Задание по аналитике

**Задание №1**

**Тема: Бизнес-процесс создания заказа в приложении "Частная пекарня".**

Цель задания:

Разработать визуальную модель бизнес-процесса создания заказа в мобильном приложении для пекарни, используя стандартную нотацию моделирования.

Инструменты для выполнения:

Для создания диаграммы бизнес-процесса была использована: Онлайн-доска Miro.

**Подготовка рабочей области:**

1. Для начала создали новую доску: "Create" → "New board"
2. Добавили заголовок: "Бизнес-процесс 'Частной пекарни' "
3. Создали основные процессы и настроили цветовую кодировку:

* Клиент: зеленый
* Приложение: красный
* Сервер: желтый
* Пекарня: оранжевый
* База данных: фиолетовый
* Доставка: голубой

**Использованные элементы:**

1. Использовали прямоугольники для действий
2. Стрелки-связи с подписями
3. Для Базы данных использовали иконку "Database" из библиотеки Miro

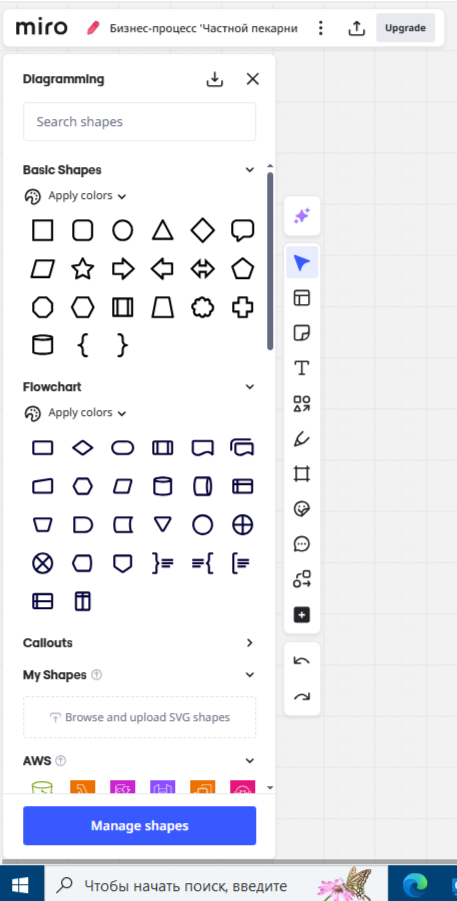


Рис.1.Панель с основными инструментами

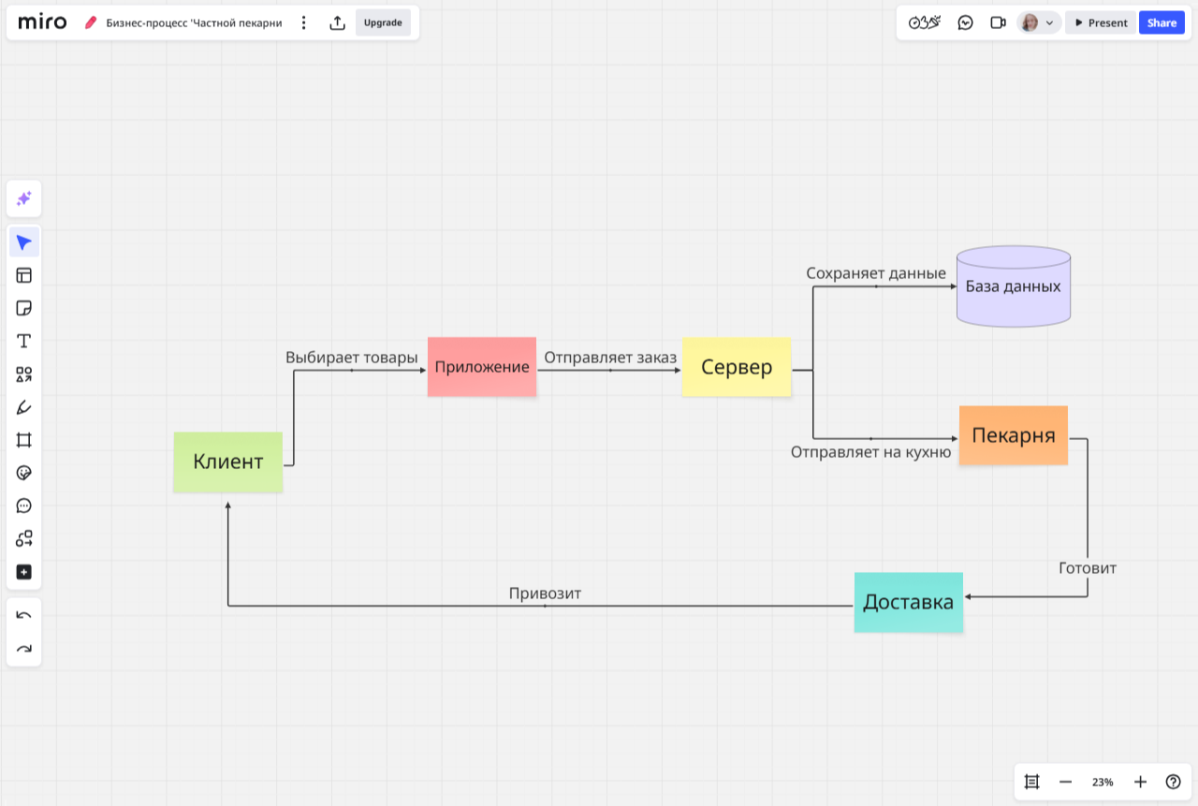


Рис.2. Бизнес-процессы создания заказа в Miro

**Задание №2**

**Тема: Процесс синхронизации данных в приложении "Частная пекарня".**

**Общее описание процесса синхронизации:**

Процесс синхронизации между клиентом и сервером в мобильном приложении "Частная пекарня" можно разделить на следующие основные этапы:

1. Инициализация соединения - установление связи между клиентом и сервером
2. Аутентификация пользователя - проверка учетных данных
3. Синхронизация данных - обмен актуальной информацией
4. Обработка изменений - применение изменений с обеих сторон
5. Подтверждение синхронизации - завершение процесса

**UML-диаграммы:**

**1.Диаграмма последовательности синхронизации:**

В рамках разработки клиент-серверного приложения "Частная пекарня" была создана UML-диаграмма последовательности, визуализирующая процесс синхронизации данных между мобильным клиентом и сервером. Для автоматической генерации диаграммы использовался инструмент PlantUML — текстовый язык описания диаграмм, который преобразует код в графическое представление.

Использованный код PlantUML представлен ниже на скриншоте:

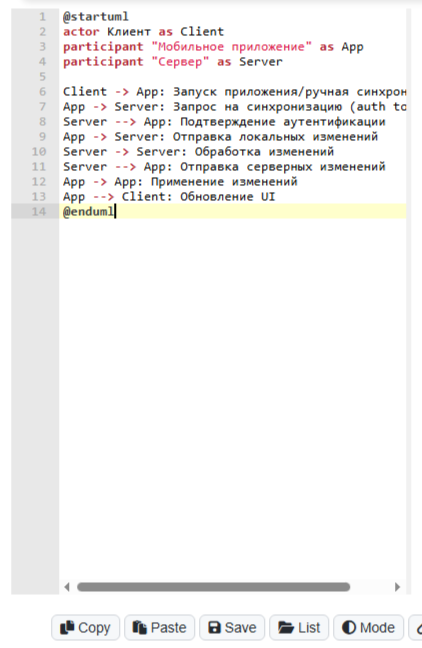


Рис.3. PlantUML-код диаграммы последовательности

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма последовательности.

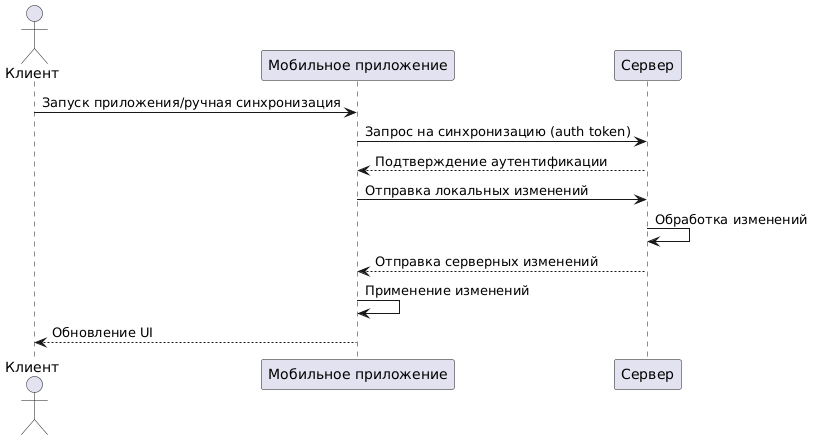


Рис.4. Диаграмма последовательности синхронизации

**Описание диаграммы последовательности синхронизации:**

Участники (Actors/Components):

