

## Prykladna-SHOP

Сайт-магазин атрибутики Факультет прикладної  
математики та інформатики Львівського  
національного університету імені Івана Франка

by Rangeln:

Білокурсський Валерій,

Євуш Софія,

Макаруха Соломія,

Романик Юля,

Сенюх Назарій.

Студенти групи ПМІ-32.

28.02.2024

## Опис проекту:

Наш проект - це веб-сайт магазину аксесуарів, спеціалізованого на продажу товарів Факультету прикладної математики та інформатики Львівського національного університету імені Івана Франка. На нашому сайті ви зможете знайти широкий асортимент товарів, які відображають принципи та ідентичність факультету.

Наш веб-сайт побудований з використанням технологій ASP.NET MVC фреймворку, що забезпечує стабільну та ефективну роботу сайту. Для зберігання даних ми використовуємо базу даних, яка дозволяє нам ефективно управляти асортиментом товарів та замовленнями клієнтів.

Фронтенд нашого сайту розроблений з використанням HTML/CSS, що забезпечує зручне та привабливе користування. Ми прагнемо зробити процес вибору та придбання товарів якнайбільш зручним та приємним для наших клієнтів.

## Ролі у системі:

- Адміністратор - має повний доступ до адміністративної панелі та може керувати всією системою.
- Покупець - це користувач, який реєструється на сайті та має можливість переглядати товари, обирати їх та оформлювати замовлення. Він не має доступу до адміністративної панелі або перегляду інформації про інших користувачів.
- Гість - неаутентифікований користувач матиме доступ лише до перегляду головної сторінки та сторінок реєстрації/входу.

## Функціональні вимоги:

1. **Реєстрація користувача:** можливість реєстрації нового облікового запису на сайті. Вимога введення обов'язкових даних, таких як ім'я, прізвище, електронна пошта, телефон тощо.
2. **Редагування профілю:** можливість зміни особистих даних користувача після реєстрації.
3. **Зміна паролю:** можливість зміни паролю користувачем.
4. **Наявність кошика:** Функція додавання товарів до кошика для подальшого оформлення замовлення. Можливість перегляду вмісту кошика та видалення товарів. Відображення загальної суми замовлення та кількості товарів у кошику.
5. **Окрема сторінка замовлень:** відображення історії замовлень користувача.
6. **Каталог:** перелік усіх доступних товарів на сайті, поділених на категорії. Можливість перегляду докладної інформації про кожен товар, такої як опис, ціна, наявність тощо.
7. **Фільтрація за видом:** можливість фільтрації товарів за різними параметрами, такими як тип товару, ціновий діапазон, виробник тощо. Зручний інтерфейс для вибору потрібних фільтрів та відображення відфільтрованих результатів.

## Нефункціональні вимоги:

1. **Мова інтерфейсу:** Програма повинна мати інтерфейс, доступний на українській а також англійській мові, для задоволення потреб різних користувачів.
2. **База даних:** Програма повинна використовувати реляційну базу даних PostgreSQL для збереження та управління інформацією користувачів.
3. **Програмне забезпечення:**

Для розробки веб-застосунків за допомогою ASP.NET MVC, нам знадобляться наступні програмні компоненти:

  - Середовище розробки (IDE): Visual Studio: Рекомендоване інтегроване середовище розробки для ASP.NET. Команда буде використовувати його безкоштовний варіант.
  - ASP.NET MVC: Це фреймворк для побудови веб-застосунків на платформі .NET. Який буде використовуватися в проекті.
  - База даних: Для зберігання даних буде викорситовуватися SQL Server.
  - Git: ведення версій проекту є обов'язковим!
4. **Зберігання історії дій користувачів:** Програма повинна мати можливість реєстрації та зберігання історії дій користувачів для вирішення суперечок або аналізу діяльності.
5. **Безпека:** Програма повинна дотримуватися усіх зазначених пунктів нижче:
  - Мінімальна довжина пароля: Вимога до того, щоб пароль користувача мав мінімальну та максимальну довжину: не менше

8 символів і не більше 40, також пароль обов'язково має містити одну велику літеру, одну цифру та один небуквенний символ.

- Заборона використання слабких паролів: Вимога до користувачів використовувати паролі, які не є очевидними, такими як "123456" або "password".
- Обмеження доступу до конфіденційних даних: Вимога до того, щоб доступ до конфіденційних даних був обмежений ізольованими ролями і правами доступу.

6. **Продуктивність:** Програма повинна гарантувати, що будь-яка сторінка відкриється протягом 3 секунд як максимум.

7. **Доступність:** Веб застосунок буде безкоштовним та доступною для будь-якого авторизованого користувача.

Роль	Опис
Адмін	Може переглядати усі замовлення, усіх користувачів. Здійснювати вихід з системи, змінювати облікові дані.
Гість	Може здійснювати вхід, реєстрацію. Можливість перегляду докладної інформації про кожен товар, такої як опис, ціна, наявність.
Користувач	Може купувати товар, переглядати історію замовлень та деталі, вихід з системи, змінювати облікові дані.

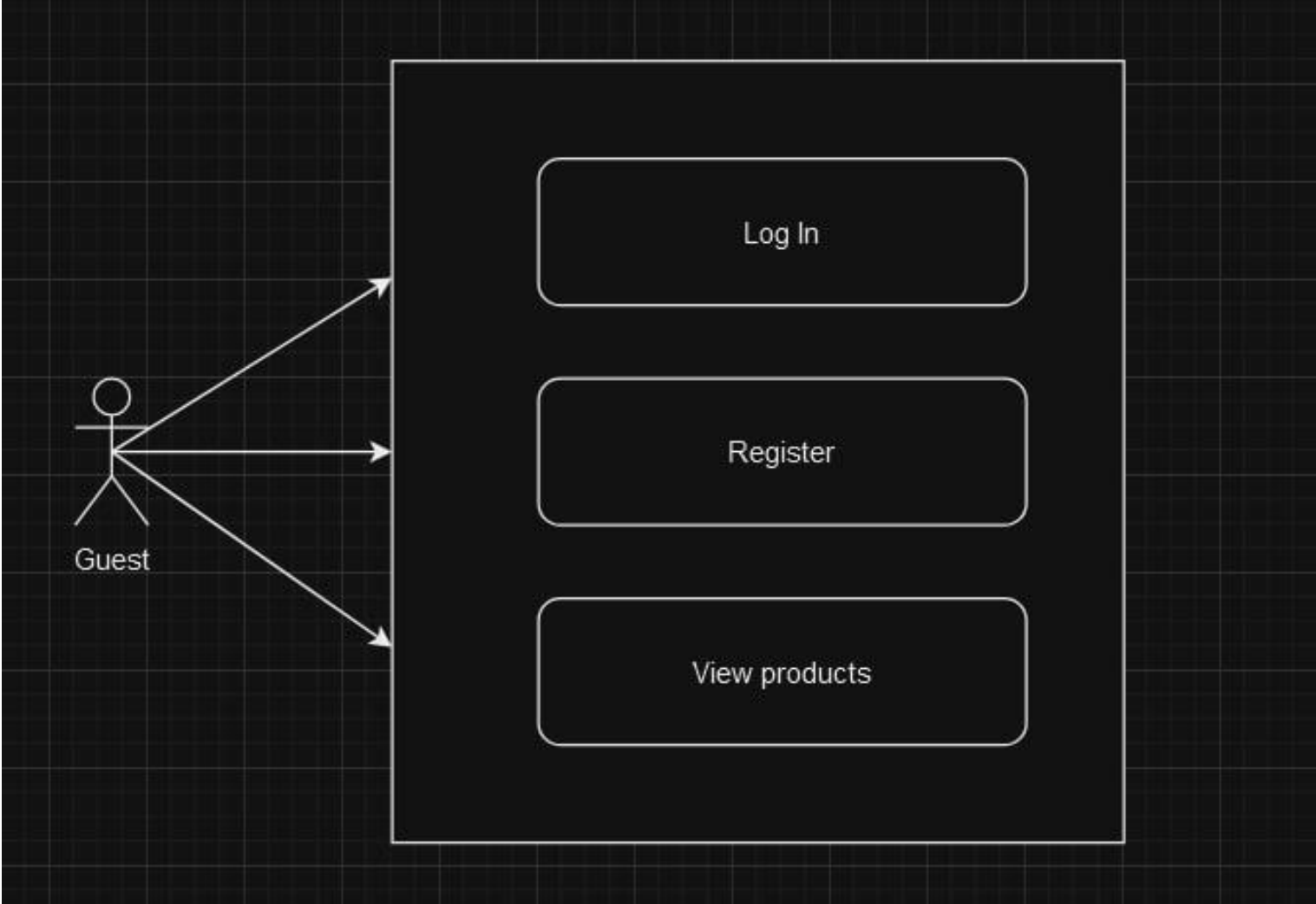
№	Роль	Use-case	Опис
1	Адмін	Перегляд усіх замовлень	Адміністратор має повний доступ до списку всіх замовлень, які були зроблені в системі. Це включає дані про кожне замовлення, такі як ідентифікатор, інформація про клієнта, дату та час замовлення, деталі товарів або послуг, їх кількість, вартість тощо. Адміністратор може швидко переглядати ці дані для забезпечення ефективного керування всіма замовленнями в системі.
2		Перегляд усіх користувачів	Адміністратор має можливість переглядати список усіх користувачів системи. Це може включати їх імена, ідентифікатори, контактну інформацію, ролі або привілеї, які вони мають в системі тощо.
3		Вихід з системи	Після виходу з системи адмін перестає мати доступ до будь-яких привілеїв, обмежень та конфіденційної інформації, пов'язаної з цим обліковим записом.
4		Зміна облікових даних	Адмін може змінити інформацію, пов'язану зі своїм обліковим записом в системі. Це може включати зміну особистої інформації, такої як ім'я, адреса електронної пошти, номер телефону, а також інші дані, які адмін ввів при реєстрації або пізніше.

