Приватний заклад вищої освіти

Одеський технологічний університет «ШАГ»

Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота

«Система управління контентом електронної комерції

(за прикладом сервісу Comfy) (Backend)»

на здобуття бакалаврського рівня вищої освіти

зі спеціальності «122 Комп’ютерні науки»

Виконавці проєкту:

№	П.І.Б.	Група	Роль
1	Єлізар’єв Вадим Антонович	КН-П-191	Розробник (backend)
2	Бондарчук Михайло Євгенович	КН-П-191	Розробник (frontend)
3	Ісаєнко Дмитро Олександрович	КН-П-191	Розробник (frontend)
4	Сідху Тіана	КН-Д-192	Дизайнер
5	Лавров Андрій Вікторович	КН-А-191	Адмін

Автор звіту: Єлізар’єв Вадим Антонович

Одеса – 2023

АНОТАЦІЯ

Дипломний проєкт присвячений розробці Backend частині онлайн-гіпермаркету на прикладі Comfy.ua. Робота включає аналіз, розробку та тестування онлайн-платформи для електронної торгівлі з великим асортиментом товарів.

Онлайн-гіпермаркет надає користувачам зручний спосіб придбати товари онлайн та пропонує широкий вибір товарів різних категорій, переважно електроніку. Платформа має різноманітні функції, включаючи реєстрацію, автентифікацію, двофакторну автентифікацію, авторизацію, пошук, фільтрацію та сортування товарів, систему відгуків та питань до товарів та інше.

Проєкт актуальний, оскільки онлайн-торгівля стає все більш популярною серед користувачів, особливо в контексті зростання електронної комерції. Розробка онлайн-гіпермаркету дозволяє забезпечити зручну платформу для покупок, широкий вибір товарів та зручність у використанні.

Одним з ключових аспектів проєкту є реалізація безпеки та захисту персональних даних користувачів. Використовуються сучасні технології шифрування даних, захисту від несанкціонованого доступу та злому системи.

Результатом проєкту буде функціональний та естетично збалансований онлайн-гіпермаркет, який забезпечить користувачам зручну платформу для покупок з широким вибором. Проєкт передбачає можливість подальшого розширення та вдосконалення з урахуванням змінних потреб ринку та клієнтів.

Для розробки проєкту на стороні Backend використовуються мова програмування C#, фреймворк ASP.NET Core 6, реляційна база даних PostgreSQL та нереляційна база даних Redis, яка використовується як кеш.

ЗМІСТ

ВСТУП

Інтернет-технології надають безліч можливостей для спрощення і поліпшення різних сфер життя, включаючи торгівлю. Зростання популярності онлайн-шопінгу та зростання попиту на зручні та ефективні способи покупок стимулюють розвиток глобального електронного ринку. У цьому контексті виникає потреба в рішеннях, що дозволяють споживачам здійснювати покупки онлайн зі зручністю та ефективністю, а продавцям - розширювати свій бізнес та досягати нових ринків та аудиторій.

Онлайн гіпермаркети виступають як відповідь на ці вимоги, поєднуючи широкий асортимент товарів, зручний інтерфейс та безліч послуг, що полегшують процес покупок.

У даному документі розглядається концепція і розробка онлайн гіпермаркету, який відповідає потребам споживачів і сприяє ефективному функціонуванню бізнесу. Основною метою дослідження є створення інноваційної цифрової платформи, яка забезпечує безперервний доступ до широкого спектра товарів, зручну навігацію та високу якість обслуговування.

Автореферат розпочинається аналізом технічних вимог до проєкту. Далі проводиться проєктування архітектури онлайн гіпермаркету, враховуючи вимоги зручності, безпеки та ефективності. Потім розглядаються етапи розробки, включаючи вибір технологій, розробку серверної частини.

Висновки дослідження демонструють результати, отримані під час розробки, тестування та використання онлайн гіпермаркету. В цілому, автореферат дозволяє зрозуміти сутність та переваги цього рішення.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

Загальний опис

Програмне забезпечення для створення інформаційної системи інтернет крамниці для організації роздрібної інтернет торгівлі.

Призначення

Надання в Інтернет просторі інформації про доступні товари (оферти продавця) та організації оформлення замовлень (акцепту покупця) через мережу Інтернет. Оперативне управління веб-контентом Інтернет крамниці. Збереження даних та регулювання доступу до них. Організація взаємодії з іншими інформаційними системами.

