МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОСТРОЗЬКА АКАДЕМІЯ»

КАФЕДРА ЕКОНОМІКО-МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗВІТ

про виробничу практику

Лободи Антоніни Олександрівни

(ПІБ студента)

122 Комп’ютерні науки

(спеціальність)

Комп’ютерні науки

(освітньо-професійна програма)

Керівник практики                    \_\_\_\_\_\_\_\_  Анатолій ВЛАСЮК

Завідувач кафедри                       \_\_\_\_\_\_\_\_ Анатолій ВЛАСЮК

Захист: \_\_\_\_\_\_\_\_\_\_

(дата)

Оцінка:

за національною шкалою \_\_\_\_\_\_\_\_\_

кількість балів \_\_\_\_\_\_\_\_\_\_

за шкалою ECTS\_\_\_\_\_\_\_\_\_\_

Острог, 2022

Зміст

[**Вступ** 3](#_Toc96766989)

[**Тема: Розробка back-end частини онлайн магазину на ASP.NET Core** 3](#_Toc96766990)

[**Порядок виконання завдань:** 3](#_Toc96766991)

[**Теоретичні відомості** 3](#_Toc96766992)

[**Опис задачі** 4](#_Toc96766993)

[**Опис реалізації** 4](#_Toc96766994)

[**Стек технологій** 5](#_Toc96766995)

[Література: 16](#_Toc96766996)

**Вступ**

**Тема: Розробка back-end частини онлайн магазину на ASP.NET Core**

**Мета роботи:** Навчитися працювати з технологією **ASP.NET.** Ознайомитися і використати в своєму проекті **Clean Architecture.**

**Порядок виконання завдань:**

1. Засвоїти теоретичний матеріал.
2. Створити проект на **ASP.NET**
3. Для виведення результатів використати **Swagger**
4. Реалізувати в проекті **Clean Architecture**
5. Створити реляційну БД третього нормального типу
6. Створити **CRUD** для кожної таблиці
7. Реалізувати валідацію вхідних данних
8. Реалізація функціоналу **back-end**
9. …

**Теоретичні відомості**

ASP.NET Core представляє технологію для створення веб-додатків на платформі .NET, яку розвиває компанія Microsoft. Як мови програмування для розробки програм на ASP.NET Core використовуються C# і F#.

ASP.NET Core дозволяє створювати веб-застосунки за допомогою різних моделей розробки.

• Перш за все це базовий ASP.NET Core, який підтримує всі основні моменти, необхідні для роботи сучасного веб-програми: маршрутизація, конфігурація, логування, можливість працювати з різними системами баз даних і т.д. У фреймворк був доданий так званий Minimal API – мінімізована спрощена модель, яка ще спростила процес розробки та написання коду програми. Решта моделей розробки працють поверх базового функціоналу ASP.NET Core

• ASP.NET Core MVC представляється у загальному вигляді побудови програми навколо трьох основних компонентів – Model (моделі), View (представлення) та Controller (контролери), де моделі відповідають за роботу з даними, контролери представляють логіку обробки запитів, а уявлення визначають візуальну складову.

• Razor Pages представляє модель, за якою за обробку запиту відповідають спеціальні сутності - сторінки Razor Pages. Кожну таку сутність можна асоціювати з окремою веб-сторінкою.

• ASP.NET Core Web API представляє реалізацію патерну REST, у якому кожному типу http-запиту (GET, POST, PUT, DELETE) призначений окремий ресурс. Такі ресурси визначаються як методів контролера Web API. Ця модель особливо підходить для односторінкових програм, але не тільки.

• Blazor представляє фреймворк, який дозволяє створювати інтерактивні програми як на стороні сервера, так і на стороні клієнта та дозволяє задіяти на рівні браузера низькорівневий код WebAssembly.

**Swagger** — це потужний, але простий у використанні набір інструментів для розробників API команд, що дає змогу розробляти весь життєвий цикл API, від проектування та документації до тестування та розгортання.

