Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

випускна кваліфікаційна робота бакалавра

<<СИСТЕМА УПРАВЛІННЯ КОНТЕНТОМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ (за прикладом сервісу Amazon)>>

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»
Виконавці проекту

№	П.І.Б.	Ролі	Група
1	Семенюк Олег Олегович	Backend	КН-П-201
2	Пшеничний Олександр Сергійович	Backend	КН-П-201
3	Вацько Ірина Олександрівна	Frontend	КН-П-201
4	Рибальченко Денис Леонідович	Frontend	КН-П-201
5	Гетманцева Вікторія Юріївна	Designer	КН-Д-201
6	Гарбуз Дар'я Вячеславівна	Designer	КН-Д-202
7	Петров Всеволід Едуардович	DevOps	КН-А-201

Автор звіту: Семенюк Олег Олегович

Одеса – 2024

АНОТАЦІЯ

Ця робота присвячена аналізу та розробці системи управління контентом електронної комерції на базі найбільшого в США інтернет магазину Amazon. Цей документ описує реалізацію та аналіз серверної частини додатку.

Реалізація інтернет-магазину є актуальною на сьогоднішній день, і на це є декілька причин, по перше це зручність, останім часом, багатьом людям буде набагато зручніше подивитися необхідні товари в інтернет-магазині використовуючи смартфони, зробити замовлення лише в декілька кліків та обрати доставку. Для покупців також є можливість ознайомитись з товаром прочитавши відгуки людей які вже купили продукт, ознайомитися з товаром за допомогою відеоогляду, подивитися детальні характеристики продукту. По-друге пандемія COVID-19 яка зіграла не малу роль, багато покупців, які раніше взагалі не користувалися онлайн-послугами, почали робити покупки онлайн, навіть після зняття карантинних обмежень, значна кількість людей продовжує робити онлайн закупівлі.

В результаті роботи буде представлений інтернет магазин, де кожен користувач може дивитися велику кількість товарів різних категорій. Шукати необхідні товари та фільтрувати за певними параметрами. Можливість додавати товар до кошику, можливість робити замовлення.

Цей документ описує роботу серверної частини додатку. Також описуються відомості про технології та інструменти які були використані під час розробки.

ЗМІСТ

ВСТУП

Онлайн-сервіси на сьогоднішній день стали невід’ємною частиною нашого життя. Вони пропонують безліч інструментів які дозволяють вирішити проблеми багатьох сфер. Однією із цих сфер є торгівля, в останні часи онлайн-шопінг набув надзвичайної популярності, дозволяючи покупцям отримати бажаний товар в декілька кліків використовуючи будь який пристрій який має вихід в інтернет. Інтернет-магазини надають покупцеві доступ до великої кількості товарів різних категорій, де він може обрати речі які дозволять вирішити його потреби.

Даний документ описує аналіз та реалізацію інтернет-магазину, який дозволить вирішити потреби покупця. Результатом цієї роботи буде сервіс, що надає зручний інтерфейс, для перегляду товарів, можливості фільтрації та сортування товарів, детальні описи та чіткі зображення, відгуки від інших користувачів сервісу.

Документ складається з технічного завдання, в якому описуються вимоги для реалізації інтернет-магазину, з загальної характеристики системи, де описуються всі її частини і як вони взаємодіють між собою, розділ який описує більш детально серверну частину додатку, в якому є відомості про технології та інструменті які були застосовані для реалізації проєкту, розділ який описує адміністративну частину, висновки та перелік джерел, використаних для вирішення задач.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ ЕЛЕКТРОННОГО МАРКЕТИНГУ (за зразком Amazon)

Загальний опис.
Програмне забезпечення (ПЗ) призначене для створення розподіленої системи управління контентом, який призначений для електронної комерції. Функції системи мають забезпечити повний життєвий цикл управління контентом: створення, актуалізація, модифікація та знищення. Головна особливість системи полягає у тому, що вона передбачає механізми узгодження контента з віддаленими джерелами даних , а також управління замовленнями, їх оформленням, статусом, а також зберіганням та переданням їх користувачу або зовнішнім системам (платіжна система, служба доставки, тощо).

