Министерство просвещения Приднестровской Молдавской Республики

ГОУ СПО «Тираспольский техникум информатики и права»

**КУРСОВАЯ РАБОТА**

по МДК.01.01 Технология разработки и защиты баз данных

на тему: «Разработка системы продуктового магазина»

**Выполнил обучающийся**:

Полякова Анна Игоревна

**Специальность** 2.09.02.07

Технология разработки и защиты баз данных

**Руководитель:**

Преподаватель

информационных дисциплин

Надькин Леонид Юрьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

«\_\_\_\_» «\_\_\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_

дата подпись руководителя

Тирасполь, 2024

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_30j0zll)

[Глава 1. Теоретическая часть разработки программного обеспечения 5](#_1fob9te)

[1.1. Описание предметной области 5](#_3znysh7)

[1.2. Анализ существующих разработок 6](#_2et92p0)

[1.3. Описание выявленных достоинств и недостатков 9](#_tyjcwt)

[1.4. Используемый язык программирования C# 10](#_3dy6vkm)

[1.5. Общая постановка задачи 11](#_1t3h5sf)

[ЗАКЛЮЧЕНИЕ 14](#_4d34og8)

# **ВВЕДЕНИЕ**

**Актуальность** темы электронной торговли становится ключевым фактором для успеха компаний. Создание собственного онлайн магазина помогает улучшить эффективность бизнеса и оставаться конкурентоспособным. Целью данной курсовой работы является разработка веб-приложения продуктового магазина с использованием ASP.NET и C#, способного управлять ассортиментом продукции и заказами, оптимизировать процессы продаж и повысить уровень обслуживания клиентов.

**Целью** данной курсовой работы является разработка информационной системы продуктового магазина способного управлять ассортиментом продукции и заказами, оптимизировать процессы продаж и повысить уровень обслуживания клиентов.

Для выполнения курсовой работы были поставлены следующие **задачи**:

1. Реализация управления данными продуктового магазина
2. Возможность учета продуктов
3. Реализация системы заказов

**Объектом** данной работы является продуктовый магазин, то есть компания, занимающаяся продажей различных товаров **Предметом** данной работы является разработка информационной системы для управления продуктовым магазином. Это включает в себя проектирование и создание программного обеспечения, которое будет оптимизировать основные процессы работы с товарами, такие как учет товаров на складе, управление заказами, реализация учетных данных пользователей. Разработанная система будет предоставлять набор функций и возможностей для управления продуктовым ассортиментом, обработки заказов и взаимодействия с клиентами.

Метод исследования, используемый в данной работе для продуктового магазина, является аналитическим и систематическим анализом.

**В первой главе** к курсовой работы анализируются теоретические аспекты, связанные с продуктовым магазином, включая его основные концепции и функции.

**Во второй главе** работы определяется постановка задачи, где обсуждаются методы решения и предлагается обзор выбранного языка программирования для разработки программного обеспечения для управления продуктовым магазином. Эта глава также содержит описание разработанной программы.

В приложении работы приведен листинг разработанной программы.

# **ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

# **Описание предметной области**

**Магазин продуктов** специализируется на предоставлении клиентам доступа к широкому ассортименту товаров первой необходимости и является важным элементом современного общества. В наше время продуктовые магазины стали ключевым местом для приобретения продуктов как для личного потребления, так и для коммерческих целей, обеспечивая быструю и удобную покупку товаров.

**Основная цель** продуктового магазина заключается в обеспечении клиентов качественными товарами и предоставлении удобной и безопасной платформы для покупок. Это включает в себя функции организации товаров, системе учета пользователей, а также управление заказами и обеспечение надежной передачи информации о продуктах.

Продуктовый магазин обслуживает разнообразные категории клиентов, включая частных покупателей, предприятия, государственные учреждения и другие организации. Он также предоставляет разнообразные функциональные возможности, например, управление ассортиментом, заказами, платежами и доставкой, для обеспечения удобства покупателей и эффективного ведения бизнеса.

**Предметная область** продуктового магазина охватывает различные процессы и операции, связанные с управлением товарным ассортиментом и обслуживанием клиентов. Основные функции включают:

1. Управление запасами и складскими запасами, чтобы гарантировать наличие товаров для клиентов.
2. Оформление заказов и обработка транзакций для проведения покупок.
3. Обеспечение безопасной и надежной платформы для покупок и передачи информации о продуктах.
4. Ведение учета продаж и анализ данных для оптимизации бизнес-процессов.
5. Предоставление услуг доставки и возможности самовывоза для удобства клиентов.
6. Реализация учетных данных пользователей и истории заказов

**Целью** разработки информационной системы для управления продуктовым магазином является повышение операционной эффективности и качества обслуживания клиентов. Эта информационная система позволяет автоматизировать и оптимизировать основные процессы работы с товарами, что приводит к ускорению обработки заказов, снижению риска недостаточного запаса товаров и улучшению взаимодействия с покупателями.

В следующих разделах работы будет проведен обзор существующих информационных систем для управления продуктовыми магазинами, их функциональных возможностей и особенностей реализации. Будет проведен анализ их преимуществ и недостатков, а также выявлены основные требования и потребности пользователей в данной области. На основе этого анализа будет предложено описание разработанной программы, учитывающей специфику управления продуктовым магазином и включающей в себя необходимый набор функций для эффективной работы.

# **Анализ существующих разработок**

Анализ различных программ продуктовых магазинов в современном рынке имеет важное значение для понимания тенденций и развития этой отрасли. Давайте рассмотрим несколько популярных программ и оценим их преимущества, функциональность и эффективность.

**Цель анализа** - состоит в выявлении ключевых тенденций и лучших практик в области управления продуктовым магазином, а также в оценке потенциала для удовлетворения потребностей клиентов.