* Клиент (Пользователь) — инициирует процесс синхронизации.
* Мобильное приложение — промежуточный слой между клиентом и сервером.
* Сервер — обрабатывает запросы и возвращает данные.

**Последовательность шагов:**

1. Инициация синхронизации

* Клиент запускает приложение или вручную инициирует синхронизацию (например, нажатием кнопки "Обновить").

2. Запрос на синхронизацию

* Мобильное приложение отправляет запрос к серверу, включая auth token для аутентификации.

3. Подтверждение аутентификации

* Сервер проверяет токен и возвращает статус (например, 200 OK или 401 Unauthorized).

4. Отправка локальных изменений

Приложение передает серверу изменения, сделанные пользователем в оффлайн-режиме:

* Новые/измененные заказы.
* Обновленные персональные данные.
* Отмененные заказы.

5. Обработка изменений на сервере

* Сервер применяет изменения к основной базе данных, разрешает конфликты (если есть) и подготавливает актуальные данные для клиента.

6. Отправка серверных изменений

Сервер возвращает клиенту:

* Обновленные статусы заказов.
* Изменения в каталоге продуктов (например, обновленные цены или доступность).
* Подтверждение изменений профиля.

7. Применение изменений в приложении

* Мобильное приложение обновляет локальную базу данных полученными данными.

8. Обновление интерфейса (UI)

Клиент видит актуальные данные:

* Список заказов с новыми статусами.
* Изменения в профиле.
* Обновленный каталог продуктов.

**2. Диаграмма состояний синхронизации:**

Использованный код PlantUML представлен ниже на скриншоте:

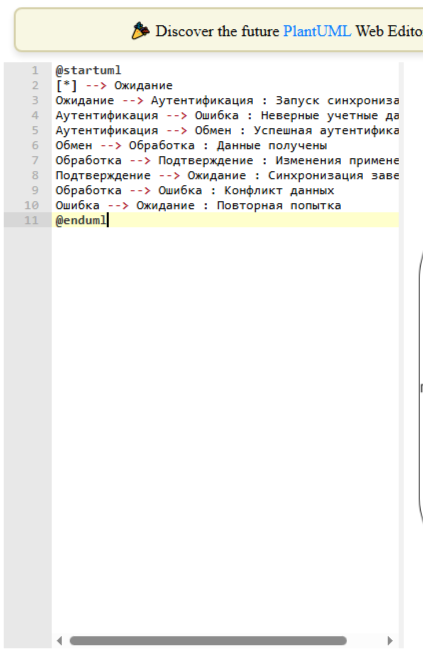


Рис.5. PlantUML-код диаграммы состояний

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма состояний.

