*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Разработка программной системы для*

*автоматизации работы магазина компьютерной техники*

*Выполнил: студент группы ПРИ-120 Парахин К.В.*

*Принял: доц. кафедры ИСПИ*

*Верешинин В.В.*

*Владимир, 2022*

**Аннотация**

В данном курсовом проекте производилась разработка программной системы для автоматизации работы и функционирования компонентов магазинов с товарами компьютерной техники.

Проект состоит из 3 основных этапов проектирования, включающих первый этап под названием «Описание предметной области», содержащий в себе такие блоки, как: словарь предметной области, описание пользователей разрабатываемой подсистемы, описание прецедентов пользователей системы, сущностей системы, второй этап под названием «Динамическое моделирование системы», включающий в себя такие блоки, как описание состояний объектов системы, описание последовательностей действий и активностей в системе, а также третий этап, состоящий из реализации спроектированной системы в виде электронного сайта с набором функционала, реализованном согласно возможностям платформы .NET.

Реализованная система может применяться для таких целей, как, например, хранение данных о активностях пользователей, хранение списка магазинов, складов и товаров на них, а также обеспечение базового функционала по ведению торговой деятельности магазина.

Реализованная система спланирована для использования различными видами пользователей, имеющих различный функционал согласно их ролям и возможностям в системе.

Курсовой проект представлен на 58 страницах, основных рисунков - 14, использованных источников – 7, приложений – 2, иллюстрационный материал на 2 листах формата А1.

**ABSTRACT**

In this course project, a software system was developed to automate the operation and functioning of components of stores with computer equipment products.

The project consists of 3 main stages of design, including the first stage called "Description of the subject area", containing such blocks as: a dictionary of the subject area, a description of users of the subsystem being developed, a description of the use cases of users of the system, entities of the system, the second stage called "Dynamic modeling of the system", including such blocks, as a description of the states of the system objects, a description of the sequences of actions and activities in the system, as well as the third stage, consisting of the implementation of the designed system in the form of an electronic website with a set of functionality implemented according to the capabilities of the .NET platform.

The implemented system can be used for such purposes as, for example, storing data on user activities, storing a list of stores, warehouses and goods on them, as well as providing basic functionality for conducting the store's trading activities.

The implemented system is planned for use by different types of users with different functionality according to their roles and capabilities in the system.

The course project is presented on 57 pages, the main drawings - 14, the sources used – 7, appendices – 2, illustrative material on 2 sheets of A1 format.

**Содержание**

[2. Описание предметной области 4](#_Toc123386394)

[2.1. Пользователи разрабатываемой подсистемы 4](#_Toc123386395)

[2.2. Словарь предметной области: 5](#_Toc123386396)

[2.3. Разработка UML-диаграмм 7](#_Toc123386397)

[2.4. Диаграмма прецедентов системы: 11](#_Toc123386398)

[2.5. Диаграмма классов системы: 13](#_Toc123386399)

[3.1. Диаграмма состояний в системе: 15](#_Toc123386400)

[3.2. Диаграмма последовательностей в системе: 18](#_Toc123386401)

[3.3. Диаграмма видов деятельности в системе: 20](#_Toc123386402)

[4.1. Общее описание реализации системы: 22](#_Toc123386403)

[4.2. Разработка backend – части системы: 23](#_Toc123386404)

[4.3. Разработка frontend – части системы: 30](#_Toc123386405)

[Заключение 38](#_Toc123386406)

[Список использованных источников 39](#_Toc123386407)

[Приложение A. Исходный код программной системы.. 40](#_Toc123386408)

[Приложение Б. Структура таблиц БД 53](#_Toc123386409)

1. **Введение**

**Цель работы**:

Разработать автоматизированную систему работы электронного компьютерного магазина, которая включает в себя подсистему хранения данных, которая будет позволять сохранять и фиксировать изменения данных о заказах, отгрузках и доставках заказов устройств компьютерной техники, о регистрации и действиях пользователей, имеющих различные привилегии в системе.

Для достижения поставленной цели необходимо решить некоторый набор задач.

**Задачи решаемые в системе:**

* хранение данных о производителях компьютерной техники
* хранение списка категорий (типов товаров)
* хранение списка товаров, имеющихся в наличии на складах
* хранение данных о пользователях, авторизованных или зарегистрированных в системе
* хранение данных о системных пользователях (работников системы), их привилегий и функций
* хранение корзины пользователей, их отзывов и оценок
* функции сбора итоговой информации о продажах компьютерной техники в и течение определенного промежутка времени (анализ маркетинга)

1. **Описание предметной области**

У каждого компьютерного устройства есть принадлежность к какому-либо типу устройств, свои технические характеристики, производитель, его цена и наличие на складах. Сеть официально получает разрешение на ведение торговой деятельности, покупает разнообразную компьютерную технику (например, ноутбуки, смартфоны, наушники, системные блоки, затем устанавливает свои цены на приобретенные товары для потенциальных покупателей в зависимости от различных факторов. Кроме того сеть может менять цену на устройства, делая скидки постоянным клиентам или в целях ускоренной продажи и продвижения определенных товаров (ведение маркетинга – чем занимается менеджер).

Сеть имеет свою собственную автоматизированную программную систему, представленную для пользователей системы в виде электронного сайта, на котором пользователи системы могут совершать определенные действия, которые заложены в их функциональных ролях в этой системе.

Например, пользователи электронного сайта имеют возможность просто авторизоваться в системе, зарегистрироваться в личном кабинете, стать постоянным пользователем услуг сети.

Также стоит сказать, что сеть открывает в городе несколько пунктов выдачи, где операторы (работники системы) могут выдавать заказы покупателям (заказ должен быть доставлен в пункт выдачи – что не является частью автоматизированной электронной системы).

Ниже перечислим основные типы пользователей данной системы.

## Пользователи разрабатываемой подсистемы

*Администратор системы –* сотрудник сети, который может редактировать содержание сайта, личного кабинета клиентов (то есть работает с блоками и сообщениями на сайте), а также осуществлять управление аккаунтами пользователей системы

*Менеджер системы –* сотрудник сети, который осуществляет бухгалтерский (экономический) учет работы сети, может регулировать поставки тех или иных товаров на склады, а также регулировать категории товаров на сайте (тем самым осуществляя маркетинг)

*Обычный пользователь системы (или покупатель) –* любой пользователь, осуществляющий доступ к системе, может просматривать электронный каталог, делать заказы и получать их, а также оставлять отзывы.

## Словарь предметной области:

*Пункт выдачи* – отдельный торговый офис сети, содержащий некоторый кадровый состав и каталог имеющихся в наличии телефонов (часто вместо этого термина используется устоявшийся вариант – магазин).

Атрибуты: адрес, контакты, кадровый состав (операторы)

*Корзина* – страница, на которую клиент отправляет товары, чтобы затем приобрести их.

Атрибуты: клиент, товар, кол-во товаров (того или иного вида), дата добавления, информация о карте.

*Заказ –* это специальный бизнес-процесс, в результате которого осуществляется поручение на доставку (и соответственно, оплату) товаров покупателю из электронного магазина.

Атрибуты: товары, покупатель, дата создания, стоимость заказа, дата доставки, адрес пункта выдачи

*Производитель* – компания, осуществляющая создание компьютерной техники, их продвижение и продажу в разных странах мира.

Атрибуты: название, описание, официальный сайт

*Тип устройства* (DeviceType)– один из основных шаблонов компьютерных устройств.

Атрибуты: название, описание

Артикул – некоторый идентификатор товара, по которому он определяется и находится в системной базе данных.

Атрибуты: товар, название, идентификатор

*Смартфон* – устройство, объединяющее в себе функции персонального органайзера и мобильного телефона

*Систе́мный блок* — это устройство, которое физически представляет собой корпус, наполненный аппаратным обеспечением для создания компьютера

*Ноутбук* — переносной компьютер, обладающий небольшими размерами и автономнойстью работы, в корпусе которого объединены типичные компоненты ПК.

*Модель устройства* - определенный вид (шаблон) устройств конкретного производителя, обладающий постоянным набором технических характеристик и стоимостью

Атрибуты: производитель, линейка, технические характеристики, базовая стоимость

*Каталог* – полный список предоставляемого ассортимента устройств, содержащий информацию о наличии различных компьютерных устройств (вообще в сети, и конкретно по наличии на складах).

Атрибуты: сеть, склаж, модель устройства, отметка о наличии

*Личный кабинет* – сервис на электронном сайте сети, в которой могут регистрироваться клиенты для удобного доступа к различным функциям системы с разных устройств и бразеров.

Атрибуты: электронный адрес, входные данные, пользователь системы, логин.

*Склад* – помещение, предназначенное для хранения товаров.

Атрибуты: адрес, товар, наличие товара(ов), их количество.

*Пользователь (клиент)* – человек, который пользуется функционалом, предоставляемыми автоматизированной электронной системой.

Атрибуты: фамилия, имя, отчество, контактный телефон, личный кабинет, скидки и бонусы

*Роль –* это набор полномочий (или привилегий), который необходим пользователю или группе пользователей для выполнения определённых рабочих задач

Атрибуты: пользователь, привилегии, функции (функционал).

*Привилегия* – это право некоторой учетной записи выполнять определенную функцию.

Атрибуты: учетная запись, пользователь, логин, роль, функции.

*Скидка* – вознаграждение сетью постоянных клиентов за частое пользование услугами торговой сети, проявляющееся в снижение стоимости некоторых товаров при пользовании картой.

Атрибуты: постоянный клиент, скидка, бонус.

* 1. **Разработка UML-диаграмм**

Начнем с разработки **диаграммы типа Use Case (диаграммы прецедентов)** в нотации UML (рис. 1). Для этого требуется выполнить анализ прецедентов работы с программной системой и промоделировать взаимоотношения автоматизируемой информационной системы и ее элементов с внешней средой.

**В ходе проектирования были выделены следующие прецеденты:**

- Управлять своим аккаунтом (пользователь системы)

- Авторизоваться

- Зарегистироваться

- Управлять своим аккаунтом

- Выбирать устройство в каталоге (покупатель)

- Использовать поиск товара на сайте

- Применять фильтрацию товаров по категориям

- Создать заказ (покупатель)

- Добавить товар в корзину

- Управлять содержимым корзины

- Оплатить заказ

- Оценить товар из заказа

- Оставить отзыв о заказе

- Управлять товарами (менеджер)

- Огранизовать поставки товаров со складов

- Продвинуть товар (добавить рекламу)

- Администрировать сайт (администратор)

- Добавить новый блок (сообщение) на сайт

- Осуществлять модерацию отзывов

**Опишем основные прецеденты пользователей в системе:**

**Прецедент «Управлять своим аккаунтом»**

Предусловие: пользователь системы авторизован в личном кабинете системы, имеет некоторый идентификатор в сети.

Действующее лицо: Пользователь системы

Основной поток: Управлять своим аккаунтом (авторизоваться – получить логин)

Клиент открывает в браузере электронный сайт магазина, заходит на страницу авторизации (в верхней правой части экрана присутствует ссылка)

Если пользователь не авторизован в системе, то система предложит ему при заходе на страницу авторизации авторизоваться.

Альтернативный поток: пользователь авторизован в системе. Тогда он может зарегистрироваться в личном кабинете (чтобы ему были доступны функции личного кабинета, а также накопление денег на балансе и возможности сохранения доставочной информации).

Если пользователь не собирается регистрироваться в системе – он может просто выйти с этой страницы и пользоваться простым функционалом системы.

**Прецеденты покупателей:**

**Прецедент «Выбирать устройство в каталоге»**

Предусловие: пользователь (клиент) авторизован в личном кабинете системы, имеет некоторый идентификатор в сети.

Действующее лицо: Покупатель

Основной поток: Выбирать устройство в каталоге

Клиент открывает в браузере электронный сайт (представление веб-приложения), на котором отображается артикул всех моделей, доступных для покупки.

Пользуется средствами поиска по каталогу и операциями фильтрации (может вводить название производителя, отмечать требуемые значения характеристик искомых устройств и т.д.)

