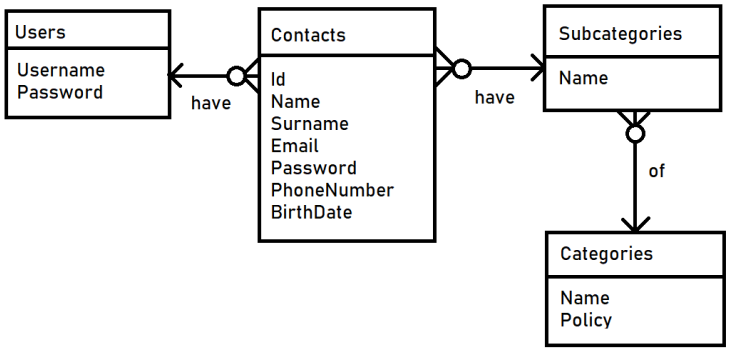
Specyfikacja techniczna PhoneFolio Jan Kaczerski

1. Krótki opis ważnych aspektów

1. Sposób działa
   Aplikacja jest wspólną przestrzenią kontaktów dla grupy użytkowników. Każdy użytkownik może przeglądać kontakty. Zalogowany użytkownik może je edytować, dodawać i usuwać. Zarejestrować się może każdy.

2. Bezpieczeństwo
   Do autoryzacji użyte zostały krótkie tokeny JWT transferowane w ciasteczkach HttpReadonly odświeżane przy jakiejkolwiek odpowiedzi od serwera (poza wylogowaniem). Aplikacja w obecnym stanie posiada podstawowe zabezpiecznia przed SQL Injecton, CSRF oraz XSS.

3. Baza danych



   W aplikacji użyta została baza przedstawiona na powyższym diagramie. Kontakt musi dostać się do swojej kategorii przez kategorię, co pozwala uniknąć niepotrzebnych referencji między kontaktem a kategorią. Metoda ta narzuca posiadanie podkategorii przez kontakt, a więc dodany został konfigurowalny rekord „pustej podkategorii". Email jest często przeszukiwaną kategorią kontaktu, więc został indeksowny. Encje mają TimeStamp ostatniej modyfikacji co pozwala na detekcję konfliktów spowodowanych rozproszonym dostępem do bazy.

2. Oprogramowanie

1. Oprogramowanie
   - ASP.Net 8.0 - VisualStudio Community 2022 17.10
   - Angular 18.1.2 - Visual Studio Code 1.92.0
   - MySql 9.0.1
   - Docker 4.32.0
2. Biblioteki
   - ASP.Net Core
   - Microsoft.EntityFrameworkCore + Tools
   - MySql.Data
   - Pomelo.EntityFramworkCore.Mysql
   - Microsoft.AspNetCore.Authentication.JwtBearer
   - ErrorOr
   - Domyślnie zainstalowane biblioteki Angulara i C#

3. Sposób kompilacji

1. Klonowanie repozytorium

```
$ git clone https://github.com/ar1valdi/PhoneFolio
```

2. Ustawienie i włączenie bazy danych
   - Wybierz w terminalu ścieżkę repozytorium

- Po wykonaniu polecania docker ps należy znaleźć utworzony kontener i przepisać jego id co następnej komendy w miejsce <id>

```
$ cd Database
$ docker-compose up -d
$ docker ps
$ docker cp ./phonepholio_dump.sql <id>:/dump.sql
$ docker-compose exec mysql sh -c 'mysql -u root -p$MYSQL_ROOT_PASSWORD PhoneFolio < /dump.sql'
```

3. Włączenie serwera .NET
   - Wybierz w terminalu ścieżkę repozytorium

```
$ cd Backend
$ dotnet restore
$ dotnet build
$ dotnet run
```

4. Włączenie serwera Angular
   - Wybierz w terminalu ścieżkę repozytorium

```
$ cd Frontend
$ npm install
$ ng serve
```

5. Aplikacja powinna być dostępna w przeglądarce pod adresem localhost:4200

4. Opis klas i metod C#

ApiController - klasa rodzic wszystkich kontrolerów, służy do nadpisania metody Problem() aby zwracała odpowiednie informacje razem z odpowiedzią.