Вимоги до функціоналу:

- Інтернет крамниця
    - Навігація по веб вітрині крамниці
    - Пошук товарів за критеріями
    - Реєстрація покупців
    - Формування кошика покупця
    - Оформлення замовлень
    - Зворотній зв'язок та відгуки
- Управління контентом інтернет крамниці
    - Навігація, перегляд, створення, редагування структури та контенту
    - Керування доступом та розподіл дозволів користувачів
    - Створення вітрини товарів
    - Управління даними користувачів
    - Оформлення замовлень
    - Модерація коментарів

- Реєстрація та авторизація користувачів
    - Користувач ПЗ має змогу зареєструватись чи увійти у систему як за допомогою авторизаційних даних (email та пароль), так і за допомогою облікових записів популярних соціальних мереж (Google, тощо)
    - Користувач ПЗ має змогу увімкнути та вимкнути функцію двофакторної автентифікації
- Особистий кабінет покупця
    - Перегляд історії замовлень
    - Кошик покупця
    - Реквізити для платежів та доставки
- Взаємодія із інтернет додатками
    - Web-API моделі та запити для управління аккаунтом користувача
    - Web-API моделі та запити для одержання, створення та редагування інформації, що формує профіль покупця
    - Web-API моделі та запити для вибору товару в корзину та оформлення замовлення користувачем
    - Web-API моделі та запити для створення питань та відгуків, а також для їх оцінок (лайків/дізлайків)
    - Web-API моделі та запити для пошуку товарів з необхідними фільтрацією та сортуванням
    - Web-API моделі та запити для отримання категорій товарів, банерів, тощо

Архітектура:

- Сервер бази даних - серверне програмне забезпечення, що забезпечує збереження та обробку поточних даних.
- Сервіс Web-API – серверне програмне забезпечення, що надає програмний інтерфейс для взаємодії (REST-API) з іншим програмним забезпеченням.
- Веб додаток керування контентом - доступний з веб-простору сайт, що забезпечує візуальний інтерфейс для менеджерів системи.
- Веб крамниця - доступний з веб-простору сайт, що дозволяє взаємодіяти з інтернет-магазином покупцям.

Технології:

- Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології та методології створення:
    - Бази даних – PostgreSQL, Redis
    - Сервіс Web-API – REST-API, ASP.NET Core, C#
    - Веб додаток керування контентом – MVC, ASP.NET Core, Razor Views, C#
    - Веб додаток для покупця – Node JS, React, Redux
- При розгортанні програмного забезпечення рекомендується задіяти хмарні технології
- При розробці програмного забезпечення необхідно задіяти зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту).
- 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.

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

В результаті аналізу технічного завдання до проєкту, архітектурним стилем додатка став SPA (Single Page Application) завдяки своїй продуктивності, сучасності та зручності для користувачей. Проєкт складається з таких частин:

- Backend
- Design
- Frontend
- DevOps

Серверна частина Backend розроблена мовою програмування C#, за допомогою фреймворку ASP.NET Core 6. Для надійного зберігання персональних було використано СУБД PostgreSQL. Вибір саме цієї системи управління базами даних був зумовлений її ефективністю та популярністю. Також ця СУБД підтримується популярним ORM (Object Relational Mapping) Entity Framework Core від Microsoft, який спрощує розробку та взаємодію з базою даних за допомогою мови запитів LINQ.

Сервер реалізує RESTful API, який обробляє мережеві запити від клієнтської частини за допомогою протоколу HTTP. Він проводить маніпуляції з даними, за потреби звертається до бази даних або до кешу і надсилає відповідь до клієнтської частини у форматі JSON. Щоб забезпечити безпеку у додатку, була реалізована авторизація та автентифікація за допомогою JWT та Refresh токенів.

Код серверної частини знаходиться за адресою: репозиторій.

Код адміністративної частини знаходиться за адресою: репозиторій.