Призначення:
ПЗ орієнтується на створення контейнеру управління інформаційними одиницями комерційного призначення. Значна увага приділяється маркетинговим інструментам, що дозволяють проводити дослідження ефективності роботи системи, зокрема, зворотному зв'язку між сторонами комерційної взаємодії. ПЗ створюється за прикладом мережевого магазину, у той же час має складатись за принципом універсальності, мати змогу змінити цільове призначення.

Вимоги до функціоналу. Обов'язкова частина
Головна сторінка
Стартовий інтерфейс ПЗ (головна сторінка або активність) має за мету популяризацію системи. На ньому має бути забезпечено відображення популярних показників як-то загальна кількість користувачів та комерційних одиниць (або їх груп), вибірка з найбільш популярного та новітнього контенту, а також нових дописів (відгуків), зовнішня реклама. Головна сторінка обмежує функції системи для незареєстрованих користувачів лише переглядом, інші функції надаються у разі авторизації користувача. Має бути забезпечений пошук контенту, групування та фільтрування результатів за різними ознаками, наявними у знайденому контенті. Для об'ємних результатів має бути забезпечена пагінація.

Реєстрація користувачів:
Користувач ПЗ має мати змогу зареєструватись за допомогою авторизаційних даних (наприклад електронної пошти та паролю).
При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - номер телефону, електронна пошта тощо. На тестовому етапі, за умови ускладнення замовлення послуг телефонії чи СМС, можливе використання лише електронної пошти. Засіб оперативного контактування (ЗОК) при реєстрації підлягає верифікації через введення коду з надісланого листа на введену пошту користувачем. Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.
За умови що користувач забув пароль, ЗОК має бути використаний для його відновлення. Схема відновлення паролю жорстко не регламентується.
Також передбачається наявність у системі користувачів з додатковими правами: модераторів та адміністраторів. Реєстрація кореневого адміністратора системи має бути реалізоване при інсталяції системи, наступним користувачам змінюють права наявні адміністратори.

Особистий кабінет користувача:
Основне робоче місце зареєстрованого користувача ПЗ. Повинен забезпечити усі необхідні засоби для

- Управління власними замовленнями: створення, видалення, перегляд усіх замовлень з можливістю більш детального огляду
- Встановлення, зміна та вилучення персональних даних, у т.ч. платіжного засобу, зміна чи відновлення авторизаційних даних (паролю)
- Перегляд поточної активності: історії пошуку, перелік відзначеного контенту (кошик), виконані замовлення.
- Формування зворотного зв'язку: надання оцінок, складання відгуків та рекомендацій

Також засобами особистого кабінету має бути передбачено можливість зміни персональних даних.

Зворотній зв'язок
Користувач має змогу встановити власну оцінку (лайк або бал) для довільного контенту, щодо якого зафіксована його активність (замовлення). За умови виконаного замовлення (купівлі) користувачеві надається можливість створювати відгуки та рекомендації щодо контенту.

Засоби пошуку, фільтрування та групування мають забезпечити врахування рейтингу контенту.

Вимоги до функціоналу. Опціональна частина

Модерація

Для забезпечення дотримання загальноприйнятих норм спілкування, а також з метою недопущення публікування контенту, який містить заборонені компоненти (ненормативні висловлювання, заклики до протиправних дії, приниження чи дискримінація, тощо), усі створені користувачами дані (відгуки) мають в обов'язковому порядку пройти погодження (модерацію). Якщо контент порушує норми, кореневий адміністратор має можливість видаляти подібний контент.

Архітектура

ПЗ має бути спроектоване за клієнт-сервеною розподіленою архітектурою з чітко відокремленими структурними вузлами системи:
Mockup - візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.
Backend - серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.
Web App - додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів усіх категорій
Mobile App (опціонально) - додаток, що встановлюється на мобільних пристроях (смартфон, планшет).
Усі модулі ПЗ (окрім візуальної моделі Mockup) повинні обов'язково включати до свого складу засоби

- розгортання на новому пристрої (інсталяції) із встановленням початкових таблиць у БД та кореневого користувача-адміністратора

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:

- root
    - Backend
    - Frontend
    - Mockup (вихідні файли або посилання на зовнішній проект)
    - DevOps

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

База даних - MySQL
Backend - ASP .NET Core Web-API.
Frontend - JS фреймворк Next.js на основі бібліотеки React
Mockup - визначається ментором з дизайну
DevOps - GitLab, Docker, MySQL, WireGuard VPN

Команда проєкту

Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проекту.
Мінімальна команда проекту - чотири учасники, у такій команді кожен з учасників відповідає за одну з частин: Backend, Frontend, Mockup, DevOps.
Дозволяється включення до команди більшої кількості учасників, у такому разі нові учасники будуть розподілені між існуючих частин (Mockup, Backend, або Frontend).