№	Роль	Use-case	Опис
1	Гість	Вхід в систему	Гість може лише переглядати загальнодоступну інформацію, що доступна на сайті або додатку.
2		Реєстрація	Гість може перетворитися на зареєстрованого користувача шляхом заповнення реєстраційної форми. Зазвичай це включає введення особистої інформації, такої як ім'я, електронна адреса та пароль. Після успішної реєстрації гість стає зареєстрованим користувачем і має доступ до додаткового функціоналу системи.
3		Перегляд товарів	Гість може переглядати асортимент товарів чи послуг, які надає система. Зазвичай це включає перегляд каталогу товарів, опис продукції, цін та інших характеристик. Однак гість не може здійснювати покупки або зберігати товари в кошику, оскільки для цього необхідно бути зареєстрованим користувачем і мати активний обліковий запис.

№	Роль	Use-case	Опис
1	Користувач	Купівля товарів	Користувач має можливість переглядати асортимент товарів або послуг, вибирати необхідні позиції та додавати їх до кошика.
2		Перегляд історії замовлень і деталі замовлення	Користувач може переглядати свою історію замовлень, де відображаються всі попередні покупки. Він також може переглядати деталі кожного окремого замовлення, такі як перелік куплених товарів, сума оплати тощо.
3		Вихід з системи	Користувач може в будь-який момент завершити сеанс роботи в системі шляхом виходу з облікового запису. Це дозволяє зберегти безпеку даних та уникнути несанкціонованого використання облікового запису у разі доступу до пристрою користувача іншими особами.
4		Зміна облікових даних	Користувач може змінити особисту інформацію, пов'язану зі своїм обліковим записом, таку як адреса електронної пошти, пароль, контактні дані тощо.



use-case діаграми:





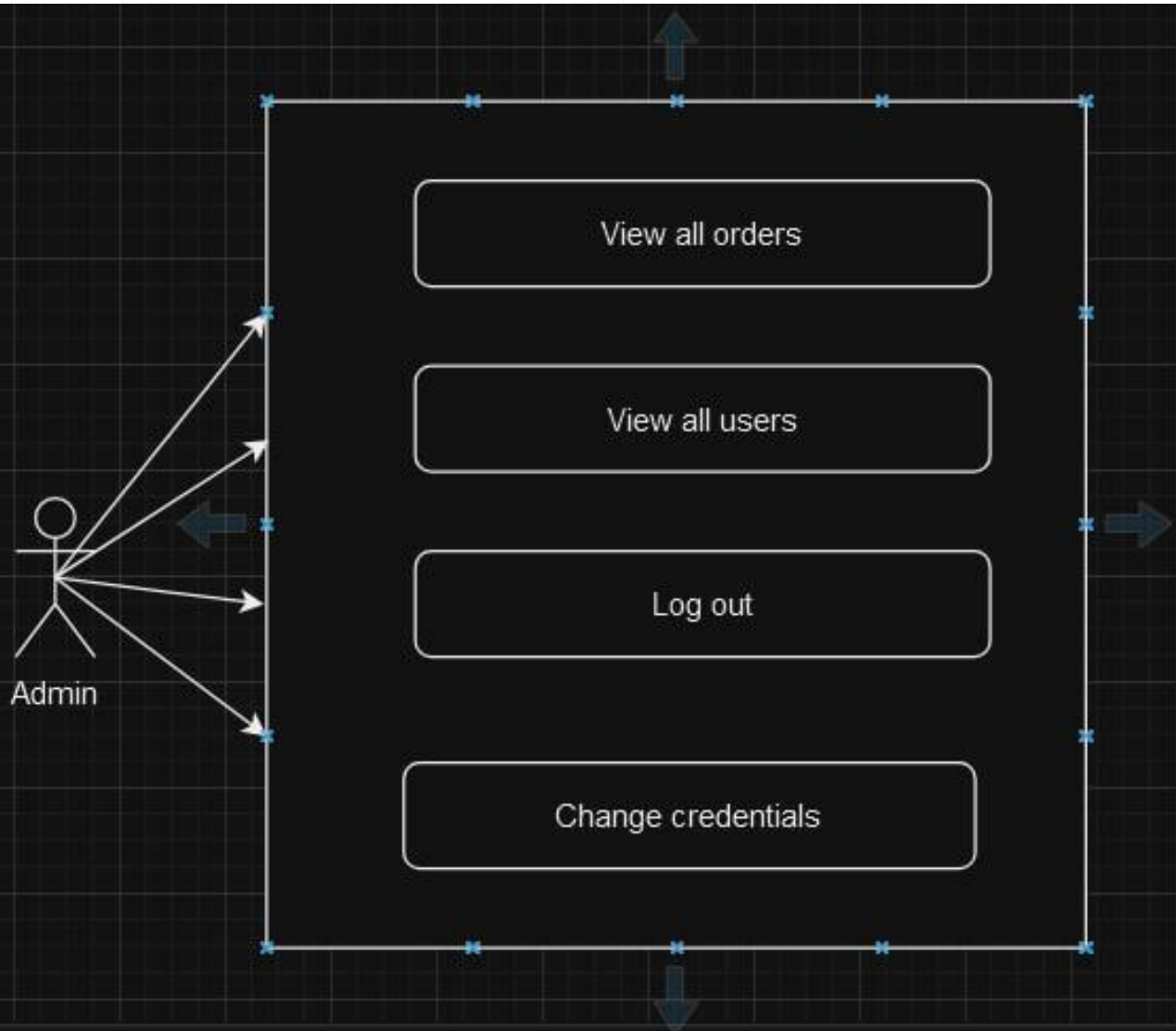
User

Buy products

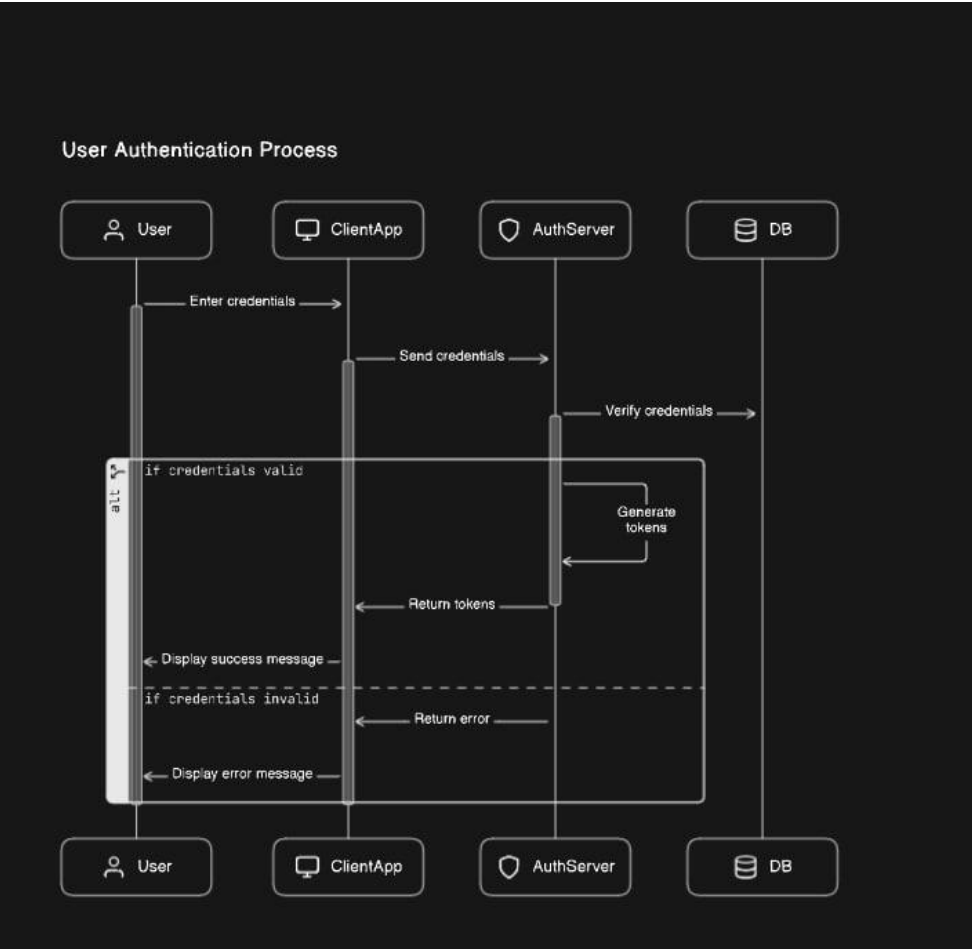
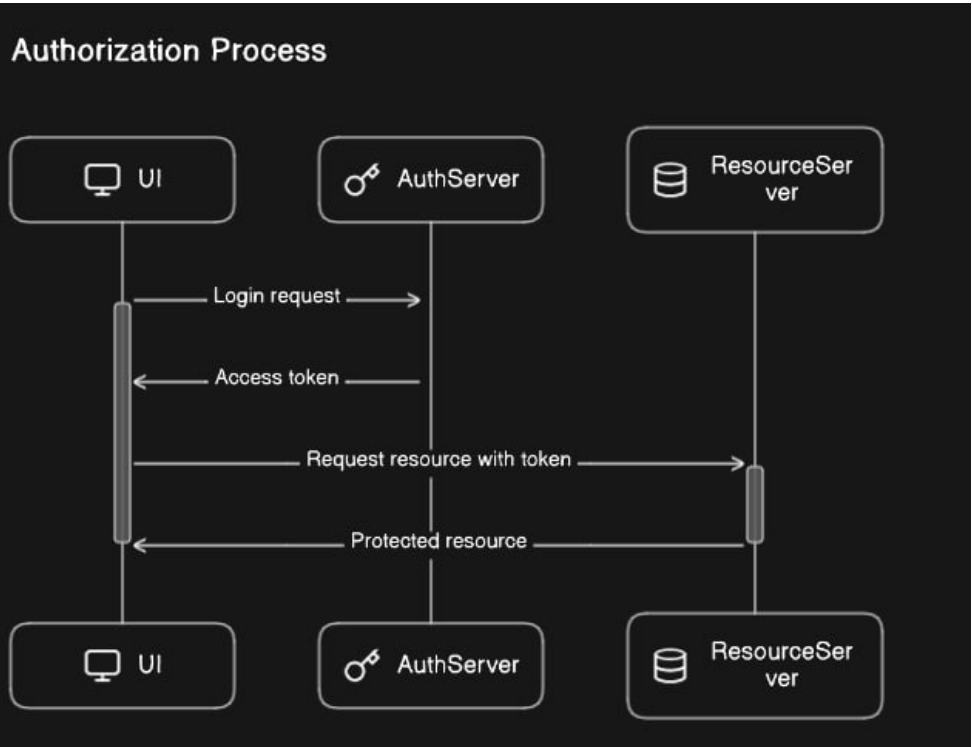
View order history and details