РОЗДІЛ 2

РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Серверна частина написана з використанням наступних технологій:

- Мова програмування C#
- Фреймворк ASP.NET Core 6
- База даних PostgreSQL
- Інструмент адміністрування бази даних PgAdmin
- Середовище розробки Visual Studio
- Платформа для тестування та розгортання додатків Docker
- Сховище даних типу «ключ-значення» Redis
- Технології: EF Core, LINQ
- Додаткові пакети: MailKit, AutoMapper, MediatR, Humanizer, NickBuhro.Translit, FluentValidation, GoogleAuthenticator, Google.Apis.Auth, AspNetCore.HealthChecks
- API, що керує користувачами, паролями тощо: ASP.NET Identity

- Unit тести до серверної частини написані з використанням:
- Мова програмування C#
- Фреймворк xUnit
- Додаткові пакети: FluentAssertions, Moq

В основі серверної частини лежить RESTful API, який реалізовано згідно з Чистою Архітектурою. При написанні проєкту використовувались принципи проєктування, такі як SOLID, DRY, та патерни проєктування, зокрема Dependency Injection, Mediator, CQRS. Це все необхідно для того, щоб мінімізувати наростання складності розробки додатка з часом.

Усі користувацькі та системні дані, необхідні для роботи додатка, зберігаються у базі даних PostgreSQL. Для маніпулювання та мапінгу даних с бази до класів в коді використовується ORM (Object-Relation Mapper) Entity Framework Core з мовою запитів LINQ (Language-Integrated Query). ORM Entity Framework Core підвищує швидкість написання коду та його читабельність.

У додатку реалізовані надійні системи реєстрації, автентифікації та авторизації, завдяки яким користувач може не хвилюватись, що від його імені зловмисник зможе щось зробити. Додаток має два види реєстрації. Перший – з використанням Email та паролю. Другий – з використанням свого акаунту у Google. При реєстрації за допомогою Email та паролю, цей Email необхідно підтвердити, ввівши код, який на нього буде надіслано. Тільки після підтвердження пошти буде створено акаунт. А до підтвердження у базі зберігаються тимчасові дані, такі як пошта, код підтвердження та кількість невдалих спроб підтвердити пошту. Якщо користувач не в змозі її підтвердити – реєстрація з даною поштою блокується на деякий час. Це необхідно для того, щоб людина не могла зареєструватись з використанням пошти, доступу до якої в неї нема. У випадку реєстрації за допомогою стороннього сервісу Google, усі дані користувача вважаються підтвердженими, тому нема потреби їх ще раз підтверджувати. Також у додатку є два види автентифікації та авторизації. Перший – за допомогою Email та паролю. Другий – за допомогою акаунту у Google. В будь-якому випадку після успішної авторизації, клієнт отримує JWT та Refresh токени, які використовує для подальшої роботи з серверною частиною. Для виконання деяких дій клієнту необхідно у запросі до API передавати Access Token (JWT), який буде перевірено на сервері. Якщо Access Token буде прострочено, то клієнт буде мати змогу оновити його за допомогою Refresh Token. Також у додатку реалізована система двофакторної автентифікації за допомогою додатку Google Authenticator. Якщо користувач увімкнув та

налаштував цю систему, то при наступній автентифікації разом з поштою та паролем йому буде необхідно ввести код з Google Authenticator.

У системі також реалізовано кеш відповідей на запити клієнту за допомогою сховища даних типу «ключ-значення» Redis. Він не є кешем, але завдяки своїй продуктивності використовується як кеш. При запиті до даних, що рідко змінюються, відповідь кешується у Redis. Наступні запити до цих даних не торкаються бази даних, а одразу беруть дані з кешу, що прискорює роботу додатка та знімає навантаження з бази даних. Оскільки код написано з використанням бібліотеки MediatR, то усі запроси проходять скрізь Mediator Pipeline. Тому кешування було вирішено помістити у цей пайплайн.

```
0 references | tapeex, 44 days ago | 1 author, 3 changes
public async Task<TResponse> Handle(TRequest request, RequestHandlerDelegate<TResponse> next, CancellationToken cancellationToken)
{
    TResponse response;
    var cachedValue = await _distributedCache.GetStringAsync(request.CacheKey, cancellationToken);
    if (string.IsNullOrEmpty(cachedValue))
    {
        response = await next.Invoke();
        cachedValue = JsonSerializer.Serialize(response);
        var options = new DistributedCacheEntryOptions
        {
            SlidingExpiration = TimeSpan.FromHours(request.ExpirationHours)
        };
        await _distributedCache.SetStringAsync(request.CacheKey, cachedValue, options, cancellationToken);
        return response;
    }

    response = JsonSerializer.Deserialize<TResponse>(cachedValue)!;
    return response;
}
```

