Министерство образования Московской области

ГБПОУ МО «Колледж «Коломна»

09.02.07

КУРСОВОЙ ПРОЕКТ

ПМ. 02 Осуществление интеграции программных модулей

Тема: Разработка информационной системы для фитнес клуба

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ККОО.КП1223.000ПЗ

Студент:	Скорочкин Н.А.
Руководитель:	Карташова Е.В.
Нормоконтролер:	Грушникова Т.Н.
Дата защиты:	Оценка

2024

Министерство образования московской области
ГБПОУ МО «Колледж «Коломна»

ЗАДАНИЕ
на курсовой проект
ПМ.02 Осуществление интеграции программных модулей
студента очной формы обучения 4 курса группы 42112

Тема: Разработка информационной системы для фитнес клуба

Разработать информационную систему в соответствии с темой и приложение для работы с ней. Разработанная информационная система должна отражать основные бизнес-процессы в соответствии с техническим заданием на разработку. Для защиты информации необходимо разграничить права пользователей системы. Разработанные формы должны обеспечить возможность пользователям осуществлять работу с данными в соответствии с их правами.

При разработке информационной системы необходимо решить следующие задачи:

– сформулировать цель разработки информационной системы;

– описать предметную область, для которой разрабатывается информационная система

– провести анализ предметной области, выявить основные бизнес-процессы и пользователей, которые будут с ней работать.

– определить круг запросов и задач, которые предполагается решать с использованием созданной информационной системы;

– написать техническое задание на разработку информационной системы;

– провести функциональное моделирование и объектно-ориентированное проектирование информационной системы;

– провести проектирование интерфейса информационной системы:

– разработать модель данных и реализовать ее в выбранной СУБД

– разработать программное решение для информационной системы;

– оценить качество разработанной системы путем тестирования основного функционала.

Объем курсовой работы:

1) Пояснительная записка (25-50 листов печатного текста формата А4):

Введение

1. Разработка системного проекта

1.1 Описание предметной области

1.2 Анализ предметной области

1.3 Техническое задание

2 Проектирование информационной системы

2.1 Функциональное моделирование

2.2 Проектирование интерфейсов приложения

3 Разработка информационной системы

3.1 Модель данных

3.2 Структура проекта

4 Оценка качества программного обеспечения

Выводы и заключение

Список литературы

Приложение А – Код программных модулей

2) Презентация для защиты курсового проекта (10-15 слайдов)

Дата выдачи задания _____

Срок окончания _____

Зав структурным подразделением __ _____ Емельянова В. А.

Руководитель ___ _____ Карташова Е.В.

СОДЕРЖАНИЕ

Введение

1. Разработка системного проекта 6

1.1 Описание предметной области

1.2 Анализ предметной области 7

1.3 Техническое задание

2 Проектирование информационной системы 13

2.1 Функциональное моделирование

2.2 Проектирование интерфейсов приложения 18

3 Разработка информационной системы 23

3.1 Модель данных

3.2 Структура проекта 25

4 Оценка качества программного обеспечения 32

Выводы и заключение 36

Список литературы 37

Приложение А – Код программных модулей 38

Целью курсового проекта является.

При разработке информационной системы необходимо решить следующие задачи:

– сформулировать цель разработки информационной системы;

– описать предметную область, для которой разрабатывается информационная система

– провести анализ предметной области, выявить основные бизнес-процессы и пользователей, которые будут с ней работать.

– определить круг запросов и задач, которые предполагается решать с использованием созданной информационной системы;

– написать техническое задание на разработку информационной системы;

– провести функциональное моделирование и объектно-ориентированное проектирование информационной системы;

– провести проектирование интерфейса информационной системы:

– разработать модель данных и реализовать ее в выбранной СУБД

– разработать программное решение для информационной системы;

– оценить качество разработанной системы путем тестирования основного функционала.

1 РАЗРАБОТКА СИСТЕМНОГО ПРОЕКТА

(1.1) Описание предметной области

Продажа музыкальных инструментов — это процесс, направленный на предоставление широкого ассортимента инструментов для любителей и профессионалов, таких как гитары, пианино, барабаны, духовые и струнные инструменты, а также электронные музыкальные устройства. В современном мире, где музыка играет важную роль в жизни многих людей, продажа музыкальных инструментов становится не просто бизнесом, а важным элементом культурного и творческого развития. Покупка нового инструмента может быть значительным вложением, но это вложение окупается с лихвой, особенно если инструмент выбран правильно и соответствует потребностям музыканта.

Продажа музыкальных инструментов также играет важную роль в поддержке музыкального образования и развития талантов. Доступ к качественным инструментам позволяет начинающим музыкантам развивать свои навыки, а профессионалам — совершенствовать мастерство. Это способствует росту музыкальной культуры и созданию новых произведений, которые обогащают нашу жизнь. Кроме того, продажа музыкальных инструментов способствует сохранению традиций и культурного наследия. Многие инструменты имеют богатую историю и являются важной частью культурного наследия разных народов. Поддержка производства и продажи таких инструментов помогает сохранить эти традиции и передать их будущим поколениям.

1.2 Анализ предметной области

Цель информационной системы для продажи музыкальных инструментов – автоматизация деятельности магазина по продаже музыкальный инструментов и др.

Задачи данной информационной системы:

1. Автоматизация регистрации данных о клиенте. Необходимо, чтобы данные о клиенте сохранялись в нашей ИС. Эти данные используются для идентификации клиента.

2. Автоматизация регистрации заказов на клиентском сайте.

Необходимо хранить всю информацию о заказе клиента. Эта информация используется как для просмотра деталей заказа, так и для дальнейшего анализа текущего мониторинга продуктов и клиентов.

3. Автоматизация складского учета. Необходимо в ИС заносить факты принятых и выданных заказов, составлять отчеты по продажам.

4. Автоматизация бухгалтерского учета. Необходимо в ИС вести бухгалтерский учет согласно действующему законодательству.

Список пользователей ИС ремонта электронной техники:

– администратор;

1.3 Техническое задание

Введение

Интернет-магазин по продаже музыкальных инструментов

Настоящее техническое задание распространяется на разработку автоматизированной информационной системы «Продажа музыкальных инструментов», предназначенной для автоматизации учета заказов, клиентов и товаров на складе.

Целью создания автоматизированной информационной системы является сокращение времени на получение сведений о наличии товаров на складе, на формирование отчетности по поступлению и реализации товаров, учета остатков товара на складе и т.д. Система обеспечивает повышение эффективности процесса обработки и движения документов, снижает затраты на ведение документооборота, повышает точность и достоверность данных.

Работа выполняется в рамках проекта автоматизации управления торговым предприятием.

Основание для разработки

Основанием для разработки является договор № _____ от _____

Организация, утвердившая договор: _____

Наименование работы: Информационная система «Продажа музыкальных инструментов»

Назначение разработки.

Информационная система «Продажа музыкальных инструментов» предназначена для автоматизации учета движения и наличия товаров, ипользуемых в заказах и на складе. Пользователями системы являются менеджеры склада, специалисты отдела учета, сотрудники отдела приема и оформления документов. Система должна обеспечить учет товаров, поступивших на склад, а также фиксацию всех операций, связанных с их перемещением, продажам и возвратом клиентам. Пользователи системы:

1. Менеджеры склада — отвечают за приемку и выдачу товаров для ремонта, их хранение и учет.

2. Специалисты отдела учета — ведут учетные записи всех операций по поступлению, ремонту и выдаче товаров, осуществляют проверку данных.

3. Сотрудники отдела приема и оформления документов — занимаются оформлением первичных документов на прием товаров на склад. Основные функции системы:

1. Учет поступления товаров на склад в интернет-магазинр по продаже музыкальных инструментов:

– Оформление и учет поступления товаров на склад на основании документов (заявка на закупку, накладная, описание состояния).

– Регистрация условий приемки товаров (дата, документ, количество, описание неисправностей).

– Ведение журнала поступления товаров, содержащего следующие данные: название документа, его дату и номер, описание товара, количество товара на момент поступления.

2. Оформление и учет ремонта и обслуживания:

– Регистрация заявок на покупку.

– Оформление этапов продажи (начало принятия заказа, упаковка, доставка).

– Фиксация информации о выполненных заказах, использованных продуктов.

3. Учет выдачи заказов после продажи:

– Оформление документов на передачу заказа и заказов клиенту после продажи.

– Ведение журнала учета выдачи заказов, в котором фиксируются: номер по порядку, дата выдачи, наименование товаров, количество, фамилия и подпись лица, выдавшего заказ.

Требования к программе

3.1. Требования к функциональным характеристикам

Автоматизированная информационная система «продажа музыкальных инструментов» должна обеспечивать выполнение следующих функций:

– ввод, хранение, поиск и обработку информации о поступлении товаров для продажи и доставки товаров после заказа и упаковки;

– ведение журнала регистрации документов, связанных с приемом товаров на склад, оформлением заказа и доставки товаров;

– своевременное получение информации о наличии товаров на складе;

Нормативно-справочная информация автоматизированной информационной системы «продажи музыкальных инструментов» представлена справочниками:

а) клиентов (юридические и физические лица);

б) товаров, доступных для продажи продаже;

в) заказов;

Первичными документами для учета товаров в системе являются:

– приходные документы — акты приема товаров, содержащие дату поступления заказа, сведения о клиенте, описание заказанных товаров, количество единиц товара;

– документы на доставку — накладные на доставку товара клиенту, содержащие дату доставки заказа, перечень заказанных товаров и общую сумму услуг.

Выходными данными являются:

– отчет по поступлению товаров на склад за определенный период, содержащий сведения о клиентах, перечень поступивших товаров, количество;

– отчет по выполненным заказам за определенный период, включающий перечень заказанных товаров, затраты на доставку по каждому товару и общую сумму;

– отчет по остаткам товаров на складе, отражающий товары, находящиеся в продаже или ожидающие пополнения;

– оборотная ведомость за период (для каждого товара — остаток на начало периода, поступление, возврат и остаток на конец периода);

– рейтинг клиентов по количеству заказов и объему предоставленных услуг;

– рейтинг товаров по частоте заказов;

Все отчеты должны формироваться на даты, заданные пользователем, и содержать итоговые значения. В приложении к техническому заданию требуется включить образцы всех первичных документов и образцы всех отчетов.

В программе необходимо предусмотреть:

– возможность резервного сохранения данных;

– наличие встроенных подсказок;

– возможность быстрого поиска необходимых документов и справочной информации.

3.2. Требования к надежности

Программное обеспечение должно соответствовать требованиям надежности и безопасности, включая:

– возможность восстановления после сбоев (отключение питания, сбои в операционной системе и т.д.);

– возможность резервного копирования информационной базы;

– разграничение пользовательских прав;

– предотвращение несанкционированного копирования программы.

Необходимо предусмотреть контроль вводимой информации, обработку или блокировку некорректных действий пользователя. Все поля первичных документов должны быть заполнены.

3.3. Требования к составу и параметрам технических средств

Минимальные системные требования для работы программного продукта:

- тактовая частота процессора – 1.4 ГГЦ
- объем оперативной памяти – 4 Гб
- объем свободного дискового пространства – 100 Мб
- разрешение монитора – 640 x 480

3.4. Требования к информационной и программной совместимости

Программа должна работать в операционных системах Windows... Все формируемые отчеты должны иметь возможность экспортирования в редактор электронных таблиц MS Office Excel.

3.5. Требования к транспортировке и хранению

Установка программы не обязаьтельна, так как оно является веб-приложение.

Программная документация также доступна на сайте разработчика в электронном виде.

3.6. Специальные требования

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя средней квалификации в области компьютерной грамотности.

Проект будет выполняться поэтапно, с совместимостью модулей, разработанных в разное время.

4. Требования к программной документации

В ходе разработки программы должна быть подготовлена следующая документация:

- текст программы;
- описание программы;

– программа и методика испытаний;

– руководство пользователя.

5. Технико-экономическое обоснование

Использование автоматизированной системы «продажа музыкальных инструментов» позволит сократить время на выполнение операций по регистрации поступления товаров на склад, оформлением заказа и доставке клиентам. Ожидается значительное уменьшение времени на подготовку отчетов, проверку данных и уменьшение ошибок, связанных с ручным вводом информации. Экономический эффект будет достигнут за счет повышения скорости и точности обработки данных, сокращения ручного труда, а также улучшения контроля за складскими операциями и бизнес-процессами.

2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

2.1 Функциональное моделирование

Функциональное моделирование диаграммы — это графическое представление функций предприятия в определённой области. На рисунке 1 представлена диаграмма IDEF0.

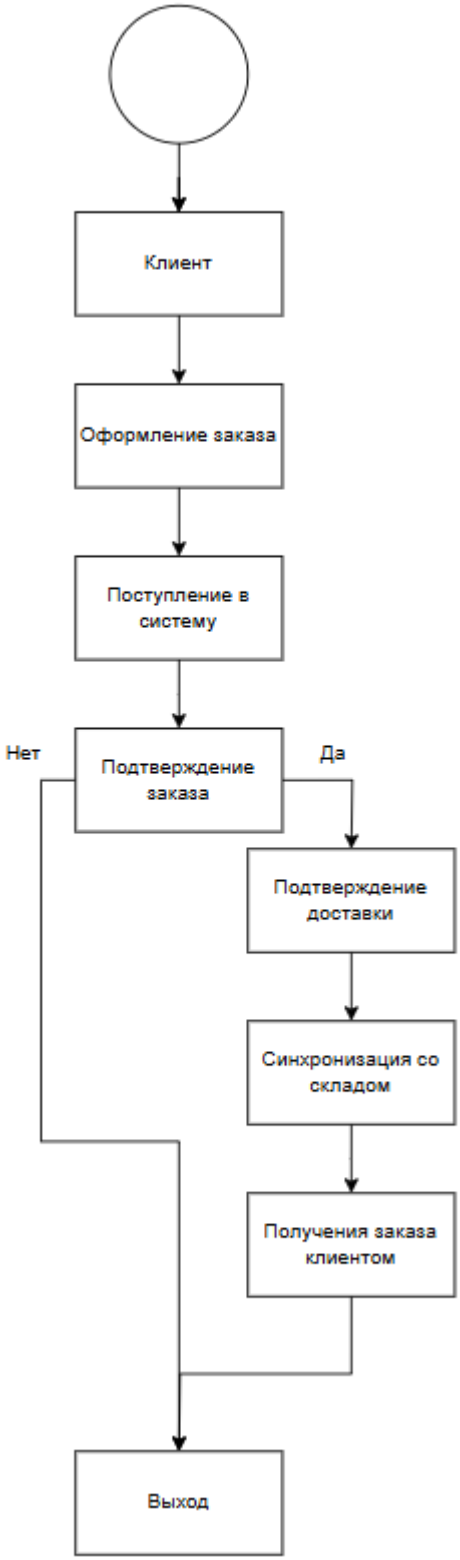


Рисунок 1 – Диаграмма IDEF0

На рисунке 2 представлена диаграмма DFD.

