytkownika
- Wyszukiwanie połączeń i przeglądanie dostępnych tras
- Rezerwacja miejsc w pojeździe
- Utworzenie rezerwacji
- Wgląd w historię rezerwacji miejsc
- Sprawdzanie przez użytkownika statusu rezerwacji
- Zmiana rezerwacji oraz jego statusu przez administratora
- Uprawnienia do administrowania
- Tworzenie kursów przez administratora
- Zarządzanie użytkownikami przez administratora
- Dodawanie przystanków przez administratora
- Dodawanie zmian tras autobusów przez administratora
- Zarządzanie użytkownikami przez administratora

2.2 Wymagania niefunkcjonalne

- Koszt wykonania systemu musi być jak najniższy. Należy dobrać tak wiele środków optymalizujących jak to możliwe.
- Niezawodność systemu, która oznacza zabezpieczenie aplikacji na wczesnym etapie projektowania i testowania. Wszystkie zadania powinny dokładne i staranne. Powinno się również ochronić aplikację przed nieupoważnione osoby.
- Możliwość oszczędzenia czasu, poprzez zarezerwowanie miejsca online

- Responsywny widok interfejsu użytkownika, aby była możliwość korzystania z różnych urządzeń, przeznaczony również na tablety i smartfony, komputery

3 Diagram przypadków użycia



- Przypadek 1: administrator niezalogowany dostaje uprawniania i dostępy, stając się administratorem zalogowanym.
- Przypadek 2: administrator zalogowany ma możliwość tworzenia tras, zmiany danych użytkowników oraz rezerwacji miejsc.
- Przypadek 3: użytkownik zalogowany ma możliwość edycji profilu, sprawdzenie rezerwacji oraz wyszukiwanie połączeń. Po wybraniu odpowiedniego połączenia, użytkownik może przejść do rezerwacji miejsca.
- Przypadek 4: użytkownik niezalogowany ma możliwość tylko wyszukiwania przejazdów stworzonych przez administratora.

4 Opis sposobów i metod testowania

4.1 Test jednostkowy

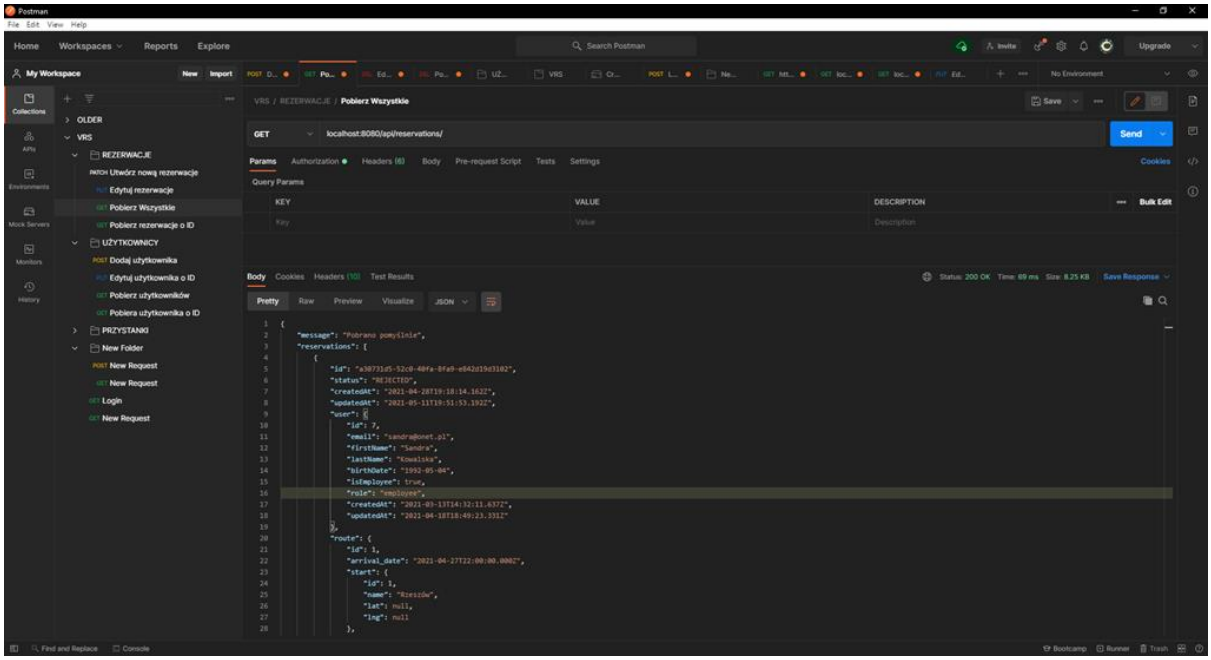
```
it('should throw an error if user already exists', async () => {
      const dto: CreateUserDto = {
        firstName: 'Test',
        lastName: 'User',
        email: 'test@example.com',
        password: 'password',
        birthDate: new Date(),
        isEmployee: false,
        role: UserRole.USER,
      };

      mockUserRepository.findOne.mockResolvedValue(dto);

      await expect(service.createOne(dto)).rejects.toThrow(
        new HttpException('Użytkownik o podanym adresie email istnieje.', HttpStatus.CONFLICT),
      );
    });
  });
```