Contacts Controller – przyjmuje żądanie, przetwarza je i zwraca klientowi informacje dotyczące klientów.
- Task<IActionResult> AddContact(AddContactRequest request) – waliduje i dodaje kontakt do bazy danych.
- Task<IActionResult> GetContactsList() – zwraca listę podstawowych danych kontaktu.
- Task<IActionResult> GetContactDetails(int id) – zwraca pełne informacje o kontakcie.
- Task<IActionResult> DeleteContact(int id) – usuwa kontakt z bazy danych, jeżeli nie istnieje, zwraca Errors.Contacts.NotFound (404).
- Task<IActionResult> EditContact(id, EditContactRequest) – waliduje i zamienia dane kontaktu pod podanym id na dane z rządania, jeżeli kontakt nie istnieje, zwraca Errors.Contacts.NotFound (404).

Dictionary Controller – przyjmuje żądanie, przetwarza je i zwraca klientowi informacje dotyczące danych słownikowych.
- Task<IActionResult> FetchCategories() – zwraca kategorie z bazy danych.
- Task<IActionResult> FetchCateogrySubcategories(categoryName) – zwraca wszystkie podka-tegorie danej kategorii z bazy danych.

ErrorsController – punkt docelowy mapowania wyjątków app.UseExceptionHandler("/errors");
- IActionResult Error() – zwraca Problem() bez śladu stosu.

UserController
- Task<IActionResult> RegisterUser(RegisterUserRequest request) – waliduje żądanie i dodaje użytkownika do bazy.
- Task<IActionResult> GetUsername() – zwraca username klienta, jeżeli nie jest zalogowany to Unauthorized().
- Task<IActionResult> Logout() – niszczy ciasteczko z tokenem JWT.
- Task<IActionResult> Login(LoginUserRequest request)– waliduje dane logowania, tworzy token JWT i wysyła go użytkownikowi.

DataContext – abstrakcja bazy danych z EntityFramework, służy do obsługi bazy danych:
- Zawarte kolekcje: Contacts, Categories, Subcategories, Users.
- OnModelCreating() – przechowuje instrukcje jak stworzyć bazę danych, używane przez EF.

SlidingExpirationMiddleware – middleware odpowiadający za przedłużanie tokenów. Znajduje się po autoryzaji.
- Task Invoke(HttpContext context); – jeżeli użytkownik jest zalogowany to przedłuża jego token, w przeciwnym razie nic nie robi.

Category – model kategorii.
- Name – nazwa kategorii.
- Policy – enum CategoryPolicy, specyfikuje jak kategoria zachowuje się względem podkateogrii.
- AllowCustomSubcategories – zwraca bool czy kategoria może mieć dowolne podkategorie.
- HasSubcategories – zwraca bool czy kategoria ma podkategorie.
- Subcategories – kolekcja do nawigacji po bazie danych.

Contact – model kontaktu: record(Id, Name, Surname, Email, Password, Subcategory, PhoneNumber, BirthDate, User).

ContactBasic – model podstawowych danych kontaktu: record(Id, Name, Surname, Password).