Рассмотрим продуктовый магазин Самокат:

Самокат (Рис. 1) - сервис заказа продуктов питания с доставкой из дарксторов. Самокат предлагает быструю доставку продуктов на дом, а также готовой еды и других товаров. Он обладает рядом дополнительных функций, таких как возможность заказа продуктов с бесплатной доставкой, а также предлагает различные акции и скидки. Самокат является одним из самых популярных сервисов доставки продуктов в России и используется миллионами пользователей. Он предоставляет доступ к своим услугам через веб-интерфейс и мобильные приложения

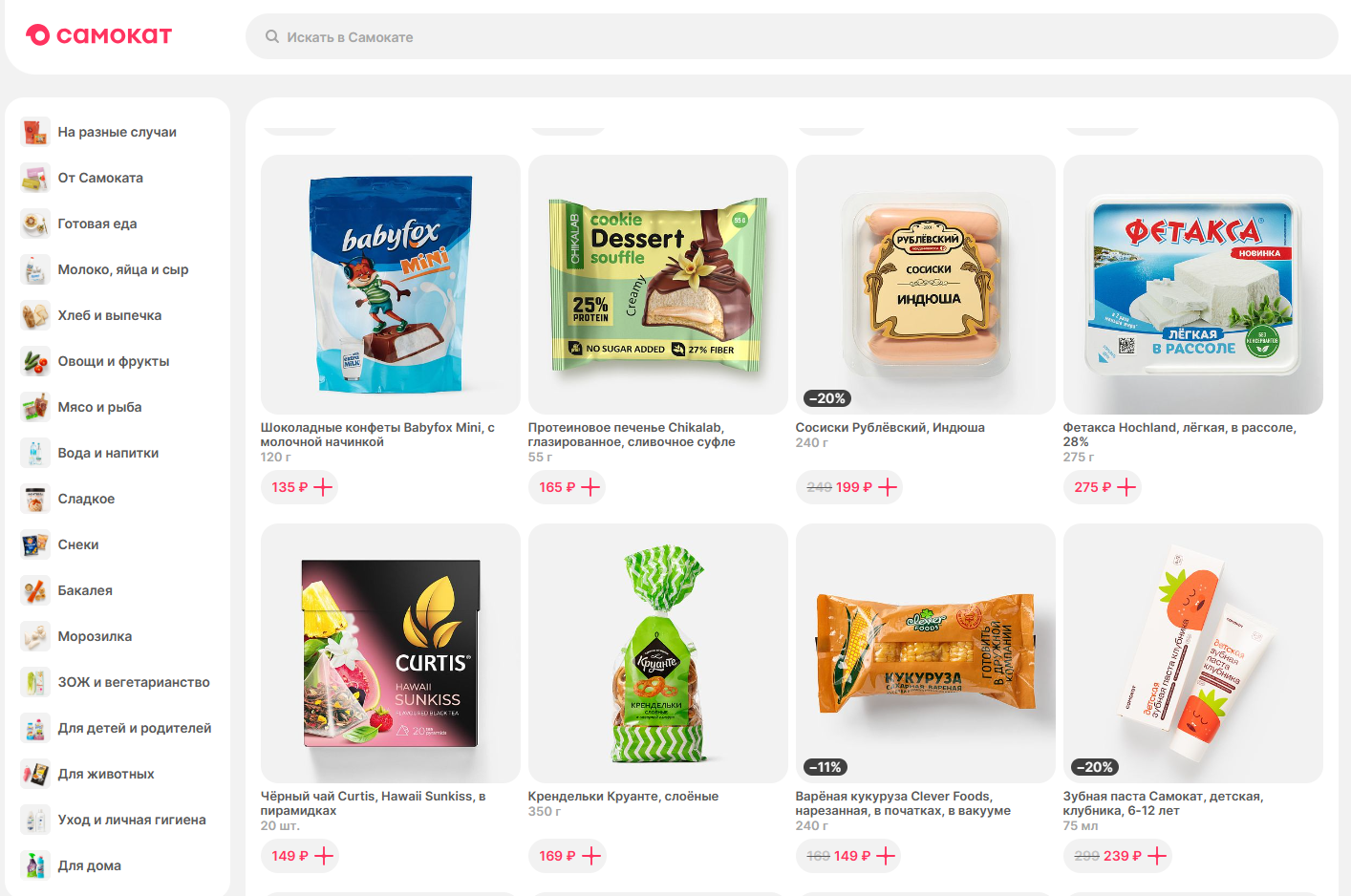
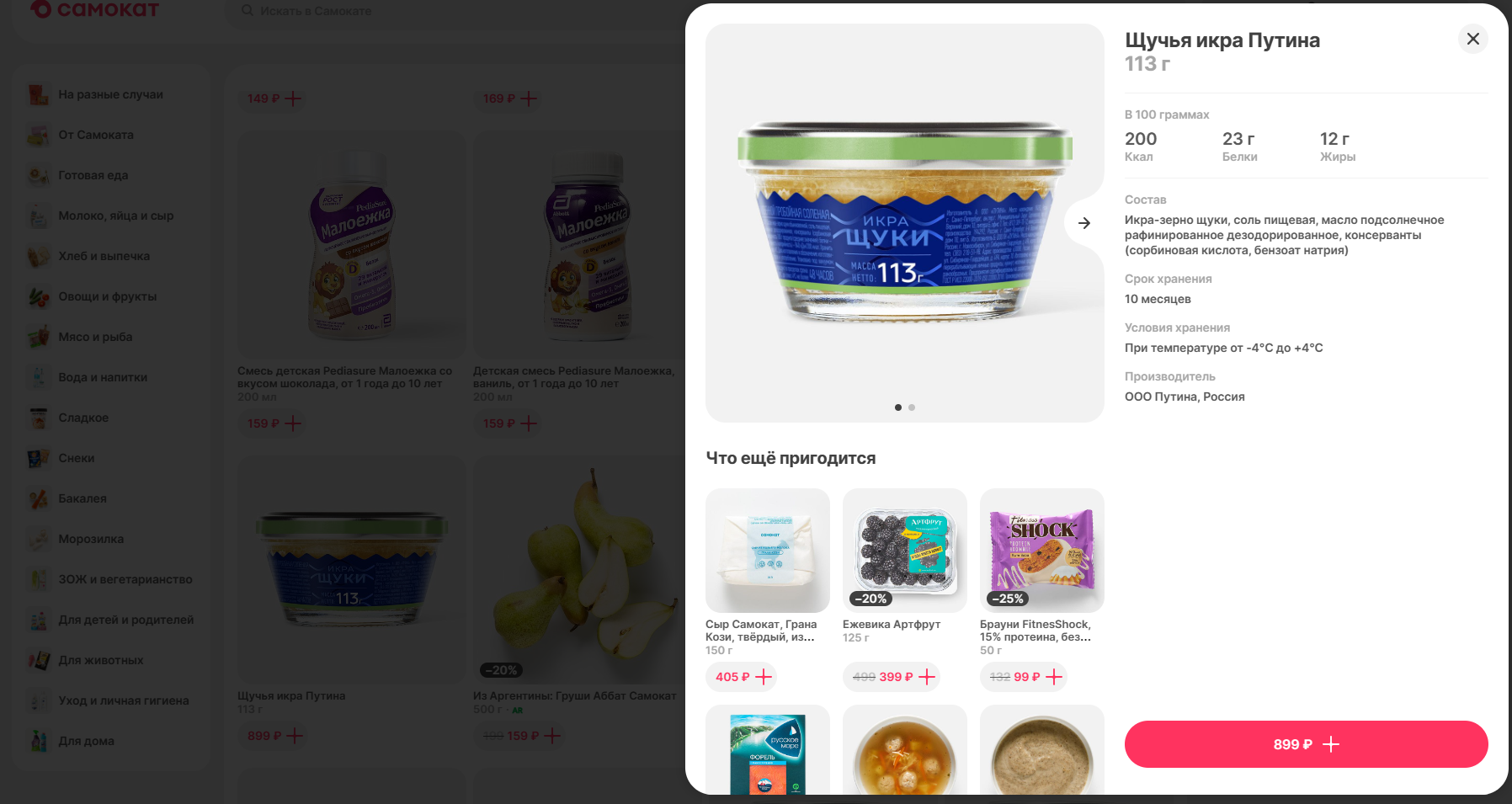