Swagger складається з поєднання безкоштовних і комерційно доступних інструментів з відкритим кодом, які дозволяють будь-кому, від технічних інженерів до аутсайдерів, створювати чудові API.

Swagger створено компанією SmartBear Software, лідером інструментів якості програмного забезпечення для команд. SmartBear стоїть за деякими з найбільших імен у сфері програмного забезпечення, включаючи Swagger, SoapUI та QAComplete.

Чиста архітуктура

**Опис задачі**

Створити бекенд онлайн-магазину різних категорій товарів (подібний до інтернет-магазину Rozetka) з його базовими функціями.

**Опис реалізації**

Розробка бекенду на технології ASP.NET Core 5 з використанням чистої архітектури і засобом примітивного відображення запитів API Swagger.

**Стек технологій**

* ASP.NET Core
* Swagger

**Біліотеки:**

* AutoMapper
* Microsoft.EntityFramworkCore
* Microsoft.EntityFramworkCore.SQLServer
* Syroot.Windows.IO.KnownFolder
* FluentValidation
* MediatR

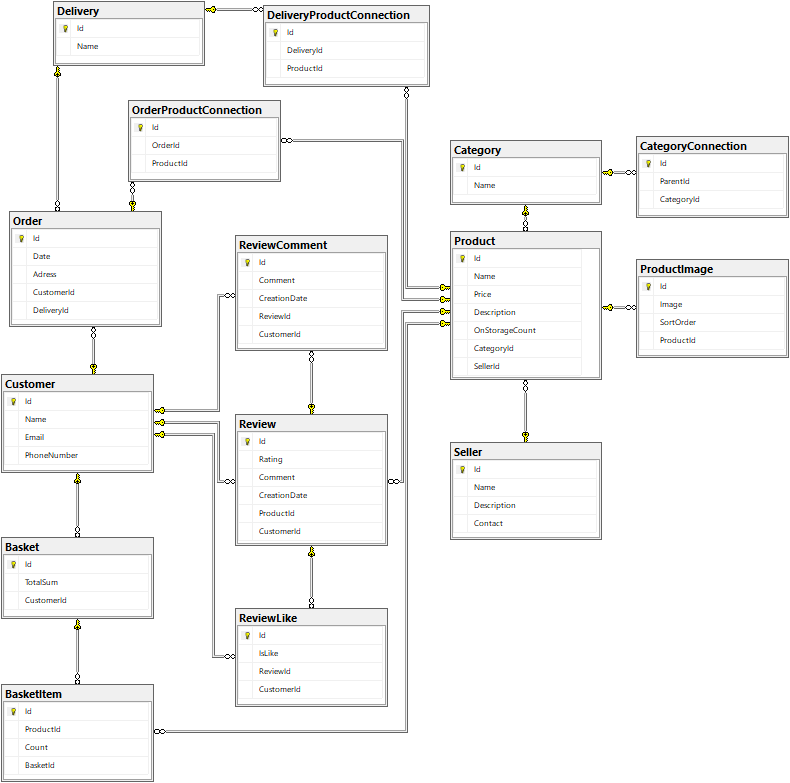


Рис.1 Модель БД

Хід роботи:

Створюєм проект на ASP.NET Core 5 і три папки Core, Infrastructure, Presentation.

В папці Core створюєм такі підпроекти:

Shop.Application – в ньому розміщуються команди для запитів, валідація, інтерфейс для бідключення до БД.

Shop.Domain – тут описуються сутності БД

В папці Infrastructure створюємо такі підпроекти:

Shop.Persistence – тут описується підключення та ініціалізація БД

В папці Presentation створюємо такі підпроекти:

DemoApi – тут описується простий API за допомогою Swagger.

Після створення базової структури бекенда, створюєм сутності таблиць БД.