Subcategory – model podkateogrii
- Name – nazwa
- Category – kategoria nadrzędna
- Contacts – kolekcja do nawigacji po bazie danych

User – model użytkownika
- Username – nazwa użytkownika
- Password – hasło
- Contacts – kolekcja do nawigacji po bazie danych

IContactsRepository – interfejs repozytorium kontaktów. Wyższy poziom abstrakcji na źródło danych.
- Task<ErrorOr<List<Contact>>> GetAllContactsAsync(); - zwraca listę kontaktów ze źródła danych.
- Task<ErrorOr<Contact>> GetContactAsync(Guid id); - zwraca szczegóły konkretnego kontaktu, jeżeli nie został znaleziony zwraca Errors.Database.NotFound.
- Task<ErrorOr<Contact>> GetContactAsync(string email); - zwraca szczegóły konkretnego kontaktu, jeżeli nie został znaleziony zwraca Errors.Database.NotFound (szuka po emailu).
- Task<ErrorOr<Created>> AddContactAsync(Contact contact); - dodaje kontakt do źródła. W przypadku konfliktu zwraca odpowiednie błędy.
- Task<ErrorOr<Updated>> UpdateContactAsync(Contact contact); - nadpisuje dane kontaktu o zgodnym id. Jeżeli nie istnieje, zwraca Errors.Database.NotFound.
- Task<ErrorOr<Deleted>> DeleteContactAsync(Guid id); - usuwa kontakt o podanym id z bazy. Jeżeli nie istnieje, zwraca Errors.Database.NotFound.

ContactsRepository – implementacja powyższego interfejsu na bazie danych.

MockContactsOneCateogryRepository – implementacja powyższego interfejsu na liście kontaktów z tylko jedną kategorią (cele testowe).

IDictionaryRepository – interfejs zapewniający poziom abstrakcji na dostęp do danych słownikowych.
- Task<ErrorOr<Created>> AddSubcategoryAsync(Subcategory subcategory); - dodaje podkategorię do źródła danych.
- Task<ErrorOr<Created>> AddCategoryAsync(Category subcategory); - dodaje kategorię do źródła.
- Task<ErrorOr<Deleted>> RemoveSubcategoryAsync(string name); - usuwa podkategorię o podanej nazwie, jeżeli nie istnieje zwraca Errors.Database.NotFound.
- Task<ErrorOr<Deleted>> RemoveCategoryAsync(string name); - usuwa kategorię o podanej nazwie, jeżeli nie istnieje zwraca Errors.Database.NotFound.
- Task<ErrorOr<Subcategory>> GetSubcategoryAsync(string name); - zwraca podkategorię o podanej nazwie. W przypadku błędów zwraca odpowiednio zmapowane błędy aplikacji (Errors.).
- Task<ErrorOr<Category>> GetCategoryAsync(string name); - zwraca kategorię o podanej nazwie. W przypadku błędów zwraca odpowiednio zmapowane błędy aplikacji (Errors.).

- Task<ErrorOr<List<Subcategory>>> GetAllSubcategoriesAsync(); - zwraca listę wszystkich podkategorii.
- Task<ErrorOr<List<Category>>> GetAllCategoriesAsync(); - zwraca listę wszystkich kategorii.

DictionaryRepository – implementacja powyższego interfejsu na bazie danych.

IUserRepository – Abstarakcja na dostęp do bazy użytkowników.
- Task<ErrorOr<Created>> AddUserAsync(User user); - dodaje użytkownika do źródła danych.
- Task<ErrorOr<User>> GetUserAsync(string username); - zwraca użytkownika o podanej nazwie użytkownika. Jeżeli nie istnieje, zwraca Errors.Database.NotFound.
- Task<ErrorOr<Deleted>> RemoveUserAsync(string username); - usuwa użytkownika. Jeżeli nie istnieje, zwraca Errors.Database.NotFound.
- Task<ErrorOr<Updated>> EditUserAsync(User user); - zmienia dane użytkownika o id = user.id na dane podane przez parametr.

UserRepository – implementacja interfejsu na bazie danych.

