

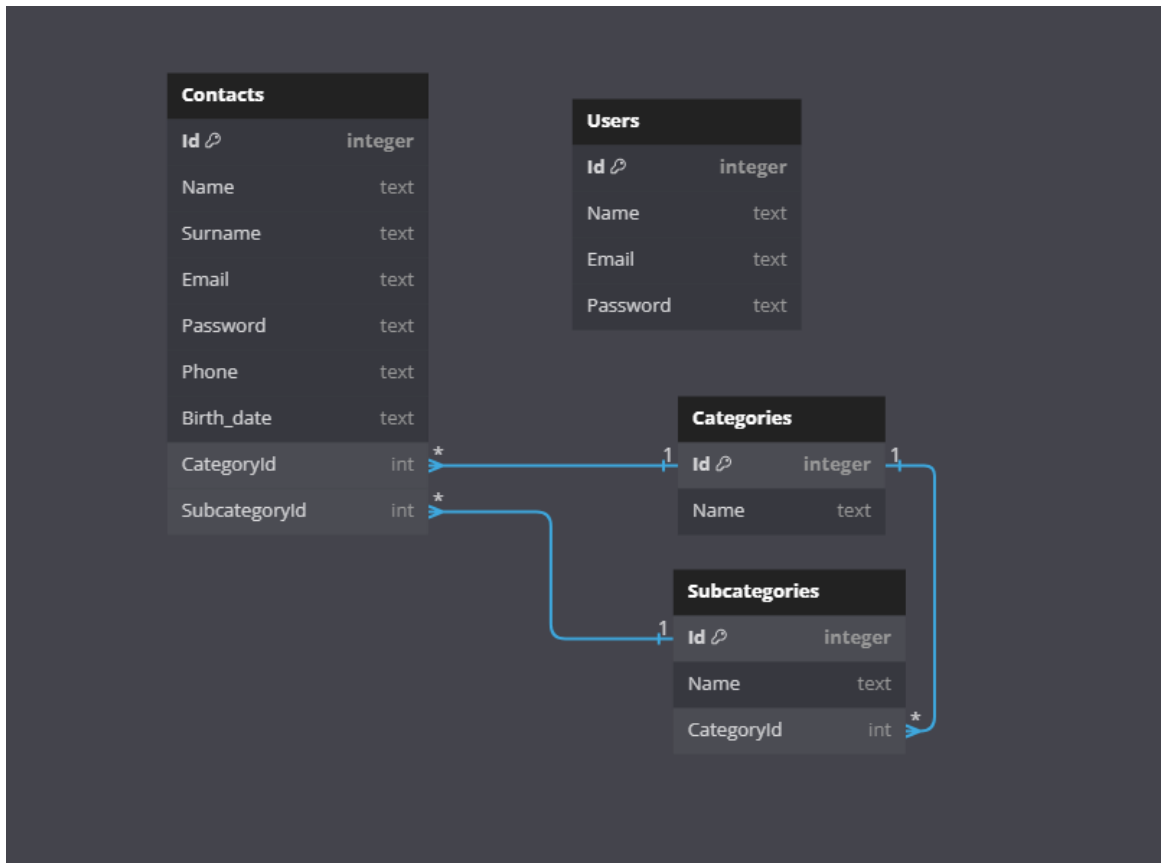
1. Opis poszczególnych klas i metod

Frontend (React.js):

Komponenty:

- **AddContact.js**
 - Formularz dodawania kontaktu
- **App.js**
 - Główny komponent odpowiada za routing oraz posiada panel nawigacyjny
- **CheckPass.js**
 - Przechowuje metodę sprawdzającą złożoność hasła
- **ContactDetails.js**
 - Wyświetla szczegóły wybranego kontaktu
- **Home.js**
 - Wyświetla listę kontaktów (niezalogowany)
- **Loggedin.js**
 - Wyświetla listę kontaktów oraz daje możliwości dodawania, aktualizacji oraz usuwania kontaktów (Wymagane logowanie)
- **Login.js**
 - Formularz logowania
- **Register.js**
 - Formularz rejestracji
- **UpdateContactForm.js**
 - Formularz aktualizacji kontaktu

Backend (.NET C#):



Controllers:

- AccountController.cs

- **[HttpPost("register")]**
public IActionResult Register(RegisterDto registerDto):
Metoda obsługuje żądania rejestracji użytkownika
- **[HttpPost("login")]**
public IActionResult Login(LoginDto loginDto)
Metoda odpowiada za logowanie za pomocą JWT
- **[HttpGet("user")]**
public IActionResult User()
Metoda zwraca zalogowanego użytkownika
- **[HttpPost("logout")]**
public IActionResult Logout()
Metoda wylogowuje użytkownika usuwając JWT

- **CategoryController.cs**

- **[HttpGet]**
public IActionResult GetCategories()
Metoda zwraca wszystkie kategorie
- **[HttpGet("{id}")]**
public IActionResult GetCategory(int id)
Metoda zwraca kategorię o podanym id
- **[HttpPost]**
public IActionResult AddCategory([FromBody] CategoryDto category)
Metoda obsługuje żądania dodania nowej kategorii
- **[HttpPut]**
public IActionResult UpdateCategory(int categoryId, [FromBody] CategoryDto category)
Metoda obsługuje żądania aktualizacji nowej kategorii
- **[HttpDelete]**
public IActionResult DeleteCategory(int categoryId)
Metoda obsługuje żądania usunięcia wybranej kategorii

- **ContactController.cs**

- **[HttpGet]**
public IActionResult GetContacts()
Metoda zwraca wszystkie kontakty
- **[HttpGet("{id}")]**
public IActionResult GetContact(int id)
Metoda zwraca kontakt o podanym id
- **[HttpPost]**
public IActionResult AddContact([FromBody] ContactDto contact)
Metoda obsługuje żądania dodania nowego kontaktu
- **[HttpPut("{contactId}")]**
public IActionResult UpdateContact(int contactId, [FromBody] ContactDto contact)
Metoda obsługuje żądania aktualizacji wybranego kontaktu
- **[HttpDelete("{contactId}")]**
public IActionResult DeleteContact(int contactId)
Metoda obsługuje żądania usunięcia wybranego kontaktu

- **SubcategoryController.cs**
 - **[HttpGet]**
public IActionResult GetSubcategories()
Metoda zwraca wszystkie podkategorie
 - **[HttpGet("{id}")]**
public IActionResult GetSubcategory(int id)
Metoda zwraca podkategorię o podanym id
 - **[HttpGet("name/{name}")]**
public IActionResult GetSubcategoryByName(string name)
Metoda zwraca podkategorię o podanej nazwie
 - **[HttpGet("category/{categoryId}")]**
public async Task<IActionResult> GetSubcategoriesByCategory(int categoryId)
Metoda zwraca podkategorie powiązane z wybraną kategorią
 - **[HttpPost]**
public IActionResult AddSubcategory([FromBody] SubcategoryDto subcategory)
Metoda obsługuje żądania dodania nowej podkategorii
 - **[HttpPut]**
public IActionResult UpdateSubcategory(int subcategoryid, [FromBody] SubcategoryDto subcategory)
Metoda obsługuje żądania aktualizacji wybranej podkategorii
 - **[HttpDelete]**
public IActionResult DeleteSubcategory(int subcategoryid)
Metoda obsługuje żądania usunięcia wybranej podkategorii

Data:

- **AppDbContext.cs**
 - Klasa dziedziczy po klasie DbContext. Odpowiada za ustanowienie połączenia z bazą danych oraz mapowanie obiektów klas i relacji na odpowiednie tabele w bazie danych.

Dto:

- **CategoryDto.cs**
- **ContactDto.cs**
- **LoginDto.cs**
- **RegisterDto.cs**
- **SubcategoryDto.cs**
 - Powyższe klasy reprezentują Data Transfer Object, odpowiadają za przechowywanie danych, które mogą być przesyłane pomiędzy warstwami aplikacji lub aplikacją, a interfejsem użytkownika

Mapper:

- **Mapping.cs**
 - Odpowiada za konfigurację mapowania między modelami, a DTO

Models:

- **Category.cs**
- **Contact.cs**
- **Subcategory.cs**
- **User.cs**
 - Klasy reprezentują encje na podstawie, których budowane są tabele w bazie danych.