Отмечает найденную интересуемую модель устройства и нажимает кнопку «Добавить в корзину» - прецедент «Добавить в корзину».

Система при этом сохраняет в базу данных выбранное устройство и при надобности перенаправляет клиента в «корзину», в которой находится список всех выбранных пользователем устройств на сайте.

Альтернативный поток: нужная модель отсутствует в каталоге (не в наличии)

На втором шаге клиент не находит требующееся устройство (например, какой-то смартфон). В этом случае он:

- либо начинает поиск устройства другой модели

- либо корректирует условия поиска или фильтрации исходного каталога

Постусловие: если требующееся устройство было все-таки найдено, то должно быть произведено создание заказа, в который будет включено это устройство (и остальные устройства из списка выбранных).

В случае, если заказ не укомплектован и не выполнен, то не гарантируется удержание данного устройства в наличии (то есть другие пользователи могут в это время заказать его сами и оно уже не будет в будущем в наличии).

**Прецедент «Создать заказ».**

Предусловие: пользователь (клиент) выбрал устройство (а), добавил их в корзину.

Действующее лицо: Покупатель

Основной поток: Создать заказ

Клиент открывает в браузере электронный сайт (представление веб-приложения), заходит на страницу под названием «Корзина», в ней выбирает те устройства, которые он в данный момент хочет купить, помечает их галочками. Затем нажимает кнопку «Составить заказ», переходит на форму оплаты заказа, осуществляет все необходимые действия и оставялет свою контакную информацию. Затем ожидает некоторое время доставку своего заказа и получает заказ: прецедент «Получить заказ».

Система при этом сохраняет в базу данных выбранный заказ, укомплектовывает его и отправляет его в выбранный пользователем пункт выдачи.

Альтернативный поток: устройства, которые пользователь хочет купить, теперь отсуствуют на сайте, и он не может их добавить в заказ.

На втором шаге клиент не находит требующееся устройство в статусе «В наличии» среди списка выбранных устройств в корзине:

- либо выбирает другие устройства и совершает заказ

- либо корректирует условия поиска и заново ищет нужное устройство («Выбирать устройство»)

Постусловие: если требующееся устройство было все-таки найдено, то должно быть произведено создание заказа, в который будет включено это устройство (и остальные устройства из списка выбранных).

Кроме того, в данном взаимодействии еще присуствуют такие прецеденты, как: «Искать устройство», «Передать заказ» (в пункт выдачи), «Получить заказ».

Кроме основного действующего лица «Клиент» в данной схеме присутствует еще «Оператор», который занимается передачей заказа клиенту.

**Прецеденты менеджера:**

**Прецедент «Управлять категориями товаров».**

Предусловие: На сайте присутствует некоторый базовый каталог товаров

Действующее лицо: Менеджер

Основной поток: Управлять категориями товаров.

Менеджер имеет возможность управлять каталогом электронного сайта, регулировать категории товаров, сформированные ранее, настраивать фильтры товаров, гибко подстраиваясь под основные интересы клиентов (например, присулшиваясь к их отзывам). Кроме этого, он управляет не только каталогом (то есть не только способами отобразить имеющиеся товары в наличии), но и может управлять отдельными товарами, выводить более востребованные товары и заниматься модерацией прочих, подобно менеджеру обычного магазина.

Альтернативный поток: менеджер не может управлять категориями, потому что такой блок не был добавлен на сайт администратором. Тогда он должен обратиться с помощью к администратору.

Ниже изображена UML – диаграмма прецедентов для основных пользователей системы интернет-магазина компьютерной техники (рис.1 – измененная по сравнению с прошлым этапом диаграмма) .

## Диаграмма прецедентов системы:

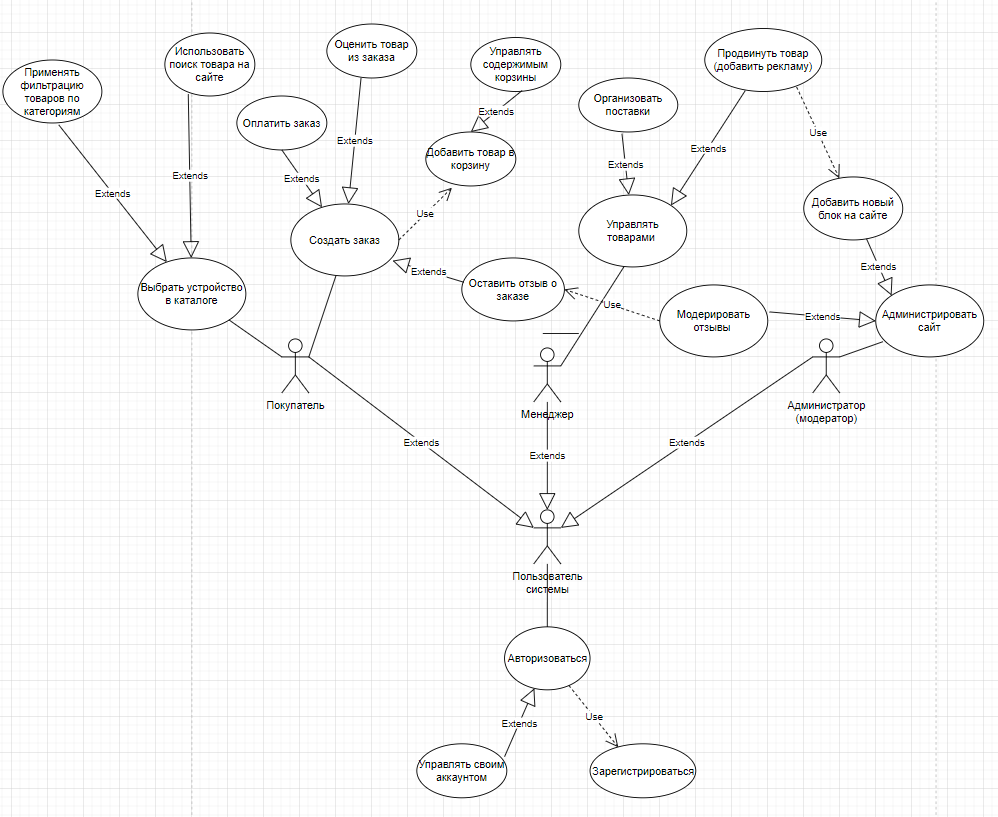


Рисунок 1. Диаграмма прецедентов

Далее разработаем **диаграмму классов (UML Class Diagram)** в нотации UML (на рис 2.)

Класс – это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой.

Выделим такие основные сущности в системе, как:

- **Producer** (производитель (и его требования))

- **DeviceType** (тип устройства: смартфон, ноутбук, системный блок, планшет, наушники и т.д.)

- **Product** (товар, являющийся некоторой моделью устройства)

- **SummUpProduct** (совокупность товаров одного артикула, которая может находиться в корзине)

- **Order** (заказ, его статус и информация о нем)

- **Basket** (корзина клиента с данными о товарах и личными клиентскими данными)

- **User** (абстрактный пользователь с его данными в системе)

- **Role** (описание привилегии, которой может быть наделен пользователь системы с набором опций. Включает в себя такие привилегии, как: покупатель (обычный пользователь), менеджер и администратор системы)

- **Option** (особая функция, которой может быть наделена некоторая привилегия. Считается также, что если какая-то более низкая привилегия обладает какой-то опцией, то все привилегии (роли) выше нее по уровню также обладают этой опцией).

- **Notification** (системное уведомление пользовтелю на сайте)

- **Rewiew** (оценки конкретного товара и отзывы о нем)

- **Warehouse** (склад для хранения товаров)

- **Shipping** (информация о доставке товаров)

Ниже изображена UML – диаграмма классов системы интернет – магаизна компьютерной техники, на которой используются такие основные виды отношений, как ассоциация, агрегация и композиция.

Примеры связей и отношений между сущностями (классами) можно посмотреть на рисунке 2 (измененная по сравнению с прошлым этапом диаграмма).

* 1. **Диаграмма классов системы:**

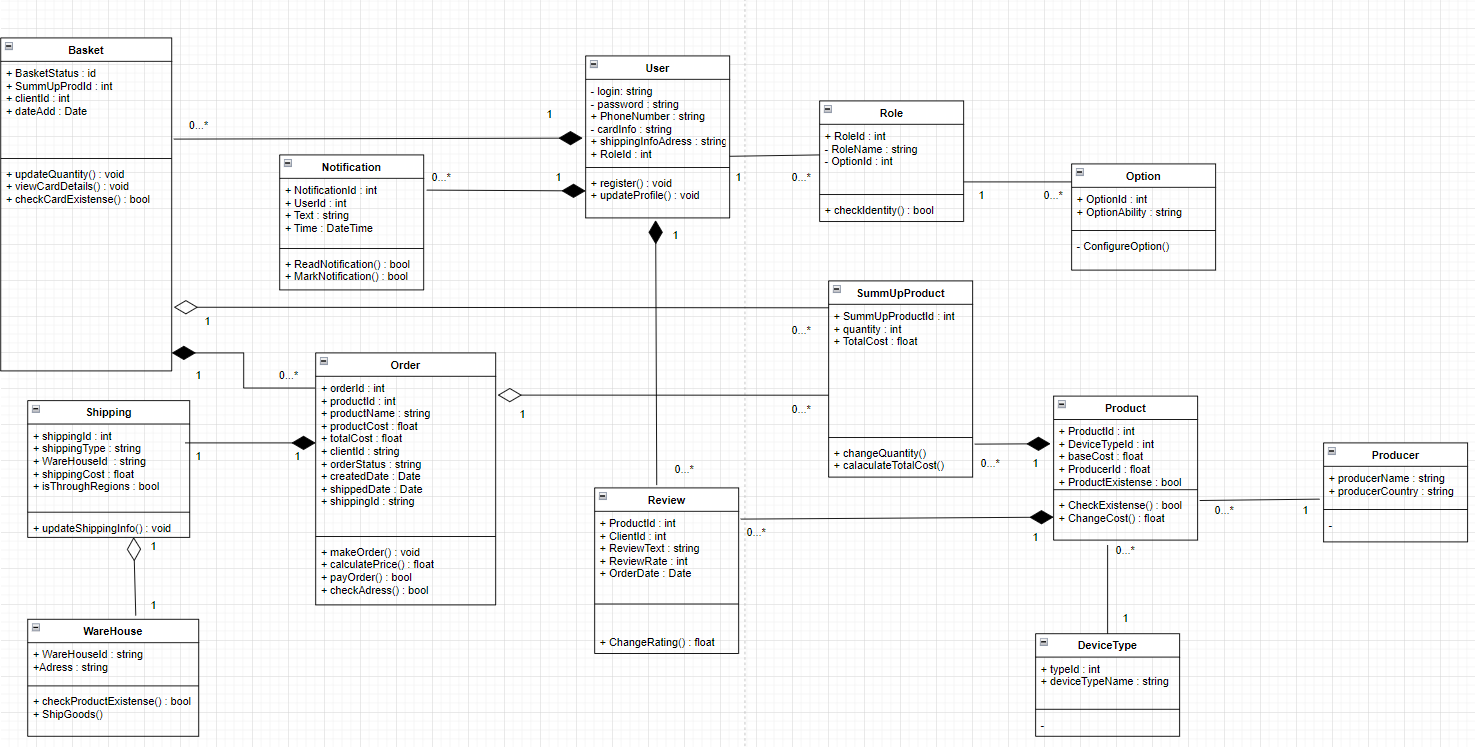


Рисунок 2. Диаграмма классов

1. **Динамическое моделирование системы**

Начнем с анализа состояний в автоматизируемой системе электронного компьютерного магазина для построения диаграммы состояний (**UML StateChart Diagram**) на рис.3:

Сделаем это на примере состояний объекта «Заказ» для данной системы (учитывая тот факт, что сущность «Заказ» построена на сущности «Корзина» (смотреть диаграмму классов – рис.2)

Состояния объекта «Заказ в системе»:

1. Покупатель собрал желаемые товары в корзине, указав их требуемое количество. Объект «Корзина» заполнен. Начинает создаваться объект «Заказ»
2. От системы создания заказа приходит форма для оформления заказа, в которой необходимо заполнить данные о способах оплаты, реквизитах пользователя, место для получения заказа.