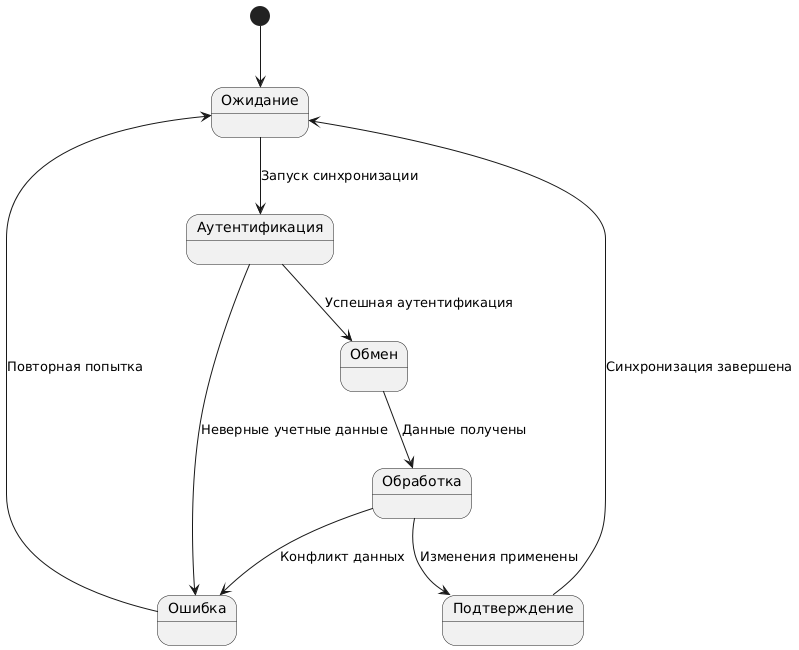


Рис.6. Диаграмма состояний синхронизации

**Описание диаграммы состояний процесса синхронизации:**

Основные состояния системы:

1) Ожидание

* Начальное состояние системы
* Ожидание команды на запуск синхронизации
* В этом состоянии система может находиться неограниченное время

2) Аутентификация

* Процесс проверки учетных данных пользователя
* Два возможных исхода:
* Успешная аутентификация
* Ошибка аутентификации (неверные учетные данные)

3) Обмен данными

* Фаза передачи данных между клиентом и сервером
* Включает:
* Отправку локальных изменений
* Получение серверных обновлений

4) Обработка данных

* Анализ и применение изменений
* Проверка на конфликты данных
* Валидация полученной информации

5) Подтверждение

* Фиксация результатов синхронизации
* Обновление меток времени последней синхронизации

6) Ошибка

* Состояние обработки исключительных ситуаций
* Может возникать на разных этапах процесса

**3. Диаграмма классов основных сущностей:**

Использованный код PlantUML представлен ниже на скриншоте:

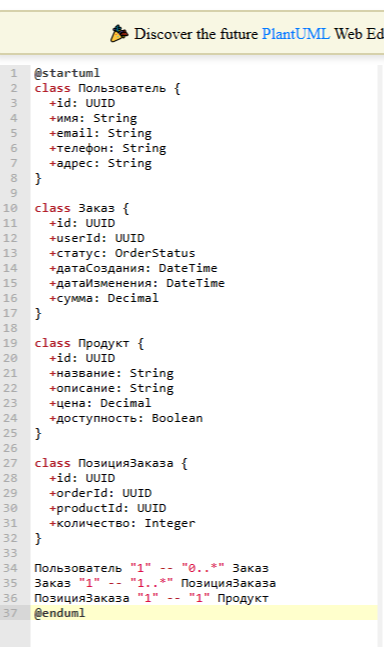


Рис.7. PlantUML-код диаграммы классов основных сущностей

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма классов основных сущностей.

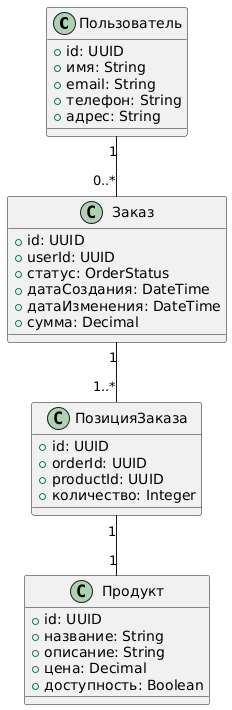


Рис.8. Диаграмма классов основных сущностей

**Описание диаграммы классов основных сущностей:**

Основные сущности и их атрибуты:

1) Пользователь

Описание: Клиент пекарни, делающий заказы

Атрибуты:

* id (UUID) - уникальный идентификатор
* имя (String) - полное имя пользователя
* email (String) - электронная почта (уникальный)
* телефон (String) - контактный номер
* адрес (String) - адрес доставки

2) Заказ

Описание: Транзакция покупки выпечки

Атрибуты:

* id (UUID) - уникальный идентификатор заказа
* userId (UUID) - ссылка на пользователя (внешний ключ)
* статус (OrderStatus) - enum: "создан", "оплачен", "доставлен" и т.д.
* датаСоздания (DateTime)
* датаИзменения (DateTime) - последнее обновление
* сумма (Decimal) - итоговая стоимость

3) ПозицияЗаказа

Описание: Конкретный товар в заказе

Атрибуты:

* id (UUID) - уникальный идентификатор
* orderId (UUID) - ссылка на заказ (внешний ключ)
* productId (UUID) - ссылка на продукт (внешний ключ)
* количество (Integer) - число единиц товара

