Specyfikacja techniczna projektu listy kontaktów
Adrian Zdankowski

1. Wykorzystane biblioteki:

1.1. Microsoft.AspNetCore.Mvc - Obsługuje kontrolery, atrybuty routingu, zwracanie wyników itd. Główna biblioteka do budowy REST API.

1.2. Microsoft.EntityFrameworkCore - ORM (Object-Relational Mapper) – pozwala pracować z bazą danych za pomocą klas C#.

1.3. Microsoft.IdentityModel.Tokens - Obsługa tokenów JWT (JSON Web Token), w tym klucze i algorytmy podpisu. Używana do szyfrowania, weryfikowania i podpisywania tokenów w aplikacjach z autoryzacją.

1.4. Microsoft.AspNetCore.Authentication.JwtBearer - Middleware do autoryzacji opartej na JWT. Sprawdza nagłówki Authorization: Bearer <token>, weryfikuje i wpuszcza użytkownika.

1.5. Microsoft.EntityFrameworkCore.InMemory - Wersja EF Core, która działa w pamięci RAM.

1.6. Swashbuckle.AspNetCore - Generuje dokumentację Swagger/OpenAPI dla API automatycznie.

1.7. System.ComponentModel.DataAnnotations - Zbiór atrybutów walidacyjnych i metadanych (np. [Required], [StringLength], [EmailAddress]). Pozwala walidować dane wejściowe w DTO i modelach.

1.8. System.Security.Claims - Reprezentuje tożsamość i role użytkownika.

1.9. System.Security.Cryptography - Zawiera klasy do szyfrowania, haszowania i pracy z kluczami kryptograficznymi.

2. Sposób kompilacji aplikacji:

Pobranie kodu z repozytorium Github:
git clone https://github.com/AdrianZdankowski/contact-app.git

Kompilacja backendu w ASP.NET:
cd contact-app/contact-app
dotnet restore
dotnet build

lub włączyć plik contact-app/contact-app.sln i skompilować przez Visual Studio.

Kompilacja frontednu w Angular:
cd contact-app-angular
npm install
ng serve

3. Opis poszczególnych klas i metod

3.1. Klasa **User** - klasa reprezentująca użytkownika aplikacji.

Atrybuty:

```
[Key]
public int Id {get; set;}

[Required]
[EmailAddress]
public string Email {get;set;}

[Required]
public byte[] PasswordHash {get;set;}

[Required]
public byte[] PasswordSalt {get;set;}
```

3.2. Klasa **Category** - klasa reprezentująca kategorię kontaktu.

Atrybuty:

```
[Key]
public int Id {get;set;}

[Required]
[MaxLength(8)]
public string Name {get;set;}
```

3.3. Klasa **Subcategory** - klasa reprezentująca podkategorię kontaktu.

Atrybuty:

```
[Key]
public int Id {get; set}

[Required]
[MaxLength(50)]
public string Name {get;set;}

[Required]
public int CategoryId {get;set;} ← klucz obcy Kategorii
```

3.4. Klasa **Contact** - reprezentuje kontakt

Atrybuty:

```
[Key]
public int Id {get;set;}

[Required]
[MaxLength(50)]
public string FirstName {get;set;}

[Required]
[MaxLength(50)]
public string LastName {get;set;}

[Required]
[EmailAddress]
public string Email { get; set; }

[Required]
public byte[] PasswordHash { get; set; }

[Required]
public byte[] PasswordSalt { get; set; }

[Required]
public int CategoryId { get; set; }
public Category Category { get; set; }

public int? SubcategoryId { get; set; } ← opcjonalne, ze względu na kategorię “Prywatny”
public Subcategory? Subcategory { get; set; }

[Required]
[Phone]
public string PhoneNumber { get; set; }

[Required]
[DataType(DataType.Date)]
public DateOnly BirthDate { get; set; }
```

3.5. Klasy **LoginDTO** i **RegisterDTO** - data transfer objects używane w logowaniu i rejestracji do aplikacji.
Atrybuty:
[Required]
[EmailAddress]
public string Email {get;set;}

[Required]
[MinLength(8)]
[RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$", ErrorMessage = "Password must be at least 8 characters long, contain a capital letter, a number and a special character")]
public string Password {get;set;}