А) Заказ не будет создан до того момента, пока не будут заполнены реквизиты и не будет выбрано доступное место получения заказа.

Б) Оплата заказа производится через внешнюю платежную систему, заказ будет обработан только после получения подтверждения о его оплате.

3) От системы проверки оформления заказа пришло сообщение (внутри системы) об отсутствии всех необходимых введенных данных.

Покупателю на сайте показывается соотвествующее уведомление, переходим к пункту 8

4) От платежной системы пришло сообщение об отказе в оплате или истек максимальный срок ожидания оплаты.

Заказ отменяется, пользователю приходит на электронную почту уведомление об отмене заказа. После этого переходим к пункту 8.

1. От платежной системы пришло подтверждение об оплате. Заказ помечается как «оплаченный».

Указанный покупателем адрес доставки проверяется во внешней системе проверки адресов, то есть заказ ожидает подтверждение о проверке адреса.

1. От системы проверки адресов пришло сообщение об ошибке в указанном для доставки адресе. Покупателю приходит уведомление на почту об отмене заказа и возврате средств, платежная система получает команду вернуть деньги за отмененный заказ на счет пользовтеля.
2. От системы проверки адресов пришло подтверждение правильности введенного адреса. Заказ помечается как «доставляемый» и переходит в состояние ожидания подтверждения о доставке.

Пользовтелю приходит уведомление на почту о передаче заказа в службу доставки.

1. Заказ помечается как «не созданный» и «ждёт» дооформления.
2. Заказ помечается как «отмененный» (и «ждёт» переоформления)
3. Заказ приходит в пункт выдачи и помечается как «досталенный». Покупатель получает соответствующее уведомление о доставке товара.
   1. **Диаграмма состояний в системе** для объекта «Заказ» (Statechart Daiagram UML):

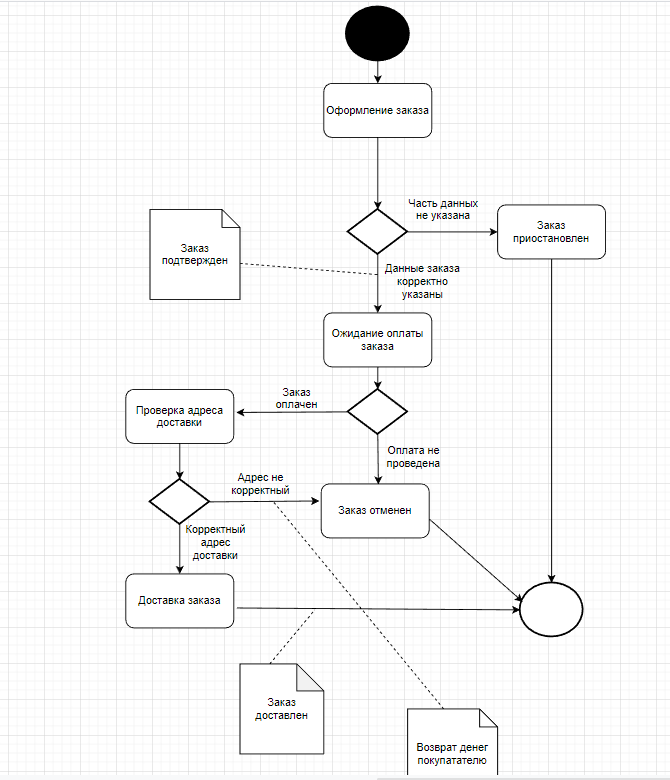


Рисунок 3. Пример диаграммы состояний системы для сущности «Заказ»

Далее проанализируем виды деятельности в системе и построим диаграмму последовательностей **(UML Sequence Diagram)** на рис.4.

Используем для это логику перехода между состояниями объекта «Заказ» из построенной и описанной ранее UML - диаграммы состояний (рис.3)

Выделим такие основные последовательности (последовательные «остановки» для создаваемого заказа в системе, вызываемые функции и сопровождаемые ими действия, а также ответные реакции системы (тоже в виде вызова функций – что представляет концептуальную схему взаимодействия сущностей в системе с их контроллерами).

Последовательные «звенья» системы оформления заказа для пользователя, который был зарегистрирован в системе:

- Покупатель

Функции:

- Login (Авторизоваться) : ответное действие – уведомление о результате проведенной авторизации

- AddProductToBasket (Добавить товар в корзину) : ответное действие – ShowProductPrompt (сообщение о изменении количества товаров данного артикула в корзине)

- ChangeProductsQuantity (Изменить количество товаров в корзине) : ответное действие – ChangeBasketConditionList (Изменить список товаров в корзине)

- OpenBasket (Открыть страницу «Корзина)»: ответное действие – DisplayBasketList (показать текущее состояние корзины на новой странице электронного сайта магазина)

- CreateOrder (Создать заказ) : ответное действие – GetOrderNotification (показать статус созданного заказа и перейти к следующим шагам подтверждения оформления заказа (заказ может иметь на этом этапе состояния «Создан» или «Не создан» (по причине указания неполноты данных или их некорректности))

- ConfirmOrder (Подтвердить заказ): ответное действие – GetResponse()

- PayOrder (Оплатить заказ) : ответное действие – GetPaymentStatus (показать состояние оплаты заказа (заказ на данном этапе может иметь состояния «Оплачен» или «Не оплачен»))

Используемые контроллеры:

- Account Controller (требуется для регистрации пользователей в системе, проверке данных, вводимых пользователем при его авторизации в системе)

Ответные действия :

* RegistrationNotification (уведомление о регистрации)
* Login\Logout Notifications (уведомление о проведенной авторизации)

- Basket Controller

Ответные действия :

– ShowProductPrompt() : Product (сообщение о изменении количества товаров данного артикула в корзине)

- DisplayBasketList() : Basket (Показать текущее состояние корзины на новой странице «Корзина»)

- Order Controller

Функции:

- CreateOrder() : Order (Проверить корректность указанного в заказе адреса и его существования)

Ответное действие системы – GetOrderNotification() – уведомление о результате проверки указанного адреса

Используемые контроллеры:

- Order Manager (система управления заказа)

Функции:

- ProcessOrder():Order- обработка заказа

- GetOrderProcessedResult() – получить результат обработки заказа

- Order Confirmer (система подтверждения заказа)

Функции:

- ConfirmOrder: Order (подтвердить заказ)

- GetResponse() – получить ответ результата подтверждения заказа

- Order Pay (система оплаты заказа)

Функции:

- PayOrder(): Order – оплатить заказ (во внешней платежной системе)

## Диаграмма последовательностей в системе – UML Sequence Diagram (для последовательности по созданию пользователей заказа, а также его подтверждению и обработке):

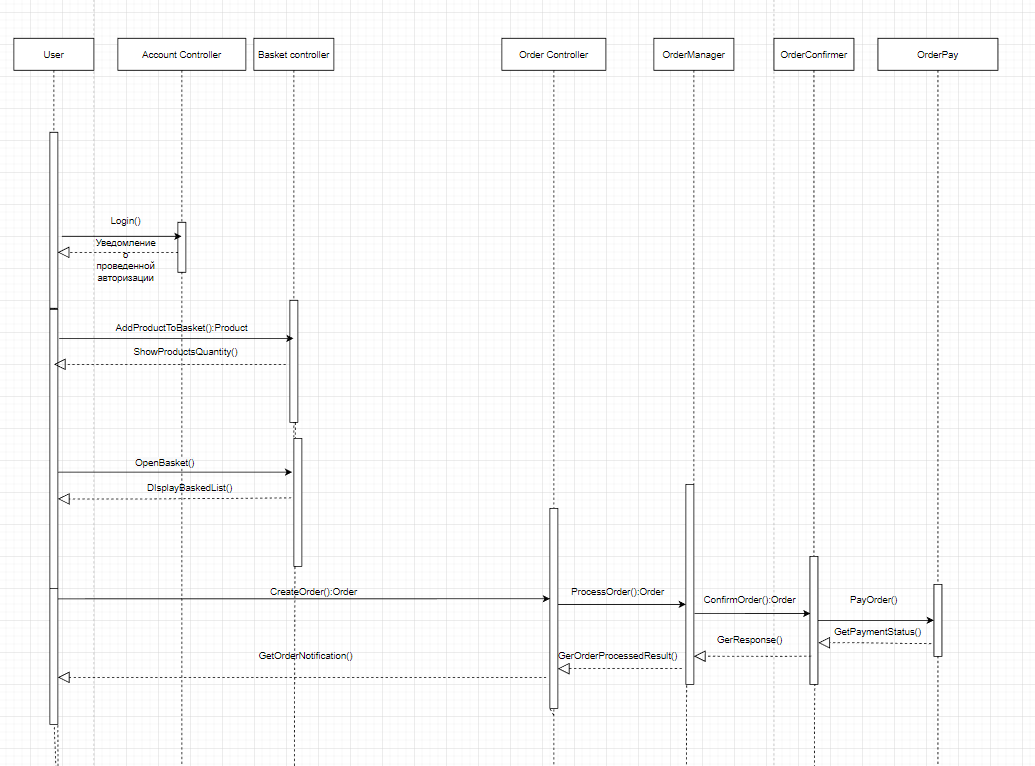


Рисунок 4. Диаграмма последовательностей в системе

Анализ видов деятельности и построение диаграммы видов деятельности (**UML Activity Diagram**) на рисунках пункта 5:

Основные действующие лица в системе: клиент (покупатель), менеджер, администратор.

Кроме этого, основные действия по оформлению заказа, составлении платежного счета и проверке его оплаты, по проверке корректности указанного адреса, будет производить автоматизированная система интернет-магазина компьютерной техники.

Менеджер в случае отсутствия каких-то товаров из заказа в наличии на внутренних складах осуществляет по своему усмотрению поставку товаров со внешних складов (складов поставщика). В случае, если заказ пришел, а клиент его не получил, то менеджер делает обратный запрос на возвращение товаров (на внутренние склады).

Администратор занмается проверкой выдачи заказов, ставит подтверждение при явке покупателя и получении им заказа, либо в случае неявки он отзывает товары из заказа.