4) Продукт

Описание: Выпечка/товар в ассортименте

Атрибуты:

* id (UUID) - уникальный идентификатор
* название (String) - наименование продукта
* описание (String) - детали (состав и т.д.)
* цена (Decimal) - текущая стоимость
* доступность (Boolean) - есть ли в наличии

**Связи между сущностями:**

* Пользователь → Заказ (1-to-many)
* Один пользователь может иметь множество заказов
* Связь реализована через userId в таблице Заказов
* Заказ → ПозицияЗаказа (1-to-many)
* Каждый заказ содержит одну или несколько позиций
* Связь через orderId в ПозицияхЗаказа
* ПозицияЗаказа → Продукт (many-to-1)
* Каждая позиция ссылается на конкретный продукт
* Связь через productId в ПозицияхЗаказа

**API методы для синхронизации:**

**1. Аутентификация (POST /api/auth)**

Назначение: Получение токена доступа для работы с API.

Используется: При первом входе в приложение или при истечении срока действия токена.

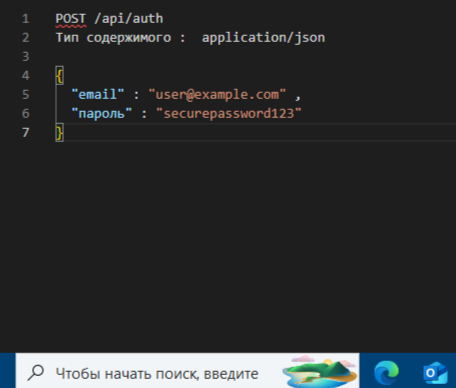


Рис.9. Запрос аутентификации

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Обязательно** | **Описание** |
| email | string | Да | Email пользователя |
| пароль | string | Да | Пароль (мин. 8 символов) |

Табл.1. Поля Body

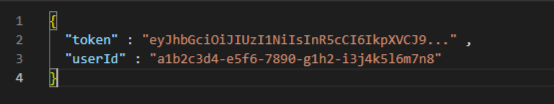


Рис.10. Успешный ответ (200 OK)

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| token | string | JWT-токен для авторизации |
| userId | string | Уникальный идентификатор пользователя |

Табл.2. Структура ответа при успешном входе в систему

**Ошибки:**

* 400 Bad Request: Неверный формат email/пароля.
* 401 Unauthorized: Неправильные учетные данные.
* 500 Internal Server Error: Ошибка сервера.

**2. Синхронизация данных (POST /api/sync)**

Назначение: Обмен изменениями между клиентом и сервером (заказы, профиль, каталог).

Используется: При запуске приложения или вручную по запросу пользователя.

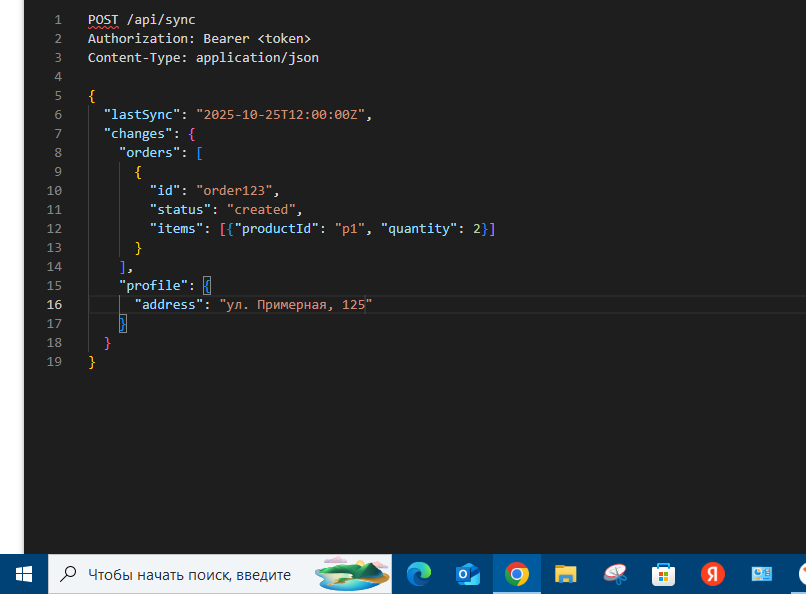


Рис.11.Запрос синхронизации данных

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| lastSync | DateTime | Время последней успешной синхронизации (ISO 8601). Если первый sync — null. |
| changes | object | Локальные изменения на клиенте. |
| changes.orders | Order[] | Список новых/измененных заказов. |
| changes.profile | UserProfile | Измененные данные профиля (адрес, телефон). |