Рис. 1 Приложение «Самокат»

Сервис “Самокат” также обладает интуитивно понятным пользовательским интерфейсом, который делает использование “Самоката” удобным даже для непрофессионалов. Он предоставляет информацию в понятной форме и облегчает взаимодействие с системой заказа и доставки продуктов

Рис. 2 Пользовательский интерфейс

**Gmail предоставляет подробные отчеты и данные** (Рис. 3) о состоянии почтового ящика, обнаруженных проблемах и выполненных действиях. Это помогает пользователям получать объективную информацию о состоянии и управлении своей электронной почтой, обеспечивая прозрачность в процессе обработки сообщений.

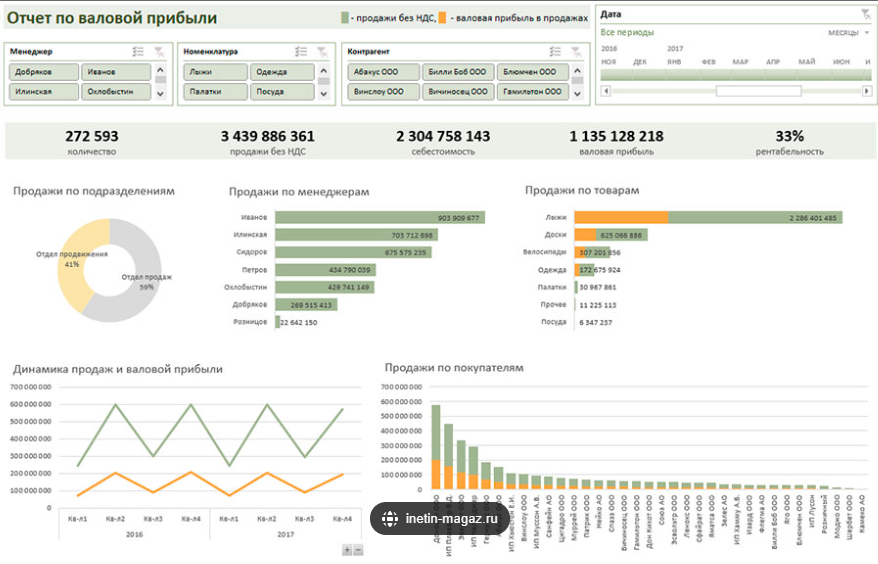


Рис. 3 Пример отчета и данных

Большим недостатком сервиса Самокат также является необходимость подключения к интернету для доступа к сервису и оформления заказов. Пользователи должны иметь доступ к интернету и убедиться в наличии соответствующего устройства и настроенного интернет-соединения, что может потребовать дополнительных усилий и ресурсов

# **Описание выявленных достоинств и недостатков**

**Описание достоинств** программы Gmail:

1. Бесплатность и масштаб: “Самокат” предоставляет свои услуги абсолютно бесплатно для пользователей и обладает огромным масштабом, позволяя делать заказы с уникальным адресом доставки
2. Большой ассортимент товаров: Пользователям предоставляется значительное количество товаров для выбора, что позволяет сохранять большие объемы информации без необходимости регулярно посещать магазины
3. Интеграция с другими сервисами: “Самокат” легко интегрируется с другими сервисами, такими как платежные системы, что обеспечивает удобство в работе и позволяет эффективно управлять всеми аспектами своих покупок через одну платформу

**Описание недостатков** программы Gmail:

1. Ограничения конфиденциальности данных: “Самокат” может использовать информацию о заказах пользователей для персонализации рекламы и улучшения своих услуг, что может вызывать опасения по поводу конфиденциальности данных пользователей
2. Ограниченные возможности работы офлайн: Несмотря на то, что “Самокат” предоставляет некоторые функции работы в автономном режиме через мобильное приложение, эти возможности ограничены и могут быть недостаточными для пользователей, которым требуется полноценная работа без подключения к интернету.
3. Ограничения на размер заказа: “Самокат” имеет ограничения на размер заказа, что может быть проблематично при заказе больших объемов товаров, требующих высокой пропускной способности и времени доставки

# **Используемый язык программирования C#**

В процессе выполнения курсовой работы использован язык программирования C# , а также интерфейс программирования приложений Windows Forms

Язык C# - это высокоуровневый язык программирования, разработанный Microsoft. Он был представлен в 2000 году и является одним из основных языков программирования в экосистеме Microsoft.NET. C# сочетает в себе элементы объектно-ориентированного программирования (ООП) и функционального программирования, и предоставляет мощные инструменты для разработки широкого спектра приложений.

C# используется во множестве областей, включая разработку настольных приложений, веб-разработку, создание мобильных приложений, игр и многое другое. Он широко применяется для разработки приложений под операционные системы Windows, macOS и Linux.

Программа на C# может содержать различные компоненты, такие как классы, структуры данных, методы, свойства и события. Существует возможность использовать библиотеки и фреймворки, предоставленные Microsoft или сторонними разработчиками, чтобы упростить разработку и добавить дополнительные возможности к вашей программе.

C# также имеет средства для обработки исключений, взаимодействия с базами данных, работы с файлами и многими другими задачами. Язык поддерживает сборку мусора для автоматического освобождения памяти и обладает мощным инструментарием для отладки кода и создания тестов.

Одним из основных преимуществ использования C# является его интеграция с .NET Framework или .NET Core, что позволяет использовать множество готовых компонентов и библиотек для создания эффективных и масштабируемых приложений.

В целом, C# является мощным и гибким языком программирования, который позволяет создавать разнообразные программы с использованием различных платформ и технологий.

ASP.NET — это фреймворк для разработки веб-приложений, разработанный компанией Microsoft, который предоставляет инструменты для создания динамических и интерактивных веб-сайтов и веб-приложений. В рамках ASP.NET, C# используется для создания веб-приложений с помощью технологии Web Forms.

С помощью C# в ASP.NET разработчики могут создавать разнообразные веб-элементы управления, такие как кнопки, текстовые поля, списки, таблицы и другие. Существует возможность размещать их на веб-страницах и настраивать их свойства, чтобы создать интерактивные пользовательские интерфейсы.