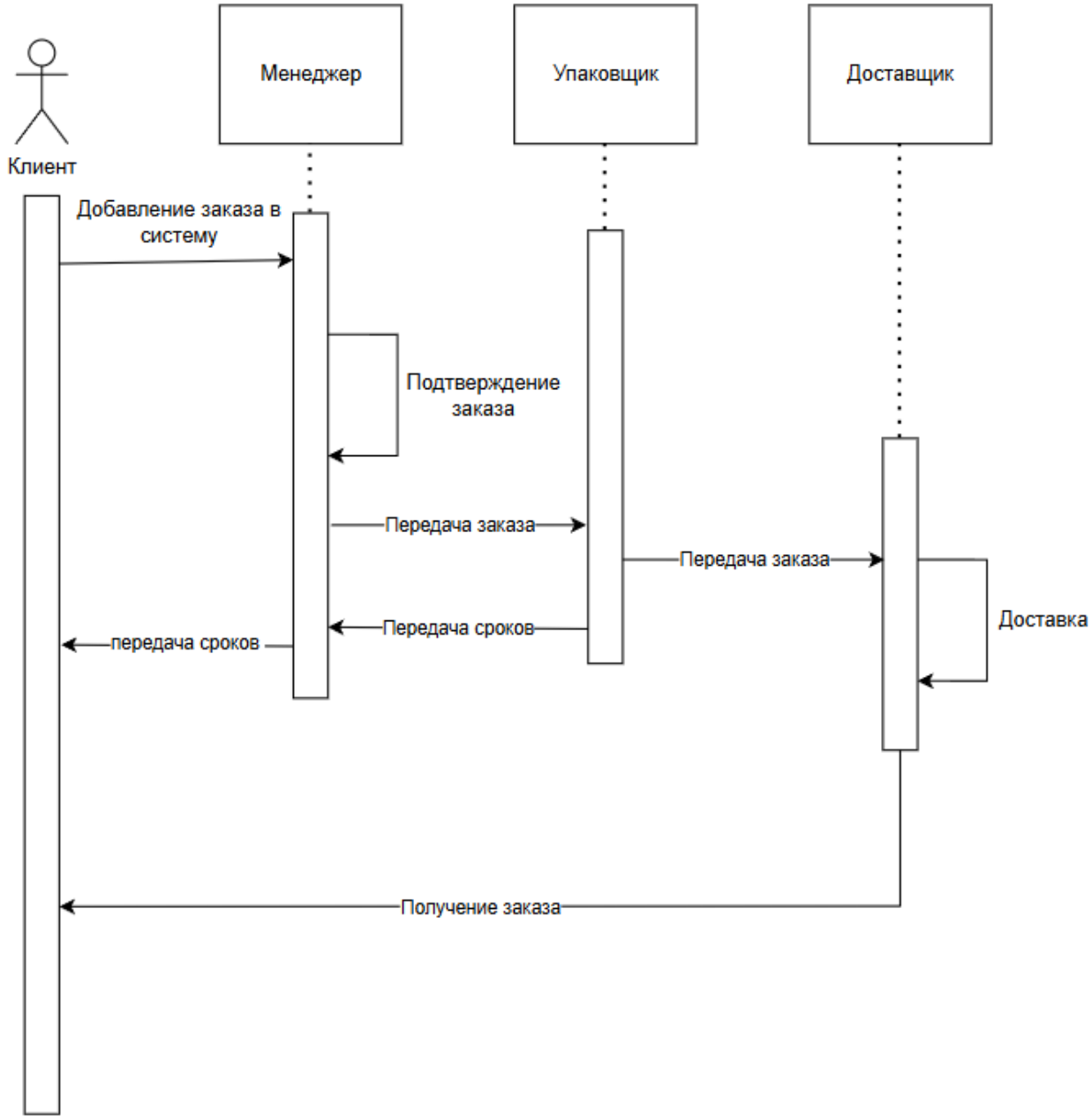


Рисунок 2 - Диаграмма DFD

На рисунке 3 представлена диаграмма деятельности.

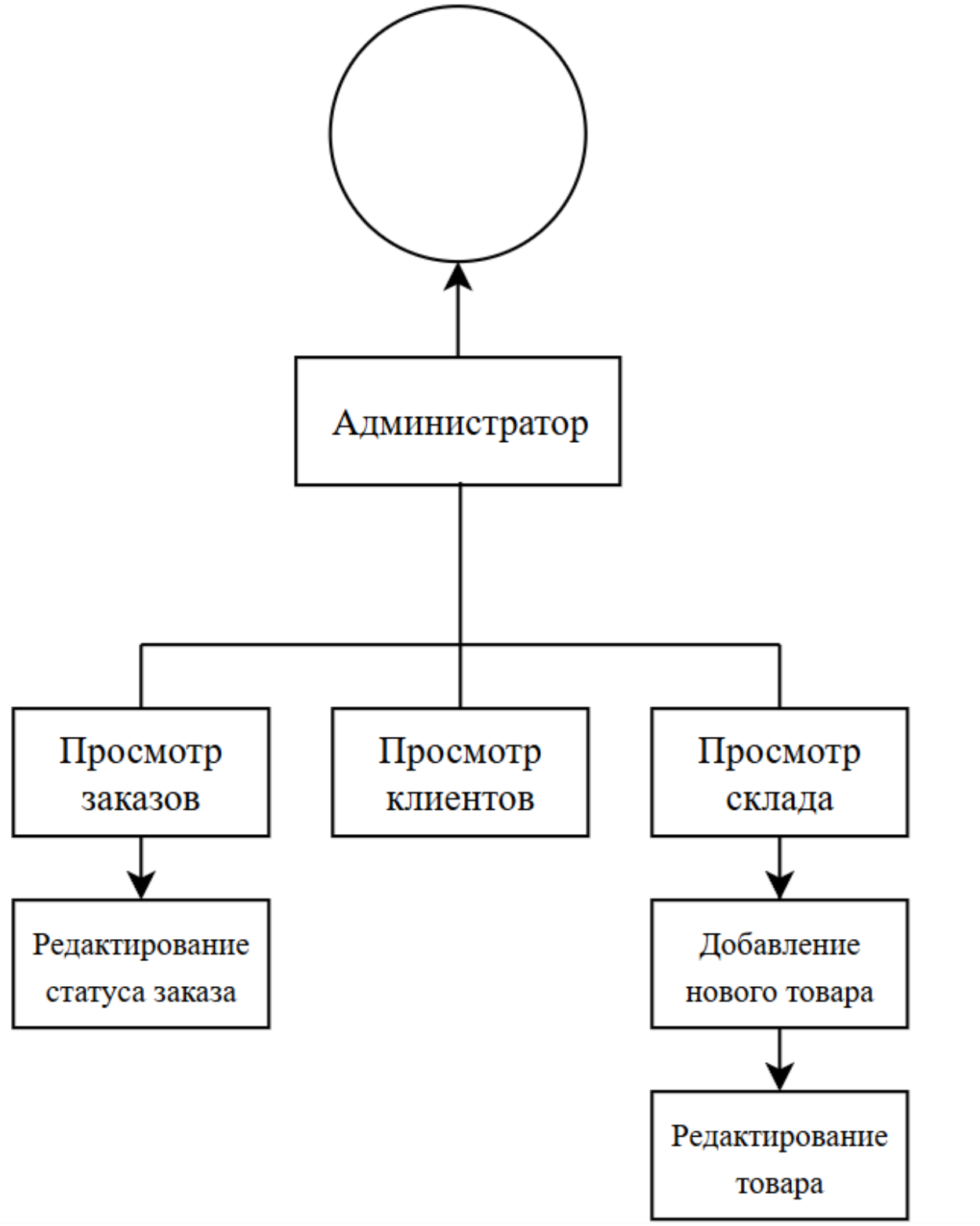


Рисунок 3 - диаграмма деятельности

На рисунке 4 представлена диаграмма последовательности.

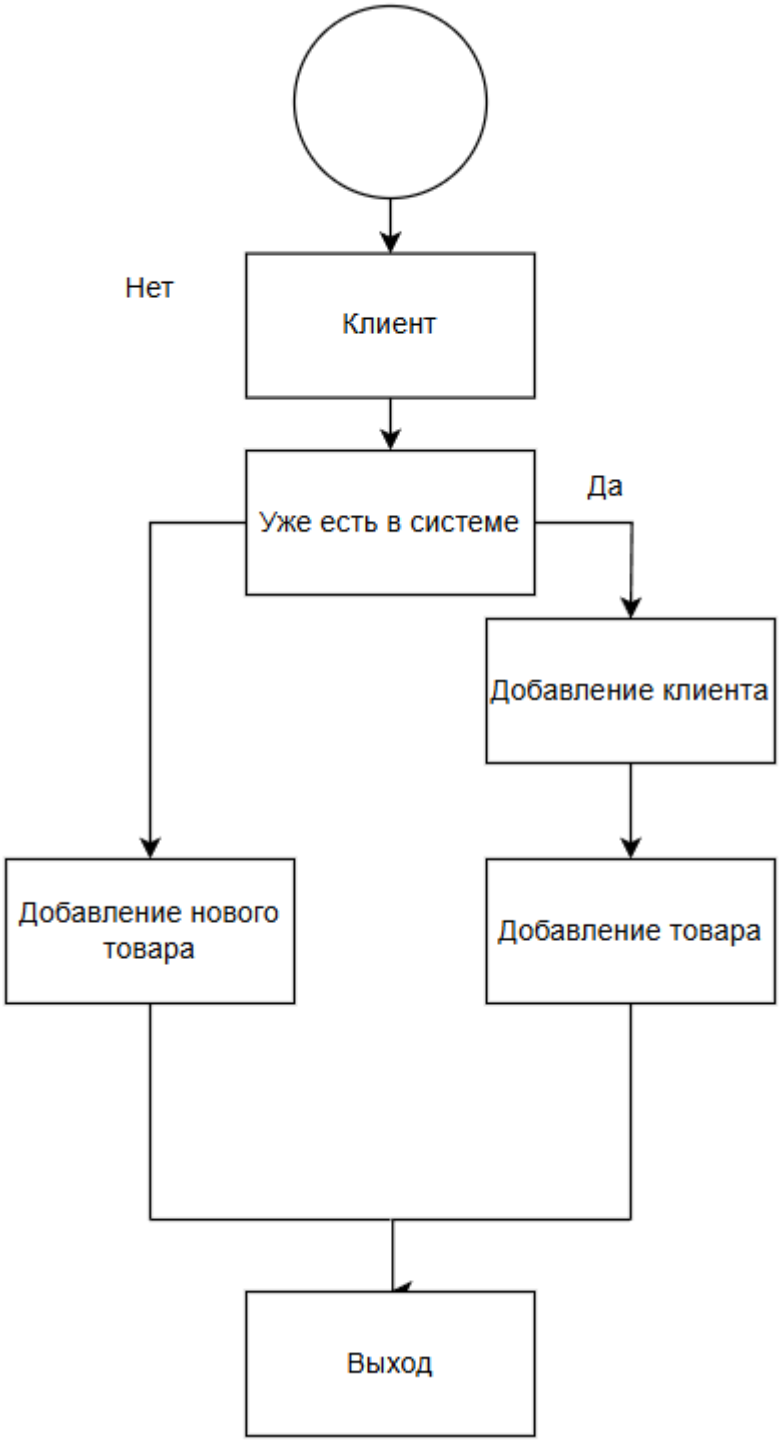


Рисунок 4 - Диаграмма последовательности

2.2 Проектирование интерфейсов приложения

На странице /home (Рисунок 5) находятся следующие элементы:

– LeftMenu – навигационное меню;
– TitleButton – кнопка перехода на страницу /home;
– HomeButton – кнопка перехода на страницу /home
– OrdersButton – кнопка перехода на страницу /orders;
– ClientsButton – кнопка перехода на страницу /clients
– InventoryButton – кнопка перехода на страницу /products
– MapButton – кнопка перехода на страницу /map
– CollapseMenu – кнопка скрывающие меню
– ColorMode – кнопка меняющая оформление страницы

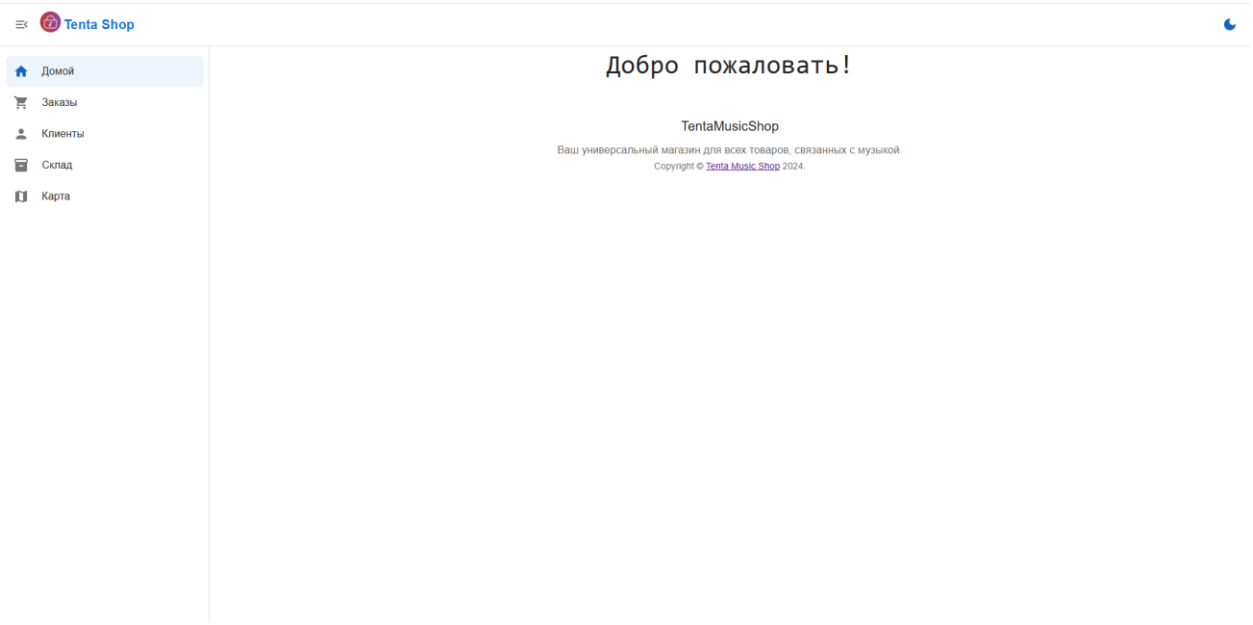


Рисунок 5 – Страница /home или /

На странице /orders (Рисунок 6) находятся следующие элементы:

– OrdersGrid – разметка для вывода текущих заказов
– InfoButton – кнопка для открытия модального окна с информацией
– CompleteButton – кнопка для подтверждения заказа

– CancelButton – кнопка для отмены заказа

– DeliveryButton – кнопка для доставки заказа

– Pagination – элемент отвечающий за пагинацию

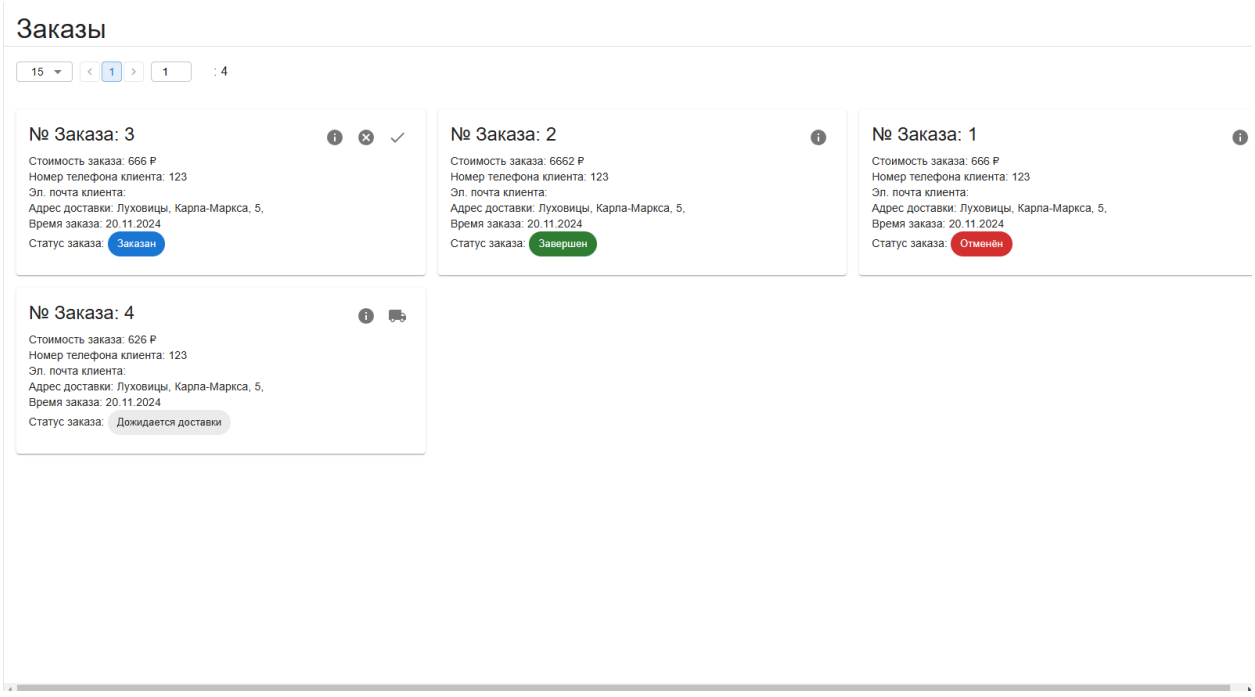


Рисунок 6 – Страница /orders

Рисунок 7 – Модальное окно заказа

На странице /products (Рисунок 8) находятся следующие элементы:

– AddButton – кнопка для добавления нового продукта

– EditButton – кнопка для редактирования текущего продукта

– ProductsGrid – сетка для вывода карточек продуктов

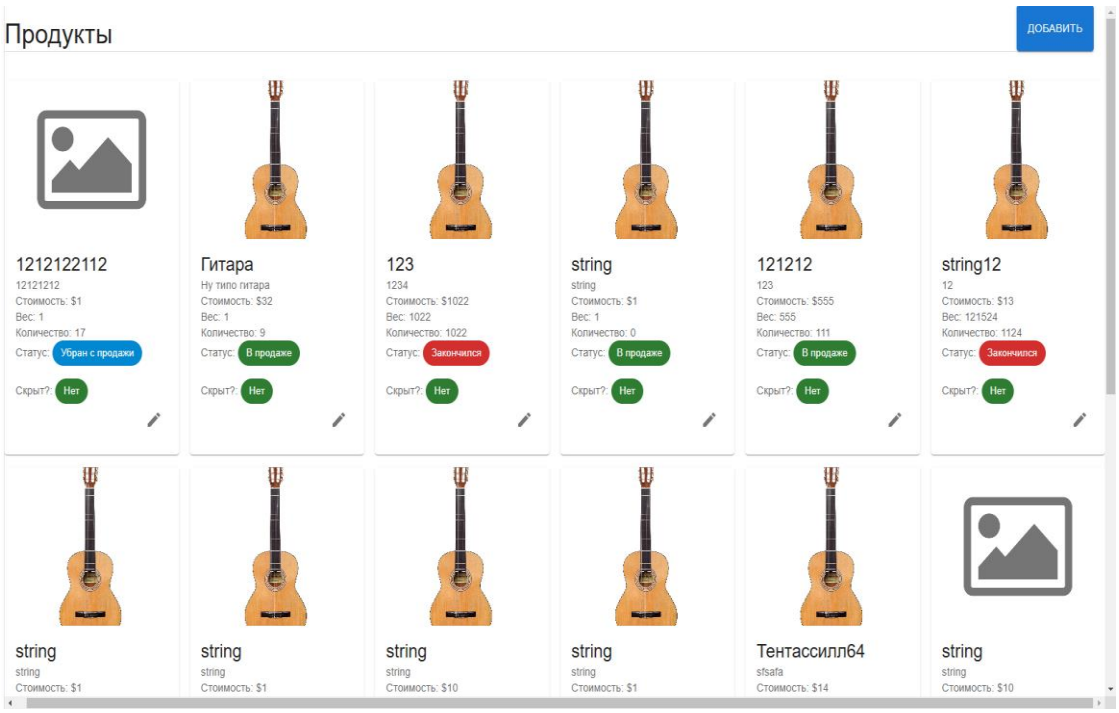


Рисунок 8 – Страница /products

На странице /products/add (Рисунок 9) находятся следующие элементы:

– ProductNameInput – текстовое поле для ввода наименования товара

– ProductDescInput – текстовое поле для ввода описания товара

– ManufacturerInput – текстовое поле для ввода производителя

– CategorySelect – выпадающий список с категориями

– ProductPriceInput – текстовое поле для ввода цены товара

– WeightInput – текстовое поле для ввода веса товара

– StateSelect – выпадающий список с состоянием товара

– CountInput – текстовое поле для ввода количество товара

– ImageList – текстовое поле с кнопкой для добавления фотографий товару

– SaveButton – кнопка для сохранения товара

– ReturnButton – кнопка для возврата назад

Рисунок 9 – Страница /products/add

На странице /clients (Рисунок 10) находятся следующие элементы:

– ClientsTable – таблица с клиентами

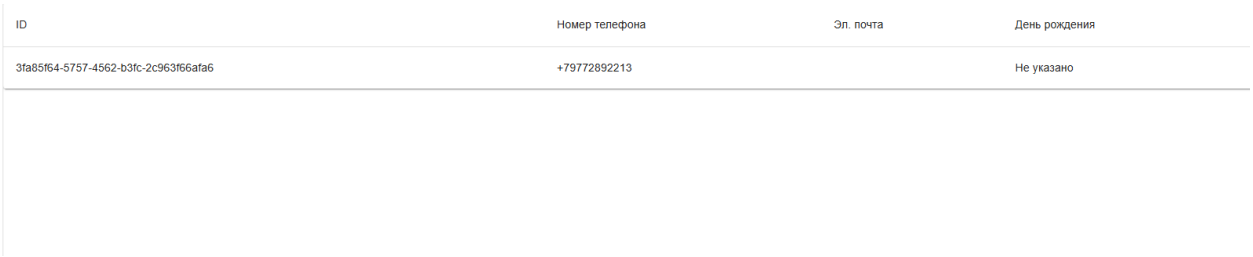


Рисунок 10 – Страница /clients

На странице /map (Рисунок 11) находятся следующие элементы:

– IFrameMap – Карта мира с точкой нашего магазина

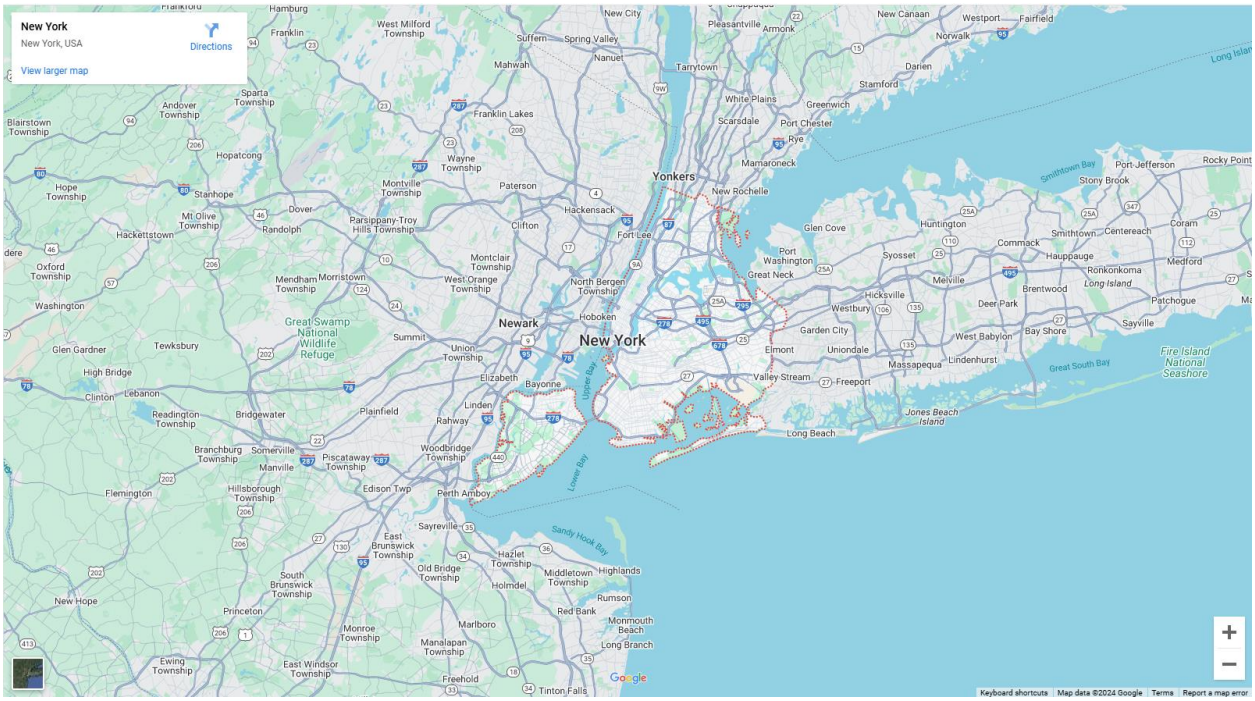


Рисунок 11 – Страница /map

3 РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

3.1 Модель данных

Модель данных реализована в СУБД DBeaver.

В результате разработки базы данных были спроектированы следующие таблицы:

1) Products (id, name, description, price, weight, manufacturer, quantity, images, status, createddatetimeutc, modifieddatetimeutc, isremoved, ishidden, categoryid, createduserid, modifieduserid);

2) Categories (id, name, isremoved, createddatetimeutc, modifieddatetimeutc, createduserid, modifieduserid);

3) Address (id, city, street, home, apartment, createddatetimeutc, modifieddatetimeutc, isremoved);

4) OrderItems (id, orderid, productid, productprice, isremoved, createddatetime, modifieddatetime, productcategoryid);

5) Users (id, phonenumber, passwordhash, email, createddatetimeutc, modifieddatetime, isremoved, birthdate)

6) Orders (id, price, clientphonenumber, clientid, addressid, createddatetimeutc, modifieddatetimeutc, completeddatetimeutc, state, isremoved, ordernumber)

На Рисунке 12 представлена база данных, на Рисунках 13 - 18 показаны таблицы.

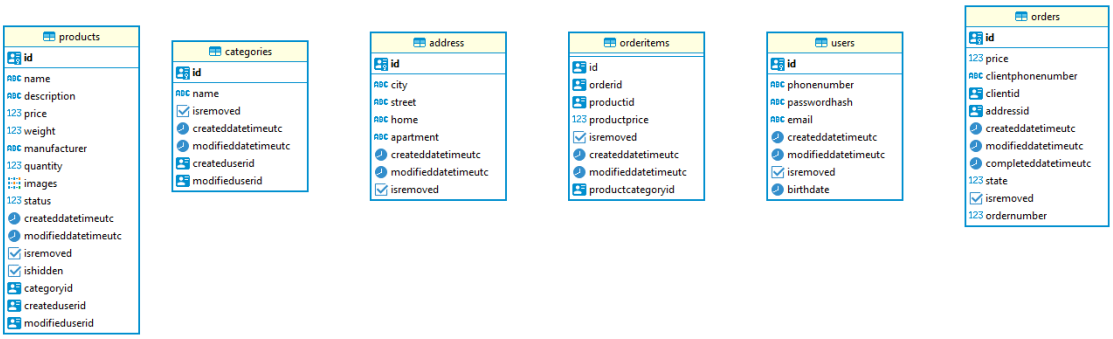


Рисунок 12 – База данных MusicShop

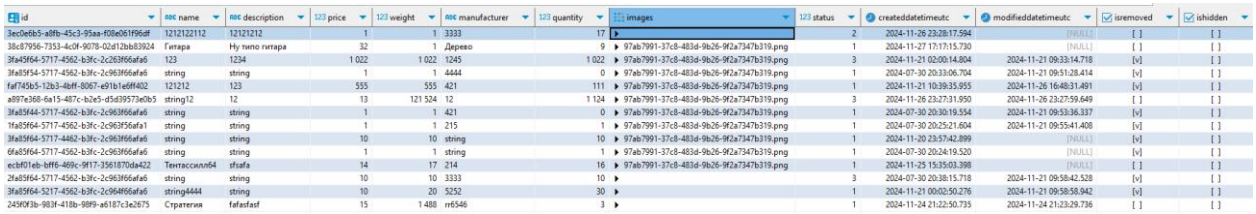


Рисунок 13 – Таблица Products

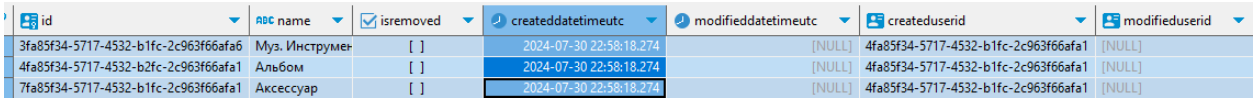


Рисунок 14 - Таблица Categories

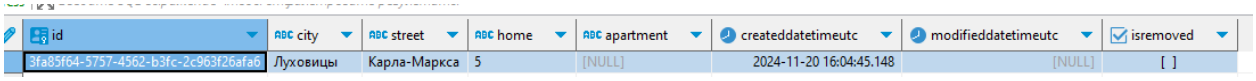


Рисунок 15 – Таблица Address

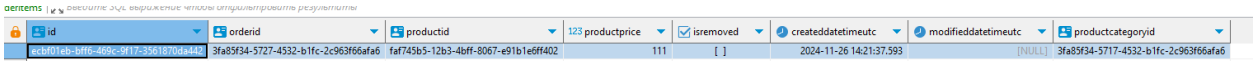
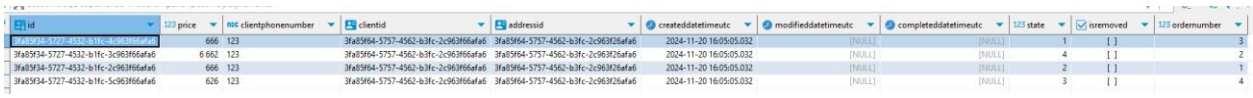