Log out

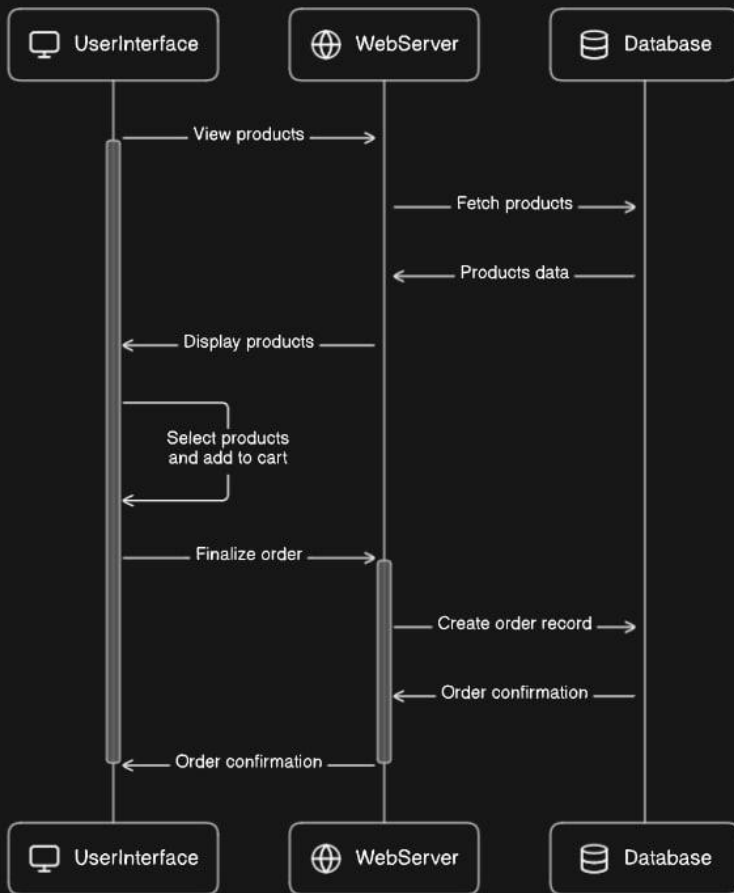
Change credentials



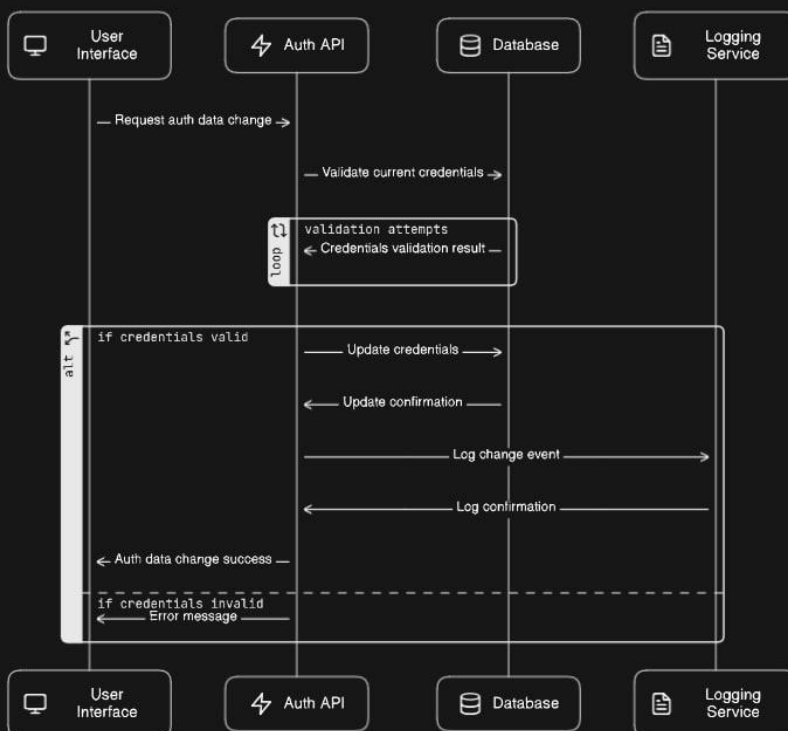
Sequence diagrams:



## Online Store Order Process

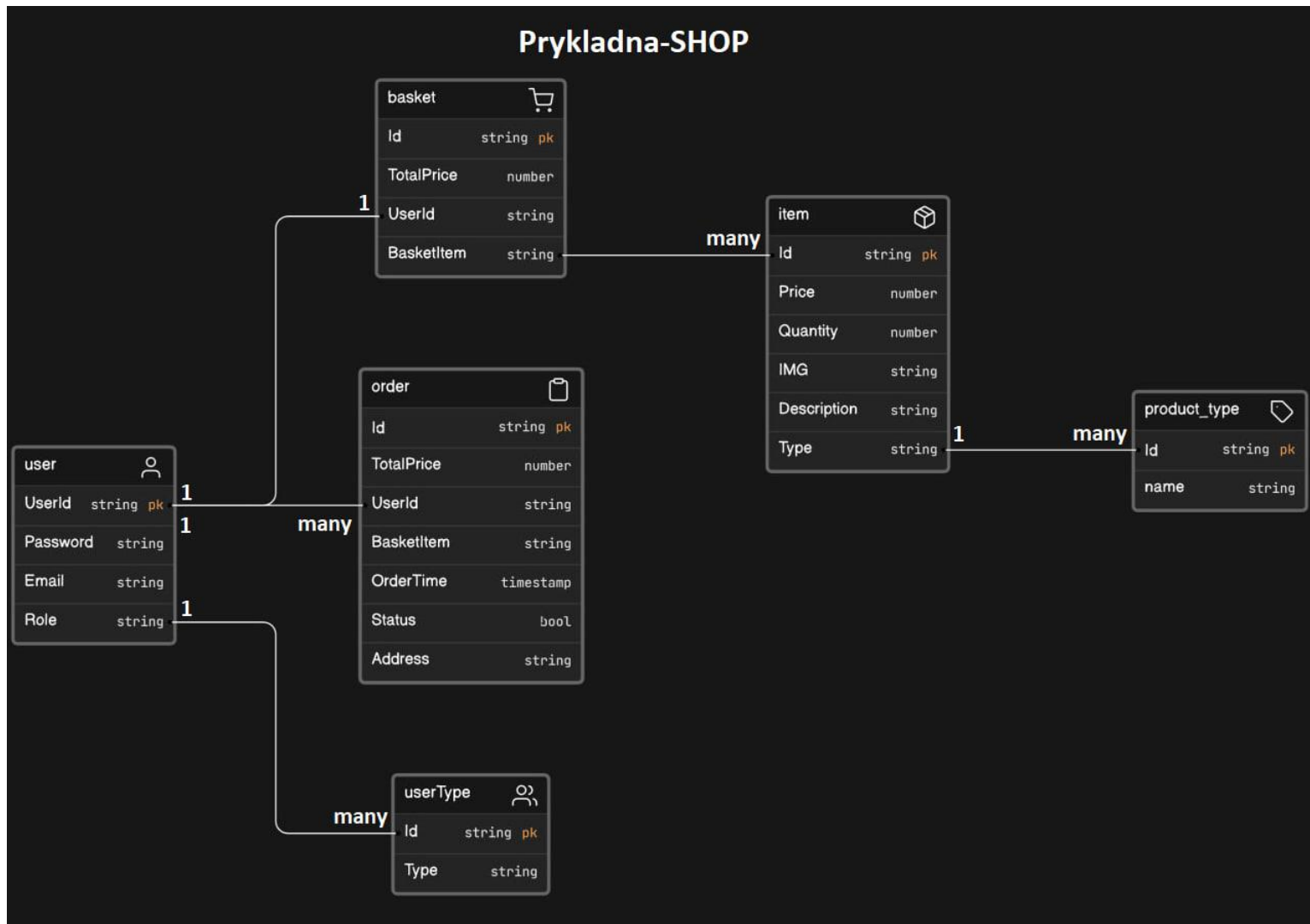


## Change Authorization Data



# ER-діаграма

Ми використовуємо модель «сутність-зв'язок» для чіткого і систематичного опису структури бази даних, щоб зрозуміти взаємозв'язки між сутностями та потреби користувачів, спрощуючи процес розробки бази даних та полегшуючи комунікацію між учасниками проекту.



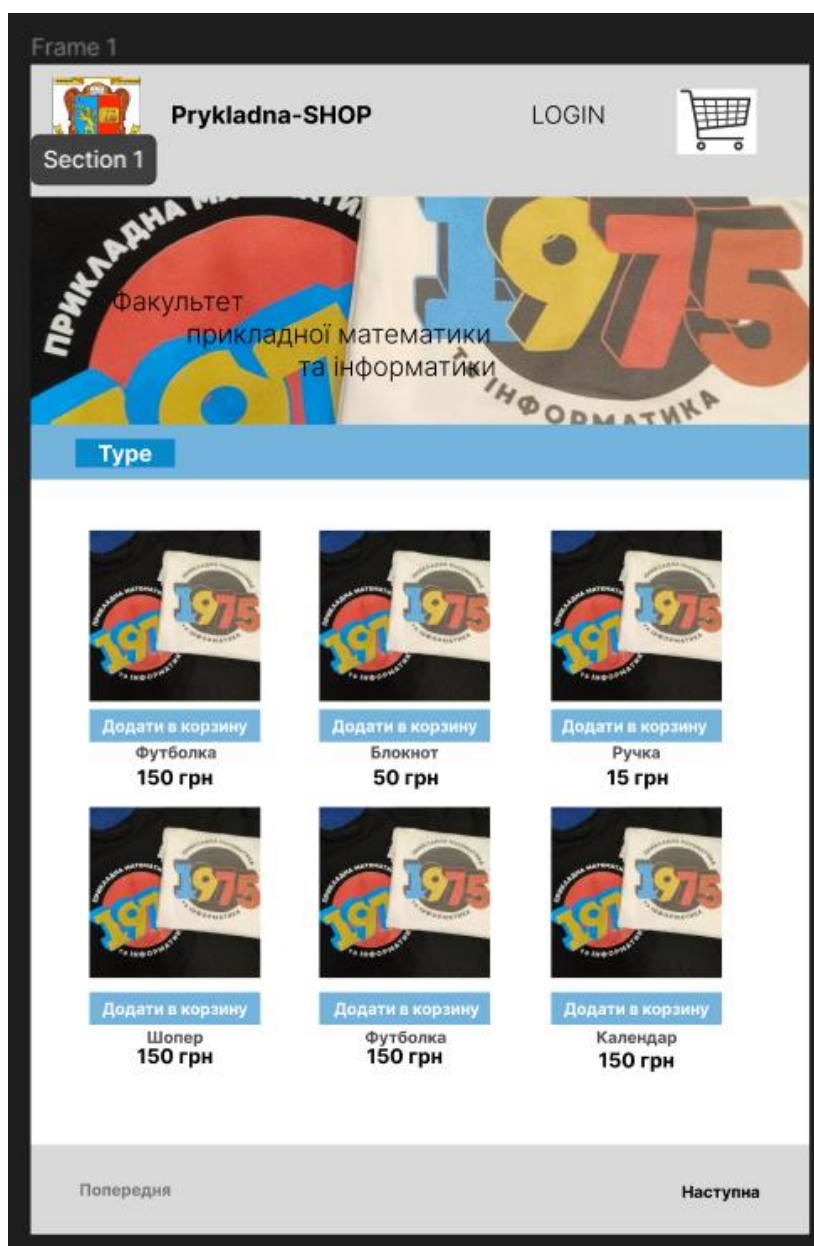
На графічному зображенні відображена структура нашої реляційної бази даних. Після успішної реєстрації користувача його дані додаються до таблиці "user". В системі передбачено три типи користувачів(" userType"): викладачі, студенти та фанати. Останні — гості, які можуть бути зацікавлені у товарах, але не мають прямого зв'язку з факультетом. Крім того, користувач може взаємодіяти з замовленнями ("order") та кошиком ("basket"). Кожен кошик має зв'язок з товаром ("item"), який обирає замовник. Також варто зазначити, що тип продукту