ASP.NET позволяет разработчикам создавать веб-приложения, которые могут взаимодействовать с базами данных, обрабатывать формы, обеспечивать аутентификацию и авторизацию пользователей, а также создавать динамический контент, который обновляется без перезагрузки страницы.

ASP.NET предоставляет возможности для создания веб-приложений, включая разработку веб API (Application Programming Interface), которые позволяют взаимодействовать с приложением через интернет посредством HTTP запросов.

Веб API в ASP.NET позволяют создавать точки доступа (эндпоинты), к которым можно обращаться из любого клиентского приложения, будь то веб-приложение, мобильное приложение или даже другой сервер.

Основные концепции и возможности веб API в ASP.NET:

1. ASP.NET Web API Framework: Это фреймворк, входящий в состав ASP.NET, который облегчает создание и развертывание веб API. Он предоставляет множество инструментов для создания API с поддержкой различных форматов обмена данными, таких как JSON и XML.
2. Маршрутизация: Существует возможность определить маршруты URL, которые указывают на методы вашего API. Это делается с помощью атрибутов маршрутизации или настройками конфигурации.
3. Контроллеры: Контроллеры в ASP.NET Web API представляют собой классы, которые обрабатывают входящие HTTP запросы и возвращают соответствующие HTTP ответы. В контроллерах определяются методы, которые будут вызываться при обращении к определенным URL.
4. Модели: Существует возможность использовать модели данных для передачи информации между клиентом и сервером. Обычно это классы C# (или других поддерживаемых языков), которые отображают структуру данных, которые передаются через API.
5. Форматы обмена данными: ASP.NET Web API поддерживает различные форматы обмена данными, такие как JSON, XML и другие. Существует возможность настроить форматы сериализации и десериализации данных по выбору.
6. Атрибуты: ASP.NET Web API использует атрибуты для настройки маршрутизации, аутентификации, авторизации и других аспектов API.
7. Аутентификация и авторизация: Существует возможность реализовать механизмы аутентификации и авторизации в вашем веб API, чтобы обеспечить безопасность и контроль доступа к ресурсам.
8. Фильтры: ASP.NET Web API позволяет использовать фильтры для добавления дополнительной логики в обработку запросов и ответов, например, логирование, кэширование или обработка ошибок.
9. Документация: Хорошая документация важна для веб API. Существует возможность использовать инструменты, такие как Swagger (Рис. 4) или OpenAPI, для автоматической генерации документации на основе кода.

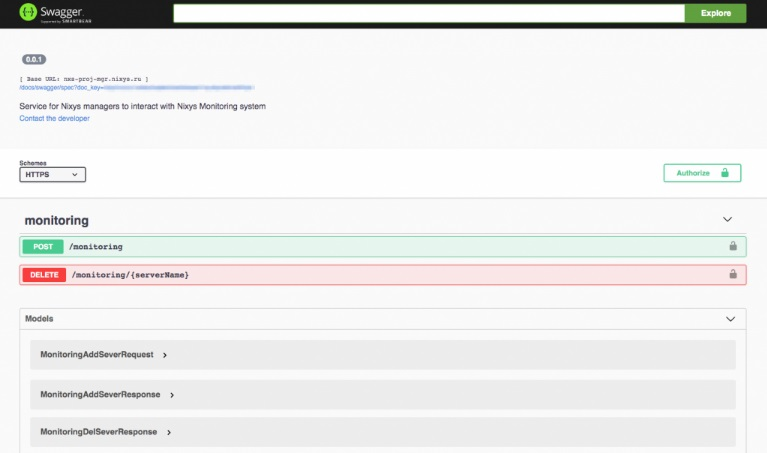


Рис. 4 Интерфейс Swagger

PostgreSQL - это мощная и широко используемая система управления реляционными базами данных (RDBMS), с открытым исходным кодом. Она разрабатывается сообществом и предоставляет богатый набор функций для хранения, управления и обработки данных. Вот основные возможности PostgreSQL:

1. **Хранение и управление данными**: PostgreSQL обеспечивает надежное хранение и эффективное управление данными различных типов. Он поддерживает широкий спектр типов данных, включая числа, строки, даты, времена, JSON, XML и многие другие.
2. **Язык SQL**: PostgreSQL полностью совместим с языком SQL (Structured Query Language) и предоставляет богатые возможности для выполнения запросов данных, создания и изменения таблиц, а также для управления доступом к данным.
3. **ранзакционная поддержка**: PostgreSQL обеспечивает полную поддержку транзакций, что позволяет группировать операции базы данных в логические единицы работы. Это обеспечивает целостность данных и защиту от потери данных в случае сбоев.
4. **Масштабируемость и производительность**: PostgreSQL предоставляет множество возможностей для масштабирования баз данных и обработки больших объемов данных. Он также предоставляет инструменты для оптимизации производительности запросов и баз данных.
5. **Безопасность**: PostgreSQL обеспечивает многоуровневую систему безопасности, которая позволяет управлять доступом к данным на уровне пользователя, базы данных и объектов базы данных. Он поддерживает аутентификацию и авторизацию пользователей, а также шифрование данных для обеспечения конфиденциальности.
6. **Резервное копирование и восстановление данных**: PostgreSQL предоставляет инструменты для создания резервных копий баз данных и их восстановления в случае сбоев или потери данных. Это обеспечивает защиту данных и возможность восстановления работы в случае чрезвычайных ситуаций.
7. **Автоматизация и планирование задач**: PostgreSQL предоставляет возможности для автоматизации и планирования регулярных задач, таких как резервное копирование, оптимизация баз данных и выполнение регулярных обслуживающих операций.
8. **Аналитика и отчетность**: PostgreSQL включает в себя инструменты для анализа данных и создания отчетов, такие как PostgreSQL Analytics, что позволяет анализировать данные и создавать отчеты для принятия бизнес-решений.
9. **Интеграция с другими приложениями**: PostgreSQL обеспечивает возможности интеграции с другими приложениями и технологиями через различные протоколы и интерфейсы, что позволяет взаимодействовать с другими системами и обеспечивать целостность данных