Рисунок 16 – Таблица OrderItems

Рисунок 17 – Таблица Users



id	price	clientphonenumber	clientid	addressid	createddatetimeutc	modifieddatetimeutc	completeddatetimeutc	state	isremoved	ordernumber
3fa85f34-5727-4532-b1fc-4c963f66afa6	666	123	3fa85f64-5757-4562-b3fc-2c963f66afa6	3fa85f64-5757-4562-b3fc-2c963f26afa6	2024-11-20 16:05:05.032	[NULL]	[NULL]	1	[ ]	3
3fa85f34-5727-4532-b1fc-3c963f66afa6	6 662	123	3fa85f64-5757-4562-b3fc-2c963f66afa6	3fa85f64-5757-4562-b3fc-2c963f26afa6	2024-11-20 16:05:05.032	[NULL]	[NULL]	4	[ ]	2
3fa85f34-5727-4532-b1fc-2c963f66afa6	666	123	3fa85f64-5757-4562-b3fc-2c963f66afa6	3fa85f64-5757-4562-b3fc-2c963f26afa6	2024-11-20 16:05:05.032	[NULL]	[NULL]	2	[ ]	1
3fa85f34-5727-4532-b1fc-5c963f66afa6	626	123	3fa85f64-5757-4562-b3fc-2c963f66afa6	3fa85f64-5757-4562-b3fc-2c963f26afa6	2024-11-20 16:05:05.032	[NULL]	[NULL]	3	[ ]	4

Рисунок 18 – Таблица Orders

3.2 Структура проекта

В процессе разработки программного решения были спроектированы следующие вещи:

1) Страницы:
   - /home;
   - /orders;
   - /clients;
   - /proudcts;
   - /products/add;
   - /products/edit/:id.
   - /map;
2) Элементы базы данных
   - Users;
   - Address;
   - Products;
   - Orders;
   - OrderItems.
   - Categories

На Рисунках 19-20 представлена вся структура программного решения.

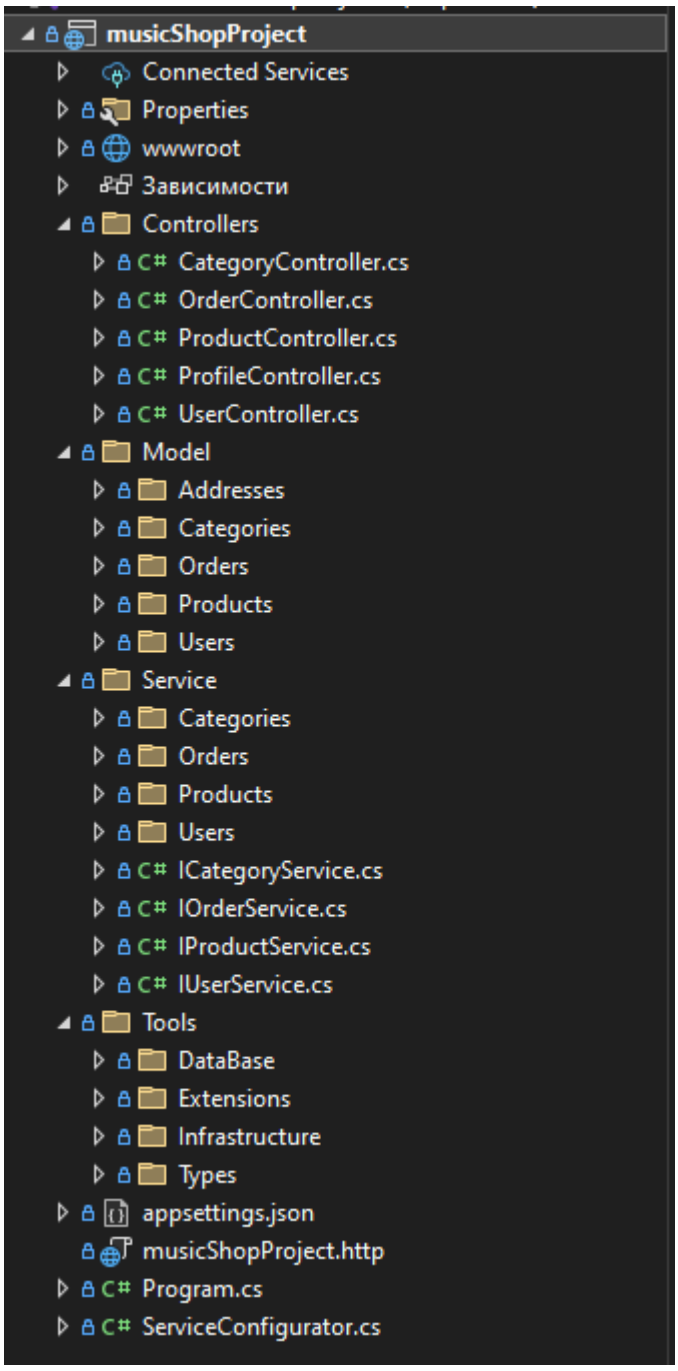


Рисунок 19 – Структура проекта C#

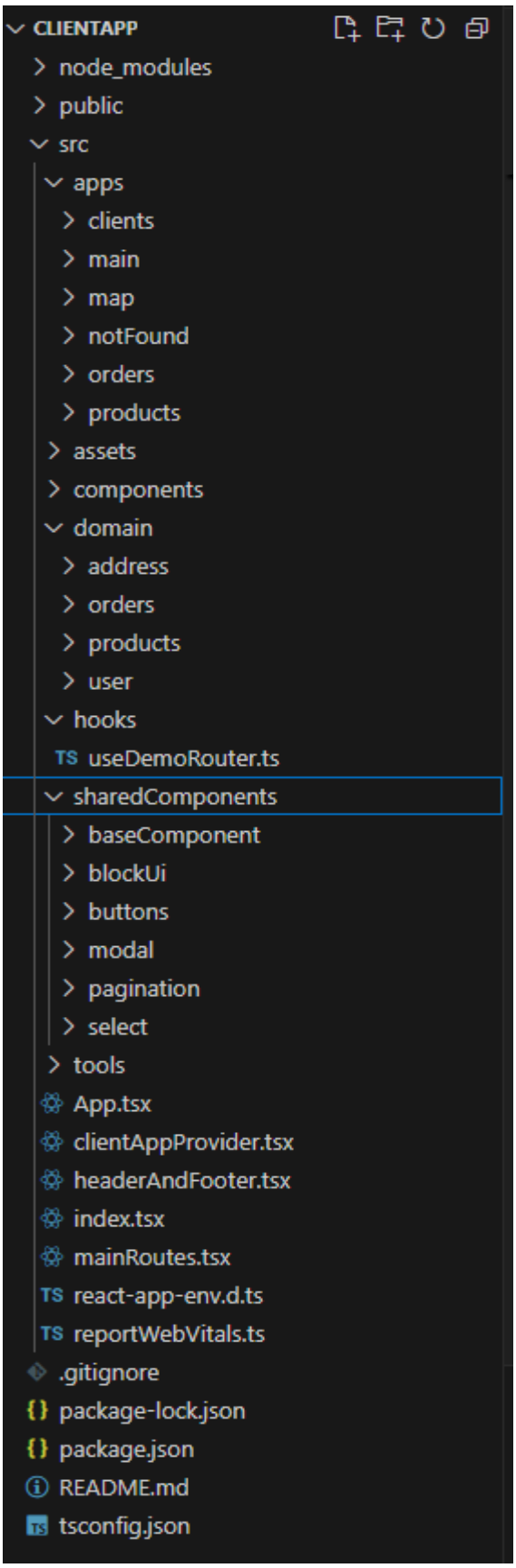


Рисунок 20 – Структура React

Класс OrderController.cs имеет следующие методы (Рисунок 21):

– private readonly _orderSerivce – поле дающие доступ к сервисе заказов;

– public ChangeOrderState – метод обращающийся к сервису заказов меняет статус заказа;

– public GetOrderPage – метод возвращающий страницу заказов

– public OrderController()– конструктор класса.

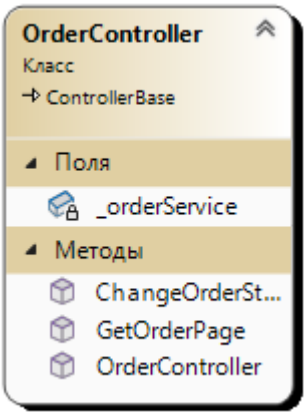


Рисунок 21 – Структура класса OrderController.cs

Класс CategoryController.cs имеет следующие методы (Рисунок 22):

– private readonly _categoryService – доступ к сервису;

– public AddCategory – добавление новой категории;

– public GetAllCategories – метод возвращающий все категории;

– public CategoryController() – конструктор класса;

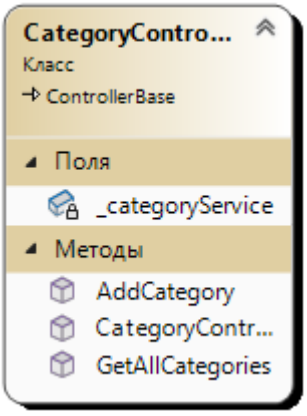


Рисунок 22 – Структура класса CategoryController

Класс ProductController.cs имеет следующие методы (Рисунок 23):

– private readonly _productService – доступ к сервису;
– public SaveProduct – метод сохраняющий продукт;
– public GetProducts – метод возвращающий массив продуктов;
– public GetProduct – метод возвращающий продукт по его id;
– public ProductController() – конструктор класса;

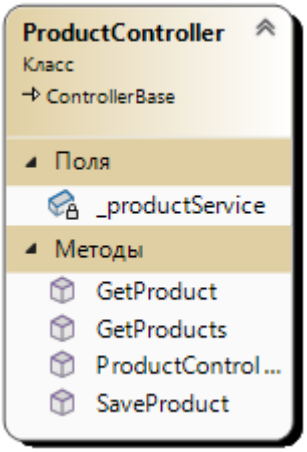


Рисунок 23 – Структура класса ProductController

Класс UserController.cs имеет следующие методы (Рисунок 24):

– private readonly _userService – доступ к сервису;

– public Register – метод регистрации;

– public Login – метод логина;

– public GetAllUsers – метод получения всех пользователей;

– public GetUser – метод получения пользователя по номеру телефона;

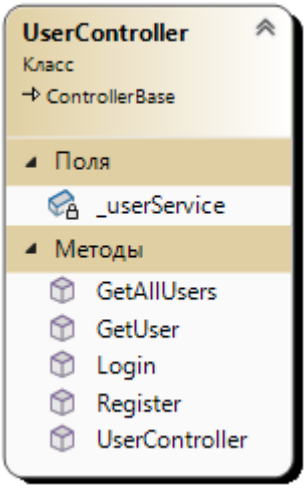


Рисунок 24 – Структура класса UserController

Класс MainConnector.cs имеет следующие методы (Рисунок 25):

– private _connectionString – строка подключения;

– public void ExecuteNonQuery – метод выполняющий запрос без возвращаемого параметра;

– public GetPage – метод возвращающий страницу элементов;

– public Get – метод возвращающий один элемент;

– public GetArray – метод возвращающий массив элементов;

– public GetAllArray – метод возвращающий все элементы таблицы в виде массива;

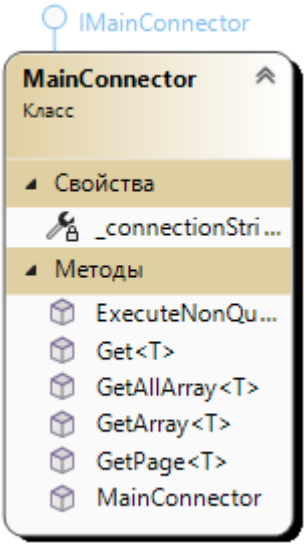


Рисунок 25 – Структура класса MainConnector

Класс OrderService.cs имеет следующие методы (Рисунок 26):

– private readonly _orderRepository – доступ к репозиторию;
– private readonly _productService – доступ к сервису продуктов;
– private readonly _userService – доступ к сервису пользователей;
– public ChangeOrderState – метод меняющий статус заказа;
– public GetOrderPage – метод возвращающий страницу;
– public OrderService() – конструктор класса;

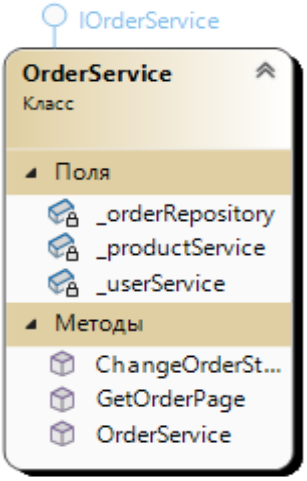


Рисунок 26 - Структура класса OrderService

4 ОЦЕНКА КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Сценарий тестирования представлен в таблице 1.

Таблица 1. Сценарий тестирования

Test case #	Название теста	Резюме испытания (краткое описание)
TC_UI_1	Проверка добавления продукта с неуказанной категорией	1. Проверка добавлени нового продукта с указанием категории. 2. Проверка невозможности создания продукта с не указанием категории.
TC_UI_2	Проверка редактирования уже созданных товаров	1. Проверка изменения данных с теми же данными; 2. Проверка изменения с новыми данными;
TC_UI_3	Проверка добавления продукта	1. Проверка добавления с правильным вводом 2. Проверка добавления с незаполненными полями.