Табл.3. Поля Body



Рис.12. Успешный ответ (200 OK)

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| serverTime | DateTime | Текущее время сервера (для следующего sync). |
| changes.orders | Order[] | Обновленные статусы заказов (например, "подтвержден"). |
| changes.products | Product[] | Изменения в каталоге (цены, доступность). |
| changes.profile | UserProfile | Обновленные бонусы/баллы лояльности. |

Табл.4. Структура ответа при успешном входе в систему

**Ошибки:**

* 401 Unauthorized: Недействительный токен.
* 409 Conflict: Конфликт данных (требуется ручное разрешение).
* 429 Too Many Requests: Слишком частые запросы.

**3. Специфические методы:**

* Создание/изменение заказа:
* Создание (POST /api/orders)

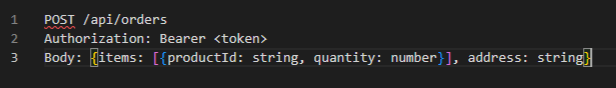
****

Рис.13. Создание заказа

* Изменение (PUT /api/orders/{id})

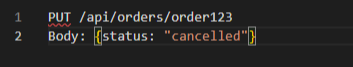


Рис.14. Изменение заказа

* Отмена заказа (POST /api/orders/{id}/cancel)

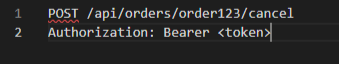


Рис.15. Отмена заказа



Рис.16. Ответ

* Обновление профиля (PUT /api/profile)

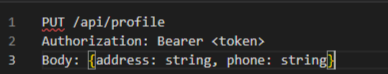


Рис.17. Обновление профиля

* Оплата заказа (POST /api/orders/{id}/pay)

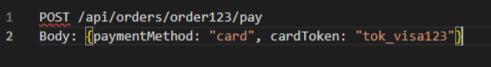


Рис.18. Оплата заказа



Рис.19. Ответ

**ER-диаграмма сущностей**

В рамках проектирования базы данных для системы "Частная пекарня" была разработана ER-диаграмма сущностей, автоматически сгенерированная с использованием декларативного языка PlantUML. Данный подход позволил совместить преимущества текстового описания структуры данных с наглядностью графического представления.

Использованный код PlantUML представлен ниже на скриншоте:

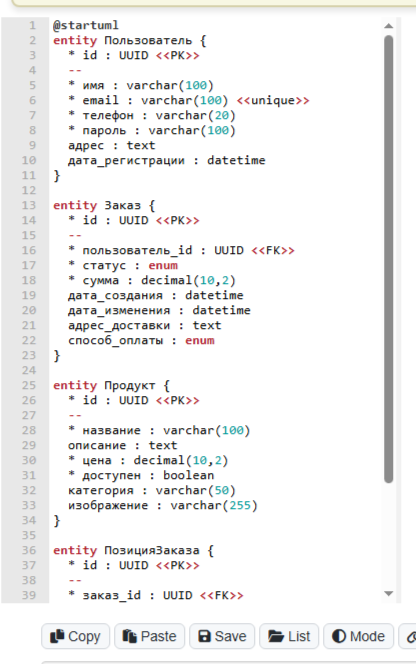
****

Рис.20. PlantUML-код ER-диаграммы сущностей

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже ER-диаграмма сущностей.