Також дані кожного запросу валідуються перш ніж звертатися до кешу або до бази даних. Валідація здійснюється за допомогою бібліотеки FluentValidation. Для кожного атрибуту запросу додаються необхідні правила. Якщо дані у запросі не відповідають встановленим правилам, то звернення до бази даних чи до кешу не буде. Це збільшує продуктивність системи. Приклад класа-валідатора, який перевіряє вхідні дані згідно з правилами бізнес-логіки:

```
1 reference | tapeex, 5 days ago | 1 author, 2 changes
public sealed class CreatePendingUserCommandValidator : AbstractValidator<CreatePendingUserCommand>
{
    0 references | tapeex, 5 days ago | 1 author, 1 change
    public CreatePendingUserCommandValidator(UserManager<User> userManager, IApplicationDbContext context)
    {
        RuleFor(x => x.Email).EmailAddress();

        RuleFor(x => x.Email).MustAsync(async (email, _) =>
        {
            var userWithEmail = await userManager.FindByEmailAsync(email);
            return userWithEmail is null;
        }).WithMessage(ValidationMessages.UserWithEmailAlreadyExists);

        RuleFor(x => x.Email).MustAsync(async (email, cancellationToken) =>
        {
            var pendingUsersCount = await context.PendingUsers.CountAsync(x => x.Email == email, cancellationToken);
            return pendingUsersCount <= 0;
        }).WithMessage(ValidationMessages.ConfirmationCodeWasAlreadySent);
    }
}
```

Код системи є асинхронним там, де це необхідно. Асинхронність необхідна у I/O-bound (Input/Output-bound) операціях. Прикладом такої операції є запит до бази даних.

Щоб не передавати клієнту непотрібну інформацію, отримані дані з бази даних автоматично перетворюються на об’єкти, які містять лише необхідну інформацію. Таке преображення називається мапінг (mapping). Для цього в коді використовується бібліотека AutoMapper. Приклад DTO класу, який мапіться зі звичайного:

```
3 references | - changes | -authors, -changes
public sealed record BrandDTO : IMapWith<Brand>
{
    0 references | tapeex, 44 days ago | 1 author, 1 change
    public int Id { get; init; }
    0 references | tapeex, 44 days ago | 1 author, 1 change
    public string Name { get; init; } = null!;
    1 reference | tapeex, 44 days ago | 1 author, 1 change
    public void Mapping(Profile profile)
    {
        profile.CreateMap<Brand, BrandDTO>();
    }
}
```

```
1 reference | tapeex, 53 days ago | 1 author, 3 changes
public async Task<IEnumerable<BannerDTO>> Handle(GetBannersQuery request, CancellationToken cancellationToken)
{
    var banners = await _context.Banners
        .AsNoTracking()
        .ToListAsync(cancellationToken);

    var result = _mapper.Map<IEnumerable<BannerDTO>>(banners);

    return result;
}
```

