Gymfinity - sklep internetowy z akcesoriami fitness

Sklep ten dedykowany jest miłośnikom siłowni i oferuje produkty takie jak ubrania, hantle, sztangi czy suplementy. Platforma oferuje możliwość przeglądania różnych kategorii produktów (bez konta), dodawanie ich do koszyka oraz finalizacje zamówienia po zalogowaniu. Zarejestrowani użytkownicy będą mieli możliwość podejrzenia historii swoich zamówień oraz zmiany danych wysyłki.

Autor: Jan Jankowski

Pierwsze uruchomienie:

Do uruchomienia projektu wymagany jest node.js

Sklonuj repozytorium projektu: https://github.com/yachu09/Gymfinity

(w Visual Studio) Widok -> terminal

cd do AngularApp5.Server

Upewnij się, że na pewno znajdujesz się w AngularApp5.Server (cała ścieżka może się różnić)

```
PS C:\Users\admin\source\repos\AngularApp5\AngularApp5.Server>
```

A następnie użyj komendy: dotnet build

Po zbudowaniu projektu uruchom go: dotnet run

w konsoli może wyświetlić się akceptacja udostępniania danych dla Angulara (y/n)

pod adresem https://localhost:52523/ znajduje się aplikacja

! Migracja nastąpi sama, lokalne bazy danych utworzone same, a dane do nich zostaną automatycznie dodane poprzez Seed. Nie używać poleceń do migracji, ani aktualizowania bazy !

W przypadku błędu związanego z node.js , który może wystąpić świeżo po jego pobraniu należy zresetować IDE i ponownie spróbować zbudować i uruchomić projekt

W przypadku błędu z pobieraniem pakietów: dotnet restore

Wykorzystane technologie:

.NET 8.0

Angular 19.0.5

Node.js 22.13.0

Baza danych: Microsoft SQL Server localdb

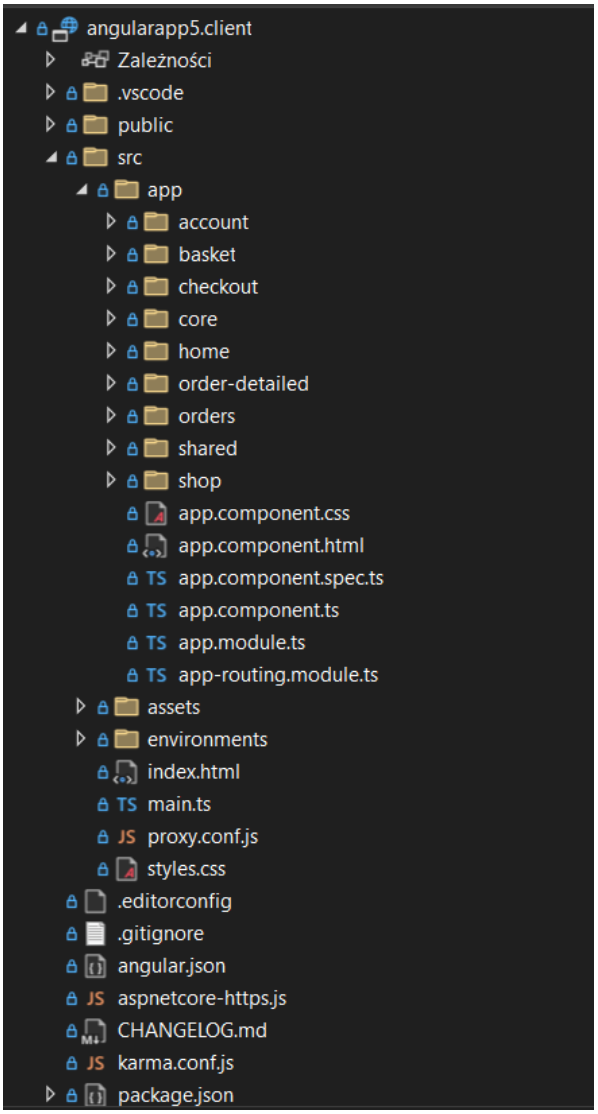
Struktura projektu:

Frontend:

Zawiera kod źródłowy Angular Single Page Application

W folderze src/app znajdują się wszystkie komponenty, moduły, serwisy oraz moduły odpowiedzialne za routing

Folder shared zawiera wszystkie wspólne części aplikacji, które exportuje SharedModule oraz interfejsy w katalogu models



Backend:

Implementuje API dla klienta aplikacji Gymfinity

Controllers – kontrolery

Data – konteksty baz danych, Seed produktów wywołujący się po pierwszym włączeniu aplikacji oraz implementacja Unit Of Work zarządzająca repozytoriami i transakcjami