Services:

- **AccountService.cs**
 - **public User Register(User user)**
Dodaje użytkownika do bazy oraz go zwraca
 - **public User GetUserByEmail(string email)**

Zwraca użytkownika o podanym emailu

- **public User GetUserById(int id)**
Zwraca użytkownika o podanym id

- **CategoryService.cs**
 - **public Category GetCategory(int id)**
Zwraca wybraną kategorię
 - **public ICollection<Category> GetCategories()**

Zwraca wszystkie kategorie

- **public bool CreateCategory(Category category)**

Dodaje kategorię do bazy danych

- **public bool UpdateCategory(Category category)**
Aktualizuje wybraną kategorię
- **public bool DeleteCategory(Category category)**
Usuwa wybraną kategorię
- **private bool Save()**

Zwraca informację czy dodanie kategorii powiodło się

- **public bool CategoryExists(int id)**

Sprawdza czy kategoria istnieje

- **ContactService.cs**

- **public Contact GetContact(int id)**

Zwraca wybrany kontakt

- **public ICollection<Contact> GetContacts()**

Zwraca wszystkie kontakty

- **public bool CreateContact(Contact contact)**

Dodaje kontakt do bazy danych

- **public bool UpdateContact(Contact contact)**

Aktualizuje wybrany kontakt

- **public bool DeleteContact(Contact contact)**

Usuwa wybrany kontakt

- **private bool Save()**

Zwraca informację czy dodanie kategorii powiodło się

- **public bool ContactExists(int id)**

Sprawdza czy kontakt istnieje

- **JwtService.cs**

- **public string Generate(int userId)**

Tworzy i zwraca token JWT na podstawie userId, podpisany przy
użyciu algorytmu HMAC SHA256 i klucza symetrycznego.

- **public JwtSecurityToken Verify(string jwt)**

Weryfikuje podpisany token JWT przy użyciu klucza symetrycznego i
zwraca zweryfikowany token

- **SubcategoryService.cs**

- **public Subcategory GetSubcategory(int id)**

Zwraca wybraną podkategorię

- **public Subcategory GetSubcategoryByName(string name)**

Zwraca podkategorię na podstawie nazwy

- **public bool CreateSubcategory(Subcategory subcategory)**

Dodaje podkategorię do bazy danych

- **public bool UpdateSubcategory(Subcategory subcategory)**

Aktualizuje wybraną podkategorię

- **public bool DeleteSubcategory(Subcategory subcategory)**

Usuwa wybraną podkategorię

- **private bool Save()**

Zwraca informację czy dodanie podkategorii powiodło się

- **public bool SubcategoryExists(int id)**

Sprawdza czy podkategoria istnieje

- **public async Task<List<Subcategory>>**

GetSubcategoriesByCategory(int categoryId)

Zwraca wszystkie podkategorie powiązane z wybraną kategorią

2. Wykorzystane biblioteki

Frontend (React.js):

- React Router DOM
- Axios

Backend (.NET C#):

- Entity Framework Core
- AutoMapper
- BCrypt.Net-Next
- Microsoft.AspNetCore.Cors
- Npgsql.EntityFrameworkCore.PostgreSQL
- System.IdentityModel.Tokens.Jwt

3. Sposób kompilacji aplikacji

- Instalacja niezbędnych aplikacji:

VisualStudio (Backend C# .NET API)

<https://visualstudio.microsoft.com/pl/downloads/>

NodeJS (Frontend React)

<https://nodejs.org/en/download/prebuilt-installer>

- Pobranie oraz kompilacja:

1. Pobranie repozytorium

git clone <https://github.com/Hikkaruu/NetPCTask.git>

2. Sposób 1:

Uruchom plik start.bat, w razie jego braku (fałszywy alarm antywirusa)

Zmień rozszerzenie start.txt na start.bat i uruchom.

Backend powinien uruchomić się na <http://localhost:5288/swagger/index.html>

Frontend powinien uruchomić się na <http://localhost:3000/>

3. Sposób 2:

Jeżeli z jakichś powodów nie uda się uruchomić aplikacji za pomocą “Sposób 1”.

Backend: Należy uruchomić projekt poprzez środowisko Visual Studio lub

Za pomocą konsoli z folderu projektu przejść na ścieżkę **Back\NetPCTask** i wykonać komendę **dotnet run**

Frontend: Za pomocą konsoli z folderu projektu przejść na ścieżkę **Front\netpctask** i wykonać komendę **npm start**