(После получения заказа и его закрытия в системе, покупатель может оставить о нем отзыв на электронном сайте и поставить оценку товарам, администратор также может модерировать эти отзывы)

Ниже будут представлена диаграмма активностей работы интернет-магазина компьютерной техники (рис. 5.1 и рис. 5.2):

* 1. **Диаграмма видов деятельности в системе:**

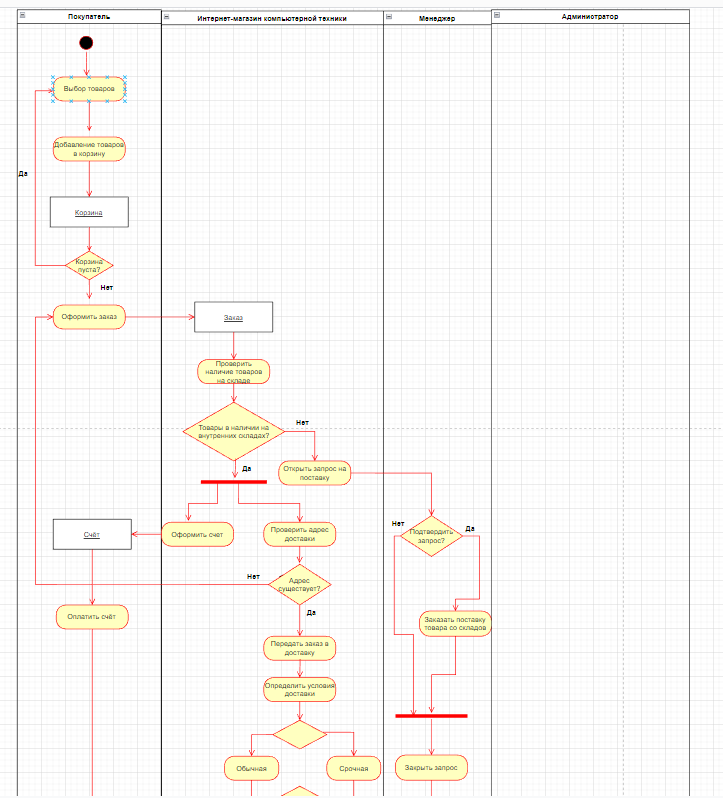


Рисунок 5.1. Диаграмма видов деятельности – часть 1

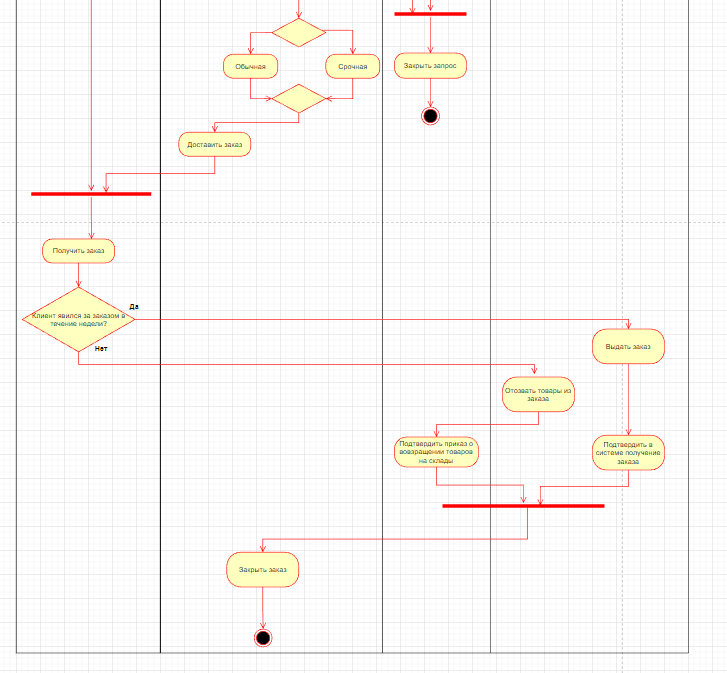


Рисунок 5.2. Диаграмма видов деятельности – часть 2

1. **Реализация системы**
   1. Общее описание реализации системы:

После проектирования системы магазина компьютерной техники требуется выполнить реализацию основных компонентов, таких как:

- База данных системы, в которой будет храниться информация о всех пользователях, зарегистрированных в системе, их корзинах и заказах, а также информация по внутреннему распорядку сети (то есть магазинов и складов, наличию в них тех или иных групп товаров, их артикула, а также всех доставок или иных внутренних действий)

- Сервис системы, представляющий набор контроллеров и зависимостей логических компонентов, благодаря взаимодейтсвию которых происходит работа системы.

- Электронный сайт для системы, который будет открываться в веб-браузере и доступен для просмотра пользователям системы. Он будет состоять из набора представлений, сгенерированных на основе созданных моделей объектов системы, а также стандартной HTML – верстки.

В процесс реализации я буду использовать такую среду программирования, как Visual Studio 2022 и платформу .NET 5. Разработка будет вестись на последней версии объектно-ориентированного языка C#.

Основой для решения будет являться создание проекта с веб-архитектурой, представляющего сервис системы. Для простоты я буду использовать встроенный в ASP.NET Core шаблон MVC (Model-View-Controller), представляющий все необходимые комопоненты, которые потребуются мне при разработке.

Дополнительно будут созданы проекты по шаблону «Библиотека классов», в которых будут размещаться остальные компоненты системы, а конкретно:

- Модели системы, на основе которых будет создаваться контекст базы данных, а также строиться синтаксис некоторых представлений.

- Объекты с данными – то есть контекст базы данных, а также миграции (файлы, используемые при генерации БД).

- Unit-тесты к компонентам системы, благодаря которым можно проверять функциональность логики системы, а также разрабатывать и прорабатывать всякие возможные сценарии их работы, то есть использовать подход TDD к проектированию логики.

Перейдем к более детальному рассмотрению этапов реализации системы.

* 1. Разработка backend – части системы:

Начнем с создания базы данных. В процессе разработки использовался такой фрэймфорк, как Microsoft Entity Framework (EF) , а также Microsoft SQL Server – СУБД для созданной БД.

Использованный подход – Entity Framework Code First (то есть бд сгенерировалась после кодирования основных зависимостей её компонентов) – на рисунке 6.1. представлены иерархия папок и файлоф с таблицами сгенерированнной БД.

Для создания базы данных использовались контекст – файл, содержащий набор «будущих таблиц» для создаваемой БД, ссылающихся на ранее разработанные модели системы, ранее описанные в процессе проектированию диаграммы классов системы.

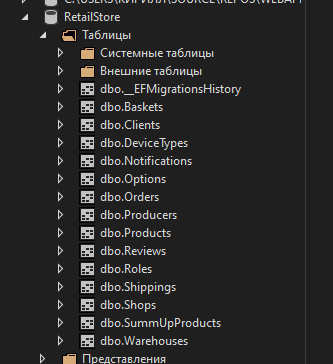


Рисунок 6.1. Файлы с таблицами и скриптами БД MS SQL Server

Кроме контекста, потребовались также миграции – файлы, которые использует Entity Framework для генерации базы данных по указанному пути – которые представлены на рисунке 6.2.

И контекст, и миграции лежат в отдельном проекте под названием Data. Контекст имеет назване RetailStoreDataContext.

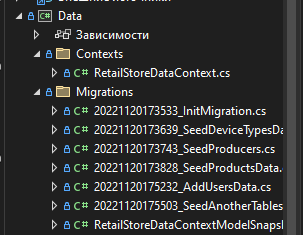


Рисунок 6.2. Файлы с миграциями в проекте Data

Для упрощения разработки я использовал встроенную в Visual Studio инструменты Microsoft SQL Server и Microsoft SQL Management Studio. В итоге, пользуясь консолью диспетчера пакетов Nuget была сгенерирована БД автоматическими миграции EF 6. Остальные миграции были написаны мною вручную, и в них находится процесс наполнения данными таблиц БД (миграции с префиксом Seed). Благодаря такому подходу любой пользователь может открыть мой проект на любом другом компьютере и с помощью ввода одной команды «Update Database» может получить исходное содержание БД с некоторыми данными. Пример показан ниже на рисунке 6.3.

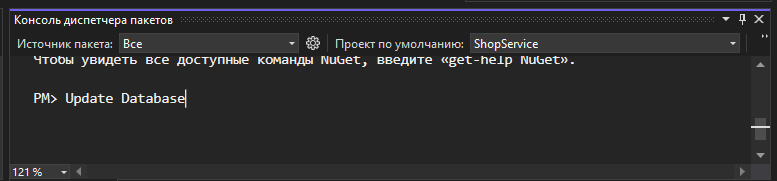


Рисунок 6.3. Консоль диспетчера пакетов NuGet

Далее перейдем к разработке сервиса системы. Как я уже говорил, я использовал шаблон MVC. Модели были мною уже написаны и находятся в отдельном проекте и благодаря внедрению зависимостей я могу получить к ним доступ из любого места проекта согласно принципу SOLID.

Список всех моделей (также указан ниже ни рисунке 6.3):

- Basket

- DeviceType

- Notification

- Option

- Order

- Producer

- Review

- Role

- Shipping

- Shop

- SummUpProduct

- User

- Warehouse

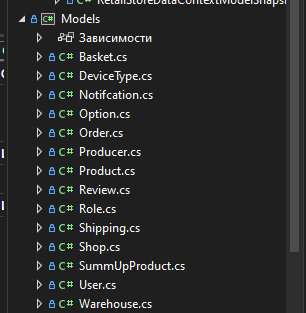


Рисунок 6.4. Файлы с классами доменных моделей в проекте Models

Далее на все эти модели, логику работы с заказом и логику работу с контекстом БД были написаны модульные тесты для проведения автоматического модульного тестирования (Unit-тесты) с использованием фрэймворка Xunit и с помощью дополнительных библиотек FluentAssertions (для сравнения результатов полученных с ожидаемыми) и MOQ (для создавания фэйковых объектов для тестируемых вспомогательных сущностей) – на рисунке 6.5. представлены файлы с тестами на модели. На рисунке 6.6. представлены файлы с тестами на логику работы с заказом. На рисунке 6.6. представлены файлы с тестами на логику репозиториев.

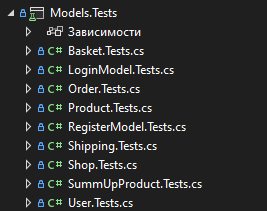


Рисунок 6.5. Модульные тесты для доменных моделей в проекте Models.Tests

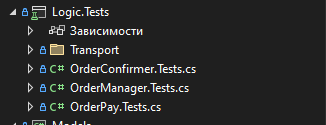


Рисунок 6.6. Модульные тесты для классов логики работы с заказом в проекте Logic.Tests

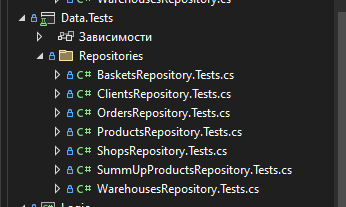


Рисунок 6.7. Модульные тесты для классов с репозиториями в проекте Data.Tests

Далее я написал несколько основных контроллеров (которые представлены ниже ни рисунке 6.8):

* HomeController – это ApiController, находящийся по адресу «api/home» и автоматически запускаемый при старте веб-приложения (а конкретно запускается его HTTP-GET метод Index – который представляет собой показ каталога товаров на сайте). По умолчанию, и авторизованные, и неавторизованные пользователь могут просматривать каталог, а также проходить по страницам с более подробной информацией о товарах. Некоторые элементы, такие как ссылки на страницы для менеджеров и администраторов, скрыты от остальных пользователей.
* AccountController – это ApiController, находящийся по адресу «api/account». Его методы используются при регистрации/авторизации пользователей на сайте, а также в процессе аутентификации (то есть определения роли пользователя в систему и, соответсвенно, их функционала на электронном сайте). По умолчанию пользователю не предлагается регистрироваться или пройти авторизацию, но при попытках выполнить действия «Добавить товар в корзину» или «Создать заказ» - пользователь будет автоматически перенаправлен на страницу авторизации. Сделано это для оптимизации работы системы, чтобы не приходилось создавать записи в базе данных и асинхронно использовать ресурсы backend – части системы.

Стоит также упомянуть про используемые средства для создания системы аутентификации и авторизации. При разработке я применил HTTP Cookies Claims, а не встроенную в MVC аутентификацию по Identity. Сделано это было с целью расщирения функционала и получения единой базы данных, в которой бы хранились как данные о сети магазинов и товаров, так и все данные о пользователях и их действиях.

- BasketController – это контроллер, который используется при всех действиях пользователя, связанных с корзиной, а конкретно с добавлением и удалением товаров в корзину, очищением корзины, а также перенаправлением на страницу Корзины. По умолчанию у авторизованного пользователя создается пустая корзина, состояние которой сохраняется все время в БД.

- OrderController – это контроллер, который используется при всех действиях пользователя, связанных с заказом, а конкретно с созданием заказа, его подвтерждением или отменой, оплатой и ожиданием уведомления о его получении. По умолчанию заказ не может создаваться пустым и создается он на основе непустой корзины авторизованного пользователя.

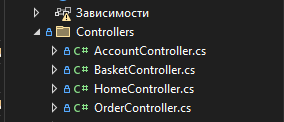


Рисунок 6.8. Файлы с основными контролеррыми в проекте ShopService(MVC)

Далее была разработана логическая система, отвечающая за создание заказа, управление им и его обработку, а также дальнейшее подтверждение (подтверждение заказа включает в себя подтверждение адреса магазина и подтверждение оплаты) – файлы представлены ниже на рисунке 6.9.