Errors – klasa matka wszystkich błędów aplikacji. Zawiera dokładne opisy występujących błędów. Przykładowy błąd:

```
public static Error PasswordNoSpecialCharacter => Error.Validation(
    code: “RequestValidation.PasswordNoSpecialCharacter”,
    description: “Password does not contain a special character”
);
```

- Code – indywidualny kod błędu
- Description – opis błędu
- Error.<> - typ błędu (zwracany kod)

IContactsService – interfejs wprowadzający abstrakcję na logikę biznesową powiązaną z kontaktami.
- Task<ErrorOr<Created>> AddContactAsync(Contact contact); - waliduje kontakt I dodaje go bazy. Jeżeli walidacja się nie powiodła, zwraca odpowieni błąd.
- Task<ErrorOr<Deleted>> DeleteContactAsync(Guid id); - usuwa kontakt o podanym id. Jeżeli nie istnieje, zwraca Errors.Contacts.NotFound.
- Task<ErrorOr<Updated>> EditContactAsync(Guid id, Contact newContactData); - waliduje nowe dane kontaktu i edytuje kontakt, jeżeli są niepoprawne zwraca odpowiedni błąd.
- Task<ErrorOr<List<Contact>>> GetAllContactsAsync(); - zwraca listę wszystkich kontaktów.
- Task<ErrorOr<Contact>> GetContactAsync(Guid id); - zwraca dane o kontakcie lub odpowiedni błąd.
- Task<ErrorOr<ContactBasic>> GetContactBasicDataAsync(Guid id); - zwraca podstawowe dane o kontakcie lub odpowiedni błąd.
- Task<List<Error>> ValidateContactDataAsync(Contact contact, string username); - sprawdza z pomocą bazy danych czy dane kontaktu są prawdiłowe, username jest nazwą aktualnie zalogowanego użytkownika.

ContactService – implementacja powyższego interfejsu.

IDictionaryService – interfejs odpowiadający za logikę biznesową związaną z danymi słownikowymi.
- Task<ErrorOr<List<Subcategory>>> GetCategorySubcategoriesAsync(string category); - zwraca wszystkie podkategorie kategorii o podanej nazwie.
- Task<ErrorOr<Subcategory>> GetSubcategoryAsync(string name); - zwraca podkategorię o podanej nazwie.
- Task<ErrorOr<Category>> GetCategoryAsync(string name); - zwraca kategorię o podanej nazwie.
- Task<ErrorOr<List<Category>>> GetAllCategoriesAsync(); - zwraca listę wszystkich kategorii.
- Task<ErrorOr<Created>> AddSubcategoryAsync(Subcategory subcategory); - dodaje podkategorię do bazy danych.
- Task<ErrorOr<Deleted>> RemoveSubcategoryAsync(string name); - usuwa podkategorię z bazy, jeżeli nie istnieje, zwraca odpowiedni błąd.

DictionaryService – implementacja powyższego interfejsu.

ITokenService – interfejs służący do obsługi tokenów JWT.
- string GenerateJwtToken(string username); - generuje token JWT.
- CookieOptions GetTokenCookieOptions(); - generuje ustawienia ciasteczka zgodne z appsettings.

TokenService – implementacja ITokenService.

IUserService – interfejs służący do obsługi logiki biznesowej związanej z użytkownikami.

- Task<ErrorOr<User>> GetUserAsync(string username); - zwraca użytkownika o podanej nazwie z bazy danych.
- Task<ErrorOr<Created>> AddUserAsync(User user); - waliduje dane użytkownika, szyfruje hasło i dodaje użytkownika do bazy. W przypadku błędu zwraca odpowiednie wyjątki.
- Task<ErrorOr<Deleted>> RemoveUserAsync(string username); - usuwa użytkownika o podanej nazwie z bazy. Jeżeli nie istnieje, zwracany jest odpowiedni wyjątek.
- Task<ErrorOr<Updated>> EditUserAsync(User user); - Edytuje użytkownika o nazwie user.username na dane obiektu user. Jeżeli obiektu nie ma w repozytorium, zwrócony zostaje odpowiedni błąd.
- Task<ErrorOr<bool>> VerifyUser(User toVerify); - Weryfikuje użytkownika. W przypadku problemów z połączeniem z bazą zwraca odpowiednie błędy.