Dtos – zawiera Data Transfer Objects dla modeli

Errors – customowa obsługa błędów odpowiedzi api

Extensions – rozszerzenia metod

Helpers – zawiera paginacje oraz profile mapowania AutoMappera

Interfaces

Middleware – znajduje się w nim implementacja ExceptionMiddleware

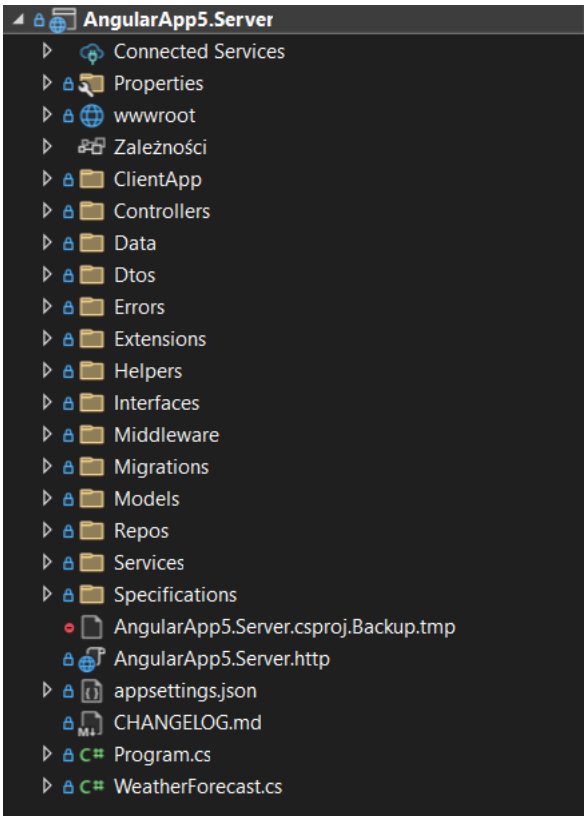
Migrations – folder przechowywujacy migracje

Models – modele bazy danych

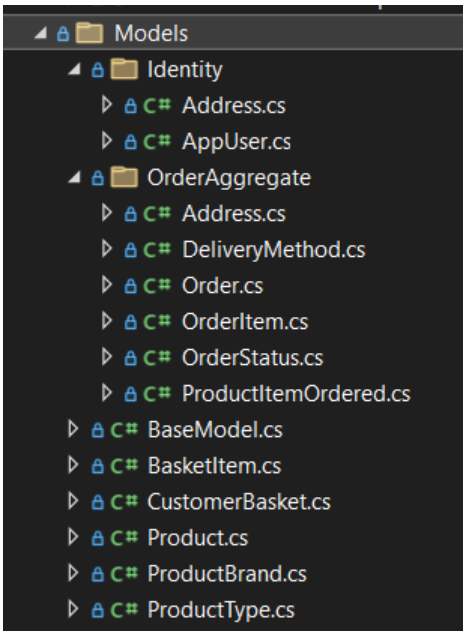
Repos- repozytoria

Service – serwisy orderów oraz tokenów JWT

Specifications – specyfikacje zapytań API



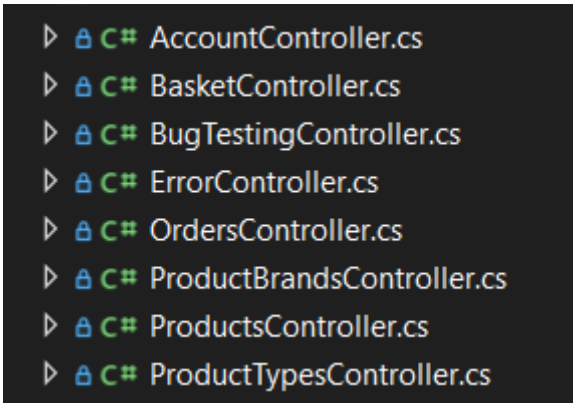
Modele:



Modele w podfolderze Identity są odpowiedzialne za dane użytkowników i należą do UserContext w bazie danych AngularApp5IdentityDB

Modele w podfolderze OrderAggregate są odpowiedzialne za wszystko co związane z zamówieniami. Razem z pozostałymi modelami od produktów oraz funkcjonalności koszyka należą do bazy danych AngularApp5DB

Kontrolery:



AccountController

GetCurrentUser() (GET) - zwraca aktualnie zalogowanego użytkownika na podstawie tokena JWT

CheckEmailExistsAsync(string email) (GET) - sprawdza czy podany adres e-mail już istnieje w bazie

GetUserAddress() (GET) - zwraca adres użytkownika, jeśli istnieje lub zwraca NoContent zamiast NotFound, jeśli adres nie jest ustawiony aby formularz w UI widniał jako pusty.

UpdateUserAddress(AddressDto address) (PUT) - aktualizuje adres zalogowanego użytkownika na podstawie przesłanych danych.

Login(LoginDto loginDto) (POST) - loguje użytkownika, sprawdzając jego dane, zwraca token JWT i dane użytkownika.