Для этих целей был создан проект Logic по шаблону проекта «Библиотека классов», в нем были созданы такие интерфейсы, как : IOrderManager.cs, IOrderConfirmer.cs, IOrderPay.cs, которые описывают основную логику, используя абстракции.

Сами классы OrderManager.cs, OrderConfirmer.cs, OrderPay.cs, соответственно, реализуют эти интерфейсы, и их методы вызываются и используются в результате запросов, которые пользователь посылает из контроллеров. Кроме этого, также присутствуют такие альтернативные варианты в логике, как например, отменить заказ или выдать заказ заочно.

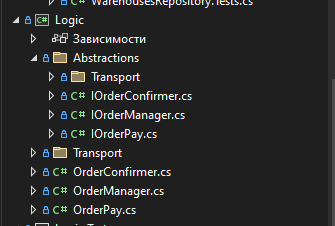


Рисунок 6.9. Файлы с классами логики для работы с заказом в проекте Logic

Для работы с контекстом базы данных и для создания более удобной и масштабируемой системы, также были созданы такие классы, как репозитории – ниже на рисунке 6.10 представлены файлы с логикой репозиториев:

IProductsRepository.cs, IBasketsRepository.cs, IOrdersRepository.cs, ISummUpProductsRepository.cs, IShopsRepository.cs, IWarehousesRepository.cs, IClientsRepository.cs, благодаря которым я произвожу различные манипуляции по вставке, обновлению, поиску, удалению данных из контекста БД (RetailStoreDataContext.cs).

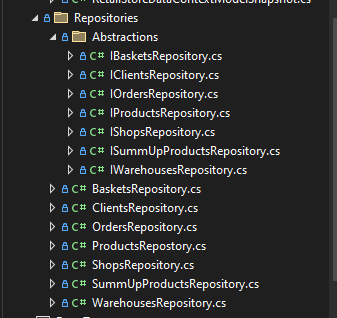


Рисунок 6.10. Файлы с классами репозиториев в проекте Data

Важнейшим звеном построения сервиса было использование механизма внедрения зависимостей, а именно механизма Dependency Injection. В файле Program.cs были внедрены зависимости на все логические классы, использующиеся в проектах решения ShopService (на примере использования зависимостей и жизненных циклов Scoped – для логики работы с контекстом БД) – пример добавления DI показан ниже на рисунке 6.11.

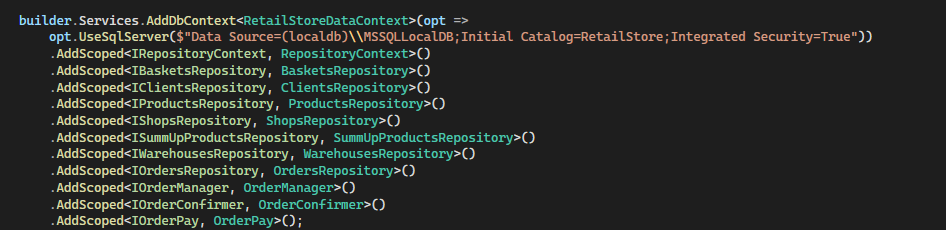


Рисунок 6.11. Пример внедрения зависимостей (DI) в сервисе

* 1. Разработка frontend – части системы:

Кроме этого еще велась разработка внешнего вида системы, а именно создания электронного сайта, который состоит из отдельных страниц (или views – представлений). В данном сервисе я использовал концепцию Razor Pages, то есть представлений, у которых модели, используемые на страницах хранятся рядом со своей моделью (и содержат основные методы, такие как, например, Post или Get).

В процессе разработки были создано несколько страниц (или views), дискретно разбитых по каталогам согласно названию контроллеров, методы (actions) которых используют эти страницы. Рассмотрим их поподробней:

- Account содержит две страницы: Register.cshtml (на рисунке 7.1.) и Login.cshtml (на рисунке 7.2.), в которых были созданы основные формы для регистрации и авторизации пользователей на сайте.

Пример страницы Register.cshtml:

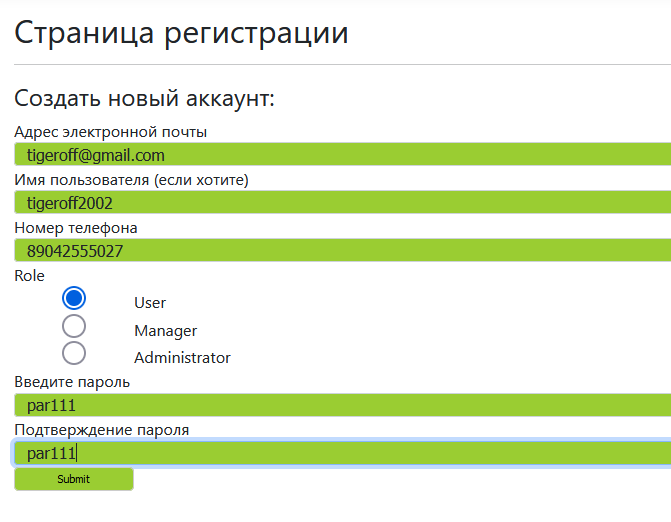


Рисунок 7.1. Представление Register

Пример страницы Login.cshtml:

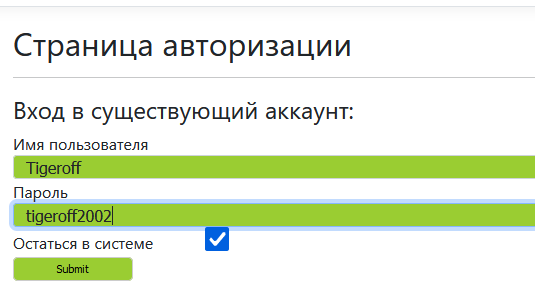


Рисунок 7.2. Представление Login

- Home содержит такие используемые в системе страницы, как:

- Index.cshtml – страница с каталогом (на рисунке 8.1.)

- Details.cshtml – страница с подробной информацией о товаре (на рисунке 8.2.)

- Первая страница с фильтрацией каталога – например, GetProductsByDeviceType.cshtml – которая подгружается на странице каталога при применении фильтра товаров по типу устройств (на рисунке 8.3.)

- Вторая страница с фильтрацией каталога – например, GetProductsByProducer.cshtml – которая подгружается на странице каталога при применении фильтра товаров по производителю (на рисунке 8.4.)

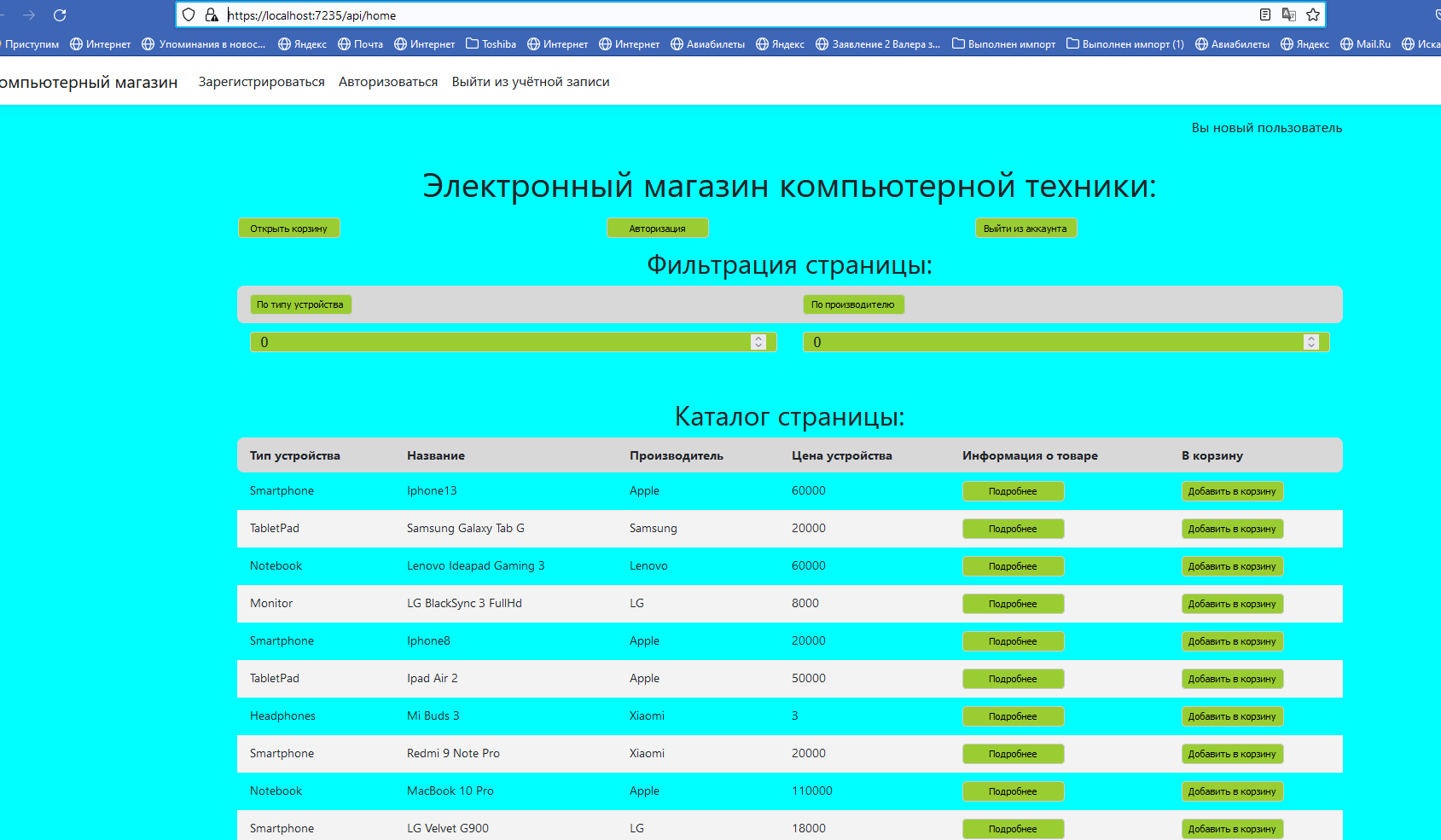


Рисунок 8.1. Представление Index

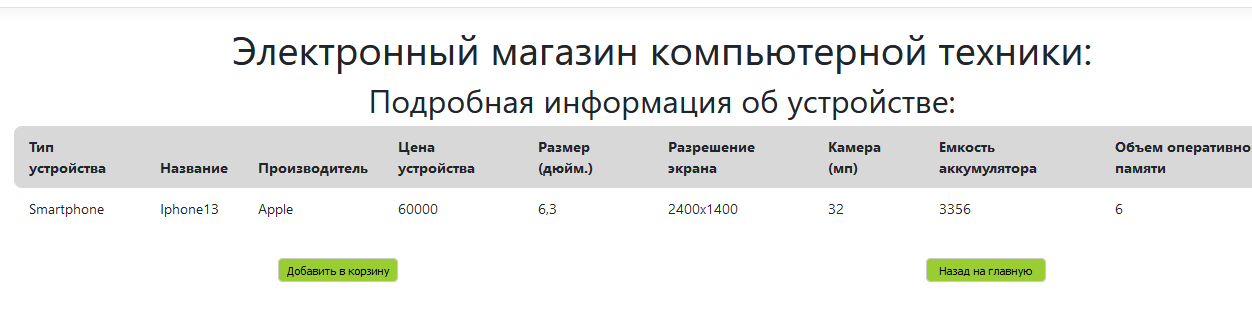


Рисунок 8.2. Представление Details



Рисунок 8.3. Представление FGetProductsByDeviceType



Рисунок 8.4. Представление GetProductsByProducer

- AdminPage.cshtml – страница администратора, содержащая некоторый его функционал, связанный с просмотром учетных записей зарегистрированных пользователей, всех их заказов, а также с функциональными возможностями по изменению статуса заказа, его отменой, а также подтверждением выдачи на руки клиенту (ниже на рисунке 9)



