Specyfikacja techniczna aplikacji PhoneBook

Mateusz Przyborski

1. Opis aplikacji

PhoneBook jest aplikacją pozwalającą na przechowywanie kontaktów dla użytkownika. Każdy użykownik aplikacji ma możliwość przeglądania listy kontaktów, natomiast zalogowany użytkownik oprócz funkcjonalności użytkownika niezalogowanego ma możliwość dodawania, usuwania oraz edycji istniejących kontaktów.

2. Zastosowane technologie

Aplikacja została podzielona na 3 główne części: frontend, backend oraz baza danych. Do implementacji oprogramowania zostały użyte poniższe narzędzia:

- Angular 19.0.5
- ASP.NET 8
- PostgreSQL 17.4
- Docker 27.0.3
- Visual Studio Code

3. Wykorzystane biblioteki

- ASP.Net Core
- Microsoft.EntityFrameworkCore.Tools 9.0.2
- Npgsql.EntityFrameworkCore.PostgreSQL 9.0.4
- Microsoft.AspNetCore.OpenApi 8.0.0
- Microsoft.AspNetCore.Authentication.JwtBearer 8.0.7
- Microsoft.IdentityModel.Tokens 7.1.2
- Swashbuckle.AspNetCore 7.3.1
- BCrypt.Net-Next

4. Opis poszczególnych klas

Aplikacja backendowa została napisana w architekturze trójwarstwowej. Do obsługi API wykorzystane zostały kontrolery, serwisy oraz repozytoria.

Opis przykładowego kontrolera **ContactController**:

Kontroler przyjmuje żądania spod /api/Contact, które następnie są przekierowywane do serwisu.
Zaimplementowane metody kontrolera:

- Task<IActionResult> GetAllContacts() – zwraca listę kontaktów z żądania GET
- Task<IActionResult> GetContactById(string id) – zwraca kontakt o podanym id z żądania GET
- Task<IActionResult> AddContact([FromBody] PutContactRequest request) – dodaje nowy kontakt (jeśli nie istnieje) do bazy danych z żądania POST
- Task<IActionResult> UpdateContact([FromBody] PutContactRequest request) – dodaje nowy kontakt (jeśli nie istnieje) do bazy danych z żądania PUT

- Task<IActionResult> UpdateContact([FromRoute] string id, [FromBody] PutContactRequest request) – modyfikuje kontakt o podanym id z żądania PUT
- Task<IActionResult> DeleteContact([FromRoute] string id) – usuwa kontakt z bazy danych z żądania DELETE

Opis serwisu **ContactService**:

Serwis korzysta z interfejsu IContactService i służy do obsługiwania logiki biznesowej kontaktów.

Metody serwisu:

- Task<Contact[]> GetAllContactsAsync() – zwraca listę kontaktów z bazy danych
- Task<Results<Ok<Contact>, NotFound>> GetContactAsync(string email) – zwraca kontakt po emailu, chyba że nie został odnaleziony w repozytorium
- Task<Results<Ok<Contact>, NotFound>> GetContactByIdAsync(string email) –zwraca kontakt po Id, chyba że nie został odnaleziony w repozytorium
- Task<Created<Contact>>AddContactAsync(Contact contact) – dodaje podkategorię kontaktu do bazy danych jeśli nie istnieje oraz dodaje kontakt do bazy danych
- Task<Results<NoContent, NotFound>> UpdateContactAsync(Contact contact) – dodaje podkategorię kontaktu do bazy danych jeśli nie istnieje, hashuje hasło kontaktu oraz modyfikuje kontakt w bazie danych
- Task<Results<NoContent, NotFound>> DeleteContactAsync(string email) – usuwa kontakt z bazy danych jeśli istnieje

Opis repozytorium **ContactRepository**:

Repozytorium modyfikuje bazę danych na podstawie metod opisanych poniżej:

- Task<Contact[]> GetAllContactsAsync() – zwraca tablicę kontaktów z bazy danych
- Task<Results<Ok<Contact>, NotFound>> GetContactAsync(string email) – zwraca kontakt z przynależącym emailem jeśli istnieje
- Task<Results<Ok<Contact>, NotFound>> GetContactByIdAsync(Guid id) – zwraca kontakt z przynależącym id jeśli istnieje
- Task<Created<Contact>> AddContactAsync(Contact contact) – dodaje kontakt do bazy danych
- Task<Results<NoContent, NotFound>> UpdateContactAsync(Contact contact) – modyfikuje kontakt lub dodaje go jeśli nie istnieje
- Task<Results<NoContent, NotFound>> DeleteContactAsync(Guid id) – usuwa kontakt z bazy danych jeśli istnieje

Model klasy **Contact**:

Zawiera pola Id, Name, Surname, Email, Password, PhoneNumber, BirthDate, Subcategory, które reprezentują kontakt. W ten sposób dane są przechowywane w bazie danych.

Analogicznie do klas powyżej działają kontrolery, serwisy, modele oraz repozytoria kategorii oraz użytkownika.

Do hashowania haseł zastosowano klasę **PasswordHasher**:

- string HashPassword(string password) – hashuje ciąg znaków z użyciem biblioteki BCrypt
- bool verifyPassword(string enteredPassword, string storedHash) – sprawdza kompatybilność zhashowanego hasła z wpisanym ciągiem znaków

Aby poprawnie obsługiwać API zastosowano klasy DTO dla kontaktów, kategorii oraz podkategorii. Posiadają one proste, uproszczone wartości dotyczące referencji do innych obiektów. Dla każdego DTO napisane zostały funkcje konwertujące klasy do DTO oraz odwrotnie.

5. Sposób kompilacji aplikacji

W celu uproszczenia procesu uruchamiania całej aplikacji wszystkie moduły zostały skonteneryzowane i uruchomione w ramach jednego docker-compose.
Przed uruchomieniem aplikacji należy zwrócić uwagę, czy port 80 nie jest zablokowany przez inny proces.

W celu uruchomienia aplikacji należy w pierwszej kolejności sklonować repozytorium na GitHub:

```
git clone https://github.com/Montaso/PhoneBook
```

Następnie dostajemy się do folderu PhoneBook komendą:

```
cd PhoneBook
```

Na końcu uruchamiamy kontenery komendą:

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

Po pobraniu oraz utworzeniu obrazów dostęp do aplikacji będzie dostępny w przeglądarce pod adresem: http://localhost:80.
W przypadku pojawienia się błędu uruchamiania aplikacji backendowej w kontenerze należy spróbować uruchomić ją ponownie.

6. Bezpieczeństwo

- W celach bezpieczeństwa logowania autoryzacja użytkownika odbywa się poprzez JSON Web Token (JWT), wystawiony na określony czas. Niezalogowany użytkownik (bez aktywnego tokenu) nie ma dostępu do funkcjonalności zalogowanego użytkownika.

- Wszystkie hasła są hashowane przed zapisaniem do bazy danych.

- Formularz zawiera podstawowe walidatory po stronie frontendu.

- CORS został ustawiony by przyjmować zapytania jedynie od http://localhost.