IUserService – implementacja IUserService.

IStringValidator – interfejs służący do walidacji stringów.

- public bool IsEmail(string email); - zwraca true jeżeli parameter jest właściwie sformatywanym mailem.
- public List<Error> IsPassword(string password); - zwraca true, jeżeli parameter zgadza się z parametrami hasła wyznaczonymi w appsettings.json.
- public bool IsPhoneNumber(string phoneNumber); - zwraca true jeżeli parameter jest właściwie sformatowanym numerem telefonu.

DefaultStringValidator – bazowy walidator stringów implementujący IStringValidator.

REQUESTS – w projekcie w folderze Requests znajdują się szablony żądań i odpowiedzi.
MAPPERS – w folderach odpowiednich serwisów znajdują się interfejsy i klasy mapujące żądania na rzeczywiste dane.

5. Opis klas i metod Angular

KOMPONENTY

- **ContactComponent** – komponent wyświetlający pojedynczy kontakt w liście kontaktówm pobiera dane o kontakcie przez Input(). Zawiera guziki umożliwiające zobaczenie szczegółów, edycji oraz usunięcia kontaktu.
- **ContactDetailsComponent** – komponent wyświetlający detale kontaktu, pobiera id z ActivatedRoute i odpytuje serwer o detale.
- **ContentHostComponent** – komponent wyświetlający ramkę na inne komponenty.
- **ErrorComponent** – komponent wyświetlający błędy aplikacji (podawane przez queryParameters).
- **CategoryPickerComponent** – komponent odpowiadający za wybór kategorii w innych formularzach. Dynamicznie zmienia wyświetlane podkateogire (lub nie wyświetla ich wcale) w zależności od wybranej kategorii. Do wybranych kategorii i podkategorii dostać się można przez gettery.
- **EditContactFormComponent** – komponent wyświetlający i obsługujący formularz edycji istniejącego kontaktu.
- **FormErrorComponent** – komponent wyświetlający i obsługujący błędy w formularzach (np. błędy walidacji po stronie klienta).
- **LoginFormComponent** – komponent wyświetlający i obsługujący formularz logowania.
- **NewContactFormComponent** – komponent wyświetlający i obsługujący formularz dodawania nowego kontaktu.
- **RegisterFormComponent** – komponent wyświetlający i obsługujący formularz rejestracji użytkownika.
- **HomeComponent** – komponent podstawowy [‘/’], odpytuje serwer o listę kontaktów i wyświetla ją. Zawiera także guzik przejścia do tworzenia nowego kontaktu.
- **AppComponent** – główny komponent aplikacji. Zawiera pasek górny z menu użytkownika i tytułem oraz wydziela miejsca wyświetlania podrzędnych komponentów.

REPOZYTORIA – każda metoda repozytoriów wysyła żądanie i zwraca obiekt Observable<> odpowiedniego typu. Obsługa odpowiedzi następuje w serwisach.

- **AuthRepository** – abstrakcja na połączenia z serwerem dotyczące autoryzacji.
    - **addUser(user: User)** – mapuje dane użytkownika na żądanie rejestracji i wysyła je.
    - **getToken(userData: User)** – mapuje dane logowania na żądanie utworzenia tokenu i wysyła je.
    - **removeToken()** – wysyła żądanie usunięcia ciasteczka z tokenem (httpreadonly, musi zostać usunięte na serwerze).