Рисунок 9. Представление AdminPage с уровнем доступа администратора

- ManagerPage.cshtml – страница менеджера, содержащая его функционал, заключающийся в возможностях осуществлять создание заявок на доставку товаров со складов в магазины, а также обратной их отрузкой, а также просмотром каталога наличия товаров в отдельных местах и их контроля (ниже на рисунке 10)

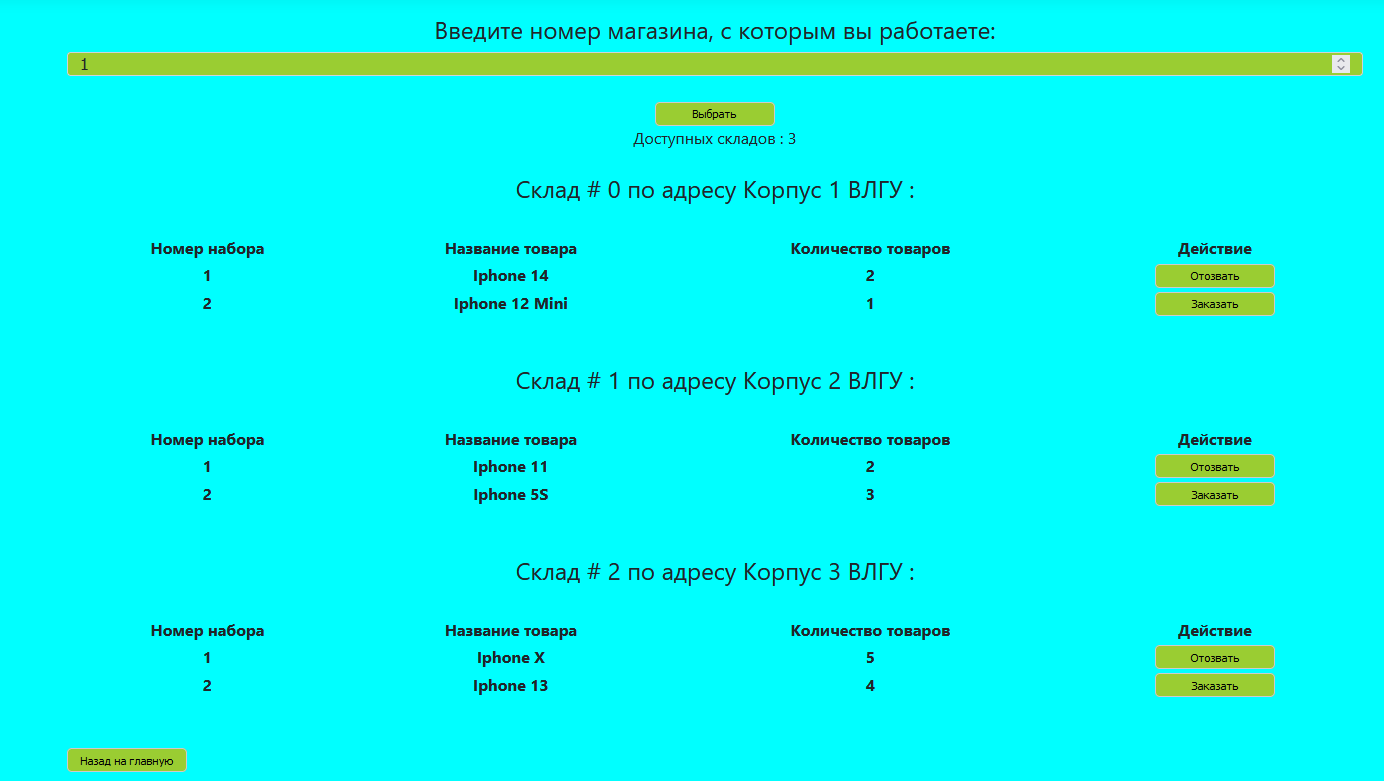


Рисунок 10. Представление ManagerPage с уровнем доступа менеджера магазина

- Basket содержит единственную страницу BasketPage.cshtml, которая представляет собой страницу с корзиной товара, на которой можно производить изменение количества конкретных товаров, а также процедуру создания заказа.

Ниже на рисунке 11.1. показан пример добавления одной единицы товара в корзину. На рисунке 11.2. представлен пример с увеличением кол-ва единиц товара в корзине. На рисунке 11.3 показан пример с очищением корзины.



Рисунок 11.1. Представление BasketPage - пример добавления единственной единицы товара



Рисунок 11.2. Представление BasketPage - пример увеличения количества единиц товара в корзине

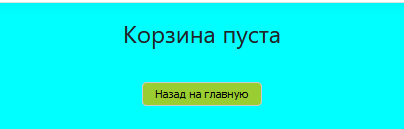


Рисунок 11.3. Представление BasketPage – пример очищения корзины

- Order содержит страницу OrderHome.cshtml (на рисунке 12.1. - с функционалом по началу создания заказа и ввода контактных данных с адресом магазина для получения заказа). А также страницу OrderConfirm.cshtml (на рисунке 12.2 - с функционалом по подтверждению и оплате заказа).

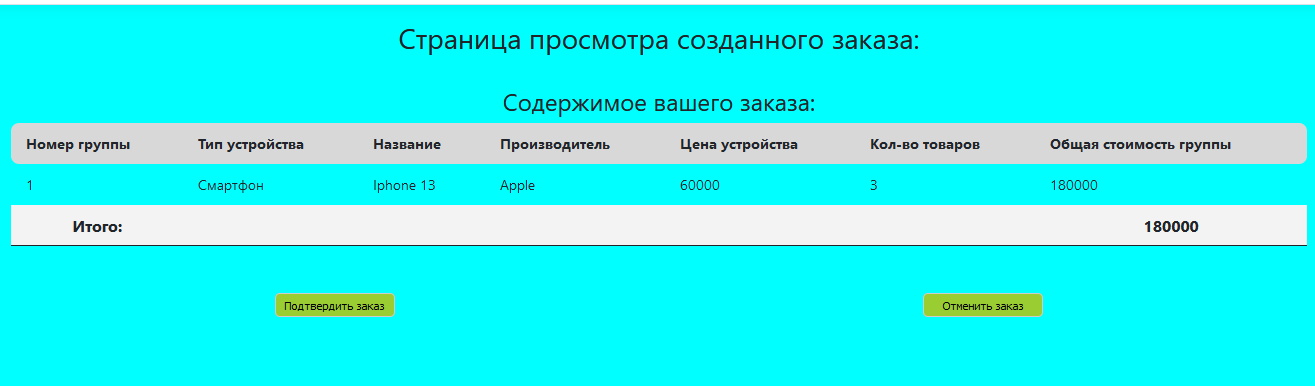


Рисунок 12.1. Представление OrderHome (создание заказа)

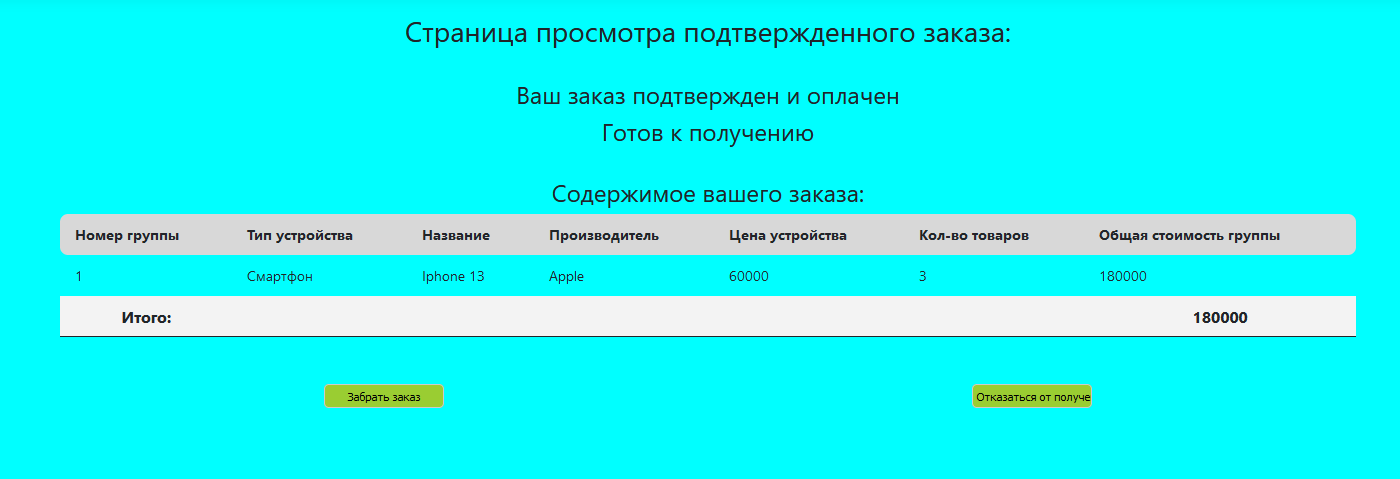


Рисунок 12.2. Представление OrderConfirm (процесс подтверждения заказа)

# Заключение

В ходе выполнения курсового проекта была разработана программная система для автоматизации работы магазина компьютерной техники.

Также была исследована предметная область, проведена работа с изучением возможностей платформы .NET и языка C#, была проведена разработка web – приложения с использованием системы контроли версий – git.

# Список использованных источников

1. Microsoft Developer Network – Сайт разработчиков Microsoft. [Электронный ресурс]:

URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/>

(Дата обращения: 08.12.2022).

1. METANIT.COM – Сайт о программировании. [Электронный ресурс]: URL: <https://metanit.com/sharp/>

(Дата обращения: 16.12.2022).

1. Habr - Сайт о программировани. [Электронный ресурс] URL: <https://habr.com/ru/hub/csharp/page3/>

(Дата обращения: 28.11.2022).

1. CLR VIA C# - Книга о программировании на языке C#.

[Электронный ресурс].

URL: <https://www.google.com/search?q=richter+c%23&biw=1920&bih=927&sxsrf>

(Дата обращения: 20.11.2022).

1. ASP.NET CORE MVC – Разработка MVC приложений в ASP.NET CORE.

[Электронный ресурс].

URL: <https://learn.microsoft.com/ru-ru/aspnet/core/mvc/overview?view=aspnetcore-7.0>

(Дата обращения: 12.12.2022).

1. ASP.NET CORE Razor Pages – Разработка Razor Pages в ASP.NET CORE:

[Электронный ресурс].

URL:<https://learn.microsoft.com/ru-ru/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio>

(Дата обращения: 13.12.2022).

**Приложение A.**

**Исходный код программной системы.**

Вся разработка велась с использованием системы контроли версий git и сервиса для хранения удаленных репозиториев Github, на котором размещен код моей программной системы по ссылке: <https://github.com/Tigeroff2002/ShopService> (рис.13 – представлен скришот курсового проекта, расположенный на сайте GitHub).