Лістинг 1 Приклад оформлення моделі сутності

|  |
| --- |
| public class Customer : EntityBase  {  public Customer()  {  this.Order = new HashSet<Order>();  }  public string Name { get; set; }  public string Email { get; set; }  public string PhoneNumber { get; set; }  public virtual ICollection<Order> Order { get; set; }  } |

Створено такий список моделей

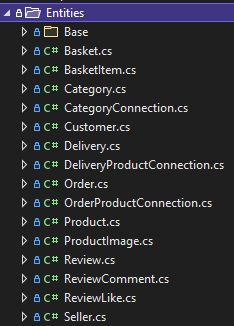


Рис. 2 Список фалів моделей

Створюємо DataBaseContext в якому реалізуєм DbContext i IDataBaseContext.

Лістинг 2 Реалізація DataBaseContext

|  |
| --- |
| public partial class DataBaseContext : DbContext, IDataBaseContext  {  public DataBaseContext(DbContextOptions<DataBaseContext> options)  : base(options)  {  }  public virtual DbSet<Basket> Basket { get; set; }  public virtual DbSet<BasketItem> BasketItem { get; set; }  public virtual DbSet<Category> Category { get; set; }  public virtual DbSet<CategoryConnection> CategoryConnection { get; set; }  public virtual DbSet<Customer> Customer { get; set; }  public virtual DbSet<Delivery> Delivery { get; set; }  public virtual DbSet<DeliveryProductConnection> DeliveryProductConnection { get; set; }  public virtual DbSet<Order> Order { get; set; }  public virtual DbSet<OrderProductConnection> OrderProductConnection { get; set; }  public virtual DbSet<Product> Product { get; set; }  public virtual DbSet<ProductImage> ProductImage { get; set; }  public virtual DbSet<Seller> Seller { get; set; }  public virtual DbSet<Review> Review { get; set; }  public virtual DbSet<ReviewComment> ReviewComment { get; set; }  public virtual DbSet<ReviewLike> ReviewLike { get; set; }  } |

Лістинг 3 Реалізуємо SQLServer підключення

|  |
| --- |
| public static class DependencyInjection  {  public static IServiceCollection AddPersistence(this IServiceCollection services,  IConfiguration configuration)  {  var connectionString = configuration["DbConnection"];  services.AddDbContext<DataBaseContext>(options =>  {  options.UseSqlServer(connectionString);  });  services.AddScoped<IDataBaseContext>(provider =>  provider.GetService<DataBaseContext>());  return services;  }  } |

В підпроекті Shop.Aplication релізуємо весь функціонал онлайн магазину у вигляді команд, які мають таку структуру:

Три папки Create, Dеlete, Update

Будова однієї папки:

Лістинг 4.1 Релізуєм Команду. Це модель за якою будуть надходити дані від користувача.

|  |
| --- |
| public class CreateCategoryCommand  : IRequest<CategoryVm>  {  public long Id { get; set; }  public string Name { get; set; }  public long ParentId { get; set; }  } |

Лістинг 4.2 Реалізуємо Handler, який оброблятиме запит і повертатиме відповідь.

|  |
| --- |
| public class CreateCategoryCommandHandler  : HandlersBase, IRequestHandler<CreateCategoryCommand, CategoryVm>  {  public CreateCategoryCommandHandler(IDataBaseContext dbContext, IMapper mapper)  : base(dbContext, mapper) { }  public async Task<CategoryVm> Handle(CreateCategoryCommand request, CancellationToken cancellationToken)  {  var category = new Category  {  Name = request.Name  };  if (!\_dbContext.Category.Any(x => x.Name == request.Name))  {  await \_dbContext.Category.AddAsync(category, cancellationToken);  }  if (request.ParentId != 0)  {  var categoryConnection = new CategoryConnection  {  ParentId = request.ParentId,  Category = category,  };  await \_dbContext.CategoryConnection.AddAsync(categoryConnection, cancellationToken);  }  await \_dbContext.SaveChangesAsync(cancellationToken);  return \_mapper.Map<CategoryVm>(category);  }  } |

Лістинг 4.3 Реалізуємо валідацію вхідних даних.