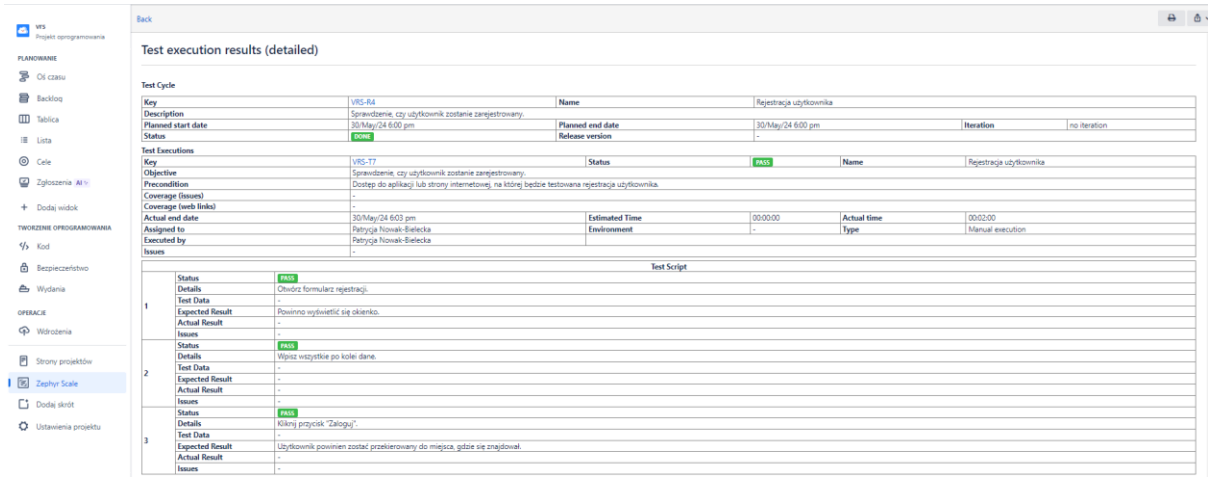
- Powyższy fragment kodu to test jednostkowy wykonany za pomocą frameworka “Jest” w środowisku JavaScript, który sprawdza zachowanie systemu przy próbie dodania nowego użytkownika, który już istnieje w bazie danych.
- Test ma na celu sprawdzenie, czy system odpowiednio radzi sobie z sytuacją, kiedy próbuje się zarejestrować użytkownika z adresem email, który już jest zarejestrowany w bazie danych. Zapewnia, że błędy są wykrywane na wczesnym etapie rozwoju oprogramowania, co ułatwia ich rozwiązywanie i zwiększa stabilność oraz niezawodność aplikacji.