    - **getCurrentUsername()** – wysyła żądanie zwracające nazwę użytkownika odpowiadającą tokenowi. Może służyćdo sprawdzania, czy użytkownik jest zalogowany.
- **ContactRepository** – abstrakcja na połączenia z serwerem dotyczących bazy danych kontaktów.
    - **getContactList()** – wysyła zapytanie o listę wszystkich kontaktów.
    - **getContactDetails(id: number)** – wysyła zapytanie o detale danego kontaktu.
    - **addContact(contact: Contact)** – mapuje kontakt na żądanie dodania go do bazy i wysyła je.
    - **removeContact(id: number)** – wysyła żądanie usunięcia kontaktu o podanym id.
    - **editContact(id: number, newData: Contact)** – mapuje kontakt na żądanie edycji isteniejącego kontaktu i wysyła je.
- **DictionaryRepository** – abstrakcja na połączenia z serwerem dotyczących danych słownikowych
    - **getCategories()** – wysyła zapytanie o listę kontaktów
    - **getCategorySubcategories(category: string)** – wysyła zapytanie o podkategorie danej kategorii

SERWISY – większość metod zwraca odpowiednio zmapowany obiekt Observable<> otrzymany z repozytorium. Jeżeli metoda zwraca coś innego, jest to przy niej opisane.

- **AuthService** – logika biznesowa związana z autoryzacją. Przetrzymuje dane o stanie tokenu użytkownika. Posiada BehaviorSubject<boolean> emitujący zmiany stanu zalogowania użytkownika.
    - **registerUser(user: User)** – dodaje użytkownika do repozytorium
    - **login(user: User)** – prosi repozytorium o token, jeżeli nie otrzymał błędu, emituje true
    - **logout()** –prosi repozytorium o usunięcie tokenu, jeżeli nie ma błędu, emituje false
    - **getUsernameFromServer() : Promise<string>** – odpytuje rezpozytorium o nazwę zalogowanego użytkownika, jeżeli nie wystąpił błąd, emituje true i zwraca nazwę, przeciwnie emituje false i zwraca error
    - **isLoggedNoFetch() : boolean** – zwraca true jeżeli użytkownik jest zalogowany na podstawie wiedzy z poprzednich odpytań serwera

- **ContactService** – logika biznesowa powiązana z kontaktami
    - **async getContactList() : Promise<Contact[]>** - oczekuje na pełną listę kontaktów z repozytorium i ją zwraca. Jeżeli wystąpił błąd, przekierowuje na ErrorComponent.
    - **async getContactDetails() : Promise<Contact>** - oczekuje na szczegóły kontaktu. Jeżeli wystąpił błąd, przekierowuje na ErrorComponent.
    - **addNewContact(contact: Contact)** – dodaje kontakt do repozytorium
    - **removeContact(id: number)** – usuwa kontakt z repozytorium
    - **editContact(contact: Contact)** – edytuje kontakt w repozytorium
    - **extractDate(isoDateTime: string) : string** – konwertuje datę w formacie ISO na datę w formacie DateOnly
- **DictionaryService** – logika biznesowa powiązana z danymi słownikowymi
    - **getAllCategories()** – zwraca wszystkie kategorie z repozytorium
    - **getCategorySubcategories(categoryName: string)** – zwraca wszystkie podkategorie kategorii
    - **async fetchCategories() : Promise<Category[]>** - zwraca kategorie z repozytorium w formie Promise<>
    - **async fetchSubcategories(categoryNmae: string) : Promise<Subcategory[]>** - zwraca podkategorie danej kategorii w formie Promise<>
    - **hasCustomSubcategoriesAllowed(category: Category)** – zwraca true jeżeli kategoria zezwala na własne podkateogorie
    - **hasSubcategoriesAllowed(category: Category)** – zwraca true jeżeli kategoria ma podkategorie
- **RedirectService** – metody ułatwiające korzystanie z routera
    - **redirectToError(error: ErrorResponse) : void** - przekierowuje do ErrorComponent podając zserializowany parameter jako queryParams

MODELE

- **Category, CategoryPolicy, Contact, ErrorResponse, Subcategory, User** – modele danych odpowiadające modelom serwerowym
- **environment** – zawiera zmienne środowiskowe
- **Folder requests** – zawiera zdefiniowane prototypy żądań