3.6. Klasa **GetContactDTO** - data transfer object używany przy zwracaniu obiektu kontaktu z danymi szczegółowymi.
Atrybuty:
public int Id {get;set;}
public string FirstName {get;set;}
public string LastName {get;set;}
public string Email {get;set;}
public string Category {get;set;}
public string? Subcategory {get;set;} ← opcjonalne, null przy kategorii “Prywatny”
public string PhoneNumber {get;set;}
public DateOnly BirthDate {get;set;}

3.7. Klasa **GetContactsDTO** - data transfer object używany przy zwracaniu obiektu kontaktu bez danych szczegółowych.
Atrybuty:
public int Id {get;set;}
public string FirstName {get;set;}
public string LastName {get;set;}
public string Email {get;set;}
public string PhoneNumber {get;set;}

3.8. Klasa **PostContactDTO** - data trasnfer object używany do tworzenia nowego kontaktu.
Atrybuty:

```
[Required]
[MaxLength(50)]
public string FirstName {get;set;}

[Required]
[MaxLength(50)]
public string LastName {get;set;}

[Required]
public string Category {get;set;}
```

public string? SubCategory {get;set;} ← opcjonalne, null przy kategorii “Prywatny”

```
[Required]
[EmailAddress]
public string Email {get;set;}

[Required]
[MaxLength(9)]
public string PhoneNumber {get;set;}

[Required]
public DateOnly BirthDate {get;set;}

[Required]
[MinLength(8)]
[RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$",
ErrorMessage = "Password must be at least 8 characters long, contain
a capital letter, a number and a special character")]
public string Password {get;set;}
```

3.9. Klasa **PutContactDTO** - data transfer object używany do aktualizacji danego kontaktu.
Atrybuty:

```
[MaxLength(50)]
public string FirstName {get;set;}

[MaxLength(50)]
public string LastName {get;set;}

[EmailAddress]
public string Email {get;set;}

[MaxLength(9)]
public string PhoneNumber {get;set;}

public DateOnly BirthDate {get; set;}

public string Category {get;set;}

public string? Subcategory {get;set;} ← opcjonalne, null przy kategorii “Prywatny”

[Required]
[MinLength(8)]
[RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$",
ErrorMessage = "Password must be at least 8 characters long, contain a capital letter, a number and a special character")]
public string OldPassword {get;set;}

[Required]
[MinLength(8)]
[RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$",
ErrorMessage = "Password must be at least 8 characters long, contain a capital letter, a number and a special character")]
public string NewPassword {get;set;}
```

3.10. Klasa **DeleteContactDTO** - data transfer object używany do usunięcia istniejacego kontaktu.
Atrybuty:
[Required]
[MinLength(8)]
[RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$", ErrorMessage = "Password must be at least 8 characters long, contain a capital letter, a number and a special character")]
public string Password {get;set;}

3.11. Klasa **AppDbContext** - klasa odpowiedzialna za połączenie aplikacji z bazą danych w Entity Framework. Tworzy tabele Users, Contacts, Categories, Subcategories.

3.11.1. Metoda OnModelCreating(ModelBuilder modelBuilder) ustawia reguły i relację między tabelami: unikalność adresu e-mail w Users i Contacts, relacje między kluczami obcymi, skutki usunięcia powiązanych rekordów.

3.12. Klasa **DataSeeder** - klasa służąca do inicjalizacji przykładowych kontaktów w bazie danych, przy uruchamianiu aplikacji.

3.12.1. Metoda Seed() - tworzy bazę jeśli nie istnieje, dodaje kategorie, podkategorie i 5 przykładowych kontaktów.

3.12.2. Metoda AddCategories() - dodaje trzy domyślne kategorie: “Służbowy”, “Prywatny”, “Inny”.

3.12.3. Metoda AddSubcategories() - dodaje podkategorie przypisane do kategorii “Służbowy” i “Inny”.

3.12.4. Metoda AddContacts() - dodaje 5 przykładowych kontaktów z różnymi kategoriami i podkategoriami.

3.13. Klasa **AuthController** obsługuje rejestrację i logowanie użytkownika w aplikacji.

3.13.1. Metoda Register(RegisterDTO dto) - rejestruje nowego użytkownika.
Endpoint: POST /api/auth/register
Sprawdza w bazie danych, czy użytkownik o podanym adresie e-mail już istnieje, jeśli tak to odrzuca żądanie.
Waliduje e-mail i hasło (minimum 8 znaków, 1 cyfra i 1 znak specjalny).
Tworzy hasło z hashem i solą.
Dodaje nowego użytkownika do bazy danych.
Tworzy token JWT i zwraca go w odpowiedzi.