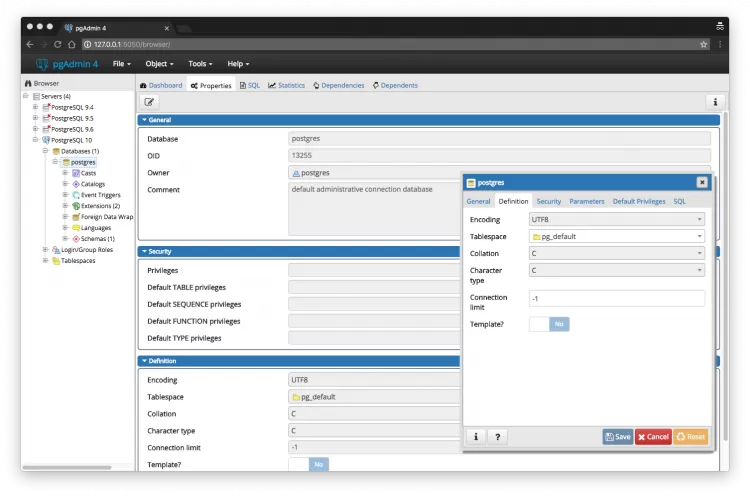


Рис. 5 Интерфейс PostgreSql

# **Общая постановка задачи**

Целью данной разработка эффективной системы для управления продуктовым магазином, которая обеспечит эффективное взаимодействие с клиентами, упростит процессы управления и повысит уровень обслуживания.. В рамках работы необходимо решить следующие задачи:

1. Определение функциональности системы: На основе анализа потребностей определить основные функции, которые должны быть реализованы в системе продуктового магазина. В программе должны быть реализованы следующие функции:
   1. **Управление пользовательской базой данных:** Возможность хранения информации о пользователях, их контактных данных, истории заказов.
   2. **Учет и управление данными продуктов:** Система должна позволять вести учет продуктов и контролировать и тд.
   3. **Добавление данных**: Возможность добавления администратором данных о продуктах, поставщиках и категориях.
   4. **Обновление базы данных**: Возможность обновлять базу данных.
   5. **Работа с базами данных PostgreSql**: Программа должна иметь возможность подключения из базы данных PostgreSql.
   6. **Регистрация и авторизация:** Система регистрировать новых и авторизировать уже существующих пользователей.
   7. **Отсутствие привязанности к конкретной СУБД:** Возможность изменять СУБД в ходе поддержки и обновления программы.
   8. **Изменение базы данных**: Возможность изменять значения в базе данных
   9. **Управление данными**: Возможность хранить и изменять информацию в базе данных
   10. **Создание новых полей в базах данных**: Программа должна иметь возможность создавать новые поля для работы с данными
2. **Разработка и реализация системы**: На основе функциональных требований разработать систему электронной почты и реализовать необходимые модули и функции. Обеспечить интеграцию с базами данных SQL Server.
3. Диаграмма прецедентов: представляет собой визуальное изображение взаимодействий между различными участниками системы и её функциональными возможностями (Рис. 6). В данном случае присутствуют два основных актера: «Пользователь» и «Продавец.

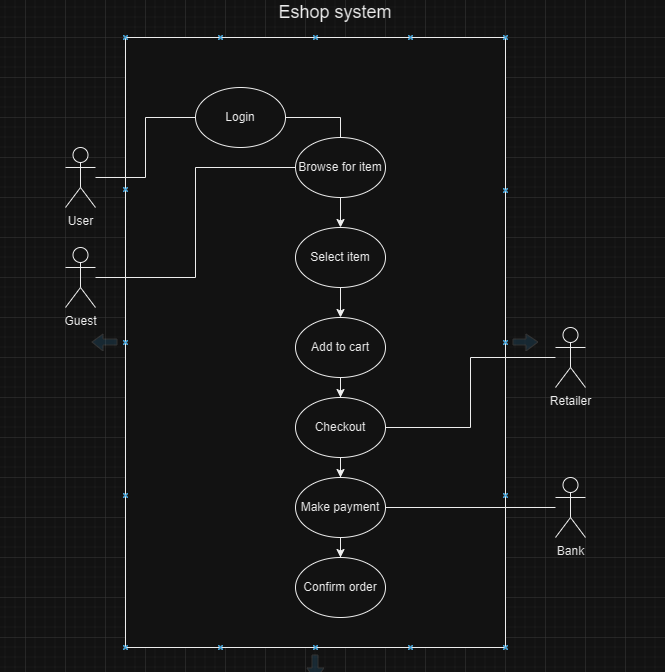


Рис. 6 Диаграмма прецендентов

Пользователь, как основной пользователь системы, имеет ряд основных действий, которые он может выполнять в системе:

Регистрация: Этот прецедент представляет собой процесс регистрации новой учётной записи в системе. Пользователь заполняет необходимые поля, такие как имя, адрес электронной почты и пароль, и отправляет данные для создания учётной записи.

Вход в систему: После регистрации или при последующих посещениях системы пользователь может войти в свою учётную запись. Этот прецедент включает в себя процесс аутентификации пользователя, где он предоставляет свои учётные данные (например, адрес электронной почты и пароль) для входа в систему.

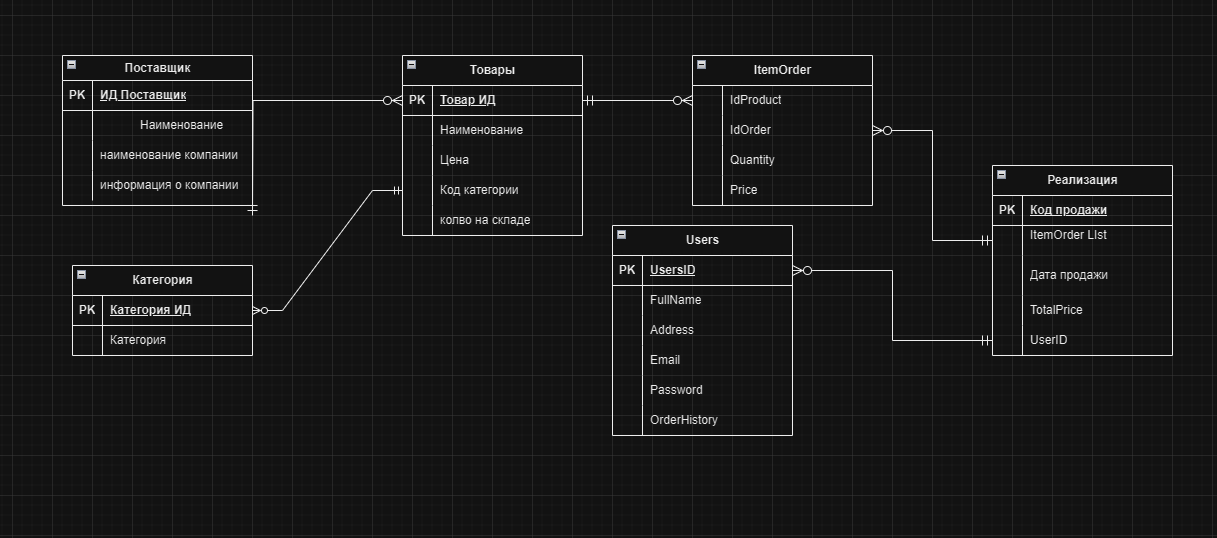
Просмотр продуктов: Пользователь может просматривать доступные продукты в магазине, просматривать их описания, цены и другие характеристики.