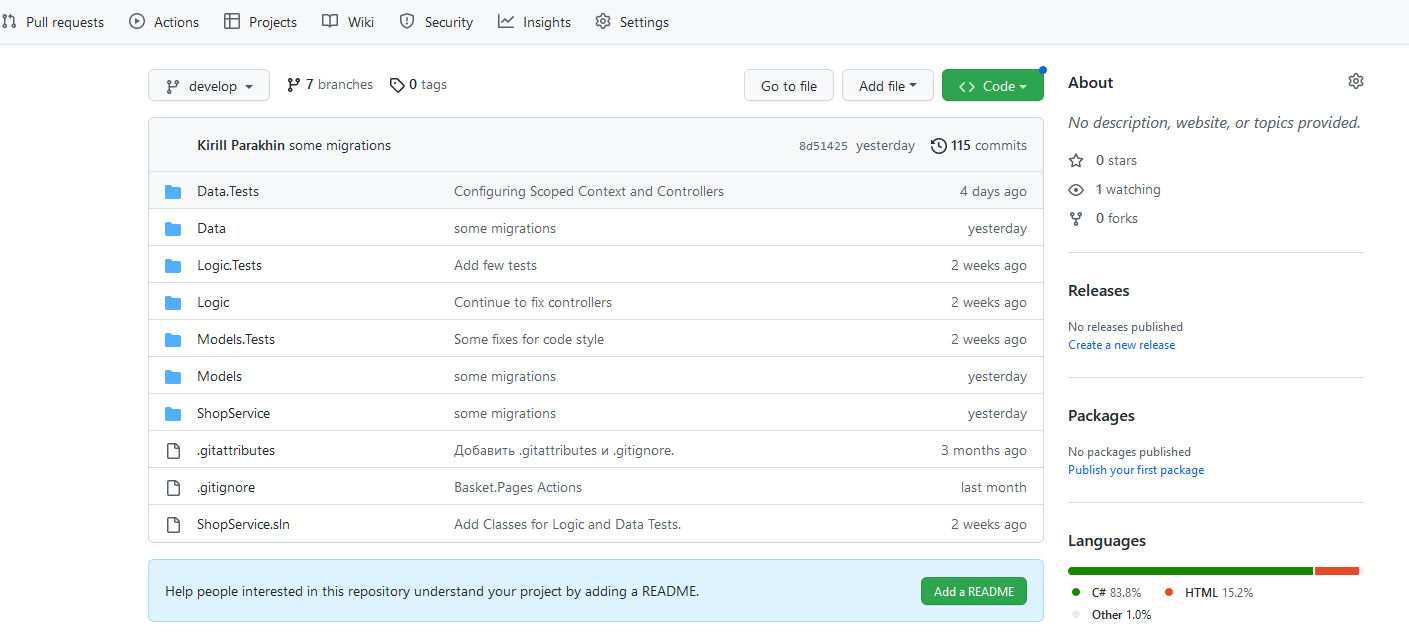


Рисунок 13. Репозиторий проекта на GitHub

**Ниже показан наиболее показательный код для разработанной программной системы.**

**Код класса AccountController.cs – пример контроллера для действий авторизации и регистрации:**

[HttpGet("login")]

public IActionResult Login()

{

var objLoginModel = new LoginModel();

return View("Login", objLoginModel);

}

public async Task<ActionResult> LoginPost(

string nickName,

string password)

{

if (string.IsNullOrEmpty(nickName))

{

throw new ArgumentException(nameof(nickName));

}

if (string.IsNullOrEmpty(password))

{

throw new ArgumentException(nameof(password));

}

var objLoginModel = new LoginModel

{

NickName = nickName,

Password = password

};

if (ModelState.IsValid)

{

var token = CancellationToken.None;

ArgumentNullException.ThrowIfNull(objLoginModel.Email);

var user = await \_repository.FindAsync(objLoginModel.Email, token)

.ConfigureAwait(false);

if (user == null)

{

ViewBag.Message = "Such user was not registered in system yet";

return View("Register", new RegisterModel());

}

else

{

var claims = new List<Claim>() {

new Claim(ClaimTypes.NameIdentifier, Convert.ToString(user.Id)),

new Claim(ClaimTypes.Name, user!.EmailAdress!),

new Claim(ClaimTypes.Role, user!.Role!.RoleCaption!)};

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal, new AuthenticationProperties()

{

IsPersistent = objLoginModel.RememberLogin

});

\_logger!.LogInformation("User was successfuly authorized!");

return RedirectToAction("Index", "Home", (new List<Product>(), user));

}

}

return RedirectToAction("Index", "Home", (new List<Product>(), new User(1, 0)));

}

[HttpGet("logout")]

public async Task<IActionResult> LogOut()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

\_logger!.LogInformation("User was successfuly unauthorized!");

return View("Login", new LoginModel());

}

[HttpGet("register")]

public IActionResult Register()

{

var objRegisterModel = new RegisterModel();

return View("Register", objRegisterModel);

}

[HttpGet("postregister")]

public async Task<IActionResult> RegisterPost(

string email,

string nickName,

int roleType,

string contactNumber,

string password,

string confirmPassword)

{

var objRegisterModel = new RegisterModel(

email,

nickName,

roleType,

contactNumber,

password,

confirmPassword);

if (ModelState.IsValid)

{

var user = new User()

{

Id = \_repository.UserCount,

NickName = objRegisterModel.NickName,

Password = objRegisterModel?.Password,

ContactNumber = objRegisterModel?.ContactNumber,

EmailAdress = objRegisterModel?.Email,

TotalPurchase = 0f,

Discount = 0f,

Role = new Role(RoleType.AuthUser)

};

if (user != null)

{

var existense = await CheckCurrentUserExistense(user);

if (false)

{

ViewBag.Message = "Such user have already registered in system";

\_logger!.LogInformation("User with these data was found in system!");

return View("Login", new LoginModel());

}

ArgumentNullException.ThrowIfNull(user!.EmailAdress);

ArgumentNullException.ThrowIfNull(user!.Role.RoleCaption);

var claims = new List<Claim>()

{

new Claim(ClaimTypes.NameIdentifier, Convert.ToString(user!.Id)),

new Claim(ClaimTypes.Name, user!.EmailAdress!),

new Claim(ClaimTypes.Role, user!.Role!.RoleCaption!

)};

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

await HttpContext.SignInAsync(

CookieAuthenticationDefaults.AuthenticationScheme,

principal,

new AuthenticationProperties()

{

IsPersistent = true

});

\_logger!.LogInformation("User was successfuly registered in system!");

return RedirectToAction("Index", "Home", (new List<Product>(), user));

}

else

{

ViewBag.Message = "Invalid Credential";

return View("Register", new RegisterModel());

}

}

return RedirectToAction("Index", "Home", (new List<Product>(), new User(1, 0)));

}

**Код класса OrderManager.cs – пример класса логики:**

public OrderManager(

ILogger<OrderManager> logger,

IOrderConfirmer confirmer,

IOrdersRepository repository)

{

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

\_confirmer = confirmer ?? throw new ArgumentNullException(nameof(confirmer));

\_repository = repository ?? throw new ArgumentNullException(nameof(repository));

\_logger.LogInformation("OrderManager is created");

}

public async Task ProcessOrdersAsync(CancellationToken cancellationToken)

{

while (!cancellationToken.IsCancellationRequested)

{

await Task.Delay(1\_000, cancellationToken)

.ConfigureAwait(false);

\_logger.LogInformation("Processor is starting now after 1s delay");

}

}

public async Task<Order> ProcessAsync(Order order, CancellationToken cancellationToken)

{

ArgumentNullException.ThrowIfNull(order);

cancellationToken.ThrowIfCancellationRequested();

\_logger.LogInformation("OrderManager is begginning to process order...");

var createdOrder = await CreateAsync(order, cancellationToken)

.ConfigureAwait(false);

\_logger.LogInformation("OrderManager is pushing order to context within repository...");

await \_repository.AddOrderAsync(createdOrder, cancellationToken)

.ConfigureAwait(false);

\_logger.LogInformation("OrderManager is confirming order...");

var orderToConfirm = await \_confirmer.ConfirmOrderAsync(order, cancellationToken)

.ConfigureAwait(false);

if (orderToConfirm.isPayd == true)

{

\_logger.LogInformation("OrderConfirmer confirmed order succesfully");

\_logger.LogInformation("OrderManager is waiting when client will take his order...");

var orderToGet = await GiveOrder(orderToConfirm, cancellationToken)

.ConfigureAwait(false);

order.isGot = orderToGet;

}

if (orderToConfirm.isGot == false && orderToConfirm.isPayd == true)

{

\_logger.LogInformation("OrderManager is automatically cancellating order...");

CancelOrder(orderToConfirm, cancellationToken);

orderToConfirm.isDeleted = true;

}

else if (orderToConfirm.isGot == true)

{

orderToConfirm.isDeleted = false;

\_logger.LogInformation("OrderManager is updating in context the order what was got by client...");

\_repository.UpdateOrder(orderToConfirm, cancellationToken);

\_repository.SaveChanges();

}

return orderToConfirm;

}

public async Task<Order> CreateAsync(Order order, CancellationToken cancellationToken)

{

ArgumentNullException.ThrowIfNull(order);

cancellationToken.ThrowIfCancellationRequested();

var orderAlreadyExists = await \_repository.FindOrderAsync(order, cancellationToken)

.ConfigureAwait(false) != null;

if (!orderAlreadyExists)

{

if (order.ResultCost != 0 && order.SummUpProducts != null)

{

if (order.SummUpProducts.Count > 0)

{

order.isReadyForConfirmation = true;

order.isReadyForPayment = false;

order.isPayd = false;

order.isGot = false;

}

}

}

\_logger.LogInformation("OrderManager created the order");

return order;

}

public async Task<bool> GiveOrder(Order order, CancellationToken cancellationToken)

{

ArgumentNullException.ThrowIfNull(order);

cancellationToken.ThrowIfCancellationRequested();

\_logger.LogInformation("Order is ready to get! Waiting for client...");

var i = 0;

while (i < 7)

{

if (RandomNumberGenerator.GetInt32(3) == 0)

{

break;

}

await Task.Delay(1\_000)

.ConfigureAwait(false);

\_logger.LogInformation($"Days gone {++i}");

}

if (i < 7)

{

\_logger.LogInformation("OrderManager gave the order to client");

return true;

}

\_logger.LogInformation("OrderManager didnt give the order to client");

return false;

}

public void CancelOrder(Order order, CancellationToken cancellationToken)

{

ArgumentNullException.ThrowIfNull(order);

cancellationToken.ThrowIfCancellationRequested();

\_repository.CancelOrder(order, cancellationToken);

\_logger.LogInformation("OrderManager cancelled and deleted the old order from db");

\_repository.SaveChanges();

}

**Код класса LoginModel.cs – пример класса модели для авторизации:**

public class LoginModel : PageModel

{

[Required]

[Display(Name = "EmailAdress")]

public string? Email { get; set; }

[Display(Name = "Имя пользователя")]

public string? NickName { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string? Password { get; set; }

[Display(Name = "Остаться в системе")]

public bool RememberLogin { get; set; }

public string? ReturnUrl { get; set; } = "/Home/Index";

public LoginModel(string email, string password)

{

if (string.IsNullOrWhiteSpace(email))

throw new ArgumentException(nameof(email));

Email = email;

if (string.IsNullOrWhiteSpace(password))

throw new ArgumentException(nameof(password));

Password = password;

RememberLogin = true;

NickName = this.NickName;

}

**Код класса RegisterModel.cs – пример класса модели для регистрации:**

public class RegisterModel : PageModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

public string? Email { get; private set; }

[Display(Name = "Имя пользователя (если хотите)")]

public string? NickName { get; private set; }

public bool isRoleChoosed { get; private set; }

public RoleType Role { get; private set; }

[Required]

[Display(Name = "Номер телефона")]

public string? ContactNumber { get; private set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Введите пароль")]

public string? Password { get; private set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

public string? ConfirmPassword { get; private set; }

public string? ReturnUrl { get; } = "/Home/Index";

public RegisterModel()

{

if (!isRoleChoosed)

Role = RoleType.AuthUser;

}

public RegisterModel(

string email,

string nickName,

int roleType,

string password,

string confirmPassword,

string contactNumber)

{

if (string.IsNullOrWhiteSpace(email))

throw new ArgumentException(nameof(email));

Email = email;

if (string.IsNullOrWhiteSpace(password))

throw new ArgumentException(nameof(password));

Password = password;

if (string.IsNullOrWhiteSpace(confirmPassword))

throw new ArgumentException(nameof(confirmPassword));

ConfirmPassword = confirmPassword;

isRoleChoosed = true;

if (!isRoleChoosed)

Role = RoleType.AuthUser;

else

Role = (RoleType)roleType;

if (string.IsNullOrWhiteSpace(contactNumber))

ContactNumber = "None";

else

ContactNumber = contactNumber;

if (string.IsNullOrWhiteSpace(nickName))

NickName = GetNameFromEmail();

else

NickName = nickName;

if (!ConfirmPassword.Equals(Password))

{

ConfirmPassword = Password;

}

}

**Пример модульных тестов для модели Product.cs:**

[Fact(DisplayName = $"{nameof(Product)} (s) can be compared correctly with false result")]