Звітність

За результатами виконання завдань із створення ПЗ кожен з учасників команди проекту складає власний (індивідуальний) звіт, що складається з практичної частини та автореферату. Практична частина є прямим результатом роботи і подається у вигляді посилання на репозиторій (чи інший URL) або у вигляді архіву на довільному носієві даних. Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.
Автореферат містить описову частину, у якій розкриваються основні концепти проекту, деталізуються завдання до виконавця, встановлюється актуальність роботи, її об'єкт та предмет, обґрунтовується вибір технологій та відзначається шлях виконання та ступінь досягнення поставлених задач, посилання на джерела інформації.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Проаналізувавши технічне завдання, командою було затверджено основні частини системи. Складається вона з:

- Frontend
- Backend
- Design
- DevOps

Frontend – частина системи яка надає візуальний інтерфейс для сайту. Взаємодіє з backend, надсилаючи HTTP-запити, для отримання або передачі відповідних до запиту даних, відповіді від сервера зберігаються у кеші, щоб зменшити час очікування одного і того ж запиту. Frontend це односторінковий додаток(SPA) який реалізований за допомогою фреймворку Next.Js та бібліотеки React, також у додаток включений автономний сервер який рендерить сторінки. У реалізації були досягнені наступні цілі:
Адаптивний дизайн – веб-сайт має адаптуватись під різні розміри екрану, зберігаючи оригінальний дизайн, але підлаштовуючись під конкретні пристрої. Увагу було приділено і для мобільних пристрої, оскільки переважна частина користувачів користується мобільними телефонами.
Послідовний дизайн – більшість елементів повинні бути стандартизовані і мати однаковий вигляд, колірну гамму, стан, тощо.

Backend це невід'ємна частина системи, яка реалізує основний функціонал сайту. Серверна частина розроблена за допомогою фреймворку ASP .NET Core. Backend реалізує REST API що дозволяє частині frontend звертатися до серверу за допомогою HTTP-запитів. Кожен запит обробляється, валідується, в залежності від результату валідації робляться наступні дії. Якщо результат валідації не задовільний, сервер повертає у відповідь на запит, структуру, яка описує кожну помилку. Задовільний результат передає запит у відповідний обробник, кожен обробник виконує свою роботу з базою даних що включає, створення, оновлення, видалення та читання даних.

Design – частина системи яка відіграє ключову роль. Її роль це не тільки створення візуальної частини, ще це аналіз цільової аудиторії, створення продуманої структури, яка допомагає розставити акценти та привернути увагу. Для виконання дизайну були виконані наступні роботи: аналіз конкурентів, була створена карта сайту, яка дозволяє визначити зв’язок

між сторінками сайту. Wireframe: схема яка визначає розташування елементів на сторінках, були розроблені такі елементи як логотип сайту, була обрана колірна гамма, графічні елементи, шрифти, іконки та Ui-Kit. Були розроблені прототипи які симулюють готовий продукт.
Для розробки цієї частини були використані наступні інструменти: Notion використовується для аналізу конкуренції, Figma інструмент за допомогою якого були розроблені макети сторінок, створення Ui-kit, карти сайту тощо, Adobe Illustrator використовувався для створення графічних елементів, Adobe InDesign створення брендбуку.