Управление данными: Этот прецедент описывает возможность пользователя изменять свои настройки профиля, администратора изменять данные продуктов, категорий и поставщиков

Создание заказа: Один из основных функциональных элементов системы - создание заказа. Пользователь может выбирать продукты из доступного каталога, добавлять их в корзину, указывать адрес доставки и оплаты, а также просматривать историю своих заказов.

Просмотр заказов: Пользователь может просматривать свои предыдущие заказы, проверять их статусы и детали доставки

1. Диаграмма базы данных: для более наглядного представления структуры системы электронной почты, разработана диаграмма базы данных (Рис. 7), которая показывает основные сущности и их взаимосвязи.

Рис. 7 Диаграмма базы данных

В представленной диаграмме базы данных присутствуют четыре таблицы: «User», «Order, «ItemOrder», «Supplier», «Category», «Product» . Эти таблицы содержат информацию о пользователях, продуктах, заказах, поставщиках, категориях а также устанавливают связи между ними с помощью первичных и внешних ключей.

Таблица «User»: содержит информацию о пользователях, включая уникальный идентификатор пользователя (UserId),, адрес электронной почты (Email), полное имя (FullName) и пароль (Password).

Таблица «Category»: содержит информацию о категориях продуктов. У каждой категории есть уникальный идентификатор (Id), а также связь с конкретным продуктом через внешний ключ (CategoryId). Дополнительные поля включают наименование категории (CategoryName).

Таблица «Supplier»: отражает информацию о поставщиках. У каждого поставщика есть свой уникальный идентификатор (Id), а также связь с продуктом через внешний ключ (SupplierId). Поля поставщиков включают наименование компании поставщика (CompanyName), информация о поcтващике (CompanyInfo)

Таблица «ItemOrder»: хранится информация о списке продуктов в заказе. Имеет свой уникальный идентификатор (Id), а также связь с соответствующим заказом через внешний ключ (ItemOrderID). Дополнительные поля включают цену за каждый продукт (price), количесвто этих продктов(Quantity)

Таблица «Order»: хранится информация о заказе. Имеет свой уникальный идентификатор (Id), а также связь с соответствующим itemOrder через внешний ключ (OrderID). Дополнительные поля включают цену за заказ (TotalPrice), дата и время заказа (OrderDate)

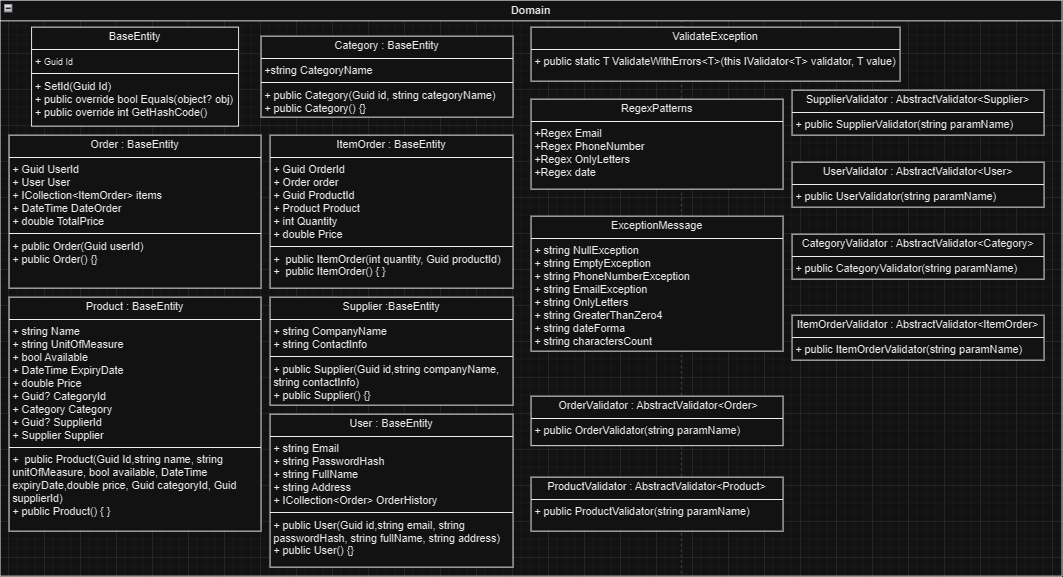
Таблица «Product»: хранится информация о продукте. Имеет свой уникальный идентификатор (Id), а также связь с соответствующим itemOrder через внешний ключ (ProductID). Дополнительные поля включают цену за товар (Price), единицы измерения кол-ва товара (UnitOfMeasure), Наименование товара (Name), доступность товара (Availible), срок годности товара (ExpireDate)

Связи между таблицами указывают на отношения между их записями. Например, связь от UserId в таблице User к UserId в таблицах Order показывает что каждый заказ связан с определенным пользователем. Связь от MessageId в таблице ыessage к MessageId в таблице Attachment показывает, что каждое вложение связано с определенным сообщением.

Таким образом, эта база данных структурирована чтобы поддерживать функциональность приложения, в котором возможен учет продуктов и оформление заказов.

1. Диаграмма классов доменного слоя (Рис.8) проекта включает в себя основные сущности и бизнес-логику, которые определяют основные объекты и их взаимодействие в системе. В этом слое сосредоточены основные классы, такие как Category, Order, ItemOrder, Product, Supplier и User, которые наследуют общий базовый класс BaseEntity. Этот базовый класс обеспечивает уникальную идентификацию для всех сущностей через идентификатор.

Кроме основных сущностей, доменный слой включает классы для валидации данных (ValidateException, RegexPatterns, ExceptionMessage) и валидаторы для каждой сущности (например, CategoryValidator, ProductValidator и т.д.). Валидация обеспечивает корректность данных и проверку их соответствия бизнес-правилам. (Рис.8)