|  |
| --- |
| public class CreateCategoryCommandValidator  : AbstractValidator<CreateCategoryCommand>  {  private CategoryExistTask existTask;  public CreateCategoryCommandValidator(IDataBaseContext dbContext)  {  existTask = new CategoryExistTask(dbContext);  RuleFor(createCategoryCammandValidator =>  createCategoryCammandValidator.Name)  .NotEmpty().WithMessage("Name is required.")  .NotNull().WithMessage("Name can not be aqueal null.");  }  } |

Запити Get мають подібну структуру. Різниця полягає в тому, що Command замінені на Query (хоча вони виконують аналогічну функцію) і тут додано VM і LookupDto моделі. Вони створенні для коректного виведення інформації на фронтенд і релізують мапінг.

Лістинг 4.4 Реалізація Vm моделі яка виводить список користувачів представлених моделлю LookupDto.

|  |
| --- |
| public class CustomersListVm  {  public IList<CustomersLookupDto> Customers { get; set; }  } |

Лістинг 4.5 Релізація LookupDto моделі і мапінгу.

|  |
| --- |
| public class CustomersLookupDto : IMapWith<Customer>  {  public long Id { get; set; }  public string Name { get; set; }  public string Email { get; set; }  public string PhoneNumber { get; set; }  public void Mapping(Profile profile)  {  profile.CreateMap<Customer, CustomersLookupDto>()  .ForMember(customerDto => customerDto.Id,  opt => opt.MapFrom(customer => customer.Id))  .ForMember(customerDto => customerDto.Name,  opt => opt.MapFrom(customer => customer.Name))  .ForMember(customerDto => customerDto.Email,  opt => opt.MapFrom(customer => customer.Email))  .ForMember(customerDto => customerDto.PhoneNumber,  opt => opt.MapFrom(customer => customer.PhoneNumber));  }  } |

Також ми в цьому підпроекті вказуємо логіку валідації, мапінгу і пагінації.

Лістинг 4.6 Релізація логіки валідації

|  |
| --- |
| public class ValidationBehavior<TRequest, TResponse>  : IPipelineBehavior<TRequest, TResponse>  where TRequest : IRequest<TResponse>  {  private readonly IEnumerable<IValidator<TRequest>> \_validators;  public ValidationBehavior(IEnumerable<IValidator<TRequest>> validators) =>  \_validators = validators;  public Task<TResponse> Handle(TRequest request,  CancellationToken cancellationToken,  RequestHandlerDelegate<TResponse> next)  {  var context = new ValidationContext<TRequest>(request);  var failures = \_validators  .Select(v => v.Validate(context))  .SelectMany(result => result.Errors)  .ToList();  if (failures.Count != 0)  {  throw new ValidationException(failures);  }  return next();  }  } |

Лістинг 4.7 Релізація логіки мапінгу

|  |
| --- |
| public class AssemblyMappingProfile : Profile  {  public AssemblyMappingProfile(Assembly assembly) =>  ApplyMappingsFromAssembly(assembly);  private void ApplyMappingsFromAssembly(Assembly assembly)  {  var types = assembly.GetExportedTypes()  .Where(t => t.GetInterfaces()  .Any(i => i.IsGenericType &&  i.GetGenericTypeDefinition() == typeof(IMapWith<>)))  .ToList();  foreach (var type in types)  {    var instance = Activator.CreateInstance(type);  var methodInfo = type.GetMethod("Mapping");  methodInfo?.Invoke(instance,new object[] { this });  }  }  } |

Лістинг 4.8 Релізація інтерфейсу мапінгу

|  |
| --- |
| public interface IMapWith<T>  {  void Mapping(Profile profile) =>  profile.CreateMap(typeof(T), GetType());  } |

Лістинг 4.9 Реалізація пагінації

|  |
| --- |
| public class PaginatedList<T> : List<T>  {  public int PageIndex { get; private set; }  public int TotalPages { get; private set; }  public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)  {  PageIndex = pageIndex;  TotalPages = (int)Math.Ceiling(count / (double)pageSize);  AddRange(items);  }  public bool HasPreviousPage => PageIndex > 1;  public bool HasNextPage => PageIndex < TotalPages;  public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)  {  var count = await source.CountAsync();  var items = await source.Skip((pageIndex - 1) \* pageSize).Take(pageSize).ToListAsync();  return new PaginatedList<T>(items, count, pageIndex, pageSize);  }  } |

В папці Exceptions створено кастомні помилки і їх валідацію.