DevOps відіграє важливу роль у системі. Його функції включають реалізацію CI/CD, який дозволяє застосувати нові зміни у частинах Frontend та Backend та одразу впровадити їх в робоче середовище. Завдяки цьому підходу можна швидко додавати новий функціонал, виправлення помилок, у системі автоматизовані процеси тестування та розгортання. Для створення інфраструктури, були застосовані сервери нашого університету, доступ до яких надається за допомогою VPN WireGuard. Для моніторингу системи серверів використовуються такі сервіси як Prometheus та Grafana

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Для розробки серверної частини були використані наступні технології та додаткові пакети:

- Мова програмування: C#
- Середовище розробки: Visual Studio 2022
- Фреймворк: ASP .NET Core 8
- Технологія: Entity Framework Core
- Додаткові пакети: MySql.EntityFrameworkCore, MediatR, Auto-Mapper, FluentValidation, Newtonsoft JSON, Slugify, JwtBearer
- СУБД: MySQL
- Середа для адміністрування бази данних: MySQL Workbench
- Система контролю версій: Git
- Платформа спільної розробки: GitHub, GitLab
- SDK: AWSSDK.S3
- Тестування: Postman

Для реалізації серверної частини використовувався фреймворк ASP .NET Core. Сервер реалізує REST API який дозволяє виконувати стандартні HTTP операції (GET, POST, PUT, DELETE) для роботи з даними.
Сервер реалізує дві частини:

- Основна частина – реалізує весь необхідний функціонал для покупця. Майже увесь функціонал доступний тільки для авторизованих користувачів.
    - Авторизація на сайті реалізована за допомогою JWT-токенів, всі токени зберігаються у базі даних. Для чого це необхідно? Нам достатньо того що токен вже зберігає інформацію про користувача, та кожен токен має свій час роботи, але є деякі ситуації, наприклад якщо користувач вийшов зі свого профілю, токен необхідно деактивувати, якщо змінити сам токен то він вже по суті буде новим. А старим токеном ще можна користуватись поки його строк дії не завершився. Для запобігання цієї ситуації, токени зберігаються в БД, для валідації токену застосовується ряд перевірок наприклад: перевірка валідності токену, перевірка токену у базі чи не деактивований він, отримати користувача за токеном, та перевірити чи не видалений це користувач, чи взагалі існує цей користувач. В адміністративній частині є додаткова перевірка на роль користувача.
    - Створення нового акаунту розподілено на декілька етапів:

        - Користувач має вказати свою електронну пошту та пароль
        - Користувач має підтвердити свою пошту
        - Користувач має надати своє Прізвище та Ім’я.

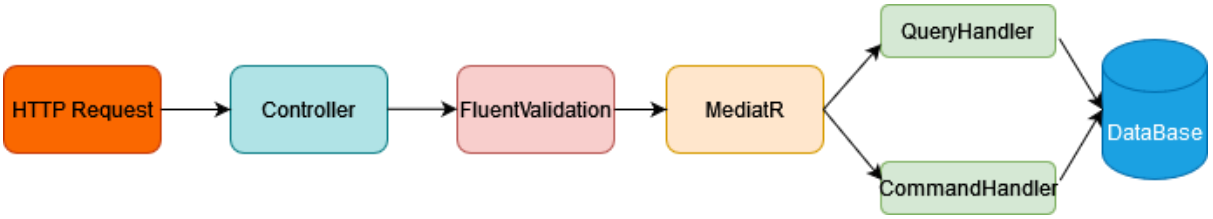
      Після першого етапу користувачу надається авторизаційний токен.
    - Авторизований користувач має змогу додавати товари у кошик, чи список бажань. Також користувач має змогу замовляти товари. Авторизований користувач має змогу залишати відгуки та оцінювати товар, є можливість ставити вподобання іншим відгукам.
    - Для усіх користувачів, є можливість дивитися товари усіх категорій, за допомогою фільтрів чи пошуку шукати необхідний товар. Читати відгуки та дивитися рейтинг товарів.
- Адміністративна частина – реалізує функціонал для адміністратора сайту:
    - Адміністратор має змогу створювати нові категорії та задавати правила для товарів які можуть знаходитись у цій категорії.
    - Адміністратори мають змогу наповнювати сайт товарами. Товари можна читати, оновлювати, створювати та видаляти.
    - Адміністратори мають змогу дивитися відгуки користувачів, та видаляти їх якщо вони порушують правила сайту.
    - Адміністратори можуть видаляти користувачів які порушують правила сайту, також має можливість дивитися користувачів, змінювати їм ролі, відновити акаунт користувачам.