[Trait("Category", "Unit")]

public void ProductsCanBeComparedCorrectlyWithFalse()

{

// Arrange

var id1 = 1;

var id2 = 2;

var producerMock = new Mock<Producer>();

var deviceTypeMock = new Mock<DeviceType>(1);

// Act

var product1 = new Product(id1, deviceTypeMock.Object, producerMock.Object);

var product2 = new Product(id2, deviceTypeMock.Object, producerMock.Object);

// Assert

product1.Equals(product2).Should().BeFalse();

}

[Fact(DisplayName = $"{nameof(Product)} (s) can be compared correctly with true result")]

[Trait("Category", "Unit")]

public void ProductsCanBeComparedCorrectlyWithTrue()

{

// Arrange

var id1 = 1;

var id2 = 1;

var producerMock = new Mock<Producer>();

var deviceTypeMock1 = new Mock<DeviceType>(1);

var deviceTypeMock2 = new Mock<DeviceType>(2);

// Act

var product1 = new Product(id1, deviceTypeMock1.Object, producerMock.Object);

var product2 = new Product(id2, deviceTypeMock2.Object, producerMock.Object);

// Assert

product1.Equals(product2).Should().BeTrue();

}

**Приложение Б.**

**Структура таблиц БД.**

Всего в базе данных 14 таблиц, скрипты их создания представлены ниже (на примере создания БД с использованием Entity Framework и СУБД Microsoft SQL Server) – ниже на рисунках 14.1-14.14 показаны скрипты создания таблиц БД:

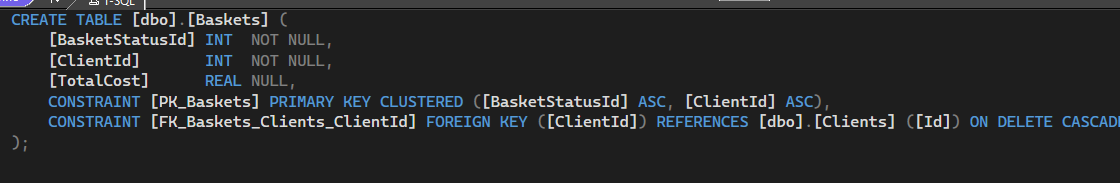


Рисунок 14.1. Скрипт создания таблицы «Baskets»

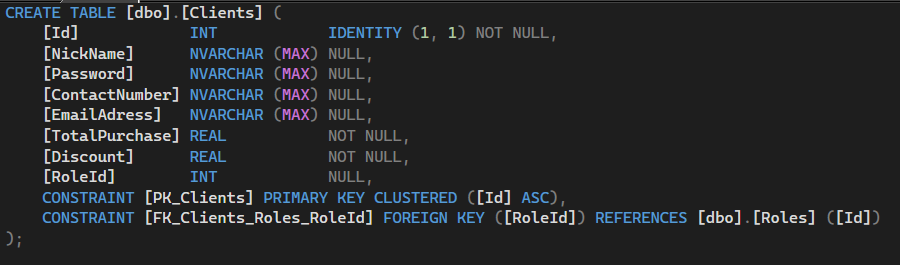


Рисунок 14.2. Скрипт создания таблицы «Clients»

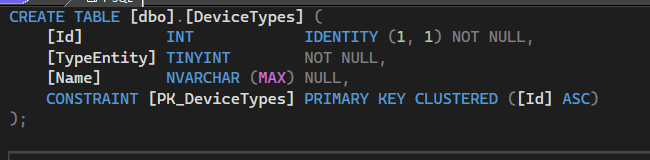


Рисунок 14.3. Скрипт создания таблицы «DeviceTypes»

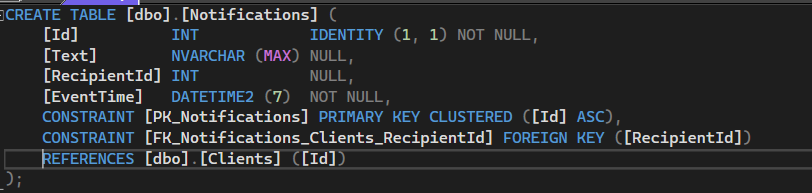


Рисунок 14.4. Скрипт создания таблицы «Notifications»

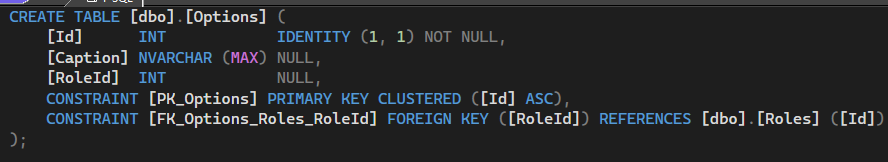


Рисунок 14.5. Скрипт создания таблицы «Options»

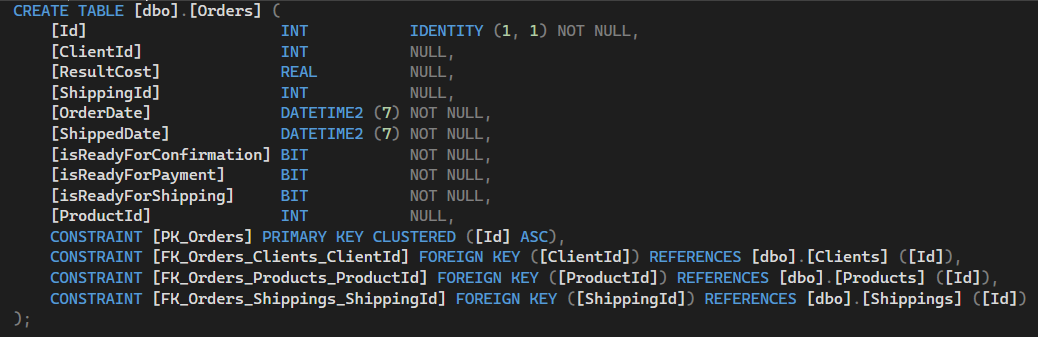


Рисунок 14.6. Скрипт создания таблицы «Orders»

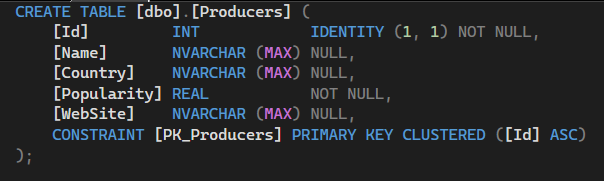


Рисунок 14.7. Скрипт создания таблицы «Producers»

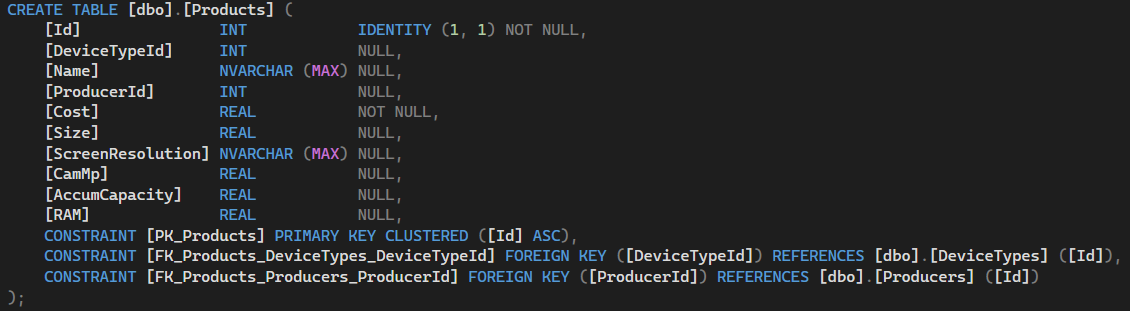


Рисунок 14.8. Скрипт создания таблицы «Products»

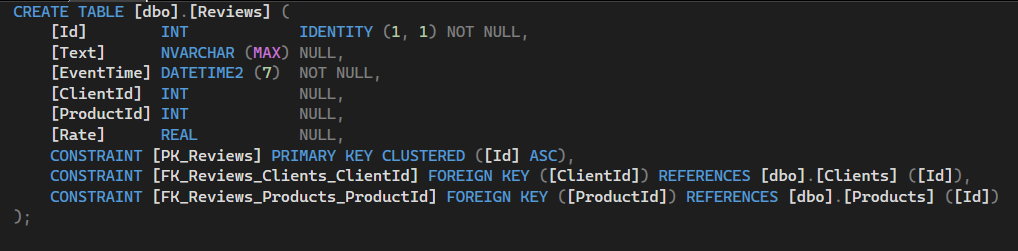


Рисунок 14.9. Скрипт создания таблицы «Reviews»

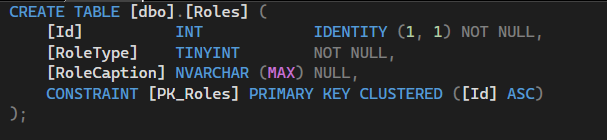


Рисунок 14.10. Скрипт создания таблицы «Roles»

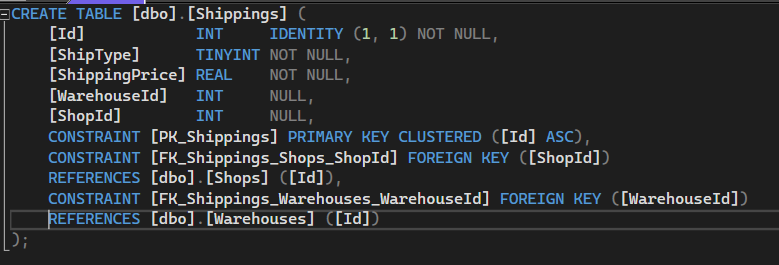


Рисунок 14.11. Скрипт создания таблицы «Shippings»

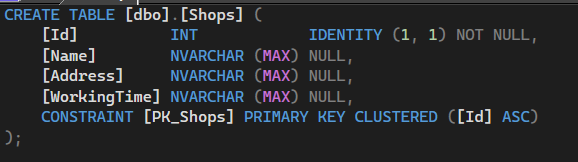


Рисунок 14.12. Скрипт создания таблицы «Shops»

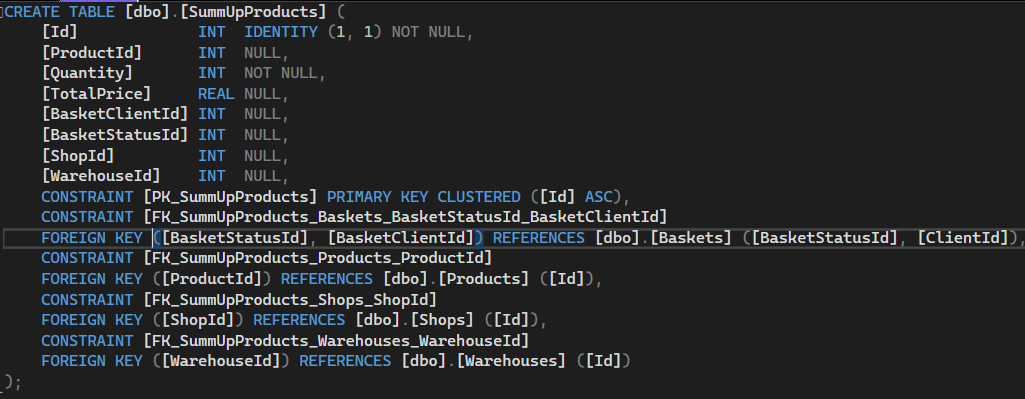


Рисунок 14.13. Скрипт создания таблицы «SummUpProducts»

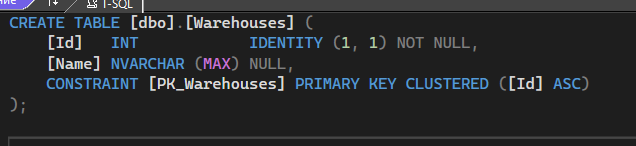


Рисунок 14.14. Скрипт создания таблицы «Warehouses»