4.2 Test integracyjny



- Do testowania endpointów można wykorzystać aplikację Postman. Umożliwia to wysyłanie żądań do API w łatwy sposób podając adres oraz typ żądania. Odpowiedź zwrotną jaką otrzymujemy od API należy sprawdzić, czy jest taka jaką zaplanowano. Co więcej pozwala zapisywać żądania w celu późniejszego użycia oraz udostępniać je innym osobom w grupie

4.3 Testy manualne



Test execution results (detailed)

Test Cycle

Key	VRS-R4	Name	Rejestracja użytkownika		
Description	Sprawdzenie, czy użytkownik zostanie zarejestrowany.				
Planned start date	30/May/24 6:00 pm	Planned end date	30/May/24 6:00 pm	Iteration	no iteration
Status	DONE	Release version	-		

Test Executions

Key	VRS-T7	Status	PASS	Name	Rejestracja użytkownika
Objective	Sprawdzenie, czy użytkownik zostanie zarejestrowany.				
Precondition	Dostęp do aplikacji lub strony internetowej, na której będzie testowana rejestracja użytkownika.				
Coverage (issues)	-				
Coverage (web links)	-				
Actual end date	30/May/24 6:03 pm	Estimated Time	00:00:00	Actual time	00:02:00
Assigned to	Patrycja Nowak-Bielecka	Environment	-	Type	Manual execution
Executed by	Patrycja Nowak-Bielecka				
Issues	-				

Test Script

1	Status	PASS
	Details	Otwórz formularz rejestracji.
	Test Data	-
	Expected Result	Powinno wyświetlić się okienko.
	Actual Result	-
	Issues	-
2	Status	PASS
	Details	Wpisz wszystkie po kolei dane.
	Test Data	-
	Expected Result	-
	Actual Result	-
	Issues	-
3	Status	PASS
	Details	Kliknij przycisk "Zaloguj".
	Test Data	-
	Expected Result	Użytkownik powinien zostać przekierowany do miejsca, gdzie się znajdował.
	Actual Result	-
	Issues	-

Test Cycle

Key	VRS-R1	**Name**	Logowanie użytkownika		
Description	-				
Planned start date	15/May/24 10:38 pm	**Planned end date**	15/May/24 10:38 pm	**Iteration**	no iteration
Status	DONE	**Release version**	-		

Test Executions

Key	VRS-T1	**Status**	PASS	**Name**	Standardowe logowanie
Objective	Sprawdzenie, czy użytkownik może zalogować się przy użyciu prawidłowych danych uwierzytelniających.				
Precondition	Użytkownik musi być zarejestrowany w systemie.				
Coverage (issues)	-				
Coverage (web links)	-				
Actual end date	15/May/24 10:54 pm	**Estimated Time**	00:00:00	**Actual time**	00:00:17
Assigned to	Patrycja Nowak-Bielecka	**Environment**	-	**Type**	Manual execution
Executed by	Patrycja Nowak-Bielecka				
Issues	-				

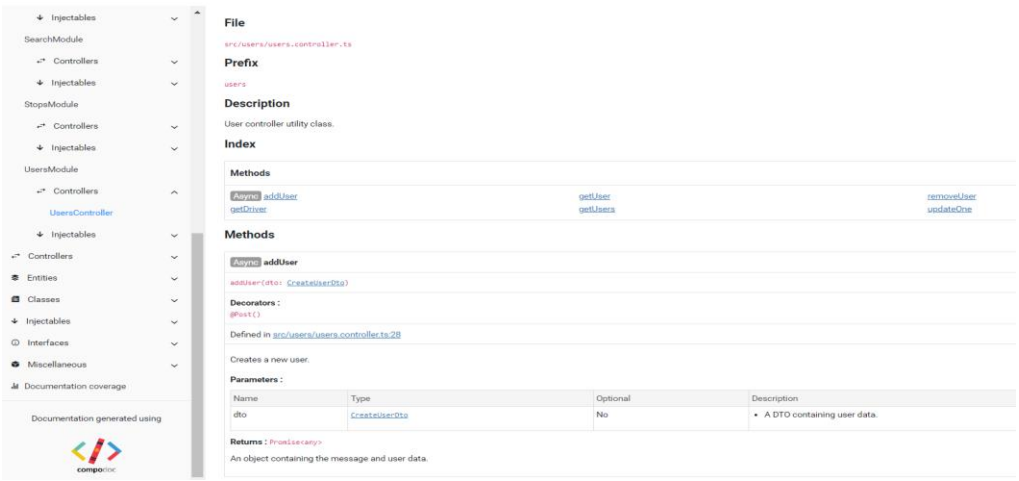
Test Script		
1	**Status**	PASS
	Details	Otwórz stronę logowania.
	Test Data	-
	Expected Result	Powinno wyświetlić się okienko logowania.
	Actual Result	-
	Issues	-
2	**Status**	PASS
	Details	Wpisz e-mail i hasło.
	Test Data	-
	Expected Result	-
	Actual Result	-
	Issues	-
3	**Status**	PASS
	Details	Kliknij przycisk "Zaloguj".
	Test Data	-
	Expected Result	Użytkownik powinien zostać przekierowany do miejsca, gdzie się znajdował.
	Actual Result	-
	Issues	-