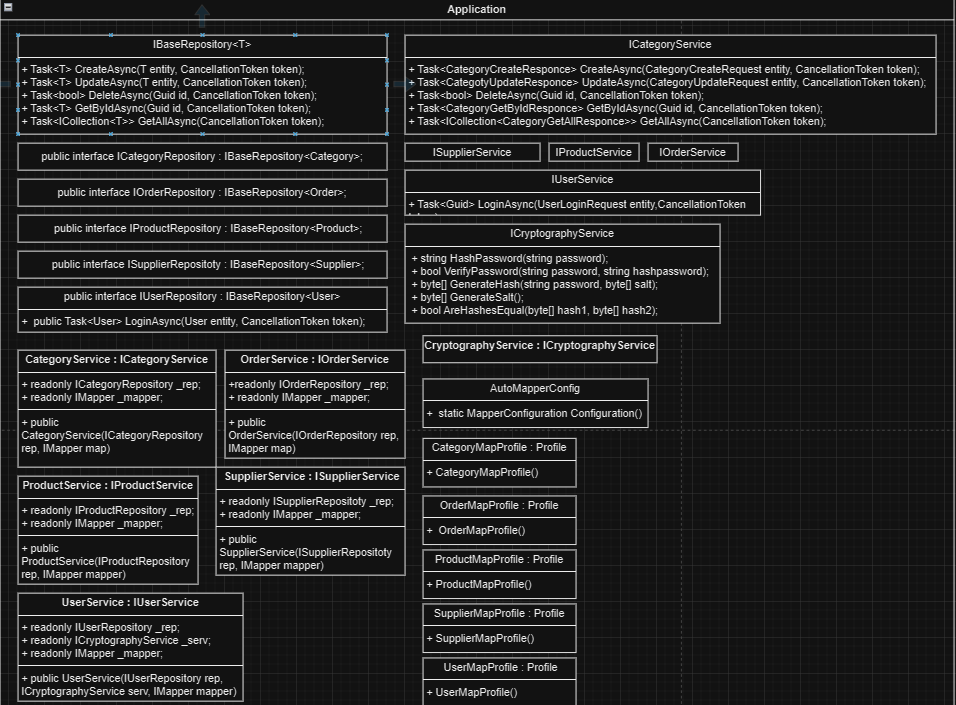
 Рис. 8 Диаграмма классов Доменного слоя

Таким образом, доменный слой формирует основу приложения, определяя ключевые объекты, их свойства и правила взаимодействия, что позволяет реализовывать бизнес-логику и обеспечивать целостность данных в системе.

1. Диаграмма классов слоя приложения (Рис. 9) включает основные компоненты, которые обеспечивают бизнес-логику и взаимодействие с данными. Интерфейсы репозиториев определяют стандартные операции создания, чтения, обновления и удаления (CRUD) для различных типов сущностей в хранилище данных. Каждый репозиторий, такой как категории, заказы, продукты, поставщики и пользователи, имеет свой собственный интерфейс, который определяет специфичные методы для работы с соответствующими данными.

Сервисы, такие как CategoryService, OrderService и т. д., предоставляют абстракцию для выполнения операций над соответствующими сущностями. Они инкапсулируют логику приложения, необходимую для обработки запросов и взаимодействия с соответствующими репозиториями. Например, ProductService обеспечивает методы для управления продуктами, включая их создание, обновление, удаление и получение, ICryptographyService - сервис безопасности, обрабатывающий пароли: хеширует, проверяет, генерирует соль и сравнивает хеши.

AutoMapperConfig отвечает за конфигурацию AutoMapper, инструмента, используемого для преобразования данных между сущностями и объектами передачи данных (DTO). Это позволяет управлять сопоставлением между различными моделями данных в приложении, упрощая передачу данных между слоями и уменьшая сложность кода.

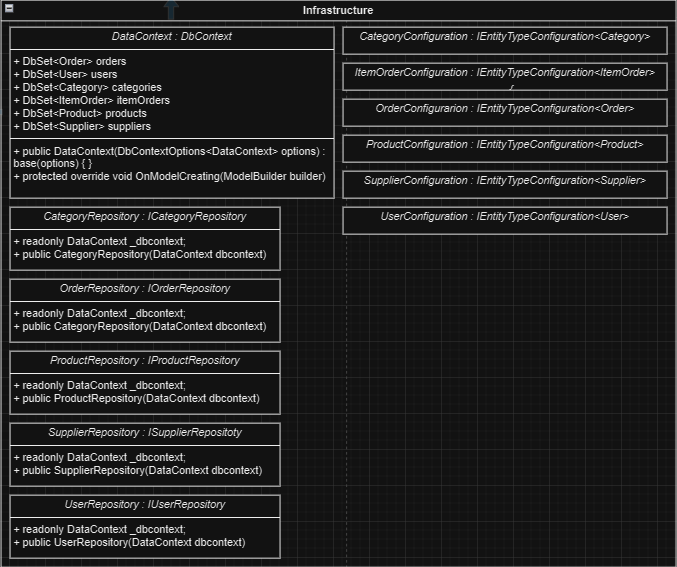
Рис. 9 Диаграмма классов слоя приложения

Таким образом на уровне приложения обеспечивается бизнес-логика и взаимодействие с данными. Репозитории и сервисы обрабатывают данные и запросы. ICryptographyService обеспечивает безопасность паролей, а AutoMapperConfig управляет сопоставлением данных.

1. Диаграмма классов слоя инфраструктуры. Инфраструктурный слой отвечает за взаимодействие с базой данных и хранение данных. Он включает в себя класс контекста базы данных, который наследуется от DbContext и содержит DbSet для каждой сущности, таких как заказы, пользователи, категории, продукты, поставщики и другие. Контекст базы данных также содержит метод OnModelCreating для конфигурации модели с помощью ModelBuilder.

Для работы с данными в инфраструктурном слое также используются репозитории. Каждый репозиторий реализует интерфейс и имеет ссылку на DataContext для доступа к базе данных. Репозитории включают репозитории для работы с категориями, заказами, продуктами, поставщиками и пользователями.

В инфраструктурном слое также определяются конфигурации для каждой сущности, реализующие интерфейс IEntityTypeConfiguration. Например, существуют конфигурации для категорий и элементов заказа, которые определяют правила и ограничения для сущностей, такие как связи и индексы.

Рис. 10 Диаграмма классов слоя инфраструктуры

Таким образом инфраструктурный слой обеспечивает надежное хранение и доступ к данным приложения через класс контекста базы данных, репозитории и конфигурации сущностей. Это ключевой компонент для создания эффективных и масштабируемых приложений.

1. Реализация чистой архитектуры (Рис.8): Реализовать чистую архитектуру в приложении с учетом ключевых требований:
   1. Разделение ответственностей : Каждый компонент должен иметь четко определенную ответственность и заниматься только одним аспектом системы. Например, бизнес-логика должна быть отделена от взаимодействия с базой данных или пользовательским интерфейсом.
   2. Зависимости должны быть обратимыми : Компоненты верхнего уровня не должны зависеть от компонентов нижнего уровня. Зависимости должны быть направлены от абстракций к деталям, что позволяет легко заменять или модифицировать конкретные реализации.
   3. Однозначность интерфейсов : Интерфейсы между компонентами должны быть явными и четко определенными. Это помогает понять, какие методы и данные доступны для использования, и избежать неявных зависимостей.
   4. Тестируемость : Код должен быть легко тестируемым без необходимости запуска всей системы. Каждый компонент должен иметь набор тестов, которые могут быть запущены автоматически.
   5. Легкость замены компонентов : Компоненты должны быть спроектированы таким образом, чтобы их можно было легко заменить новой реализацией без изменения остальной части системы. Например, замена базы данных должна быть минимально инвазивной операцией.
   6. Легкость понимания и поддержки: Структура приложения должна быть понятной для разработчиков, которые присоединяются к проекту. Код должен быть хорошо документирован, и его структура должна соответствовать бизнес-логике системы.
   7. Соблюдение принципов SOLID: Каждый компонент должен следовать принципам SOLID, таким как Single Responsibility Principle и Open/Closed Principle.