("product\_type") визначає клас, який зберігає перелік категорій товарів та передає ці категорії у товар ("item").

## Мокап системи



Нижче наведено приблизний вигляд нашого проекту після завершення його виконання.

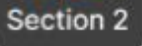
- Головна сторінка: Користувачі зможуть зареєструватися або увійти до свого облікового запису. На головній сторінці буде відображена інформація про найпопулярніші товари.






- Кошик: Користувачі зможуть додавати товари до кошика. У кошику буде можливість змінювати кількість товарів, переглядати інформацію та робити зміни у замовленні.

Frame 2

**Prykladna-SHOP**user1@gmail.com

Section 2



Товар	Ціна	Кількість	Сума
 Футболка	150 грн	<input type="text" value="1"/>	150 грн
 Ручка	15 грн	<input type="text" value="1"/>	15 грн

Продовжити покупку


Оновити

Перевірити




- Історія замовлень: На сторінці історії замовлень користувачі зможуть переглядати список своїх минулих замовлень разом з детальною інформацією про кожне замовлення, таку як дата оформлення, статус доставки, перелік товарів і загальна вартість.

Frame 6



**Prykladna-SHOP**

user1@gmail.com

### Історія покупок

Номер замовлення	Дата	Сума	Статус
1002	03/04/2024 10:56:43	165 грн	в очікуванні

### Адреса доставки


м.Городок  
Відділення №1  
вул.Львівська, 38

### Деталі замовлення

	Футболка	150 грн	1	150 грн
	Ручка	15 грн	1	15 грн


- Обліковий запис користувача: Користувачі зможуть переглядати, редагувати свій профіль.

Frame 5



Prykladna-SHOP

user1@gmail.com



Керування обліковим записом

Змінити налаштування облікового запису

Профіль

Пароль

Профіль

Ім'я користувача

user1@gmail.com

Електронна адреса


user1@gmail.com

[Send verification email](#)

Номер телефону


Зберегти

Frame 7



Prykladna-SHOP

user1@gmail.com



Керування обліковим записом

Змінити налаштування облікового запису

Профіль

Пароль

Змінити пароль

Поточний пароль

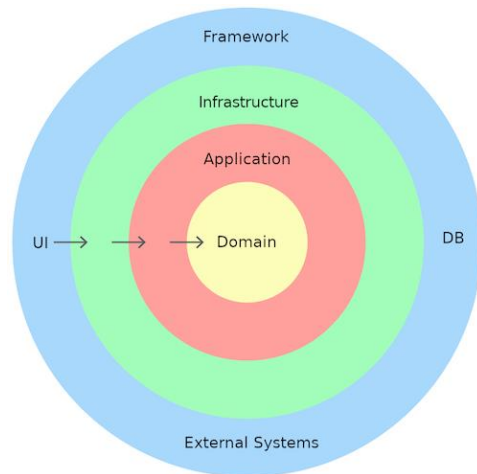
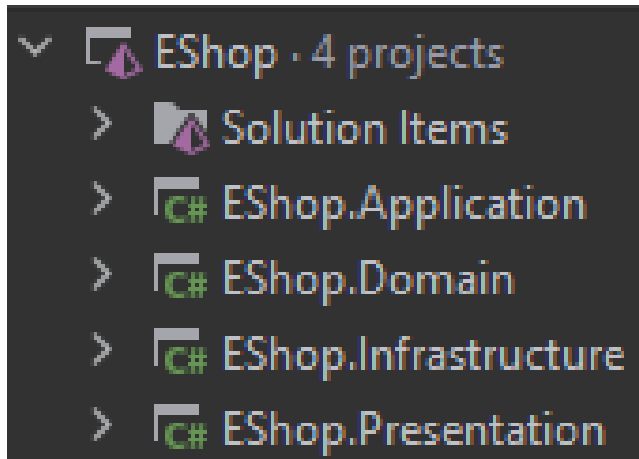
Новий пароль

Підтвердити новий пароль

Змінити

## Виконання 4 завдання

- Наш проект буде працювати за принципом Clean Architecture:



Clean Architecture - це архітектурний підхід для розробки програмного забезпечення, який покликаний розділити код на окремі шари, що дозволяє підтримувати його чистим, зрозумілим і легко змінюваним.

- Згідно вимог проекту, наша програма буде працювати з використанням бази даних PostgreSQL.

```
services:
  eshop.db:
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: eshop
      POSTGRES_PASSWORD: eshop
      POSTGRES_DB: EShop
    volumes:
      - ./volumes/eshop-db-data:/var/lib/postgresql/data
```

eshop.db відповідає за базу даних для нашого вебзастосунку.

Основні характеристики та налаштування цього сервісу вказані в конфігураційному файлі 'Docker Compose', який зберігається у 'Soulution Items'.

- Після цього ми підключили ORM до нашого проекту з використанням 'EntityFrameworkCore'.

```
using EShop.Infrastructure.Database;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace EShop.Infrastructure;

public static class DependencyInjection
{
    public static IServiceCollection AddInfrastructure(this IServiceCollection services, IConfiguration configuration)
    {
        services.AddDbContext<ApplicationDbContext>(optionsAction: options =>
            options.UseNpgsql(
                configuration.GetConnectionString(name: "DefaultConnection")));

        return services;
    }
}
```

У методі 'AddInfrastructure' визначено впровадження залежностей для служб, де використовується метод 'AddDbContext'. Цей метод налаштовує контекст бази даних 'ApplicationDbContext', використовуючи конфігурацію, яка передається через 'IConfiguration', і зв'язує його з підключенням до бази даних через 'Entity Framework Core'.