Register(RegisterDto registerDto) (POST) - rejestruje nowego użytkownika sprawdzając, czy adres e-mail nie jest zajęty. Tworzy nowego użytkownika w bazie i zwraca token JWT oraz dane usera

BasketController

GetBasketById(string id) (GET) - pobiera koszyk użytkownika lub tworzy nowy jeśli nie istnieje

PostBasket(CustomerBasket newBasket) (POST) - tworzy nowy koszyk na podstawie przesłanych danych

UpdateBasket(CustomerBasket basket) (POST, /update) - aktualizuje koszyk użytkownika podmieniając dane starego

DeleteBasketAsync(string id) (DELETE) - usuwa koszyk użytkownika

BugTestingController (wykorzystywany w celu testowania)

GetNotFoundRequest() (GET, /notfound) - zwraca błąd 404 NotFound

GetServerError() (GET, /servererror) - symuluje błąd serwera i zwraca kod 500

GetBadRequest() (GET, /badrequest) - zwraca błąd 400 Bad Request

GetValidationError() (GET, /validationerror) - zwraca błędy walidacji dla testów.

ErrorController – rzuca wyjątek w celu testowania mechanizmów obsługi błędów

OrdersController

GetOrdersForUser() (GET) - zwraca listę zamówień zalogowanego użytkownika

GetOrderByIdForUser(int id) (GET, /id) - zwraca zamówienie użytkownika na podstawie id

CreateOrder(OrderDto orderDto) (POST) - tworzy nowe zamówienie na podstawie przesłanych danych

GetDeliveryMethods() (GET, /deliveryMethods) – zwraca sposoby dostawy

ProductsController

GetProducts(`productParams`) (GET) - zwraca stronę paginacji z indeksem oraz wielkoscia strony i lista produktów zawartej na niej z możliwością filtrowania według marki i typu oraz rodzaju sortowania za pomocą specyfikacji

GetProduct(int id) (GET, /id) - zwraca szczegóły produktu na podstawie jego id

GetBrands() (GET, /brands) - zwraca listę marek

GetTypes() (GET, /types) - zwraca listę typów produktów

Inne nie wymienione wyżej kontrolery zawarte w rozwiązaniu projektu były wykorzystywane tylko we wczesnych etapach budowy i testowania API

System użytkowników:

System użytkowników opiera się na Microsoft Identity oraz tokenach JWT. W celu obsługi tokenów zaimplementowany został TokenService, a kontroler funkcjonuje na UserManager oraz SignInManager z Identity. Podczas logowania/rejestracji szyfrowany jest nowy token i zostaje przypisany użytkownikowi. Tokeny użytkowników przechowywane są w pamięci lokalnej przeglądarki i wykorzystywane do autoryzacji zapytań w API. Podczas wylogowania się z aplikacji token zostaje usunięty z pamięci. Zalogowany użytkownik ma dostęp do zamawiania produktów, aktualizowania swoich danych do wysyłki oraz przeglądania historii zamówień.

Ciekawe funkcjonalności:

Implementacja koszyka - koszyk jest elementem wspólnym dla gości jak i użytkowników. Ma on swoje id i jest przechowywany w pamięci przeglądarki przez co jeśli dodamy produkty do koszyka przed zalogowaniem lub utworzeniem konta, nasze dodane produkty wciąż znajdują się w koszyku. Koszyk znajduje się w bazie i jest on likwidowany w przypadku usunięcia z niego wszystkich produktów lub zrealizowania zamówienia.

Etapy składania zamówienia – są utworzone za pomocą Stepper który implementuje reaktywne formularze. Jest to ciekawsze rozwiązanie niż klasyczne formularze

Wdrożenie Pop-up okienek Toastr w celu powiadamiania użytkownika o błędach lub złożonych zamówieniach

Zaawansowana paginacja, filtrowanie oraz sortowania produktów - Metoda GetProducts w kontrolerze od produktów pobiera z zapytania specyfikacje danych o wielkości strony, numerze strony, rodzaju sortowania oraz typach i markach a następnie tworzy na ich podstawie specyfikacja i przekazuje do repozytorium, a po otrzymaniu wyników tworzy obiekt Pagination

```
public class Pagination<T> where T : class
{
    1 odwołanie
    public Pagination(int pageIndex, int pageSize, int count, IReadOnlyList<T> data)
    {
        PageIndex = pageIndex;
        PageSize = pageSize;
        Count = count;
        Data = data;
    }

    1 odwołanie
    public int PageIndex { get; set; }
    1 odwołanie
    public int PageSize { get; set; }
    1 odwołanie
    public int Count { get; set; }
    1 odwołanie
    public IReadOnlyList<T> Data { get; set; }
}
```

Wzorzec Unit of Work oraz repozytorium generyczne – implementacja UnitOfWork pozwala na zarządzanie bazą danych z jednej instancji StoreContext, dzięki czemu możliwa jest transakcyjność i spójność danych. Dzięki temu w przypadku zapisywania wielu danych do bazy jak np. w przypadku

tworzenia zamówienia zapiszą się albo wszystkie zmiany albo żadne.
Repozytorium generyczne natomiast pozwala na przyjmowanie różnych typów i wykonywania na nich tych samych metod co pozwala na dużą skalowalność systemu.