Тест – кейсы представлены в таблицах 2 – 4.

Таблица 2. Тест-кейс #1

Элементы тест-кейса	Значения
Название проекта	Информационная система «продажа музыкальных инструментов»
Номер версии	V 1.0.0.1
Имя тестера	Михеев Владислав Артемович
Даты тестирования	10.10 -11.11
Test Case #	TC_UI_1
Приоритет теста	Высокий
Название теста	Проверка добавления продукта с неуказанной категорией
Резюме испытания	1. Проверка создания продукта с указанием категории; 2. Проверка невозможности создать продукт с не указанием категории товара;
Шаги тестирования	1. Заполнить форму продукта и выбрать категорию товара. 2. Заполнить форму продукта, но не выбрать категорию товара;
Данные тестирования	1. Взять за основу продукт Гитара 2. Взять за основу продукт Гитара
Ожидаемый результат	1. При нажатии кнопки добавить: Успешно сохранено, переброс на страницу продуктов; 2. При нажатии кнопки добавить: Безуспешно, вылезло вспомогательное окно о незаполненности поля категории;
Фактический результат	1. При нажатии кнопки добавить: Успешно сохранено, переброс на страницу продуктов; 2. При нажатии кнопки добавить: Безуспешно, вылезло вспомогательное окно о незаполненности поля категории;
Предпосылки	Запуск приложения и переход на форму добавления
Постусловия	Подключение к базе данных MusicShop
Статус	-
Комментарии	-

Таблица 3. Тест-кейс #2

Элементы тест-кейса	Значения
Название проекта	Информационная система «продажа музыкальных инструментов»
Номер версии	V 1.0.0.1
Имя тестера	Михеев Владислав Артемович
Даты тестирования	10.10 -11.11
Test Case #	TC_UI_2
Приоритет теста	Высокий
Название теста	Проверка редактирования уже созданных товаров
Резюме испытания	1. Проверка изменения данных с не выданным товаром; 2. Проверка невозможности изменения данных в уже выданным товаром;
Шаги тестирования	1. Открыть приложение и перейти в раздел редактирования товаров. 2. Выбрать товар, который уже существует в базе данных. 3. Ввести те же данные, что и были ранее, и сохранить изменения. 4. Проверить, что данные не изменились в базе данных. 5. Ввести новые данные для того же товара и сохранить изменения. 6. Проверить, что данные изменились в базе данных.
Данные тестирования	1. Все поля заполнены теми же данными, что и были ранее. 2. Все поля заполнены новыми данными
Ожидаемый результат	1. Изменения не сохранятся, так как данные остались теми же. 2. Изменения сохранятся в базе данных.
Фактический результат	1. Изменения не сохранились, так как данные остались теми же. 2. Изменения сохранились в базе данных;
Предпосылки	Запуск приложение и появление страницы информации о товаре.
Постусловия	Подключение к базе данных MusicShop
Статус	-
Комментарии	-

Таблица 4. Тест-кейс #3

Элементы тест-кейса	Значения
Название проекта	Информационная система «продажа музыкальных инструментов»
Номер версии	V 1.0.0.1
Имя тестера	Михеев Владислав Артемович
Даты тестирования	10.10 -11.11
Test Case #	TC_UI_3
Приоритет теста	Высокий
Название теста	Проверка добавления клиента и продукта
Резюме испытания	1. Проверка добавления с правильным вводом; 2. Проверка добавления с незаполненными полями;
Шаги тестирования	1. Проверка добавления с правильным вводом; 2. Проверка добавления с незаполненными полями;
Данные тестирования	1. Все поля заполнены и продукт добавился в базу данных; 2. Одно из полей не было заполнено;
Ожидаемый результат	1. Запись о продукте сохранится; 2. Запись о пациенте не сохранится и появится всплывающие окно об ошибки;
Фактический результат	1. Запись о продукте сохранилась; 2. Запись о клиенте не сохранилась и вывелась ошибка;
Предпосылки	Подключение к базе данных MusicShop
Постусловия	-
Статус	-
Комментарии	-

ВЫВОДЫ И ЗАКЛЮЧЕНИЕ

В процессе создания курсового проекта, были достигнуты все выше поставленные цели:

– Создание программного обеспечения, которое позволяет эффективно управлять процессом ремонта электронной техники.

– Оптимизация рабочих процессов для сокращения времени ремонта и уменьшения количества ошибок.

– Реализация механизмов обратной связи для клиентов.

Для этого были выполнены следующие задачи:

– сформулирована цель разработки информационной системы;

– описана предметная область, для которой разрабатывается информационная система

– проведён анализ предметной области, выявлены основные бизнес-процессы и пользователи, которые будут с ней работать.

– определён круг запросов и задач, которые предполагается решать с использованием созданной информационной системы;

– написано техническое задание на разработку информационной системы;

– проведено функциональное моделирование и объектно-ориентированное проектирование информационной системы;

– проведено проектирование интерфейса информационной системы:

– разработана модель данных и реализована в выбранной СУБД

– разработано программное решение для информационной системы;

– оценено качество разработанной системы путем тестирования основного функционала.

СПИСОК ЛИТЕРАТУРЫ

1) Абрамян А. В. Разработка пользовательского интерфейса на основе технологии Windows Presentation Foundation : учебник для студ. Учреждений сред. Проф. Образования. - М: Издательство Южного федерального университета, 2019.

2) Ганенко А. П. Оформление текстовых и графических материалов при подготовке дипломных проектов, курсовых и письменных экзаменационных работ (требования ЕСКД) : учебно-метод. Пособие для студ. Учреждения сред. Проф. Образования – М: Издательский центр «Академия», 2020.

Электронные ресурсы:

3) Metanit – язык программирования C# и платформа .NET [Электронный ресурс] Режим доступа: свободный https://metanit.com/sharp/ (Дата обращения 03.11.2024)

4) Национальный открытый университет [Электронный ресурс] Режим доступа: свободный https://www.intuit.ru/ (Дата обращения 03.11.2024)

5) Professor Web [Электронный ресурс] Режим доступа: свободный https://professorweb.ru/ (Дата обращения 03.11.2024)

Приложение А – Код программных модулей

CategoryController.cs:

```
using Microsoft.AspNetCore.Mvc;
using musicShopProject.Model.Categories;
using musicShopProject.Service;
using musicShopProject.Tools.Types;

namespace musicShopProject.Controllers;

public class CategoryController : ControllerBase
{
    private readonly ICategoryService _categoryService;
    public CategoryController(ICategoryService categoryService)
    {
        _categoryService = categoryService;
    }

    [HttpPost("category/save")]
    public Result AddCategory([FromBody] CategoryBlank categoryBlank, Guid
requestedUserId)
    {
        return _categoryService.AddCategory(categoryBlank, requestedUserId);
    }

    [HttpGet("Category/Get/All")]
    public Category[] GetAllCategories()
    {
        return _categoryService.GetAllCategories();
    }
}
```

OrderController.cs:

```
using Microsoft.AspNetCore.Mvc;
using musicShopProject.Model.Orders;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Service;
using musicShopProject.Service.Orders;
using musicShopProject.Tools.Types;

namespace musicShopProject.Controllers
{
    public class OrderController : ControllerBase
    {
        private readonly IOrderService _orderService;
        public OrderController(IOrderService orderService)
        {
            _orderService = orderService;
        }

        [HttpGet("Get/Page")]
        public PagedResult<Order> GetOrderPage([FromQuery] Int32 page, Int32
pageSize)
        {
            return _orderService.GetOrderPage(page, pageSize);
        }
```

```
        public record ChangeOrderStateRequest(OrderState State, Guid OrderId);
        [HttpPost("Order/Change/State")]
        public Result ChangeOrderState([FromBody] ChangeOrderStateRequest
changeOrderStateRequest)
        {
            return _orderService.ChangeOrderState(changeOrderStateRequest.State,
changeOrderStateRequest.OrderId);
        }
    }
}
```

ProductController.cs:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using musicShopProject.Model.Categories;
using musicShopProject.Model.Products;
using musicShopProject.Service;
using musicShopProject.Tools.Types;

namespace musicShopProject.Controllers;

public class ProductController : ControllerBase
{
    private readonly IProductService _productService;
    public ProductController(IProductService productService)
    {
        _productService = productService;
    }

    [HttpPost("Product/Save")]
    public Result SaveProduct([FromBody] ProductBlank blank)
    {
        return _productService.AddProduct(blank, Guid.NewGuid());
    }

    [HttpGet("Products/Get/All")]
    public Product[] GetProducts([FromQuery] Guid? categoryId)
    {
        return _productService.GetProducts(categoryId);
    }

    [HttpGet("Product/Get")]
    public Product? GetProduct([FromQuery] Guid productId)
    {
        return _productService?.GetProduct(productId);
    }
}
```

UserController.cs:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using musicShopProject.Model.Users;
using musicShopProject.Service;
using musicShopProject.Tools.Types;

namespace musicShopProject.Controllers;

public class UserController : ControllerBase
{
    private readonly IUserService _userService;

    public UserController(IUserService userService)
    {
```

```
        _userService = userService;
    }

    [HttpPost("register")]
    public Result Register([FromBody] UserBlank blank)
    {
       return _userService.Register(blank);
    }

    public record LoginRequest(String? PhoneNumber, String? Password);

    [HttpPost("login")]
    public Result Login([FromBody] LoginRequest loginRequest)
    {
        return _userService.Login(loginRequest.PhoneNumber, loginRequest.Password);
    }

    [AllowAnonymous]
    [HttpGet("Users/Get/All")]
    public User[] GetAllUsers()
    {
        return _userService.GetAllUsers();
    }

    [HttpGet("user/phone")]
    public User? GetUser([FromQuery] String phoneNumber)
    {
       return _userService.GetUser(phoneNumber);
    }
}
```

Address.cs:

```
namespace musicShopProject.Model.Addresses
{
    public class Address
    {
        public Guid Id { get; }
        public String City { get; }
        public String Street { get; }
        public String Home {  get; }
        public String? Apartment { get; }

        public Address(Guid id, String city, String street, String home, String?
apartment)
        {
            Id = id;
            City = city;
            Street = street;
            Home = home;
            Apartment = apartment;
        }
    }
}
```

Category.cs:

```
namespace musicShopProject.Model.Categories;

public class Category
{
    public Guid Id { get; }
    public String Name { get; }

    public Category(Guid id, String name)
```

```
    {
        Id = id;
        Name = name;
    }
}
```

CategoryBlank.cs:

```
namespace musicShopProject.Model.Categories;

public partial class CategoryBlank
{
    public Guid? Id { get; set; }
    public String? Name { get; set; }
}

public partial class CategoryBlank
{
    public class Validated
    {
        public Guid Id { get; }
        public String Name { get; }

        public Validated(Guid id, String name)
        {
            Id = id;
            Name = name;
        }
    }
}
```

OrderState.cs:

```
namespace musicShopProject.Model.Orders.enums
{
    public enum OrderState
    {
        Ordered = 1,
        Canceled = 2,
        OnDelivery = 3,
        Completed = 4
    }
}
```

Order.cs:

```
using musicShopProject.Model.Addresses;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Model.Users;
using musicShopProject.Service.Orders.Repository.Model;

namespace musicShopProject.Model.Orders;

public class Order
{
    public Guid Id { get; }
    public Decimal Price { get; }
    public String ClientPhoneNumber { get; }
    public User Client { get; }
    public Address Address { get; }
    public DateTime? CompletedDateTimeUtc { get; }
    public DateTime CreatedDateTimeUtc { get;  }
    public OrderState State { get; }
    public Int32 OrderNumber { get; }
    public OrderItem[] OrderItems { get; }
```

```
    public Order(OrderDB db, Address address, User user, OrderItem[] items)
    {
        Id = db.Id;
        Price = db.Price;
        ClientPhoneNumber = db.ClientPhoneNumber;
        Client = user;
        Address = address;
        CompletedDateTimeUtc = db.CompletedDateTimeUtc;
        CreatedDateTimeUtc = db.CreatedDateTimeUtc;
        State = db.State;
        OrderNumber = db.OrderNumber;
        OrderItems = items;
    }
}
```

OrderItem.cs:

```
using musicShopProject.Model.Products;
using musicShopProject.Service.Orders.Repository.Model;

namespace musicShopProject.Model.Orders
{
    public class OrderItem
    {
        public Guid Id { get; }
        public Guid OrderId { get; }
        public Product Product { get; }
        public Decimal ProductPrice { get; }
        public Guid ProductCategoryId { get; set; }

        public OrderItem(OrderItemDB db, Product product)
        {
            Id = db.Id;
            OrderId = db.OrderId;
            Product = product;
            ProductPrice = db.ProductPrice;
            ProductCategoryId = db.ProductCategoryId;
        }
    }
}
```

Product.cs:

```
namespace musicShopProject.Model.Products;

public class Product
{
    public Guid Id { get; }
    public String Name { get; }
    public String Description { get; }
    public ProductStatus Status { get; }
    public String Manufacturer { get; }
    public Decimal Price { get; }
    public Decimal Weight { get; }
    public Guid CategoryId { get; }
    public Boolean IsHidden { get; }
    public Int32 Quantity { get; }
    public String[] Images {  get; }

    public Product(
        Guid id, String name, String description,
        ProductStatus status, String manufacturer, Decimal price, Decimal weight,
        Guid categoryId, Boolean isHidden, Int32 quantity, String[] images)
    {
```

```
            Id = id;
            Name = name;
            Description = description;
            Status = status;
            Manufacturer = manufacturer;
            Price = price;
            Weight = weight;
            CategoryId = categoryId;
            IsHidden = isHidden;
            Quantity = quantity;
            Images = images;
        }
    }
```

ProductBlank.cs:

```
namespace musicShopProject.Model.Products;

public partial class ProductBlank
{
    public Guid? Id { get; set; }
    public String? Name { get; set; }
    public String? Description { get; set; }
    public Guid? CategoryId { get; set; }
    public Decimal? Price { get; set; }
    public Decimal? Weight { get; set; }
    public String? Manufacturer { get; set; }
    public Int32? Quantity { get; set; }
    public String[]? Images {  get; set; }
    public Int32? Status { get; set; }
    public Boolean? IsHidden { get; set; }
}
public partial class ProductBlank
{
    public class Validated
    {
        public Guid Id { get; }
        public String Name { get; }
        public String Description { get; }
        public Guid CategoryId { get; }
        public Decimal Price { get; }
        public Decimal Weight { get; }
        public String Manufacturer { get; }
        public Int32 Quantity { get; }
        public String[] Images { get; }
        public Int32 Status { get; }
        public Boolean IsHidden { get; }
        //public Guid UserId { get; }

        public Validated(
            Guid id, String name, String description, Decimal price,
            Guid categoryId, Decimal weight, String manufacturer,
            Int32 quantity, String[] image, Int32 status, Boolean isHidden
            //Guid userId
        )
        {
            Id = id;
            Name = name;
            Description = description;
            CategoryId = categoryId;
            Price = price;
            Weight = weight;
            Manufacturer = manufacturer;
```

```
            Quantity = quantity;
            Images = image;
            Status = status;
            IsHidden = isHidden;
            //UserId = userId;
        }
    }
}
```