- У більшості IDE аналізатор кодів підключений автоматично, проте ми добавили Roslynator:

```
<PackageReference Include="Roslynator.Analyzers" Version="4.11.0">
  <PrivateAssets>all</PrivateAssets>
  <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
</PackageReference>
```

```

Roslynator.CSharp.Analyzers
Roslynator.CSharp.Analyzers.CodeFixes
Roslynator_Analyzers_Roslynator.Common
Roslynator_Analyzers_Roslynator.Core
Roslynator_Analyzers_Roslynator.CSharp
Roslynator_Analyzers_Roslynator.CSharp.Workspaces
Roslynator_Analyzers_Roslynator.Workspaces.Common
Roslynator_Analyzers_Roslynator.Workspaces.Core

```

- Додали налаштування структурного логування з використанням бібліотеки Serilog:

```

"Serilog": {
  "Using": [ "Serilog.Sinks.Console", "Serilog.Sinks.Seq" ],
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Information"
    }
  },
  "WriteTo": [
    { "Name": "Console" },
    {
      "Name": "Seq",
      "Args": {
        "serverUrl": "http://eshop.seq:5341"
      }
    }
  ],
  "Enrich": [ "FromLogContext", "WithMachineName", "WithThreadId" ]
},

```

```

<PackageReference Include="Serilog.AspNetCore" Version="8.0.1" />
<PackageReference Include="Serilog.Sinks.Seq" Version="7.0.0" />

```

- Реалізували таблиці відповідно до ER-діаграми:

3 usages Valerii 1 exposing API

```
public class Basket
{
    public Guid Id { get; set; }
    public decimal TotalPrice { get; set; }
    public Guid UserId { get; set; }
    public User User { get; set; }

    [JsonIgnore]
    public ICollection<Item> Items { get; set; }
}
```

```
public class Item
{
    public Guid Id { get; set; }
    public decimal Price { get; set; }
    public int Quantity { get; set; }
    public string Description { get; set; }
    public string Img { get; set; }
    public Guid TypeId { get; set; }

    [JsonIgnore]
    public ProductType Type { get; set; }
    [JsonIgnore]
    public ICollection<Basket> Baskets { get; set; }
}
```

```
public class Order
{
    public Guid Id { get; set; }
    public decimal TotalPrice { get; set; }
    public DateTimeOffset OrderTime { get; set; }
    public Status Type { get; set; }
    public string Address { get; set; }
    public Guid BasketId { get; set; }
    public Basket Basket { get; set; }
    public Guid UserId { get; set; }
    public User User { get; set; }
}
```

```
public class ProductType
{
    public Guid Id { get; set; }
    public string Name { get; set; }
}
```

```
public class UserType
{
    public Guid Id { get; set; }
    public string Type { get; set; }

    [JsonIgnore]
    public ICollection<User> Users { get; set; }
}
```

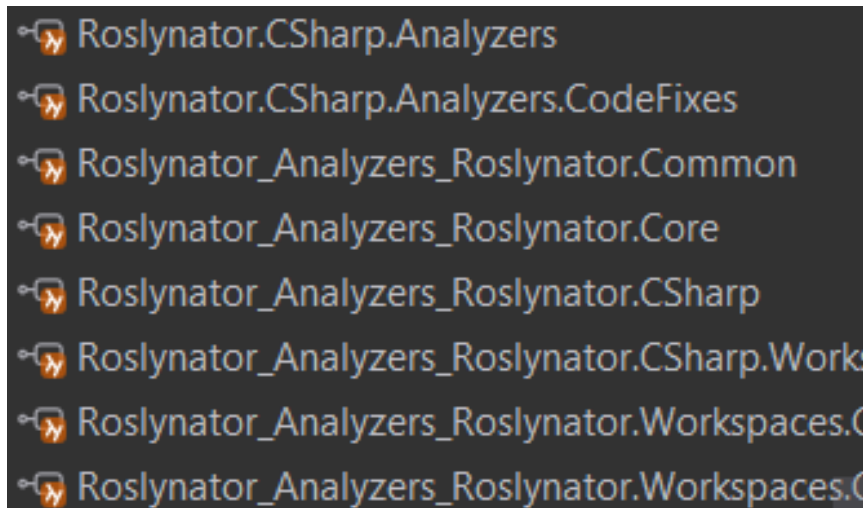
```
public class User
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    [EmailAddress]
    public string Email { get; set; }
    public string Password { get; set; }
    public string PasswordSalt { get; set; }
    public Guid UserId { get; set; }
    public UserType UserType { get; set; }
}
```

```
public class ApplicationDbContext : DbContext
{
     Valerii
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {
    }

    public DbSet<Item> Items { get; set; }
    public DbSet<ProductType> ProductTypes { get; set; }
    public DbSet<User> Users { get; set; }
    public DbSet<Basket> Baskets { get; set; }
    public DbSet<UserType> UserTypes { get; set; }
    public DbSet<Order> Orders { get; set; }
}
```

## Завдання номер 5

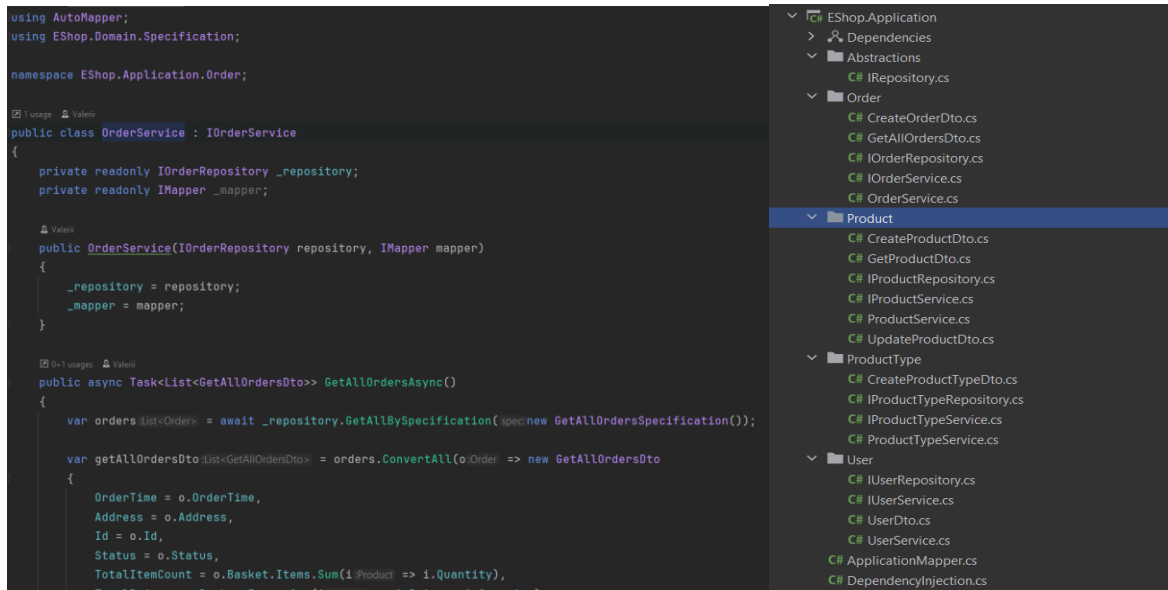
Для початку ми перевірили роботу нашого аналізатору коду Roslynator який ми підключили того разу.



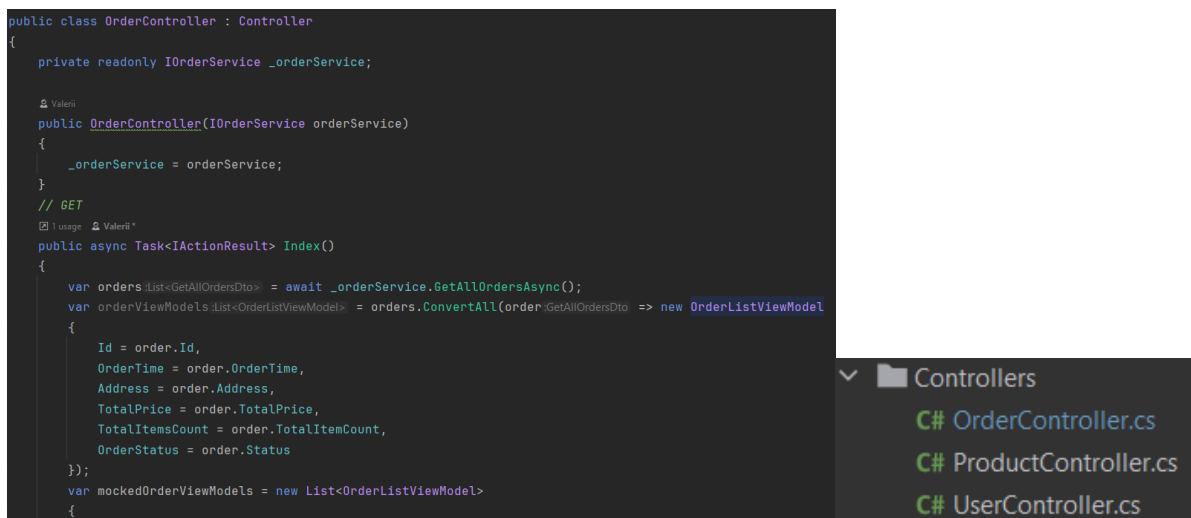
Як бачимо, він успішно показує 'помилки' під час написання чистого коду.

