1. **ПРОЕКТИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**
   1. **Описание предметной области**

Интернет-магазин одежды - это электронная платформа, предоставляющая возможность покупки товаров из различных категорий одежды, обуви и аксессуаров. В таких магазинах представлен широкий ассортимент товаров для мужчин, женщин и детей, включая различные стили, бренды, размеры и цвета.

Покупатели могут легко найти необходимые товары, просматривая каталог, фильтруя по цене, размеру, цвету и другим параметрам. Также доступны подробные описания товаров, фотографии и отзывы покупателей, что позволяет сделать правильный выбор.

Интернет-магазины одежды обеспечивают удобные условия покупки, включая быструю доставку, возможность возврата товара, различные способы оплаты и консультации специалистов по любым вопросам. Также часто проводятся акции, распродажи и специальные предложения для постоянных клиентов.

Таким образом, интернет-магазины одежды предоставляют возможность удобно и быстро приобрести модные и качественные товары, не выходя из дома, что делает процесс шопинга приятным и эффективным.

* 1. **Создание диаграммы классов**

Создание диаграммы классов является одним из этапов проектирования программного обеспечения в объектно-ориентированном программировании. Диаграмма классов представляет собой визуальное представление структуры классов и их взаимосвязей в рамках предметной области.

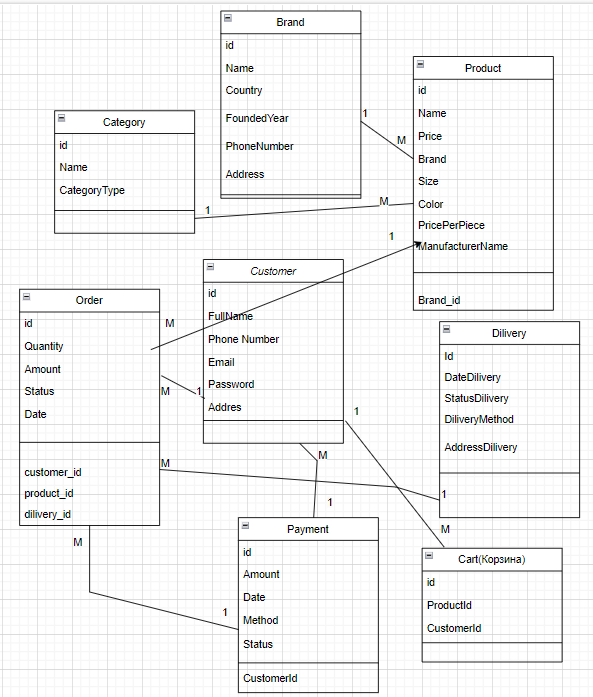
****

Рис. 1. Диаграмма классов интернет-магазина одежды

На данной диаграмме классов представлены следующие объекты:

- Brand: Представляет бренд, под которым продаются товары.

- Product: Представляет товар, доступный для продажи в магазине.

- Customer: Представляет клиента, который совершает покупки в магазине.

- Order: Представляет заказ, сделанный клиентом.

- Delivery: Представляет способ доставки заказа.

- Payment: Представляет метод оплаты заказа.

- Cart: Представляет корзину, где клиент хранит товары, которые хочет приобрести.

Связи между классами определены следующим образом:

- Brand имеет связь "один ко многим" с Product, что означает, что один бренд может иметь много продуктов.

- Product имеет связь "один ко многим" с Order, что указывает на то, что один продукт может быть включен в несколько заказов. Такие связи присутствуют и у других классов.

* 1. **Создание GitHub-репозитории проекта**

Создание репозитория проекта на GitHub - это процесс размещения исходного кода проекта на хостинге GitHub, который является популярной платформой для хранения и управления репозиториями кода. Создание репозитория позволяет совместно работать над проектом, отслеживать изменения кода, управлять версиями и делиться кодом с другими пользователями.

Для создания репозитория проекта на GitHub необходимо выполнить следующие шаги:

1. Зарегистрироваться на GitHub, если еще нет аккаунта.

2. Зайти на GitHub и войти в аккаунт.

3. Нажать на кнопку "New" (Создать) в правом верхнем углу страницы.

4. Ввести название репозитория, описание и выбрать опции видимости (публичный или приватный).

5. Дополнительно можно выбрать опции создания файла README.md, добавления файла .gitignore и выбрать лицензию.

6. Нажать на кнопку "Create repository" (Создать репозиторий).

После создания репозитория будут предоставлены инструкции по добавлению исходного кода, работе с Git, пушу и пуллу изменений и другим возможностям GitHub.

1. **ВВЕДЕНИЕ В ASP.NET WEB API**
   1. **Организация структуры проекта по Clean Architecture**

Проект интернет-магазина основан на принципах Clean Architecture. Clean Architecture представляет собой набор принципов проектирования программного обеспечения, направленных на создание более гибких, устойчивых и тестируемых приложений.

Структура проекта интернет-магазина включает четыре основных слоя:

- Application: Этот слой содержит бизнес-логику приложения, которая не зависит от конкретных технологий или инфраструктуры.

- Domain: Здесь находятся доменные модели и бизнес-правила приложения.

- Infrastructure: В этом слое расположен код, который взаимодействует с внешними технологиями и инфраструктурой, такими как базы данных, веб-сервисы и файловые системы.

- WebApi: Данный слой содержит код, обеспечивающий интерфейс RESTful API для взаимодействия с приложением.