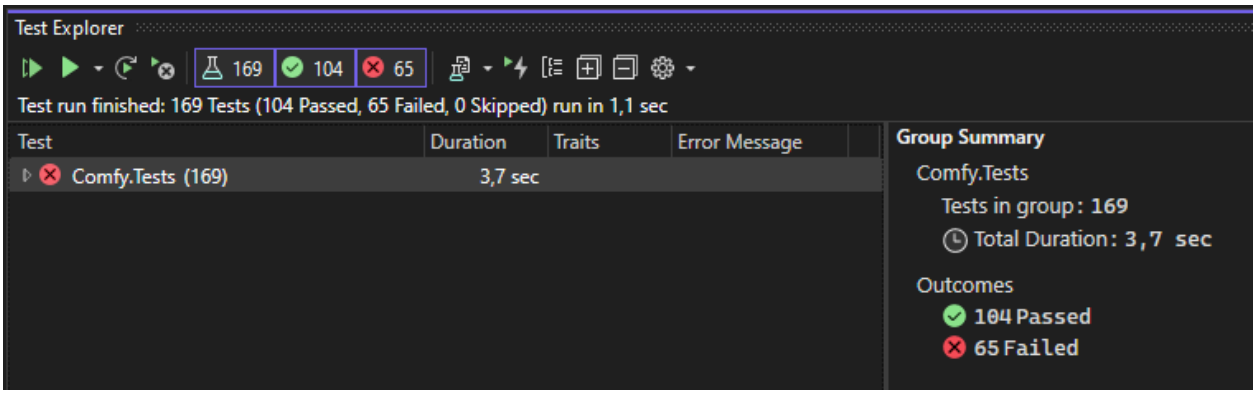
Пошук товарів з урахуванням фільтрації та сортування доволі складний механізм. Але код, який це реалізує, вийшов доволі гнучким. Та якщо потрібно буде додати параметр до сортування, то потрібно буде додати лише однин рядок кода. Ось так виглядає код сортування товарів:

```
2 references | tapeex, 1 day ago | 1 author, 1 change
private static Expression<Func<Product, object>> GetSortProperty(GetProductsQuery request)
{
    return request.SortColumn switch
    {
        "name" => x => x.Name,
        "price" => x => x.Price,
        "rating" => x => x.Rating,
        _ => x => x.Id
    };
}
```

```
if (request.SortOrder == "desc")
{
    products = products.OrderByDescending(GetSortProperty(request));
}
else
{
    products = products.OrderBy(GetSortProperty(request));
}
```

До коду серверної частини також були написані Unit тести, які перевіряють коректність роботи системи, враховуючи вхідні дані. Завдяки тому, що серверна частина реалізує патерни Mediator та CQRS (Command and Query Responsibility) було дуже просто робити Mock об’єкти усіх команд та запитів, що дозволило доволі швидко покрити тестами більшу частину коду. В ході

написання тестів я зробив скриншот, який показує кількість виконаних тестів.



Адміністратори мають можливість переглянути інформацію щодо стану здоров'я системи та її компонентів. Це реалізовано завдяки HealthChecks сервісу.

Загалом, серверна частина має 42 кінцеві точки. Усі вони необхідні для функціонування системи. Їх перелік можна побачити тут: скріншот.

РОЗДІЛ 3

АДМІНІСТРАТИВНА ЧАСТИНА

Адміністративна частина написана з використанням наступних технологій:

- Мова програмування C#
- Фреймворк ASP.NET Core 6
- База даних PostgreSQL
- Інструмент адміністрування бази даних PgAdmin
- Середовище розробки Visual Studio
- Платформа для тестування та розгортання додатків Docker
- Сховище даних типу «ключ-значення» Redis
- Технології: EF Core, LINQ
- Додаткові пакети: AspNetCore.HealthChecks, AWSSDK.S3, Humanizer, Mapster, MediatR, NickBuhro.Translit
- API, що керує користувачами, паролями тощо: ASP.NET Identity

В основі адміністративної частини (адміністративної панелі) лежить проєкт типу MVC (Model View Controller). Цей проєкт відокремлено від серверної частини (Web Api), тобто є окремим проєктом. Адміністративна панель необхідна для керування контентом в інтернет-магазині: додавання та видалення товарів, перевірка коментарів, створення категорій, банерів тощо. Цей проєкт також написано з урахуванням принципів проєктування, таких як SOLID, та з використанням деяких патернів проєктування, зокрема DI та CQRS.

Усі користувацькі та системні дані, необхідні для роботи додатка, зберігаються у базі даних PostgreSQL. Причому це та ж сама база даних, якою користується серверна частина (Web Api). Для маніпулювання та мапінгу

даних с бази до класів в коді в адміністративній частині також використовується ORM Entity Framework Core з мовою запитів LINQ.

Адміністративна панель дозволяє створювати та редагувати товари. Кожен товар має зображення. Ці зображення було прийняте рішення зберігати в AWS S3 – хмарний сервіс зберігання даних. Для цього було використано їх API за допомогою бібліотеки AWSSDK.S3.