Після цього кожен з нас реалізував по 1 use-case для нашого проекту, 4 з них для відображення даних і 1 для створення. Приклад:





Після цього ми реалізували контролери, які будуть приймати запит, відправляти на опрацювання та повертати результати. Приклад:

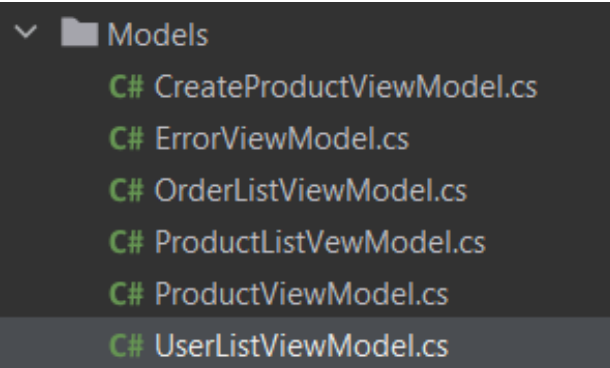


Далі ми дописали класи сутностей для наших моделей:

```
using EShop.Domain.Models;

namespace EShop.Presentation.Models;

public class UserListViewModel
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public UserType UserType { get; set; }
}
```



Далі ми дописали функціонал у наших сервісах(приклад):

```
public class ProductService : IProductService
{
    private readonly IProductRepository _repository;
    private readonly IMapper _mapper;
    private readonly string _uploadPath = Path.Combine("wwwroot", "images", "products");

    public ProductService(IProductRepository repository, IMapper mapper)
    {
        _repository = repository;
        _mapper = mapper;
    }

    public async Task<GetProductDto> GetProductById(Guid id)
    {
        var product = await _repository.GetBySpecification(spec: new GetProductDetailsSpecification(id));
        if (product == null)
        {
            throw new Exception("Product not found");
        }

        var productDto = _mapper.Map<GetProductDto>(product);
        return productDto;
    }
}
```

Під час написання коду ми дотримувалися написання ‘чистого’ коду.

Також ми дописали класи репозиторіїв до нашого проекту:

```
namespace EShop.Application.ProductType;

public interface IProductTypeService
{
    Task<List<Domain.Models.ProductType>> GetProductTypes();
    Task<Domain.Models.ProductType> CreateProductType(CreateProductTypeDto productType);
    Task Delete(Guid id);
}
```

Класи інтерфейсів репозиторії в ASP.NET в основному визначають, як ми можемо спілкуватися з базою даних у нашому веб-застосунку.

Також у нашому проєкті є логування дій та помилок користувача.

```
builder.Host.UseSerilog((configureLogger: (context, loggerConfig) => loggerConfig.ReadFrom.Configuration(context.Configuration)));
```

Ну і на кінець ми реалізували шаблони відображення для виконання use-case користувача з використанням Razor view engine у ASP.NET

```
Model CreateProductViewModel
@{
    ViewBag.Title = "title";
    Layout = "_Layout";
}

<h1>Create New Product</h1>

<form asp-action="Create" method="post" enctype="multipart/form-data">
    <div class="form-group">
        <label for="Price">Price:</label>
        <input type="number" class="form-control" id="Price" name="Price" value="@Model.Price">
    </div>
    <div class="form-group">
        <label for="Quantity">Quantity:</label>
        <input type="number" class="form-control" id="Quantity" name="Quantity" value="@Model.Quantity">
    </div>
    <div class="form-group">
        <label for="Description">Description:</label>
        <input type="text" class="form-control" id="Description" name="Description" value="@Model.Description">
    </div>
    <div class="form-group">
        <label for="Image">Image:</label>
        <input type="file" class="form-control-file" id="Image" name="Image">
    </div>
</form>
```

Views

- Order
- Product
- Shared
- User
- \_ViewImports.cshtml
- \_ViewStart.cshtml

# Проміжний результат виконання проекту 'Prykladna-SHOP'

08.05.2024

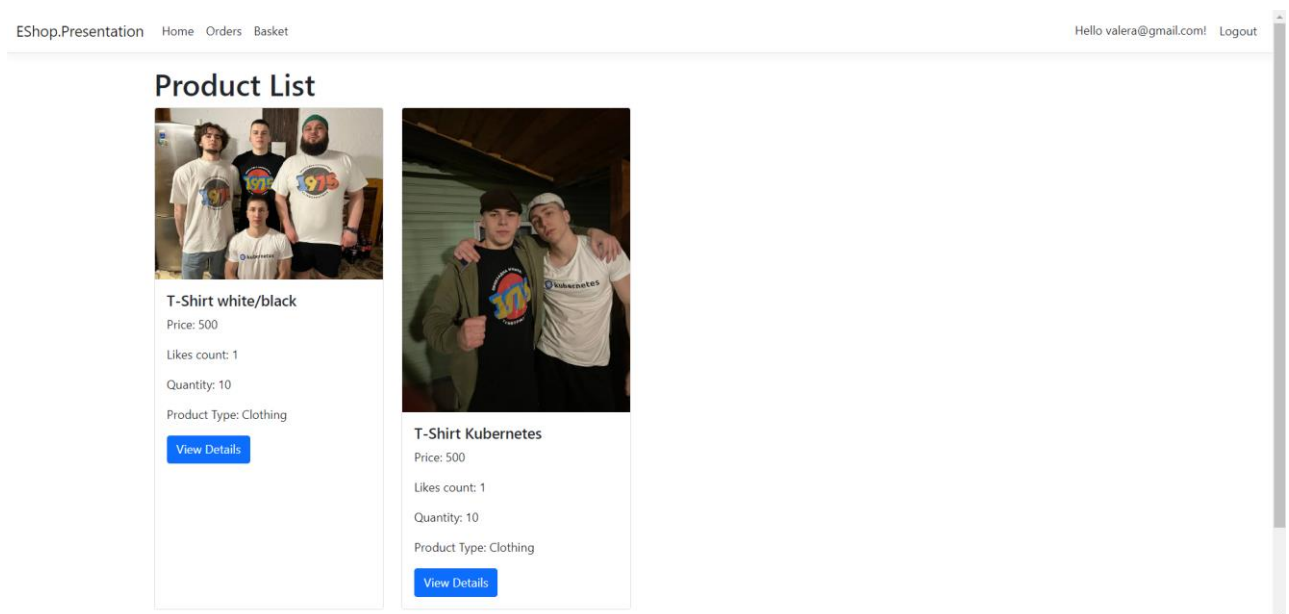
Протягом останніх 8 тижнів ми впроваджували Use Cases, підключали та використовували базу даних, розробляли фронтенд, додавали користувачів та інше...

Сьогодні ми представляємо готову першу демонстраційну версію нашого стартапу для нашого факультету, який у майбутньому може стати чимось 'більше'!

Проте, настав час повернутися до розробки та перевірити, як все виглядає та працює.

## Дизайн та Користувацький Інтерфейс

Основна сторінка:



Сторінка з інформацією про товар:

[Back](#)

## Product Details



T-Shirt white/black

Price: \$500

Quantity: 10

Product Type: Clothing

Likes count: 1

Quantity:

[Add to Cart](#)[Unlike](#)

Сторінка для замовлення:

## Order List

Order ID	Order Time	Address	Total Price	Total Items Count	Order Status	Action
44370813-9a49-4b87-9322-bc4f2cd155d3	25.04.2024 15:24:30 +00:00	Львів	\$500	1	NotConfirmed	<a href="#">Confirm</a> <a href="#">Cancel</a> <a href="#">Change Address</a>

Сторінки для реєстрації та авторизації клієнта:

## Register

Create a new account.