3.13.2. Metoda Login(LoginDTO dto) - loguje użytkownika do aplikacji.
Endpoint: POST /api/auth/login
Sprawdza w bazie danych, czy użytkownik o podanym adresie e-mail już istnieje, jeśli nie to odrzuca żądanie.
Jeśli użytkownik jest już zalogowany odrzuca żądanie.

Sprawdza poprawność hasła przez porównanie hashy.
Jeśli dane są poprawne zwraca token JWT w odpowiedzi.

3.13.3. Metoda CreateToken(User user) tworzy token JWT.
Tworzy klucz i podpisuje token.
Ustawia ważność tokenu na 15 minut.
Zwraca token w formacie string.

3.14. Klasa **ContactController** - obsługuje operacje CRUD (tworzenie, pobieranie, aktualizacja, usuwanie) dla kontaktów. Wymaga autoryzacji dla operacji modyfikujących dane (POST, PUT, DELETE).

3.14.1. Metoda GetAllContacts() - zwraca listę wszystkich kontaktów w wersji uproszczonej (GetContactsDTO)
Endpoint: GET /api/contact

3.14.2. Metoda GetContact(int id) - zwraca szczegóły konkretnego kontaktu (GetContactDTO) na podstawie jego Id. Jeśli kontakt nie istnieje zostanie zwrócony kod 404 Not Found.
Endpoint: GET /api/contact/{id}

3.14.3. Metoda CreateContact(PostContactDTO dto) - tworzy nowy kontakt. Wymaga autoryzacji JWT. Sprawdza i przypisuje kategorie do kontaktu, weryfikuje lub dynamicznie tworzy podkategorie, hashuje hasło i zwraca kod 201 Created. Gdy jakiś atrybut jest niezgodny zwracany jest BadRequest.
Endpoint: POST /api/contact

3.14.4. Metoda UpdateContact(int id, PutContactDTO dto) - aktualizuje kontakt z podanym numerem Id. Wymaga autoryzacji JWT i hasła kontaktu. Waliduje dane, ewentualnie tworzy nową podkategorię gdy podano kategorię “Inny”. Zwraca 200 OK przy pomyślnej aktualizacji, BadRequest/Unauthorized/NotFound w przypadku błędów.
Endpoint: PUT /api/contact/{id}

3.14.5. Metoda DeleteContact(int id, DeleteContactDTO dto) - usuwa kontakt o podanym numerze Id. Wymaga autoryzacji JWT i hasła kontaktu. Jeśli udało się usunąć kontakt zwracany jest kod 200 OK, w przeciwnym wypadku NotFound lub BadRequest.

4. Klasy w Angular

4.1. **AppComponent** - główny komponent aplikacji Angular.
Kontroluje stan logowania użytkownika, reaguje na zmiany trasy, udostępnia przycisk wylogowania.

4.2. **AuthService** - klasa odpowiada za logowanie i rejestrację użytkownika przez API. Zawiera odpowiednio metody:
register(email: string, password: string): Observable<any> - rejestracja użytkownika przez API.
login(email: string, password: string): Observable<any> - logowanie

użytkownika przez API.
setAuthToken(token: string) - zapisuje token JWT w sessionStorage
logout() - usuwa token z sessionStorage

4.3. **ContactService** - klasa odpowiada za wysyłanie żądań CRUD do API.
Metody:
getContacts(): Observable<GetContactsDTO[]> - wysyła żądanie GET zwracające liste kontaktów.
getContacgetContact(id: number): Observable<GetContactDTO> wysyła żądanie GET zwracające szczegóły kontaktu o danym numerze Id.
createContact(contact: PostContactDTO): Observable<ApiResponse> - wysyła żądanie POST dodające nowy kontakt.
updateContact(id: number, dto: PutContactDTO): Observable<ApiResponse> - wysyła żądanie PUT aktualizujący kontakt o danym numerze Id.
deleteContact(id: number, dto: DeleteContactDTO): Observable<ApiResponse> - wysyła żądanie DELETE usuwające kontakt o danym numerze Id.

4.4. Klasy: **RegisterComponent**, **LoginComponent**, **ContactListComponent**, **ContactDetailsComponen**t, **CreateContactComponent**, **EditContactComponent**, **DeleteContactComponen**t dostarczają formularze służące do wprowadzenia danych do obiektów DTO, w celu wysłania odpowiedniego żądania do API, oraz prezentacje odebranych danych z żądań.