Взаимодействие между слоями осуществляется через интерфейсы. Это означает, что код в одном слое не зависит от конкретной реализации в другом слое. Такой подход облегчает внесение изменений в один слой без воздействия на другие слои.

* 1. **Создание моделей приложения**

В данной части будет представлено описание моделей приложения интернет-магазина, построенного на основе принципов Clean Architecture.

Приложение состоит из четырех основных слоев: WebApi, Infrastructure, Domain, Application. Доменный слой содержит модели, которые представляют основные сущности приложения:

- BaseEntity: базовый класс для всех сущностей

- Brand: бренд

- Cart: корзина

- Category: категория

- Customer: клиент

- Delivery: доставка

- Order: заказ

- Payment: оплата

- Product: продукт

Сущности содержат свойства и методы, которые используются для доступа к данным и их управления. Эти сущности применяются в различных частях приложения, таких как пользовательский интерфейс, бизнес-логика и слой доступа к данным.

1. **РАБОТА С БАЗОЙ ДАННЫХ**
   1. **Реализация архитектурного паттерна репозиторий**

Архитектурный паттерн "Репозиторий" (Repository) в C# часто используется для абстрагирования доступа к данным в приложении, что позволяет разделить бизнес-логику и слой доступа к данным. На проекте интернет-магазина написан следующий код:

public interface IBaseRepository<TEntity> where TEntity : BaseEntity

{

Task<TEntity> GetAsync(Guid id, CancellationToken token = default);

Task<IEnumerable<TEntity>> GetAllAsync(CancellationToken token = default);

Task<TEntity> CreateAsync(TEntity entity, CancellationToken token = default);

Task<bool> UpdateAsync(TEntity entity, CancellationToken token = default);

Task<bool> DeleteAsync(TEntity entity, CancellationToken token = default);

}

Данный код представляет интерфейс IBaseRepository, который является интерфейсом репозитория для работы с сущностями. В интерфейсе заданы методы для выполнения базовых операций с объектами определенного типа TEntity, где TEntity должен быть производным от класса BaseEntity.

Описание методов интерфейса IBaseRepository:

1. GetAsync(Guid id, CancellationToken token = default): Метод для асинхронного получения сущности по указанному идентификатору (id). Возвращает объект типа TEntity с указанным id. Данный метод может быть прерван при передаче токена отмены (token).

2. GetAllAsync(CancellationToken token = default): Метод для асинхронного получения всех сущностей определенного типа TEntity. Возвращает коллекцию объектов типа TEntity. Опционально можно передать токен отмены (token) для отмены операции.

3. CreateAsync(TEntity entity, CancellationToken token = default): Метод для асинхронного создания новой сущности определенного типа TEntity. Принимает объект типа TEntity, который необходимо создать. Возвращает созданный объект типа TEntity. Можно указать токен отмены (token) для отмены операции.

4. UpdateAsync(TEntity entity, CancellationToken token = default): Метод для асинхронного обновления сущности определенного типа TEntity. Принимает объект типа TEntity, который необходимо обновить. Возвращает булево значение, указывающее на успешность операции обновления. Можно передать токен отмены (token) для отмены операции.

5. DeleteAsync(TEntity entity, CancellationToken token = default): Метод для асинхронного удаления сущности определенного типа TEntity. Принимает объект типа TEntity, который необходимо удалить. Возвращает булево значение, указывающее на успешность операции удаления. Опционально можно передать токен отмены (token) для отмены операции.

Этот интерфейс определяет базовые методы, которые должны быть реализованы в репозитории для работы с объектами определенного типа TEntity, обеспечивая единый интерфейс доступа к данным для различных типов сущностей.

* 1. **Использование EntityFramework для работы с базой данных**

Entity Framework (EF) - это технология в рамках платформы .NET, которая предоставляет разработчикам удобный инструмент для взаимодействия с базой данных с использованием объектно-ориентированной модели (ORM). Она позволяет программистам работать с данными, представленными в виде объектов, а не делать прямые запросы к таблицам и записям в базе данных. На проекте интернет-магазина написан следующий код для использования Entity Framework:

public DbSet<Cart> Carts { get; set; }

public DbSet<Category> Categories { get; set; }

public DbSet<Customer> Customers { get; set; }

public DbSet<Delivery> Deliveries { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<Payment> Payments { get; set; }

public DbSet<Product> Products { get; set; }

Этот код представляет собой часть класса контекста данных (DbContext) в приложении, который использует Entity Framework для работы с базой данных. Каждое свойство типа DbSet<TEntity> представляет коллекцию сущностей определенного типа TEntity, соответствующих таблицам в базе данных. Это помогает упростить доступ к данным и использовать объектно-ориентированный подход при работе с базой данных, делая процесс разработки более удобным и эффективным.

* 1. **Миграция данных**

Миграция данных - это процесс переноса данных из одного источника данных в другой или изменение структуры данных в базе данных.

Миграция данных является важной составляющей процесса обновления и совершенствования системы, а также перехода на новые технологии и источники данных. Она помогает сохранить целостность данных и обеспечить их доступность и актуальность при изменении бизнес-требований и выходе новых версий программного обеспечения.

1. **Реализация методов контроллеров для обработки HTTP-запросов**
   1. Создание сервисов
   2. Создание DTO
   3. Маппинг объектов с использованием библиотеки AutoMapper
   4. Создание методов контроллеров для CRUD операций