Для архітектури серверної частини був використаний стиль CQRS. Цей підхід дозволяє відділити операції читання від операції запису. Таким чином операції серверу діляться на дві частини

- Команди – це частина додатку яка вносить зміни у стан додатку (у нашому додатку це база даних). Команди виконують створення, видалення, зміну екземплярів сутності(таблиці) бази даних. Наприклад: реєстрація нового користувача, при реєстрації ми маємо створити нового користувача, внести його дані у базу даних, ця задача вважається командою.
- Запити – це частина додатку, яка не вносить ніяких змін у стан додатку. У нашому додатку запити використовуються для створення різних вибірок з бази даних. Наприклад: отримати дані про авторизованого користувача, ця задача не потребує ніяких змін, а

лише отримати одного користувача, і повернути його дані, ця задача вважається запитом.

Вихідний додаток складається з компонентів, які ніяк не впливають на роботу один одного, у такий спосіб можна легко масштабувати проєкт. Завдяки такому розділенню, є можливість окремо оптимізувати процеси читання(для таких операцій застосувати кешування відповідей) а операції запису зробити надійними, забезпечуючи валідацією даних і обробку змін. Для реалізації CQRS був використаний додатковий пакет MediatR.



У схемі вище продемонстровано етапи обробки HTTP-запиту на сервері. Розглянемо окремо усі етапи через котрі проходять запити.

- По-перше необхідно розглянути об’єкт запиту який приходить у контроллер:

```
public class GetProductsQueryRequest : IRequest<Result<List<ProductCardProfile>>>
{

    public int pageSize { get; set; }

    public int pageIndex { get; set; }

    public string? searchQuery { get; set; }

    public bool? discount { get; set; }

    public int? categoryId { get; set; }

    public string? rating { get; set; }

    public string? price { get; set; }

    public string? orderBy { get; set; }
}
```

На зображенні демонструється об’єкт запиту який фільтрує товари за певними параметрами

Кожен об’єкт запиту має реалізувати дженерік інтерфейс IRequest<T> де T це об’єкт який буде повертатися після завершення роботи обробника запиту(Query/Command Handler) як результат. За допомогою Nullable-типів (наприклад: string? searchQuery) ми вказуємо які поля є опціональними.

У нашій системі повертається об’єкт Result який зберігає відомості

про успіх виконання операції, це статус код, повідомлення, та дані. Далі цей об’єкт додається у відповідь на запит.

- Сервер отримує запит та за ендпоінтом визанчає який контролер має обробити цей запит. Контролер за ендпоінтом визначає який екшн має обробити запит. Метод контролеру(action) – обробляє HTTP-запит.

```
[HttpGet]
0 references | dsgnrr, 14 days ago | 1 author, 2 changes
public async Task<IActionResult> GetProducts([FromQuery] GetProductsQueryRequest request)
{
    var validationErrors = _prodByCategoryValidator.GetErrors(request);
    if (validationErrors != null)
    {
        return _responseService.SendResponse(HttpContext, StatusCodes.Status400BadRequest, "Bad
          request", validationErrors);
    }
    var response = await _mediator.Send(request);
    if (response.isSuccess)
    {
        return _responseService.SendResponse(HttpContext, StatusCodes.Status200OK, "Ok", response.data,
            new() { currentPage = request.pageIndex, pagesCount = response.pagesCount });
    }
    return _responseService.SendResponse(HttpContext, response.statusCode, response.message,
      response.data);
}
```

На зображенні демонструється метод(екшн) який обробляє запит отримання товарів за певними параметрами.
У нашій системі методи контролеру роблять лише дві задачі, передає об’єкт запит у валідатор, за результатом валідації визначається робиться наступне, при відсутності помилок, параметри запиту передаються у MediatR який визначає за типом запиту відповідний обробник. Після завершення роботи обробника повертається результат роботи, який відправляється у якості відповіді на запит.