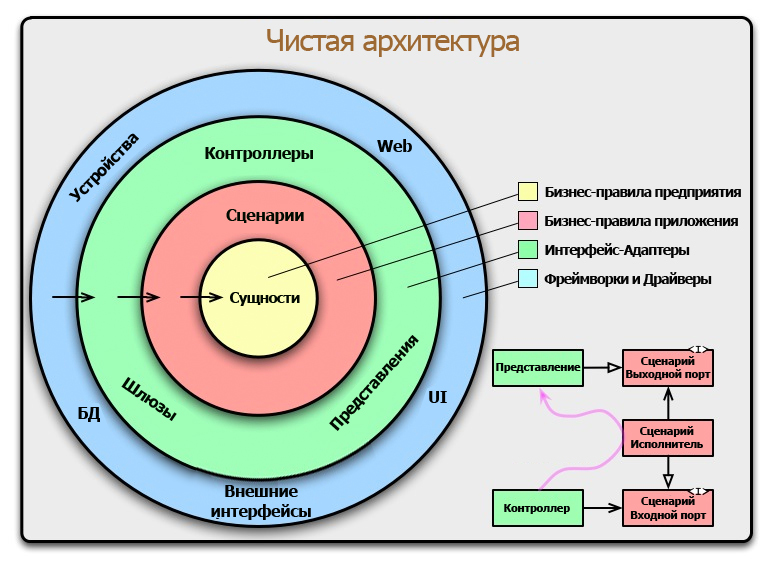


Рис. 7 Структура чистой архитектуры

**Итогом работы** должна быть разработанная, реализованная и протестированная система продуктового магазина, способная эффективно управлять данными и реализовывать систему заказов.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы разработана информационная система для продуктового магазина создана с целью оптимизации ключевых процессов работы. Программа разработана на языке программирования C# с учетом требований к системам электронной коммерции. В ней предусмотрены все необходимые разделы и функции для эффективной работы магазина.

Система обладает интуитивно понятным интерфейсом, включающим все необходимые модули, такие как управление продуктами, учет заказов, добавление новых товаров, управление пользователями и администрирование магазина.

Внедрение разработанной программы значительно упростит управление продуктовым ассортиментом, управление заказами и обслуживание клиентов. Это позволит повысить эффективность работы магазина, сократить количество ошибок и увеличить удовлетворенность клиентов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и учебно-методическая литература

1) Борисенко, В.В. (2016). Основы программирования. Интернет-Университет Информационных Технологий.

2) Туральчук, К.А. (2019). Параллельное программирование с помощью языка C#. Интернет-Университет Информационных Технологий.

3) Золотарёв, О.В. (2013). Технология внедрения корпоративных информационных систем: методические указания к лабораторным работам. Российский новый университет.

Интернет-ресурсы

4) Metanit С#: (<https://metanit.com/sharp>)

5)Соммервиль, И. (2018). Объектно-ориентированное мышление. Pearson Education.

6)Шилдт, Г. (2018). C# 8.0. Полное руководство. Вильямс.

7)Макконнелл, С. (2015). Совершенный код. Практическое руководство по разработке программного обеспечения. Вильямс.

8)Мартин, Р. (2019). Чистый код: создание, анализ и рефакторинг. Питер.

9)Freeman, A., Robson, D. (2014). Head First Design Patterns. O'Reilly Media.

ПРИЛОЖЕНИЕ

|  |  |
| --- | --- |
| public class Category : BaseEntity  {public string CategoryName { get; set; }  public Category(Guid id, string categoryName)  {SetId(id);  CategoryName =categoryName;  new CategoryValidator(nameof(Category)).ValidateWithErrors(this);}  public Category() {}  }  public class ItemOrder  {public Guid OrderId { get; set; }  public virtual Order order { get; set; }  public Guid ProductId { get; set; }  public virtual Product Product { get; set; }  public int Quantity { get; set; }  public double Price { get; set; }  public ItemOrder() { }  public ItemOrder(int quantity, Guid productId)  {Quantity = quantity;  ProductId = productId;  new ItemOrderValidator(nameof(ItemOrder)).ValidateWithErrors(this); }}  public class Order : BaseEntity  {public Guid UserId { get; set; }  public virtual User User { get; set; }  public virtual ICollection<ItemOrder> items { get; set; }  public DateTime DateOrder { get; set; }  public double TotalPrice { get; set; }  public Order() { }  public Order(Guid userId)  { UserId = userId;new OrderValidator(nameof(Order)).ValidateWithErrors(this);}}  public class Product : BaseEntity  {public string Name { get; set; }  public string UnitOfMeasure { get; set; }  public bool Available { get; set; }  public DateTime ExpiryDate { get; set; }  public double Price { get; set; }  public Guid? CategoryId { get; set; }  public virtual Category Category { get; set; }  public Guid? SupplierId { get; set; } | public virtual Supplier Supplier { get; set; }  public Product() { }  public Product(Guid Id,string name, string unitOfMeasure, bool available, DateTime expiryDate,double price, Guid categoryId, Guid supplierId)  {SetId(Id); Name = name;  UnitOfMeasure = unitOfMeasure;  Available = available;  ExpiryDate = expiryDate;  Price = price;  CategoryId = categoryId;  SupplierId = supplierId;new ProductValidator(nameof(Product)).ValidateWithErrors(this);}}  public class Supplier : BaseEntity  {public string CompanyName { get; set; }  public string ContactInfo { get; set; }  public Supplier(Guid id,string companyName, string contactInfo)  { SetId(id);CompanyName = companyName;  ContactInfo =contactInfo;  new SupplierValidator(nameof(Supplier)).ValidateWithErrors(this);}public Supplier(){}}  public class User : BaseEntity  { public string Email { get; set; }  public string PasswordHash { get; set; }  public string FullName { get; set; }  public string Address { get; set; }  public virtual ICollection<Order> OrderHistory { get; set; }  public User() { }  public User(Guid id,string email, string passwordHash, string fullName, string address)  {SetId(id);  Email = email;  PasswordHash = passwordHash;  FullName = fullName;  Address = address;  new UserValidator(nameof(User)).ValidateWithErrors(this); }} |