Contact App - Specyfikacja Techniczna

Arkadiusz Flisikowski

4 czerwca 2025

1 Struktura projektu

1.1 Wykorzystane technologie

- C# z ASP .Net Core
- Angular

1.2 Struktura aplikacji C#

Aplikacja została podzielona na 3 mikroserwisy:

- authMicroservice - odpowiedzialny za autoryzację oraz uwierzytelnianie użytkowników
- contactMicroservice - odpowiedzialny za główną logikę aplikacji
- gatewayMicroservice - stanowiący główny punkt wejścia do aplikacji, odpowiedzialny za przekierowywanie żądań do innych mikroserwisów

1.3 GatewayMicroservice

Główne elementy tego mikroserwisu to:

- plik ocelot.json - zostały w nim zdefiniowane zasady przekierowywania żądań do odpowiednich mikroserwisów
- plik Program.cs - w tym pliku została ustawiona polityka CORS, aby aplikacja napisana przy użyciu Angulara mogła wysyłać żądania do aplikacji C#

1.4 AuthMicroservice

Główne elementy tego mikroserwisu to:

- klasa User - jest modelem użytkownika
- interfejs IAuthService - zawiera definicję metod, które muszą być zaimplementowane w serwisie, aby proces autoryzacji oraz uwierzytelnienia przebiegał prawidłowo. Te metody to:
  - registerAsync - umożliwia rejestrację użytkownika
  - loginAsync - umożliwia uwierzytelnienie użytkownika
  - refreshTokenAsync - odpowiada za weryfikację Refresh Tokena
- klasa AuthService - odpowiada za implementację metod zdefiniowanych w interfejsie IAuthService oraz dodatkowo generuje nowy Jwt Token oraz Refresh Token
- klasa AuthController - kontroler, który odbiera żądania API od aplikacji frontendowej oraz przekazuje obsługę logiki tych żądań do serwisu
- klasa AuthDbContext - klasa odpowiedzialna za połączenie z bazą danych (na potrzeby tego projektu została użyta baza danych wbudowana w pamięci)

- klasy DTO (Data Transfer Object) - klasy implementujące modele służące do przesyłania pomiędzy aplikacjami. Te klasy to:
  - LoginRequest
  - LoginResponse
  - RefreshTokenRequest
  - RegisterRequest

1.5 ContactMicroservice

Główne elementy tego mikroserwisu to:

- klasy Contact, Category oraz Subcategory - modele identyfikujące pojęcia biznesowe. Modele są ze sobą w relacjach:
  - klasa Category jest w relacji 1:N z klasą Contact
  - klasa Category jest w relacji 1:N z klasą Subcategory
  - klasa Subcategory jest w relacji 0..1:N z klasą Contact
- interfejs IContactService - zawiera definicję metod, które muszą być zaimplementowane w serwisie, aby realizacja logiki biznesowej aplikacji była możliwa do zrealizowania. Te metody to:
  - getAllContactsAsync - zwraca podstawowe informacje o wszystkich zapisanych kontaktach
  - getContactByIdAsync - zwraca szczegółowe informacje na temat konkretnego kontaktu
  - deleteContactAsync - usuwa konkretny kontakt z listy kontaktów, jeśli tylko wprowadzone hasło kontaktu jest prawidłowe oraz proces autoryzacji użytkownika przebiegł pomyślnie
  - addNewContactAsync - dodaje kontakt do listy kontaktów, jeśli proces autoryzacji użytkownika powiódł się
  - updateContactAsync - edytuje konkretny kontakt, jeśli użytkownik wprowadzi prawidłowe hasło kontaktu oraz przejdzie pomyślnie proces autoryzacji
- klasa ContactService - odpowiada za zaimplementowanie metod zdefiniowanych w interfejsie IContactService
- interfejs ICategoryService - zawiera definicję metod, które muszą być zaimplementowane w serwisie, aby logika aplikacji związana z kategoriami oraz podkategoriami była możliwa do zrealizowania. Te metody to:
  - getCategories - zwraca listę kategorii zdefiniowanych w aplikacji
  - getSubcategoriesByCategories - zwraca listę podkategorii, które należą do konkretnej kategorii
- klasa CategoryService - odpowiada za zaimplementowanie metod zdefiniowanych w interfejsie ICategoryService
- klasa CategoryController - odpowiada za odbieranie żądań API związanych z kategoriami od aplikacji frontendowej oraz przekazywanie ich obsługi serwisowi CategoryService
- klasa ContactController - odpowiada za odbieranie żądań API związanych z kontaktami od aplikacji frontendowej oraz przekazywanie ich obsługi serwisowi ContactService
- klasa ContactDbContext - odpowiada za połączenie aplikacji z bazą danych (na potrzeby tego projektu została wybrana baza danych wbudowana w pamięci) oraz zdefiniowanie relacji pomiędzy encjami
- klasa DatabaseInitializer - odpowiada za stworzenie oraz zapisanie w bazie danych przykładowych danych przy uruchomieniu aplikacji
- klasa ContactMappingProfile - odpowiada za mapowanie obiektów DTO do oryginalnych encji zdefiniowanych w aplikacji