ProductStatus.cs:

```
namespace musicShopProject.Model.Products;

public enum ProductStatus
{
    OnSale = 1,
    TemporarilyOutOfSale = 2,
    OutOfStock = 3
}
```

Password.cs:

```
using System.Security.Cryptography;
using System.Text;

namespace musicShopProject.Model.Users;

public class Password
{
    public String Value { get; }
    public String Hash => GetHashString(Value);

    public Password(String value)
    {
        Value = value;
    }

    private String GetHashString(String str)
    {
        Byte[] bytes = Encoding.Unicode.GetBytes(str);
        MD5 md5 = MD5.Create();

        Byte[] byteHash = md5.ComputeHash(bytes);
        String hash = Convert.ToHexString(byteHash);

        return hash;
    }
}
```

User.cs:

```
using System.Text.Json.Serialization;

namespace musicShopProject.Model.Users;

//Client
public class User
{
    public Guid Id { get; }
    public String PhoneNumber { get; }
    public String? Email { get; }
    public DateOnly? BirthDate { get; }
    [JsonIgnore]
```

```
    public String PasswordHash { get; }

    public User(Guid id, String phoneNumber, String? email, DateOnly? birthDate)
    {
        Id = id;
        PhoneNumber = phoneNumber;
        Email = email;
        BirthDate = birthDate;
    }
}
```

UserBlank.cs:

```
namespace musicShopProject.Model.Users;

public partial class UserBlank
{
    public Guid? Id { get; set; }
    public String? PhoneNumber { get; set; }
    public String? Password { get; set; }
    public Boolean PasswordBeChanged { get; set; }
}

public partial class UserBlank
{
    public class Validated
    {
        public Guid Id { get; }
        public String PhoneNumber { get; }
        public Password Password { get; }
        public Boolean PasswordBeChanged { get; }

        public Validated(Guid id, Password password, String phoneNumber, Boolean
passwordBeChanged)
        {
            Id = id;
            Password = password;
            PhoneNumber = phoneNumber;
            PasswordBeChanged = passwordBeChanged;
        }
    }
  }
```

CategoryService.cs:

```
using musicShopProject.Model.Categories;
using musicShopProject.Service.Categories.Repository;
using musicShopProject.Tools.Extensions;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Categories;

public class CategoryService : ICategoryService
{
    private readonly ICategoryRepository _categoryRepository;
    public CategoryService(ICategoryRepository categoryRepository)
    {
        _categoryRepository = categoryRepository;
    }

    public Result AddCategory(CategoryBlank categoryBlank, Guid requestedUserId)
    {
        categoryBlank.Id ??= Guid.NewGuid();
        return SaveCategory(categoryBlank, requestedUserId);
```

```
    }

    private Result SaveCategory(CategoryBlank categoryBlank, Guid requestedUserId)
    {
        Result result = ValidateCategoryBlank(categoryBlank, out
CategoryBlank.Validated validatedBlank);
        if (!result.IsSuccess) return Result.Fail(result.ErrorsAsString);

        _categoryRepository.SaveCategory(validatedBlank, requestedUserId);

        return Result.Success();
    }

    private Result ValidateCategoryBlank(CategoryBlank categoryBlank, out
CategoryBlank.Validated validatedBlank)
    {
        validatedBlank = null!;

        if (categoryBlank.Id is not { } id) throw new Exception("id отсутствует");

        if (categoryBlank.Name.IsNullOrWhiteSpace()) return Result.Fail("Не указанно
имя категории");

        validatedBlank = new CategoryBlank.Validated(id, categoryBlank.Name!);

        return Result.Success();
    }


    public Category[] GetAllCategories()
    {
        return _categoryRepository.GetAllCategories();
    }
}
```

ICategoryRepository.cs:

```
using musicShopProject.Model.Categories;

namespace musicShopProject.Service.Categories.Repository;

public interface ICategoryRepository
{
    void SaveCategory(CategoryBlank.Validated validatedBlank, Guid requestedUserId);
    Category[] GetAllCategories();
}
```

CategoryRepository.cs:

```
using musicShopProject.Model.Categories;
using musicShopProject.Service.Categories.Repository.Converters;
using musicShopProject.Service.Categories.Repository.Models;
using musicShopProject.Tools.DataBase.Interfaces;
using Npgsql;

namespace musicShopProject.Service.Categories.Repository;

public class CategoryRepository : ICategoryRepository
{
    private readonly IMainConnector _mainConnector;
    public CategoryRepository(IMainConnector mainConnector)
    {
        _mainConnector = mainConnector;
    }
```

```
    public void SaveCategory(CategoryBlank.Validated validatedBlank, Guid
requestedUserId)
    {
        String query = @"INSERT INTO categories (
                        id, name, isremoved, createddatetimeutc, modifieddatetimeutc,
                        createduserid, modifieduserid
                        )
                        VALUES ( @p_id, @p_name, false, @p_datetime, null, @p_userid,
null)
                        ON CONFLICT (id)
                        DO UPDATE SET
                        name = @p_name,
                        modifieddatetimeutc = @p_datetime,
                        modifieduserid = @p_userid";

        NpgsqlParameter[] parameters =
        {
            new("p_id", validatedBlank.Id),
            new("p_name", validatedBlank.Name),
            new("p_datetime", DateTime.UtcNow),
            new("p_userid", requestedUserId)
        };

        _mainConnector.ExecuteNonQuery(query, parameters);
    }

    public Category[] GetAllCategories()
    {
        String query = @"SELECT * FROM categories";

        Category[] categories =
_mainConnector.GetAllArray<CategoryDB>(query).Select(c => c.ToCategory()).ToArray();

        return categories;
    }
}
```

CategoryDB.cs:

```
namespace musicShopProject.Service.Categories.Repository.Models;

public class CategoryDB
{
    public Guid Id { get; set; }
    public String Name { get; set; }
    public Boolean IsRemoved { get; set; }
    public DateTime CreatedDateTime { get; set; }
    public DateTime? ModifiedDateTime { get; set; }
    public Guid CreatedUserId { get; set; }
    public Guid? ModifiedUserId { get; set; }
}
```

CategoryConverter.cs:

```
using musicShopProject.Model.Categories;
using musicShopProject.Service.Categories.Repository.Models;

namespace musicShopProject.Service.Categories.Repository.Converters;

public static class CategoryConverter
{
    public static Category ToCategory(this CategoryDB db)
```

```
    {
        return new Category(db.Id, db.Name);
    }
}
```

OrderService.cs:

```
using musicShopProject.Model.Addresses;
using musicShopProject.Model.Orders;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Model.Products;
using musicShopProject.Model.Users;
using musicShopProject.Service.Orders.Repository;
using musicShopProject.Service.Orders.Repository.Model;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Orders;

public class OrderService : IOrderService
{
    private readonly IOrderRepository _orderRepository;
    private readonly IUserService _userService;
    private readonly IProductService _productService;
    public OrderService(IOrderRepository orderRepository, IUserService userService,
IProductService productService)
    {
        _orderRepository = orderRepository;
        _userService = userService;
        _productService = productService;
    }

    public PagedResult<Order> GetOrderPage(Int32 page, Int32 pageSize)
    {
        PagedResult<OrderDB> orderDBs = _orderRepository.GetOrderPage(page,
pageSize);

        Guid[] addressesIds = orderDBs.Values.Select(order =>
order.AddressId).ToArray();

        Address[] addresses = _orderRepository.GetAddresses(addressesIds);

        Guid[] usersIds = orderDBs.Values.Select(order => order.ClientId).ToArray();

        User[] users = _userService.GetUsers(usersIds);

        List<Order> orders = new List<Order>();

        foreach (OrderDB orderDB in orderDBs.Values)
        {
            User? user = users.FirstOrDefault(x => x.Id == orderDB.ClientId);
            Address? address = addresses.FirstOrDefault(x => x.Id ==
orderDB.AddressId);
            OrderItemDB[] orderItemDBs = _orderRepository.GetOrderItems(orderDB.Id);

            List<OrderItem> orderItems = new List<OrderItem>();

            foreach(OrderItemDB orderItemDB in orderItemDBs)
            {
                Product? product =
_productService.GetProduct(orderItemDB.ProductId);
                orderItems.Add(new(orderItemDB, product));
            }
```

```
            orders.Add(new(orderDB, address, user, orderItems.ToArray()));
        }

        return new PagedResult<Order>(orders, orderDBs.TotalRows);
    }

    public Result ChangeOrderState(OrderState orderState, Guid orderId)
    {
        _orderRepository.ChangeOrderState(orderState, orderId);

        return Result.Success();
    }
}
```

IOrderRepository.cs:

```
using musicShopProject.Model.Addresses;
using musicShopProject.Model.Orders;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Model.Users;
using musicShopProject.Service.Orders.Repository.Model;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Orders.Repository
{
    public interface IOrderRepository
    {
        PagedResult<OrderDB> GetOrderPage(Int32 page, Int32 pageSize);
        Address GetOrderAddress(Guid addressId);
        Address[] GetAddresses(Guid[] addressesIds);
        OrderItemDB[] GetOrderItems(Guid orderId);
        User GetOrderClient(Guid clientId);
        void ChangeOrderState(OrderState orderState, Guid orderId);
    }
}
```

OrderRepository.cs:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.RazorPages;
using musicShopProject.Model.Addresses;
using musicShopProject.Model.Orders;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Model.Users;
using musicShopProject.Service.Orders.Repository.Converter;
using musicShopProject.Service.Orders.Repository.Model;
using musicShopProject.Service.Users.Repository.Converters;
using musicShopProject.Service.Users.Repository.Models;
using musicShopProject.Tools.DataBase.Interfaces;
using musicShopProject.Tools.Types;
using Npgsql;

namespace musicShopProject.Service.Orders.Repository;

public class OrderRepository : IOrderRepository
{
    private readonly IMainConnector _mainConnector;
    public OrderRepository(IMainConnector mainConnector)
    {
        _mainConnector = mainConnector;
    }
```

```
    public Address GetOrderAddress(Guid addressId)
    {
        String query = @"select * from address
                         where id = @p_addressid";

        NpgsqlParameter[] parameters =
        {
            new("p_addressid", addressId)
        };

        Address address = _mainConnector.Get<AddressDB>(query,
parameters).ToAddress();

        return address;
    }

    public Address[] GetAddresses(Guid[] addressesIds)
    {
        String query = @"select * from address
                        where id = ANY(@p_addressesids)";

        NpgsqlParameter[] parameters =
        {
            new("p_addressesids", addressesIds)
        };

        Address[] addresses = _mainConnector
            .GetArray<AddressDB>(query, parameters)
            .Select(ad => ad.ToAddress())
            .ToArray();

        return addresses;
    }

    public OrderItemDB[] GetOrderItems(Guid orderId)
    {
        String query = @"select * from orderitems
                         where orderid = @p_orderid";

        NpgsqlParameter[] parameters =
        {
            new("p_orderid", orderId)
        };

        OrderItemDB[] orderItemDBs = _mainConnector.GetArray<OrderItemDB>(query,
parameters);

        return orderItemDBs;
    }

    public User GetOrderClient(Guid clientId)
    {
        String query = @"select * from users
                         where id = @p_clientid";

        NpgsqlParameter[] parameters =
        {
            new("p_clientid", clientId)
        };

        User user = _mainConnector.Get<UserDB>(query, parameters).ToUser();
```

```
            return user;
        }

        public void ChangeOrderState(OrderState orderState, Guid orderId)
        {
            String query = @"update orders
                             set state = @p_orderstate
                             where id = @p_orderid";

            NpgsqlParameter[] parameters =
            {
                new("p_orderstate", (Int32)orderState),
                new("p_orderid", orderId)
            };

            _mainConnector.ExecuteNonQuery(query, parameters);
        }

        public PagedResult<OrderDB> GetOrderPage(Int32 page, Int32 pageSize)
        {
            Int32 startIndex = (page - 1) * pageSize;

            String countQuery = @"SELECT COUNT(*) FROM orders WHERE isremoved = false";

            String dataQuery = @"SELECT * FROM orders
                             WHERE isremoved = false
                             ORDER BY createddatetimeutc DESC
                             OFFSET @p_startindex
                             LIMIT @p_pagesize";

            NpgsqlParameter[] parameters =
            {
                new("p_startindex", startIndex),
                new("p_pagesize", pageSize)
            };

            Int32 totalRows = _mainConnector.Get<Int32>(countQuery);

            OrderDB[] resultValues = _mainConnector.GetArray<OrderDB>(dataQuery,
parameters);

            return new PagedResult<OrderDB>(resultValues, totalRows);
        }
}
```

AddressDB.cs:

```
namespace musicShopProject.Service.Orders.Repository.Model
{
    public class AddressDB
    {
        public Guid Id { get; set; }
        public String City { get; set; }
        public String Street { get; set; }
        public String Home { get; set; }
        public String? Apartment { get; set; }
        public DateTime CreatedDateTimeUtc { get; set; }
        public DateTime? ModifiedDateTimeUtc { get; set;}
        public Boolean IsRemoved { get; set; }
    }
}
```

OrderDB.cs:

```
using musicShopProject.Model.Orders.enums;

namespace musicShopProject.Service.Orders.Repository.Model;

public class OrderDB
{
    public Guid Id { get; set; }
    public Decimal Price { get; set; }
    public String ClientPhoneNumber { get; set; }
    public Guid ClientId { get; set; }
    public Guid AddressId { get; set; }
    public DateTime CreatedDateTimeUtc { get; set; }
    public DateTime? ModifiedDateTimeUtc { get; set; }
    public DateTime? CompletedDateTimeUtc { get; set; }
    public OrderState State { get; set; }
    public Boolean IsRemoved { get; set; }
    public Int32 OrderNumber { get; set; }
}
```

OrderItemDB.cs:

```
namespace musicShopProject.Service.Orders.Repository.Model
{
    public class OrderItemDB
    {
        public Guid Id { get; set; }
        public Guid OrderId { get; set; }
        public Guid ProductId { get; set; }
        public Decimal ProductPrice { get; set; }
        public Boolean IsRemoved { get; set; }
        public DateTime CreatedDateTimeUtc { get; set; }
        public DateTime ModifiedDateTimeUtc { get; set; }
        public Guid ProductCategoryId { get; set; }
    }
}
```

AddressConverter.cs:

```
using musicShopProject.Model.Addresses;
using musicShopProject.Service.Orders.Repository.Model;

namespace musicShopProject.Service.Orders.Repository.Converter
{
    public static class AddressConverter
    {
        public static Address ToAddress(this AddressDB db)
        {
            return new Address(db.Id, db.City, db.Street, db.Home, db.Apartment);
        }
    }
}
```

ProductService.cs:

```
using musicShopProject.Model.Products;
using musicShopProject.Service.Products.Repository;
using musicShopProject.Tools.Extensions;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Products;
```

```
public class ProductService : IProductService
{
    private readonly IProductRepository _productRepository;
    private readonly IImageService _imageService;

    public ProductService(IProductRepository productRepository, IImageService
imageService)
    {
        _productRepository = productRepository;
        _imageService = imageService;
    }

    public Result AddProduct(ProductBlank blank, Guid requestedUserId)
    {
        blank.Id ??= Guid.NewGuid();
        return SaveProduct(blank, requestedUserId);
    }

    public Product[] GetProducts(Guid? categoryId = null)
    {
        return _productRepository.GetProducts(categoryId);
    }

    private Result SaveProduct(ProductBlank blank, Guid requestedUserId)
    {
        Result validateResult = ValidateProductBlank(blank, out
ProductBlank.Validated validatedProduct);
        if (!validateResult.IsSuccess) return Result.Fail(validateResult.Errors);

        _productRepository.SaveProduct(validatedProduct, requestedUserId);

        return Result.Success();
    }

    private Result ValidateProductBlank(ProductBlank blank, out
ProductBlank.Validated validatedProduct)
    {
        validatedProduct = null!;

        if (blank.Id is not {} id) throw new Exception("id продукта пуст");

        if (blank.Name.IsNullOrWhiteSpace()) return Result.Fail("Укажите название
товара");
        if (blank.CategoryId is not {} categoryId) return Result.Fail("Укажите
категорию товара");
        if (blank.Description.IsNullOrWhiteSpace()) return Result.Fail("Укажите
описание товара");

        if (blank.Price is not {} price) return Result.Fail("Укажите цену");
        if (price <= 0) return Result.Fail("Цена не может быть меньше или равна 0");

        if (blank.Weight is not { } weight) return Result.Fail("Укажите вес");
        if (weight <= 0) return Result.Fail("Вес не может быть меньше или равен 0");

        if (blank.Images.Length < 0) return Result.Fail("Добавьте фотографии");

        if (blank.Manufacturer.IsNullOrWhiteSpace()) return Result.Fail("Укажите
производителя");

        if (blank.Quantity is not { } quantity) return Result.Fail("Укажите
количество");
```

```
        if (quantity < 0) return Result.Fail("Количество не может быть меньше 0");

        if (blank.Status is not { } status) return Result.Fail("Укажите статус");

        if (blank.IsHidden is not { } isHidden) return Result.Fail("Укажите
видимость");

        _imageService.Save(blank.Images, out String[] imagesPaths);

        blank.Images = imagesPaths ?? blank.Images;

        validatedProduct = new ProductBlank.Validated(
            id, blank.Name!, blank.Description!, price,
            categoryId, weight, blank.Manufacturer!, quantity,
            blank.Images!, status, isHidden
        );

        return Result.Success();
    }

    public Result UpdateProduct(ProductBlank blank, Guid requestedUserId)
    {
        return SaveProduct(blank, requestedUserId);
    }

    public Product? GetProduct(Guid productId)
    {
        return _productRepository.GetProduct(productId);
    }
}
```

ImageServie.cs:

```
using musicShopProject.Tools.Types;
using System.Net;
using System.IO;

namespace musicShopProject.Service.Products
{
    public class ImageService : IImageService
    {
        private readonly IWebHostEnvironment _environment;

        public ImageService(IWebHostEnvironment environment)
        {
            _environment = environment;
        }

        public Result Save(String[] images, out String[] imagesPaths)
        {
            imagesPaths = null!;

            Result result = ValidateImages(images);
            if (!result.IsSuccess) return Result.Fail(result.ErrorsAsString);

            List<String> imagePaths = new();

            String baseDirectory = AppContext.BaseDirectory;

            String relativePath = Path.Combine("ClientApp", "public", "images",
"products");
```

```
            String targetDirectory = Path.Combine(baseDirectory, "..", "..", "..",
relativePath);

            if (!Directory.Exists(targetDirectory))
            {
                Directory.CreateDirectory(targetDirectory);
            }

            foreach (String image in images)
            {
                using WebClient webClient = new();

                byte[] imageBytes = webClient.DownloadData(image);

                String fileName = Guid.NewGuid().ToString() + ".png";
                String filePath = Path.Combine(targetDirectory, fileName);

                File.WriteAllBytes(filePath, imageBytes);

                imagePaths.Add(fileName);
            }

            imagesPaths = imagePaths.ToArray();

            return Result.Success();
        }

        private Result ValidateImages(String[] images)
        {
            foreach (String image in images)
            {
                if (!Uri.IsWellFormedUriString(image, UriKind.Absolute)) return
Result.Fail($"Неправильная ссылка: {image}");
            }
            return Result.Success();
        }
    }
}
```

IImageService.cs:

```
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Products;

public interface IImageService
{
    Result Save(String[] Images, out String[] imagesPaths);
}
```

ProductRepository.cs:

```
using musicShopProject.Model.Products;
using musicShopProject.Service.Products.Repository.Converters;
using musicShopProject.Service.Products.Repository.Models;
using musicShopProject.Tools.DataBase.Interfaces;
using Npgsql;

namespace musicShopProject.Service.Products.Repository;

public class ProductRepository : IProductRepository
{
```

```
    private readonly IMainConnector _mainConnector;

    public ProductRepository(IMainConnector mainConnector)
    {
        _mainConnector = mainConnector;
    }
    public void SaveProduct(ProductBlank.Validated blank, Guid requestedUserId)
    {
        String query = @"INSERT INTO products (
            id, name, description,
            price, weight, manufacturer, quantity,
            categoryid, images, status, createddatetimeutc, modifieddatetimeutc,
            isremoved, ishidden, createduserid, modifieduserid)
            VALUES (
            @p_id, @p_name, @p_description, @p_price, @p_weight,
            @p_manufacturer, @p_quantity, @p_category,
            @p_images, @p_status, @p_datetimeutc, null, false,
            false, @p_requesteduserid, null
            )
            ON CONFLICT (id)
            DO UPDATE SET
            name = @p_name,
            description = @p_description,
            price = @p_price,
            weight = @p_weight,
            manufacturer = @p_manufacturer,
            quantity = @p_quantity,
            categoryid = @p_category,
            images  = @p_images,
            status = @p_status,
            modifieddatetimeutc = @p_datetimeutc,
            modifieduserid = @p_requesteduserid";

        NpgsqlParameter[] parameters =
        {
            new("p_id", blank.Id),
            new("p_name", blank.Name),
            new("p_description", blank.Description),
            new("p_datetimeutc", DateTime.UtcNow),
            new("p_price", blank.Price),
            new("p_weight", blank.Weight),
            new("p_manufacturer", blank.Manufacturer),
            new("p_quantity", blank.Quantity),
            new("p_category",blank.CategoryId),
            new("p_images", blank.Images),
            new("p_status", blank.Status),
            new("p_requesteduserid", requestedUserId)
        };

        _mainConnector.ExecuteNonQuery(query, parameters);
    }

    public Product[] GetProducts(Guid? categoryId = null)
    {
        String query = @"SELECT * FROM products
                  WHERE (COALESCE(@p_categoryid::uuid, categoryid::uuid) =
categoryid::uuid)";


        NpgsqlParameter[] parameters =
        {
            new("p_categoryid", categoryId)
```

```
        };

        return _mainConnector.GetArray<ProductDB>(query, parameters).Select(product
=> product.ToProduct()).ToArray();
    }

    public Product? GetProduct(Guid productId)
    {
        String query = @"SELECT * FROM products
                 WHERE id = @p_productid";

        NpgsqlParameter[] parameters =
        {
            new("p_productid", productId)
        };

        return _mainConnector.Get<ProductDB>(query, parameters)?.ToProduct();
    }
}
```

IProductRepository.cs:

```
using musicShopProject.Model.Products;

namespace musicShopProject.Service.Products.Repository
{
    public interface IProductRepository
    {
        void SaveProduct(ProductBlank.Validated blank, Guid requestedUserId);
        Product[] GetProducts(Guid? categoryId = null);
        Product? GetProduct(Guid productId);
    }
}
```

ProductDB.cs:

```
using musicShopProject.Model.Products;

namespace musicShopProject.Service.Products.Repository.Models;

public class ProductDB
{
    public Guid Id { get; set; }
    public String Name { get; set; }
    public String Description { get; set; }
    public Decimal Price { get; set; }
    public Decimal Weight { get; set; }
    public String Manufacturer { get; set; }
    public Int32 Quantity { get; set; }
    public Guid CategoryId { get; set;}
    public String[] Images {  get; set; }
    public ProductStatus Status { get; set; }
    public DateTime CreatedDateTimeUtc { get; set; }
    public DateTime? ModifiedDateTimeUtc { get; set; }
    public Boolean IsRemoved { get; set; }
    public Boolean IsHidden { get; set; }
    public Guid CreatedUserId { get; set; }
    public Guid? ModifiedUserId { get; set; }
}
```

ProductConverter.cs:

```
using musicShopProject.Model.Products;
using musicShopProject.Service.Products.Repository.Models;

namespace musicShopProject.Service.Products.Repository.Converters
{
    public static class ProductConverter
    {
        public static Product ToProduct(this ProductDB db)
        {
            return new Product(db.Id, db.Name, db.Description, db.Status,
db.Manufacturer, db.Price, db.Weight, db.CategoryId, db.IsHidden, db.Quantity,
db.Images);
        }
    }
}
```

UserService.cs:

```
using musicShopProject.Model.Users;
using musicShopProject.Service.Users.Repository;
using musicShopProject.Tools.Extensions;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service.Users;

public class UserService : IUserService
{
    private readonly IUserRepository _userRepository;

    public UserService(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    public Result Login(String? phoneNumber, String? password)
    {
        if (phoneNumber.IsNullOrWhiteSpace()) return Result.Fail("Укажите логин");
        if (password.IsNullOrWhiteSpace()) return Result.Fail("Укажите пароль");

        User? existUser = _userRepository.GetUserByPhoneNumber(phoneNumber!,
password!.GetHash());
        if(existUser is null) return Result.Fail("Такого пользователя не
существует");

        return Result.Success();
    }

    public Result Register(UserBlank blank)
    {
         blank.Id ??= Guid.NewGuid();

        Result validateResult = ValidateUserBlank(blank, out UserBlank.Validated
validatedBlank);
        if (!validateResult.IsSuccess) return Result.Fail(validateResult.Errors);

        _userRepository.Save(validatedBlank);

        return Result.Success();
    }

    private Result ValidateUserBlank(UserBlank blank, out UserBlank.Validated
validatedUser)
```

```
    {
        validatedUser = null!;

        if (blank.Id is not { } id) throw new Exception("ID null у пользователя");

        User? existUser = GetUser(id);
        Boolean isCreating = existUser is null;

        if (blank.Password.IsNullOrWhiteSpace() && blank.PasswordBeChanged) return
Result.Fail("Укажите пароль");
        if (blank.PhoneNumber.IsNullOrWhiteSpace()) return Result.Fail("Укажите
номер телефона");

        //Валидация номера

        Password password = new(
            isCreating
            ? blank.Password!
            : blank.Password ?? existUser!.PasswordHash
        );

        validatedUser = new UserBlank.Validated(id,  password, blank.PhoneNumber!,
blank.PasswordBeChanged!);

        return Result.Success();
    }

    public User? GetUser(Guid id)
    {
        return _userRepository.GetUser(id);
    }

    public Result UpdatePassword(String? phoneNumber, String? oldPassword, String?
newPassword)
    {
        if (phoneNumber.IsNullOrWhiteSpace()) return Result.Fail("Укажите номер
телефона");
        if (oldPassword.IsNullOrWhiteSpace()) return Result.Fail("Укажите старый
пароль");
        if (newPassword.IsNullOrWhiteSpace()) return Result.Fail("Укажите новый
пароль");

        User? user = _userRepository.GetUserByEmail(phoneNumber!,
oldPassword!.GetHash());
        if (user is null) return Result.Fail("Пользователя с такими данными не
существует");

        _userRepository.UpdateUserPassword(user.Id, newPassword!.GetHash());

        return Result.Success();
    }

    public User[] GetAllUsers()
    {
        return _userRepository.GetAllUsers();
    }

    public User[] GetUsers(Guid[] usersIds)
    {
        return _userRepository.GetUsers(usersIds);
    }
```

```
    public User? GetUser(String phoneNumber)
    {
        return _userRepository.GetUser(phoneNumber);
    }

    public User? GetUserByToken(string email)
    {
        throw new NotImplementedException();
    }
}
```

UserRepository.cs:

```
using musicShopProject.Model.Users;
using musicShopProject.Service.Users.Repository.Converters;
using musicShopProject.Service.Users.Repository.Models;
using musicShopProject.Tools.DataBase.Interfaces;
using Npgsql;

namespace musicShopProject.Service.Users.Repository;


public class UserRepository : IUserRepository
{
    private readonly IMainConnector _mainConnector;

    public UserRepository(IMainConnector mainConnector)
    {
        _mainConnector = mainConnector;
    }

    public void Save(UserBlank.Validated validatedBlank)
    {
        String query = @$"INSERT INTO users (id, phonenumber, passwordhash, email,
                        createddatetimeutc, modifieddatetimeutc, isremoved,
birthdate)
                        VALUES (@p_id, @p_phonenumber, @p_passwordhash, null,
@p_datetimeutc,
                        null, false, null)
                        ON CONFLICT (id)
                        DO UPDATE SET
                        passwordhash = CASE WHEN @p_passwordbechanged THEN
@p_passwordhash ELSE users.passwordhash END,
                        modifieddatetimeutc = @p_datetimeutc";

        NpgsqlParameter[] parameters = {
            new ("p_id", validatedBlank.Id),
            new ("p_phonenumber", validatedBlank.PhoneNumber),
            new ("p_passwordhash", validatedBlank.Password.Hash),
            new ("p_datetimeutc", DateTime.UtcNow),
            new ("p_passwordbechanged", validatedBlank.PasswordBeChanged)
        };

        _mainConnector.ExecuteNonQuery(query, parameters);
    }

    public User? GetUser(String phoneNumber)
    {
        String query = "SELECT * FROM users WHERE phonenumber = @p_phonenumber";

        NpgsqlParameter[] parameters = { new("p_phonenumber", phoneNumber) };
        UserDB? userDB = _mainConnector.Get<UserDB?>(query, parameters);
```