[Register](#)

## Login

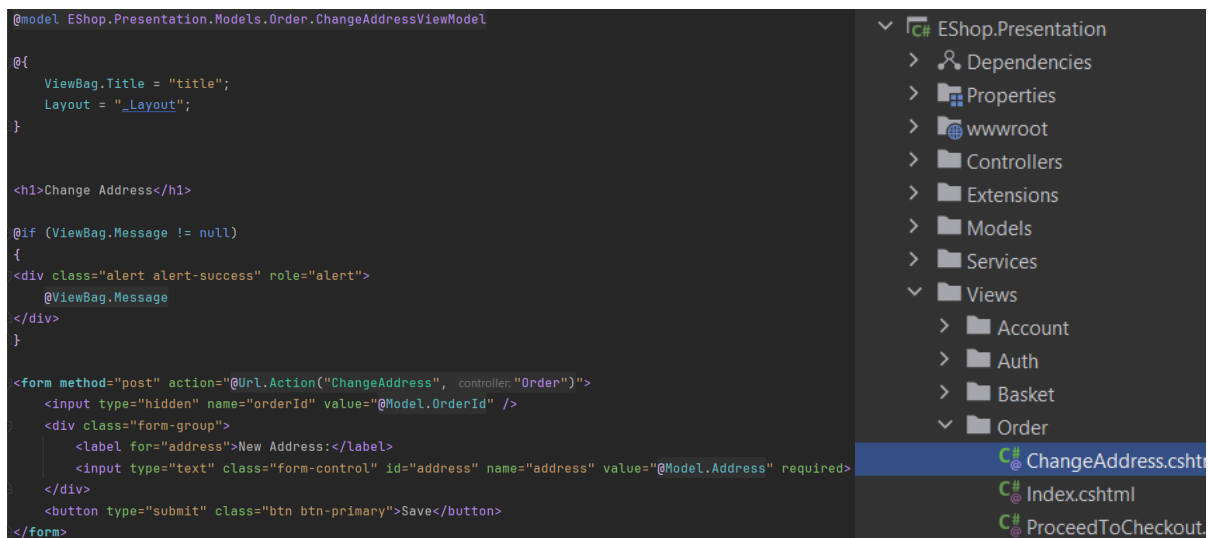
Use a local account to log in.

  
test@gmail.com  
.....☐ Remember me?[Log in](#)[Forgot your password?](#)[Register as a new user](#)

Фронтенд програми знаходиться у файлах cshtml у папці View, яка відноситься до області Presentation. Для стилізації використовується Bootstrap, який вже є вбудованим у структуру проекту. Крім того, для

повернення та відображення даних у проєкті використовується технологія Razor.

\*Приклад коду в програмі:



```
@model EShop.Presentation.Models.Order.ChangeAddressViewModel

@{
    ViewBag.Title = "title";
    Layout = "_Layout";
}

<h1>Change Address</h1>

@if (ViewBag.Message != null)
{
    <div class="alert alert-success" role="alert">
        @ViewBag.Message
    </div>
}

<form method="post" action="@Url.Action("ChangeAddress", controller: "Order")">
    <input type="hidden" name="orderId" value="@Model.OrderId" />
    <div class="form-group">
        <label for="address">New Address:</label>
        <input type="text" class="form-control" id="address" name="address" value="@Model.Address" required>
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
</form>
```

The right sidebar shows the file explorer for the 'EShop.Presentation' project, with the 'Views' folder expanded and 'ChangeAddress.cshtml' selected.

## Бекенд та Логіка Додатку

Наша команда дотримувалась попередньо визначених 'Юз кейс' діаграм та вимог до проєкту. Для зберігання, обробки, додавання та перегляду даних в проєкті ми використовуємо Postgres. У папці Presentation знаходиться папка Controllers, де відбувається обробка всіх запитів клієнта та подальше передавання інформації. У папці Application розміщені всі сервіси та їхні інтерфейси. Також варто відзначити, що ми використовуємо юніт-тести, для перевірки правильності та чіткості виконання коду. До того ж, ми використовуємо Docker, де ми маємо окремі контейнери, що містять дані бази даних та запуск сервера. Запуск відбувається за допомогою Dockerfile та Docker Compose файлу. Наш проєкт відповідає всім вимогам чистої архітектури та архітектурному патерну MVC!

## \*Приклад коду в програмі:

```
public async Task<IActionResult> AddItem(Guid productId, decimal price, int quantity)
{
    var userId String = User.FindFirstValue(claimType: ClaimTypes.NameIdentifier);
    if (userId == null) {
        return RedirectToAction("Login", controllerName: "Auth");
    }
    await _basketService.AddItemToBasket(userId, productId, price, quantity);
    return RedirectToAction("Index");
}

[HttpPost]
public async Task<IActionResult> UpdateQuantity(Guid basketId, Guid basketItemId, int quantity)
{
    await _basketService.SetQuantities(basketId, basketItemId, quantity);
    return RedirectToAction("Index");
}

[HttpPost]
public async Task<IActionResult> RemoveItem(Guid basketId, Guid itemId)
{
    await _basketService.RemoveItemFromBasket(basketId, itemId);
    return RedirectToAction("Index");
}
```

- ✓ **C# EShop.Presentation**
  - > Dependencies
  - > Properties
  - > wwwroot
  - ✓ **Controllers**
    - C# AccountController.cs
    - C# AuthController.cs
    - C# BasketController.cs
    - C# OrderController.cs
    - C# ProductController.cs
    - C# UserController.cs

```
public async Task ChangeOrderStatus(Guid orderId, Status status)
{
    var order = await _repository.GetByIdAsync(orderId);
    if (order is null)
    {
        throw new Exception(message: "Order not found");
    }
    order.Status = status;
    await _repository.UpdateAsync(order);
}
```

- ✓ **C# EShop.Application**
  - > Dependencies
  - > Abstractions
  - > Basket
  - ✓ **Order**
    - C# CreateOrderDto.cs
    - C# GetAllOrdersDto.cs
    - C# GetOrderDto.cs
    - C# IOrderRepository.cs
    - C# IOrderService.cs
    - C# OrderService.cs

```
[Fact]
public async Task SetQuantities_WhenBasketNotExists_ShouldThrowException()
{
    // Arrange
    var basketId = Guid.NewGuid();
    var itemId = Guid.NewGuid();
    _basketRepository.AnyAsync(basketId).Returns(returnThis: Task.FromResult(false));

    // Act
    Func<Task> act = async () => await _basketServices.SetQuantities(basketId, itemId, quantity: 5);

    // Assert
    await act.Should().ThrowAsync<Exception>().WithMessage(expectedWildcardPattern: "Basket not found");
}
```

- ✓ **C# EShop.UnitTests**
  - > Dependencies
  - C# BasketTests.cs
  - C# GlobalUsings.cs
  - C# OrderTests.cs
  - C# ProductTests.cs
  - C# ProductTypeTests.cs
  - C# UserTests.cs

		software-engineering-pmi32	Running (2/3)	2.64%	45 minutes ago				
		eshop.db-1	Running	0.64%	5500:5432	45 minutes ago			
		eshop.presentation-1	Exited (143)	0%	5000:8080	45 minutes ago			
		eshop.seq-1	Running	2%	5341:5341 <a href="#">Show all ports (2)</a>	45 minutes ago			

## Підсумок

Ось так виглядає попередній проект перед релізом продукту. Він буде доступний для користувачів на будь-якому браузері, а також зручний для використання на мобільних телефонах, оскільки підтримує розширення від 375 px до +4k px! Тому не зволікайте, а придбайте продукцію прикладної швидше!

*\*Приклад коду в програмі - код в прикладі демонструє архітектуру та функціонал проекту. Він носить ілюстративний характер і не є реальним! Цей приклад коду показує, як виглядає використання продукту користувачем з середини.*