- klasa DTO (Data Transfer Object) - klasy implementujące modele służące do przesyłania pomiędzy aplikacjami. Te klasy to:
  - ContactRequest
  - DeleteContactRequest
  - CategoryResponse
  - ContactDetailsResponse
  - ContactResponse
  - SubcategoryResponse

1.6 Struktura aplikacji Angular

- komponenty:
  - login-component - odpowiedzialny za widok logowania
  - register-component - odpowiedzialny za widok rejestracji
  - contact-add-component - odpowiedzialny za widok dodawania nowego kontaktu
  - contact-delete-component - odpowiedzialny za widok usuwania istniejącego kontaktu
  - contact-details-component - odpowiedzialny za widok prezentujący szczegóły istniejącego kontaktu
  - contact-update-component - odpowiedzialny za widok edytowania istniejącego kontaktu
  - contacts-component - odpowiedzialny za wyświetlanie listy kontaktów
  - footer-component - stopka aplikacji
  - header-component - header aplikacji
  - nav-component - pasek nawigacyjny aplikacji - umożliwia użytkownikowi logowanie/rejestrację bądź wylogowanie w zależności od stopnia uwierzytelnienia użytkownika
- modele - klasy, które odwzorowują klasy DTO na backendzie, aby komunikacja pomiędzy częściami aplikacji mogła się odbywać bez problemów
- serwisy:
  - authService - odpowiedzialny za kontrolowanie stopnia uwierzytelnienia użytkownika poprzez zarządzanie Jwt Tokenem oraz Refresh-Token. Dodatkowo jest odpowiedzialny za wysyłań żądań API związanych z autoryzacją oraz uwierzytelnianiem do aplikacji backendowej
  - categoryService - odpowiedzialny za wysyłanie żądań API związanych z kategoriami oraz podkategoriami do aplikacji backendowej
  - contactService - odpowiedzialny za wysyłanie żądań API związanych z kontaktami do aplikacji backendowej
- authGuard - odpowiedzialny za blokowanie dostępu do nieautoryzowanych widoków aplikacji
- authInterceptor - odpowiedzialny za dołączanie Jwt Tokena do nagłówków żądań API. W przypadku wygaśnięcia Jwt Tokena zadaniem authInterceptora jest wysłanie Refresh-Tokena, aby użytkownik nie musiał się ponownie logować

2 Wykorzystane Biblioteki

Podczas realizacji projektu wykorzystano następujące biblioteki

- Microsoft.AspNetCore.OpenApi - umożliwia generowanie i dokumentowanie interfejsów API zgodnie ze specyfikacją OpenAPI (Swagger)
- Ocelot - gateway API, który służy do zarządzania routingiem
- Scalar.AspNetCore - rozszerza funkcjonalności ASP.NET Core, ułatwiając zarządzanie skalowalnością i obsługę zapytań

- System.IdentityModel.Tokens.Jwt - służy do tworzenia i weryfikacji tokenów JWT, które są wykorzystywane do uwierzytelniania i autoryzacji
- Microsoft.EntityFrameworkCore - framework ORM pozwalający na mapowanie obiektowo-relacyjne i łatwe operacje na bazie danych
- AutoMapper - automatyzuje mapowanie obiektów pomiędzy różnymi warstwami aplikacji, upraszczając konwersję DTO i modeli domenowych
- Microsoft.AspNetCore.Authentication.JwtBearer - służy do obsługi uwierzytelniania za pomocą tokenów JWT w nagłówkach HTTP

3 Uruchomienie aplikacji

- Sklonuj repozytorium
  - git clone https://github.com/arekflis/Contact-App.git
  - cd Contact-App
- Uruchom authMicroservice
  - cd backend/authMicroservice/authMicroservice
  - dotnet restore
  - dotnet run
  - aplikacja powinna być dostępna pod adresem http://localhost:5265
- Uruchom contactMicroservice
  - cd backend/contactMicroservice/contactMicroservice
  - dotnet restore
  - dotnet run
  - aplikacja powinna być dostępna pod adresem http://localhost:5266
- Uruchom gatewayMicroservice
  - cd backend/gatewayMicroservice/gatewayMicroservice
  - dotnet restore
  - dotnet run
  - aplikacja powinna być dostępna pod adresem http://localhost:5236
- Uruchom frontend w Angular
  - cd frontend/contact-app-frontend
  - ng serve
  - aplikacja powinna być dostępna pod adresem http://localhost:4200