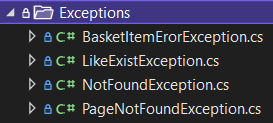


Рис. 3 Список створених Exceptions

В папці Interfaces розміщується інтерфейс IDataBaseContext.

Лістинг 4.10 Реалізація інтерфейсу IDataBaseContext

|  |
| --- |
| public interface IDataBaseContext  {  DbSet<Basket> Basket { get; set; }  DbSet<BasketItem> BasketItem { get; set; }  DbSet<Category> Category { get; set; }  DbSet<CategoryConnection> CategoryConnection { get; set; }  DbSet<Customer> Customer { get; set; }  DbSet<Delivery> Delivery { get; set; }  DbSet<DeliveryProductConnection> DeliveryProductConnection { get; set; }  DbSet<Order> Order { get; set; }  DbSet<OrderProductConnection> OrderProductConnection { get; set; }  DbSet<Product> Product { get; set; }  DbSet<ProductImage> ProductImage { get; set; }  DbSet<Seller> Seller { get; set; }  DbSet<Review> Review { get; set; }  DbSet<ReviewComment> ReviewComment { get; set; }  DbSet<ReviewLike> ReviewLike { get; set; }  Task<int> SaveChangesAsync(CancellationToken cancellationToken);  } |

Лістинг 4.11 Релізація підключення сервісів пагінації і валідації.

|  |
| --- |
| public static class DependencyInjection  {  public static IServiceCollection AddApplication(  this IServiceCollection service)  {  service.AddMediatR(Assembly.GetExecutingAssembly());  service.AddValidatorsFromAssemblies(new[] { Assembly.GetExecutingAssembly() });  service.AddTransient(typeof(IPipelineBehavior<,>),  typeof(ValidationBehavior<,>));  return service;  }  } |

В підпроекті DemoApi релізовуємо котролери для запитів, правила для виведення помилок для користувача, і моделі для отримання даних з фронтенду.

BasketController – містить такі запити Create, Delete, Get

BasketItemController - Create, Delete, Update

CategoryConnectionController - Create, Delete, Update, Getlist, GetSubcategoriesList

CategoryController - Create, Delete, Update, Getlist, getPaginatedlist

CustomerController - Create, Delete, Update, Getlist

DeliveryController - Create, Delete, Update, Getlist, Get

DeliveryProductConnectionController- Create, Delete, Update, Getlist

OrderController - Create, Delete, Update, Get, GetOrdersListForCustomer, GetOrdersList, GetOrdersPaginatedListForCustomer, GetOrdersPaginatedList

OrderProductConnectionController - Create, Delete, Update, Get, Getproduct

ProductController - Create, Delete, Update, Getlist, Get, GetPaginatedList, GetListByCategory, GetSortListByPriceIncrease, GetSortListByPriceFalling, GetSortListByRating, filterByRating, filterBySeller, filterByPrice

ProductImageController - Create, Delete, GetProductImagesList, GetProductImagesListForProduct

ReviewCommentController - Create, Delete, Get, GetReviewCommentsList, GetReviewCommentForReview

ReviewController - Create, Delete, GetReviewsList, GetReviewsListForProduct, GetReviewsListForProductByDate, GetReviewsListForProductByLike

ReviewLikeController - Create, Delete, Get, GetLikesForReview

SearchController - Search

SellerController - Create, Delete, Update, Get, Getlist

В папці Middleware описано виведення помилки у формати json

В папці Models місцятся модельки для введення даних.

**Висновок**

Під час проходження виробничої практики

# Література:

<https://habr.com/ru/post/464185/>

https://metanit.com/sharp/mvc.php