- FluentValidation – це додатковий пакет, який був встановлений за допомогою менеджеру пакетів платформи .NET NuGet. У нашій системі для кожного запиту створюється свій валідатор, роль валідатора виконує саме FluentValidation.

```
1 reference | dsgnrr, 14 days ago | 1 author, 3 changes
public class GetProductsByCategoryValidator : AbstractValidator<GetProductsQueryRequest>
{
    – references | dsgnrr, 14 days ago | 1 author, 3 changes
    public GetProductsByCategoryValidator()
    {
        RuleFor(x => x.pageSize).NotEmpty().GreaterThan(0);
        RuleFor(x => x.pageIndex).NotEmpty().GreaterThan(0);
        RuleFor(x => x.price).Must((price) =>
        {
            var parsedPrice = price!.Split('-');
            if (parsedPrice.Length != 2) return false;
            foreach (var item in parsedPrice)
            {
                if (!int.TryParse(item, out var _)) return false;
            }
            if (int.Parse(parsedPrice[0]) > int.Parse(parsedPrice[1])) return false;
            return true;
        }).When(x => x.price != null).WithMessage("Inccorect format. Use: minPrice-maxPrice");
        RuleFor(x => x.rating).Must((rating) =>
        {
            var parsedRating = rating!.Split(",");
            foreach (var item in parsedRating)
            {
                if (int.TryParse(item, out var rate))
                {
                    if (rate < 1 || rate > 5) return false;
                }
                else return false;
            }
            return true;
        }).When(x => x.rating != null).WithMessage("Rating must be integer. Rating must be greater than 0 and less than 6");
    }
}
```

Для створення необхідно створити клас валідатор успадкувавши батьківський клас дженерік AbstractValidator<T> де T це об’єкт для якого будуть прописані правила. Для кожного параметру запита необхідно прописати своє правило валідації. Валідатори використовуються як DI сервіси, тому в класі який створює об’єкт додатку, необхідно прописати створення сервісу валідації

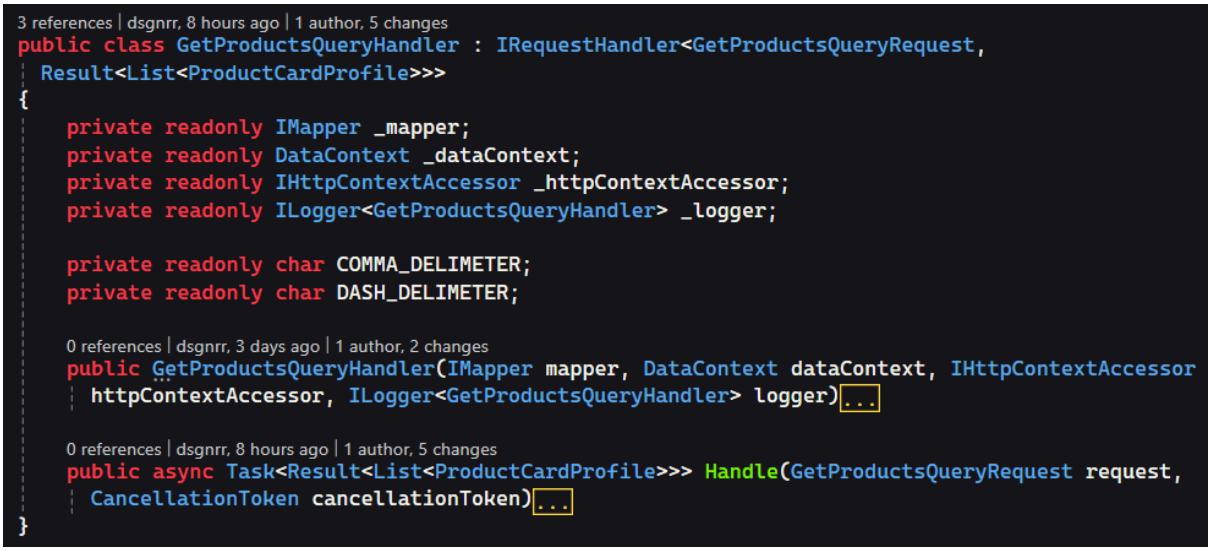
```
builder.Services.AddValidatorsFromAssemblyContaining<Program>();
```

До сервісів будуть включені всі валідатори які знаходяться в одній збірці з классом Program який будує додаток.

- MediatR – спеціальний об’єкт який знаходиться у методі контролера, та надсилає об’єкт запиту у відповідний обробник. Медіатор, по типу об’єкту запита визначає у який саме обробник необхідно передати об’єкт запиту. Як і валідатор mediatR необхідно зареєструвати у DI сервісі додатку.

```
builder.Services.AddMediatR(cfg => cfg.RegisterServicesFromAssembly(typeof(Program).Assembly));
```

- Query/Command Handlers – це спеціальні обробники які виконують команду або запит.