Кожен товар також має унікальну url-адресу. Вона формується з назви товару та його артикулу. Назва товару перетворюється до необхідного формату завдяки методам із бібліотек Humanizer.Core та NickBuhro.Translit. Приклади url-адрес товарів:

```
smartfon-apple-iphone-12-128gb-purple-1000005
smartfon-apple-iphone-12-128gb-black-1000006
smartfon-samsung-galaxy-a54-5g-6-128gb-awesome-graphite-1000007
smartfon-samsung-galaxy-a54-5g-8-256gb-light-violet-1000008
smartfon-samsung-galaxy-m14-5g-4-64gb-blue-1000009
```

Оскільки в адміністративній панелі відбувається зміна даних товарів, банерів та інших даних, які могуть зберігатися у кешу, то й інвалідація кешу виконується безпосередньо тут. Тобто кеш створюється при запросі до web api серверної частини, та видаляється при зміні закешованих даних в адміністративній панелі.

Для мапінгу класів в адміністративній частині була використана бібліотека Mapster. Вона робить усе те саме, що й AutoMapper. Але Mapster є більш продуктивним.

Адміністративна панель також дозволяє керувати усіма користувачами, зокрема й адміністраторами. Ролі адміністраторів поділені на адміністратора, старшого адміністратора та власника. Власник має найбільші можливості та може створити акаунт з будь-якими правами доступу. Старший адміністратор

не може створити акаунт з правами власника. А звичайний адміністратор взагалі не може керувати користувачами.

Оскільки серверна частина дозволяє користувачам писати відгуки та питання, то адміністративна частина має мати можливість керувати ними. Тому було прийнято рішення не публікувати на сайті коментарі, які ще не пройшли перевірку адміністрацією. Це реалізовано за допомогою звичайного атрибуту IsActive у таблиці з коментарями. Якщо коментар не порушує правила сайту, то адміністратори роблять його активним. І тоді він буде відображатись на сайті. Також є можливість деактивувати коментарі, якщо їх помилково активували. Це все необхідно для фільтрації контенту на сайті, щоб не допустити, наприклад, спам-атак.

В адміністративній панелі також є можливість перевірити здоров’я сервісів, від яких залежить система. Це також реалізовано завдяки сервісу HealthChecks.

Розробляючи цей проєкт, я прагнув дати адміністраторам можливість керувати усім контентом на сайті. Ця можливість наразі реалізована. Товари, користувачі, акції, банери, категорії та все інше створюється, редагується чи видаляється в адміністративній панелі.

ВИСНОВКИ

Було реалізовано два окремих проєкти: програмний інтерфейс (Web Api) та адміністративна панель. Обидва проєкти мають відмінності, наприклад різні шаблони для побудови системи – патерн MVC для адміністративної частини та Чиста Архітектура для Web Api. Але є й схожість. Наприклад, обидва проєкти було реалізовано за допомогою мови програмування C#, фреймворку ASP.NET Core шостої версії та ORM Entity Framework Core.

Обидва проєкти мають гарний рівень безпеки, який, наприклад, Web Api забезпечується JWT токенами та двофакторною автентифікацією, а також можливістю зареєструватись за допомогою стороннього сервісу, такого як Google. Даними користувачей в обох проєктах керує ASP.NET Identity – вбудована в ASP.NET Core система автентифікації, авторизації, зберігання та обробки даних користувачей.

Та, звичайно, дуже багато уваги було привернуто до якості програмного коду. Купу разів проводився рефакторинг, для виявлення code-smells чи просто неефективних алгоритмів. Архітектура додатків також є доволі сучасною, особливо Чиста Архітектура у Web Api.

Попри масштабну кількість виконаної роботи, проєкт в цілому ще покращувати. Є багато речей, які ще можна додати до нього. Але вже зараз – це проєкт з багатьма цікавими рішеннями.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

- C#
- .NET Core
- ASP.NET Core
- Entity Framework Core
- Razor
- XUnit
- MailKit
- AutoMapper
- MediatR
- Humanizer
- NickBuhro.Translit
- FluentValidation
- GoogleAuthenticator
- Google.Apis.Auth
- AspNetCore.HealthChecks
- FluentAssertions
- Moq
- AWSSDK.S3
- Mapster
- Visual Studio
- PostgreSQL
- Docker
- Redis
- Postman

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Microsoft technical documentation
- Stackoverflow
- Medium
- ChatGPT
- YouTube
- EF Core Tutorial
- Google authentication documentation
- GitHub
- C# Corner
- JWT documentation & debugger
- IANA JWT specification
- Metanit
- Docker documentation