```
        return userDB?.ToUser();
    }

    public User? GetUserByEmail(String phoneNumber, String passwordHash)
    {
        String query = $"SELECT * FROM users WHERE phonenumber = @p_phonenumber AND
passwordhash = @p_passwordhash";

        NpgsqlParameter[] parameters =
        {
            new("p_phonenumber", phoneNumber),
            new ("p_passwordhash",passwordHash)
        };

        UserDB? userDB = _mainConnector.Get<UserDB?>(query, parameters);

        return userDB?.ToUser();
    }

    public void UpdateUserPassword(Guid id, String newPassword)
    {
        String query = @"UPDATE users
                       SET passwordhash = @p_passwordhash
                       WHERE id = @p_id";

        NpgsqlParameter[] parameters =
        {
            new("p_passwordhash", newPassword),
            new("p_id",id)
        };

        _mainConnector.ExecuteNonQuery(query,parameters);
    }

    public User? GetUserByPhoneNumber(String phoneNumber, String passwordHash)
    {
        String query = @"SELECT * FROM users
                         WHERE phonenumber = @p_phonenumber AND passwordhash =
@p_passwordhash";

        NpgsqlParameter[] parametrs =
        {
            new ("p_phonenumber", phoneNumber),
            new ("p_passwordhash",passwordHash)
        };

        UserDB? userDB = _mainConnector.Get<UserDB?>(query, parametrs);

        return userDB?.ToUser();
    }

    public User? GetUser(Guid id)
    {
        String query = $"SELECT * FROM users WHERE id = @p_id";

        NpgsqlParameter parameter = new("p_id",id);

        UserDB? userDB = _mainConnector.Get<UserDB?>(query, parameter);

        return userDB?.ToUser();
    }
```

```
    public User[] GetUsers(Guid[] ids)
    {
        String query = "SELECT * FROM users WHERE id = ANY(@p_ids)";

        NpgsqlParameter parameter = new("p_ids",ids);

        return _mainConnector.GetArray<UserDB>(query, parameter)
            .Select(u => u.ToUser()).ToArray();
    }

    public User[] GetAllUsers()
    {
        String query = "SELECT * FROM users";

        IEnumerable<UserDB> userDBs = _mainConnector.GetArray<UserDB>(query);

        return userDBs.Select(userDB => userDB.ToUser()).ToArray();
    }
}
```

IUserRepository.cs:

```
using musicShopProject.Model.Users;

namespace musicShopProject.Service.Users.Repository;

public interface IUserRepository
{
    void Save(UserBlank.Validated validatedBlank);

    User? GetUserByEmail(String email, String passwordHash);

    User? GetUser(String phoneNumber);

    User? GetUserByPhoneNumber(String phoneNumber, String passwordHash);

    void UpdateUserPassword(Guid id, String newPassword);

    User[] GetUsers(Guid[] ids);

    User? GetUser(Guid id);

    User[] GetAllUsers();
}
```

UserDB.cs:

```
namespace musicShopProject.Service.Users.Repository.Models;

public class UserDB
{
    public Guid Id { get; set; }
    public String PhoneNumber { get; set; }
    public String Password { get; set; }
    public String? Email { get; set; }
    public DateTime CreateDateTimeUtc { get; set; }
    public DateTime? ModifiedDateTimeUtc { get; set; }
    public Boolean IsRemoved { get; set; }
    public DateOnly? BirthDate { get; set; }
}
```

UserConverter.cs:

```
using musicShopProject.Model.Users;
using musicShopProject.Service.Users.Repository.Models;

namespace musicShopProject.Service.Users.Repository.Converters;

public static class UserConverter
{
    public static User ToUser(this UserDB db)
    {
        return new User(db.Id, db.PhoneNumber, db.Email, db.BirthDate);
    }
}
```

ICategoryService.cs:

```
using musicShopProject.Model.Categories;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service
{
    public interface ICategoryService
    {
        Result AddCategory(CategoryBlank categoryBlank, Guid requestedUserId);
        Category[] GetAllCategories();
    }
}
```

IOrderService.cs:

```
using musicShopProject.Model.Orders;
using musicShopProject.Model.Orders.enums;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service;

public interface IOrderService
{
    PagedResult<Order> GetOrderPage(Int32 page, Int32 pageSize);
    Result ChangeOrderState(OrderState orderState, Guid orderId);
}
```

IProductSerivce.cs:

```
using musicShopProject.Model.Products;
using musicShopProject.Tools.Types;

namespace musicShopProject.Service;

public interface IProductService
{
    Result AddProduct(ProductBlank blank, Guid requestedUserId);
    Product[] GetProducts(Guid? categoryId = null);
    Result UpdateProduct(ProductBlank blank, Guid requestedUserId);
    Product? GetProduct(Guid productId);
}
```

IUserService.cs:

```
using musicShopProject.Model.Users;
using musicShopProject.Tools.Types;
```

```
namespace musicShopProject.Service;
public interface IUserService
{
    Result Login(String? phoneNumber, String? password);
    Result Register(UserBlank blank);
    Result UpdatePassword(String? email, String? oldPassword, String? newPassword);
    User[] GetAllUsers();
    User[] GetUsers(Guid[] usersIds);
    User? GetUser(String phoneNumber);
    User? GetUserByToken(String email);
}
```

IMainConnector.cs:

```
using musicShopProject.Tools.Types;
using Npgsql;

namespace musicShopProject.Tools.DataBase.Interfaces;

public interface IMainConnector
{
    void ExecuteNonQuery(String query, params NpgsqlParameter[] parameters);

    T? Get<T>(String query, params NpgsqlParameter[] parameters);

    Page<T> GetPage<T>(String query, params NpgsqlParameter[] parameters);

    T[] GetArray<T>(String query, params NpgsqlParameter[] parameters);

    public T[] GetAllArray<T>(String query);
}
```

DataBaseFunc.cs:

```
using Npgsql;

namespace musicShopProject.Tools.DataBase;

public static class DataBaseFuncs
{
    public static void AddParameter<T> (this NpgsqlCommand command,String name, T?
value)
    {
        if (value is not null) command.Parameters.AddWithValue(name, value);
        else command.Parameters.AddWithValue(name,DBNull.Value);
    }
}
```

MainConnector.cs:

```
using Dapper;
using musicShopProject.Tools.DataBase.Interfaces;
using musicShopProject.Tools.Types;
using Npgsql;

namespace musicShopProject.Tools.DataBase;

public class MainConnector : IMainConnector
{
    private String _connectionString { get; }

    public MainConnector(String connectionString)
```

```
    {
        _connectionString = connectionString;
    }

    public void ExecuteNonQuery(String query, params NpgsqlParameter[] parameters)
    {
        using NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        connection.Open();

        using NpgsqlCommand command = new NpgsqlCommand(query, connection);
        foreach (NpgsqlParameter parameter in parameters)
        {
            command.AddParameter(parameter.ParameterName, parameter.Value);
        }

        command.ExecuteNonQuery();

        connection.Close();
        connection.Dispose();
    }

    public Page<T> GetPage<T>(String query, params NpgsqlParameter[] parameters)
    {
        DynamicParameters dynamicParameters = new();
        foreach (NpgsqlParameter parameter in parameters)
        {
            dynamicParameters.Add(parameter.ParameterName, parameter.Value,
parameter.DbType, parameter.Direction, parameter.Size);
        }

        using NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        connection.Open();

        Int32 totalRows = connection.QueryFirstOrDefault<Int32>(query,
dynamicParameters);
        List<T> values = connection.Query<T>(query, dynamicParameters).ToList();

        return new Page<T> { Values =  values, TotalRows = totalRows };

    }

    public T? Get<T>(String query, params NpgsqlParameter[] parameters)
    {
        DynamicParameters dynamicParameters = new();
        foreach (NpgsqlParameter parameter in parameters)
        {
            dynamicParameters.Add(parameter.ParameterName, parameter.Value,
parameter.DbType, parameter.Direction, parameter.Size);
        }

        using NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        connection.Open();

        return connection.QueryFirstOrDefault<T>(query, dynamicParameters);
    }

    public T[] GetArray<T>(String query, params NpgsqlParameter[] parameters)
    {
        DynamicParameters dynamicParameters = new();
        foreach (NpgsqlParameter parameter in parameters)
        {
```

```
            dynamicParameters.Add(parameter.ParameterName, parameter.Value,
parameter.DbType, parameter.Direction, parameter.Size);
        }

        using NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        connection.Open();

        return connection.Query<T>(query, dynamicParameters).ToArray();

    }

    public T[] GetAllArray<T>(String query)
    {
        using NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        connection.Open();

        return connection.Query<T>(query).ToArray();
    }
}
```

ApplicationExtensions.cs:

```
using Microsoft.AspNetCore.Diagnostics;
using System.Net;

namespace musicShopProject.Tools.Extensions;

public static class ApplicationExtensions
{
    public static void UseExceptionsHandler(this IApplicationBuilder app)
    {
        app.UseExceptionHandler(builder =>
        {
            builder.Run(async context =>
            {
                Exception? exception =
context.Features.Get<IExceptionHandlerFeature>()?.Error;


                context.Response.Clear();
                context.Response.StatusCode =
(Int32)HttpStatusCode.InternalServerError;
                context.Response.ContentType = "application/json";
            });
        });
    }
}
```

HttpExtensions.cs:

```
using Microsoft.AspNetCore.Mvc.Controllers;

namespace musicShopProject.Tools.Extensions;

public static class HttpExtensions
{
    public static Boolean IsAjaxRequest(this HttpRequest request)
    {
        if (request == null)
            throw new ArgumentNullException(nameof(request));

        if (request.Headers != null)
```

```
            return request.Headers["X-Requested-With"] == "XMLHttpRequest";

        return false;
    }

    public static Boolean EndpointHasAttribute<T>(this HttpContext context) where T
: Attribute
    {
        Boolean? isEndpointHasAttribute = context.GetEndpoint()
            ?.Metadata
            .GetMetadata<ControllerActionDescriptor>()
            ?.MethodInfo
            .GetCustomAttributes(inherit: true)
            .OfType<T>()
            .Any();

        return isEndpointHasAttribute ?? false;
    }


    public static T[] GetAttributes<T>(this HttpContext context) where T : Attribute
    {
        return context.GetEndpoint()
            ?.Metadata
            .GetMetadata<ControllerActionDescriptor>()
            ?.MethodInfo
            .GetCustomAttributes(inherit: true)
            .OfType<T>()
            .ToArray() ?? new T[0];
    }
}
```

StringExtensions.cs:

```
using System.Security.Cryptography;
using System.Text;

namespace musicShopProject.Tools.Extensions;

public static class StringExtensions
{
    public static Boolean IsNullOrWhiteSpace(this String? str)
    {
        return String.IsNullOrWhiteSpace(str);
    }

    public static String GetHash(this String input)
    {
        using MD5 md5 = MD5.Create();

        Byte[] inputBytes = Encoding.Unicode.GetBytes(input);
        Byte[] hashBytes = md5.ComputeHash(inputBytes);

        return Convert.ToHexString(hashBytes);
    }
}
```

Page.cs:

```
namespace musicShopProject.Tools.Types;

public class Page<T>
```

```
{
    public List<T> Values { get; set; }
    public Int32 TotalRows { get; set; }
}
```

PagedResult.cs:

```
namespace musicShopProject.Tools.Types;

public static class PagedResult
{
    public static PagedResult<T> Empty<T>()
    {
        return new PagedResult<T>(Array.Empty<T>(), 0);
    }

    public static PagedResult<T> Create<T>(IEnumerable<T> values, Int32 totalRows)
    {
        return new PagedResult<T>(values, totalRows);
    }
}

public class PagedResult<T>
{
    public List<T> Values { get; }

    public Int32 TotalRows { get; }

    public PagedResult(IEnumerable<T> values, Int32 totalRows)
    {
        Values = new List<T>(values ?? Array.Empty<T>());
        TotalRows = totalRows;
    }

    public void Deconstruct(out T[] values, out Int32 totalRows)
    {
        values = Values.ToArray();
        totalRows = TotalRows;
    }
}
```

Result.cs:

```
namespace musicShopProject.Tools.Types;

public class Result
{
    public Boolean IsSuccess => Errors.Length == 0;
    public String[] Errors { get;}
    public String ErrorsAsString => String.Join(", ", Errors);

    public Result(String[] errors)
    {
        Errors = errors;
    }

    public static Result Success()
    {
        return new Result([]);
    }

    public static Result Fail(String error)
```

```
    {
        return new Result([error]);
    }

    public static Result Fail(IEnumerable<String> error)
    {
        return new Result(error.ToArray());
    }
}
```

ServiceConfigurator.cs:

```
using musicShopProject.Service;
using musicShopProject.Service.Categories;
using musicShopProject.Service.Categories.Repository;
using musicShopProject.Service.Orders;
using musicShopProject.Service.Orders.Repository;
using musicShopProject.Service.Products;
using musicShopProject.Service.Products.Repository;
using musicShopProject.Service.Users;
using musicShopProject.Service.Users.Repository;
using musicShopProject.Tools.DataBase;
using musicShopProject.Tools.DataBase.Interfaces;

namespace musicShopProject;

public static class ServiceConfigurator
{
    public static void Initialize(this IServiceCollection collection, String
environment)
    {
        #region Services

        collection.AddSingleton<IUserService, UserService>();
        collection.AddSingleton<IProductService, ProductService>();
        collection.AddSingleton<IImageService, ImageService>();
        collection.AddSingleton<ICategoryService, CategoryService>();
        collection.AddSingleton<IOrderService, OrderService>();

        #endregion Services

        #region Repositories

        collection.AddSingleton<IUserRepository, UserRepository>();
        collection.AddSingleton<IProductRepository, ProductRepository>();
        collection.AddSingleton<ICategoryRepository, CategoryRepository>();
        collection.AddSingleton<IOrderRepository, OrderRepository>();

        #endregion Repositories

        IConfiguration configuration = new ConfigurationBuilder()
        .AddJsonFile($"appsettings.{environment}.json", optional: false)
        .Build();

        collection.AddSingleton<IMainConnector>(new
MainConnector(configuration.GetConnectionString("Main")!));
    }
}
```

Programm.cs:

```
using Microsoft.Extensions.FileProviders;
```

```
using musicShopProject;
using musicShopProject.Tools.Extensions;
using musicShopProject.Tools.Infrastructure;

WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

builder.Services.Initialize(builder.Environment.EnvironmentName);

builder.Services.AddControllers();
builder.Services.AddSwaggerGen();

WebApplication app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseCors(builder => builder
            .AllowAnyOrigin()
            .AllowAnyHeader()
            .AllowAnyMethod()
);

app.UseExceptionsHandler();
app.UseStaticFiles();
app.MapControllers();
app.UseRouting();

app.Run();
```