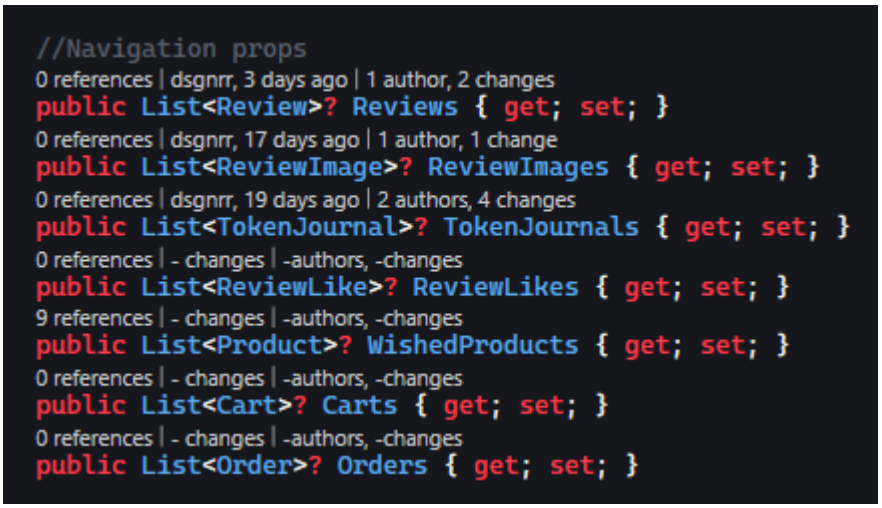


На зображенні демонструється класс обробника запиту. Кожен обробник має реалізувати інтерфейс з одним методом Handle. IRequestHandler<TRequest, TResponse> інтерфейс дженерік, де TRequest це об’єкт запиту, TResponse об’єкт відповіді.
Handle метод асинхронний тому що в ньому виконується робота з базою даних, таким чином робота з БД виконується більш ефективно, зменшуючи блокування потоків.

Для зберігання інформації про сутності системи використовується реляційна база даних(БД) MySql. Для роботи з БД використовувався підхід code first. Для того щоб працювати з БД у такий спосіб, була використана ORM Entity Framework Core, за її допомогою були розроблені усі сутності БД. Перед тим як створити БД, вона попередньо були спроєктована слідуючи трьом принципам нормальних форм.

На зображенні демонструється приклад однієї із сутностей БД. Користувач.

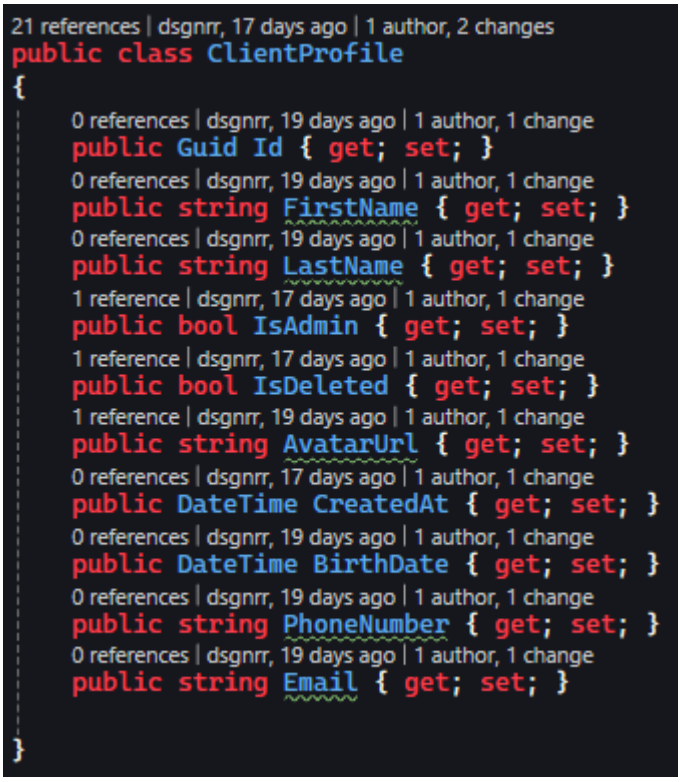
Для побудови зв’язків між сутностями використовуються навігаційні параметри:



Для створення запитів до бази даних була використана мова інтегрованих запитів LINQ, яка дозволяє будувати ланцюг запитів.