- Powyższe zdjęcia przedstawiają raport z testów manualnych

5 Implementacja

```
11
12  /**
13   * User controller utility class.
14   */
15  @Controller('users')
16  export class UsersController {
17
18      constructor(
19          @Inject(UsersService) private usersService: UsersService
20      ) {}
21
22      /**
23          * Creates a new user.
24          * @param dto - A DTO containing user data.
25          * @returns An object containing the message and user data.
26      */
27      @Post()
28      async addUser(@Body() dto: CreateUserDto): Promise<any> {
29
30          const user = await this.usersService.createOne(dto);
31          return {
32              message: "Użytkownik stworzony pomyślnie",
33              user
34          }
35      }
36
```

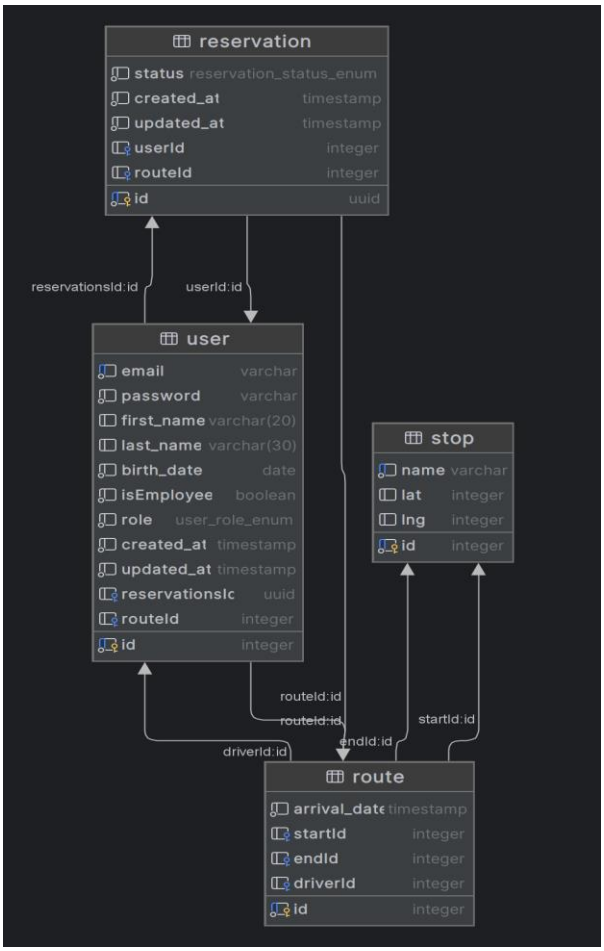
Rys. 1. Metoda odpowiadająca za dodawanie użytkownika



Rys. 2. Fragment dokumentacji wygenerowany przez compodoc

W projekcie zostały wykorzystany typ bazy danych: SQL (ang. ***S**tructured **Q**uery **L**anguage*). Ten typ pozwala nie tylko w łatwy sposób przechowywać duże ilości informacji, ale robić też to wydajnie i tanim kosztem. Dlatego ten typ jest często wykorzystywany. Dane są przechowywane w tabelach, które składają się z kolumn i wierszy.

W projekcie został wykorzystany silnik bazy PostgreSQL. Cechuje się otwartym kodem źródłowym i umożliwia jego modyfikację w zależności od potrzeb. Jako zalety tego silnika, to niezawodność i funkcjonalność. Jest doskonały do wykonywania złożonych zapytań.



Rys. 3. Schemat bazy danych (diagram ERD)