Для великої вибірки даних обов'язково застосовується пагінація. Методи що роблять подібні вибірки, містять у запиті інформацію про пагінацію(кількість елементів які потрібно отримати, поточна сторінка). Кожен з цих методів також включає параметри фільтру, які додатково дозволяють відфільтрувати. За замовчуванням сортування виконується за датою створення, але є можливість вказати стовпець за яким потрібно сортувати дані.

У сутностях які будуть повертатися є зайві дані, для того щоб їх не повертати у сутності над полями можна прописати атрибут [JsonIgnore], і такі дані у відповіді не будуть відображатися, але є різні ситуації де деякі дані знадобляться, а у іншій ситуації ці ж дані будуть зайвими.
Для вирішення цієї проблеми використовується проекція сутностей БД на їх профілі які містять лише необхідні дані, реалізовано це за допомогою додаткового пакету AutoMapper. На кожну ситуацію я можу створити профіль для сутності і легко в одному місці її змінювати, також AutoMapper дозволяє прописувати конфігурацію для кожної властивості профіля.



На зображенні демонструється профіль який використовується у кабінеті користувача.
Окремо фронтенд-розробників була створена спеціальна документація, яка описує кожен ендпоінт та його вимоги у проєкті. Документ містить інформацію про структури відповіді кожного ендпоінту, є інформацію про

успішне виконання, а також яка структура повертається у разі не успішного виконання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ АДМІНІСТРАТИВНОЇ ЧАСТИНИ

Адміністративна частина реалізована в одному додатку з основною частиною сайту. Для цієї частини всі операції також розділені на Команди та Запити.

Адміністративна частина необхідна для керування контентом на сайті. Адміністратор має змогу створювати нові категорії та задавати правила для товарів які можуть бути у цій категорії. Для кожної категорії додається опис, та своє зображення, усі зображення зберігаються на S3 бакеті, за допомогою AWSSDK.S3 був реалізований сервіс який дозволяє завантажувати зображення на бакет, категорія лише зберігає посилання на це зображення.

Адміністратор має змогу створювати нові товари, змінювати та видаляти вже існуючі. Товари як і категорії містять в собі зображення, вони також зберігаються на бакеті AWS сервісу S3.

ВИСНОВКИ

Було зроблено дуже багато роботи з точки зору аналізу та самої реалізації. На старті було дуже багато ідей, командою обговорювались всі можливості сайту. Багато функціоналу залишилось на майбутню реалізацію, одна із таких комбінації для товару, кожна комбінація зберігає свої характеристики, та кожна з цих комбінацій відносилась до одного товару. На майбутню реалізацію залишився і сервіс для продавця, кожен продавець мав би змогу створювати свої товари, передивлятися статистику за існуючими товарами.

За допомогою архітектурного патерну CQRS, проєкт можна легко масштабувати розділяючи кожну операцію на команду або запит, при цьому не буде ніякого впливу вже на існуючий функціонал, кожна частина серверу відповідає за свою роботу.

На сьогодні ми маємо додаток який дає змогу передивлятись товари, шукати, фільтрувати за певними характеристиками, дивитись відгуки, дивитись на рейтинг товарів. Для авторизованих користувачів є можливість зберігати товари у своїх списках бажань, додавати у кошик товари які сподобалися, та робити замовлення чи відміняти їх.
Також авторизований користувач має змогу змінювати дані свого профілю, і має можливість видалити свій акаунт.

На сайті є можливість створити новий акаунт покупця, для авторизації користувачів був впроваджений JWT-токен. Усі паролі користувачів шифруються за допомогою хеш-функції bcrypt, яка кожен раз генерує унікальний хеш, це означає якщо у декількох користувачів повторюються паролі, у кожного все одно буде свій хеш. Такий підхід ускладнює процес злому паролю аналізуючи кожен хеш.

Репозиторій з вихідним кодом серверної частини опублікований на GitHub.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Мова програмування C#
- ASP .NET Core
- Entity Framework Core
- EF LINQ
- CQRS
- MediatR
- Auto-Mapper
- FluentValidation
- AWSSDK
- Postman
- MySQL
- ChatGPT
- GitHub
- Git docs
- Stack Overflow
- Slugify
- NewtonsoftJSON
- YouTube
- Medium
- Microsoft Learn