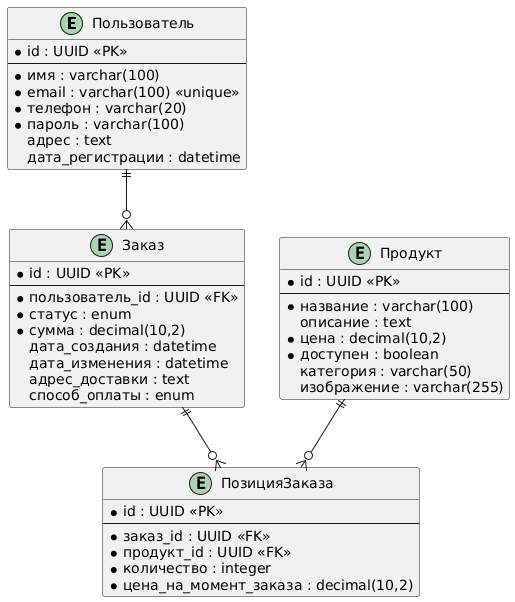


Рис.21. ER-Диаграмма сущностей

**Описание ER-диаграммы сущностей:**

Основные сущности системы:

1. Пользователь

Описание: Клиент, делающий заказы в системе

Атрибуты:

* id (UUID) - первичный ключ («PK»)
* имя (varchar(100)) - полное имя
* email (varchar(100)) - уникальный («unique»)
* телефон (varchar(20)) - контактный номер
* пароль (varchar(100)) - хэшированный
* адрес (text) - для доставки
* дата\_регистрации (datetime) - автоматически заполняемое

2. Заказ

Описание: Транзакция покупки

Атрибуты:

* id (UUID) - первичный ключ
* пользователь\_id (UUID) - внешний ключ («FK») к Пользователю
* статус (enum) - например: "новый", "оплачен", "доставлен"
* сумма (decimal(10,2)) - итоговая стоимость
* дата\_создания (datetime)
* дата\_изменения (datetime)
* адрес\_доставки (text) - может отличаться от профильного
* способ\_оплаты (enum) - "карта", "наличные" и др.

3. Продукт

Описание: Товар в ассортименте

Атрибуты:

* id (UUID) - первичный ключ
* название (varchar(100)) - наименование
* описание (text) - детали состава
* цена (decimal(10,2)) - актуальная стоимость
* доступен (boolean) - флаг наличия
* категория (varchar(50)) - тип выпечки
* изображение (varchar(255)) - путь к файлу

4. ПозицияЗаказа (Опечатка в оригинале: "ПоэмцияЗаказа")

Описание: Конкретный товар в заказе

Атрибуты:

* id (UUID) - первичный ключ
* заказ\_id (UUID) - внешний ключ к Заказу
* продукт\_id (UUID) - внешний ключ к Продукту
* количество (integer) - число единиц
* цена\_на\_момент\_заказа (decimal(10,2)) - фиксация цены

**Связи между сущностями:**

* Пользователь → Заказ (1-to-many)
* Один пользователь может иметь много заказов
* Реализовано через пользователь\_id в Заказе
* Заказ → ПозицияЗаказа (1-to-many)
* Каждый заказ содержит несколько позиций
* Связь через заказ\_id в ПозицииЗаказа
* Продукт → ПозицияЗаказа (1-to-many)
* Один продукт может быть во многих позициях
* Связь через продукт\_id в ПозицииЗаказа

**Задание №3**

**Тема: Прототип экрана "Создание заказа" для мобильного приложения "Частная пекарня".**

Экран "Создание заказа" предназначен для оформления заказа выпечки. Пользователь должен иметь возможность:

1. Выбирать товары из каталога.
2. Указывать количество и параметры (например, размер, добавки).
3. Видеть итоговую стоимость.
4. Оформлять доставку или самовывоз.
5. Оплачивать заказ.

**Прототип экрана**

В рамках данного задания была выполнена разработка интерактивного макета экрана оформления заказов для мобильного приложения "Частная пекарня". Реализация прототипа выполнена средствами HTML и CSS в интегрированной среде разработки Visual Studio Code."

**Процесс разработки:**

В редакторе Visual Studio Code была создана новая папка проекта под названием "практика 2025"

Внутри папки созданы два файла:

* index.html - для HTML-разметки
* styles.css - для стилей

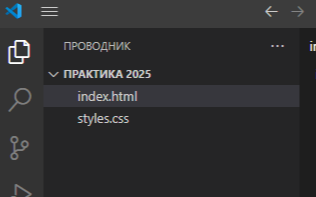


Рис.22.Созданные файлы внутри папки

В файл index.html была вставлена HTML-разметка, включающая:

* Шапку с навигацией и индикатором корзины
* Поле поиска товаров
* Список товаров с кнопками добавления
* Секцию корзины с перечнем выбранных позиций
* Блоки выбора способа доставки и оплаты
* Кнопку оформления заказа

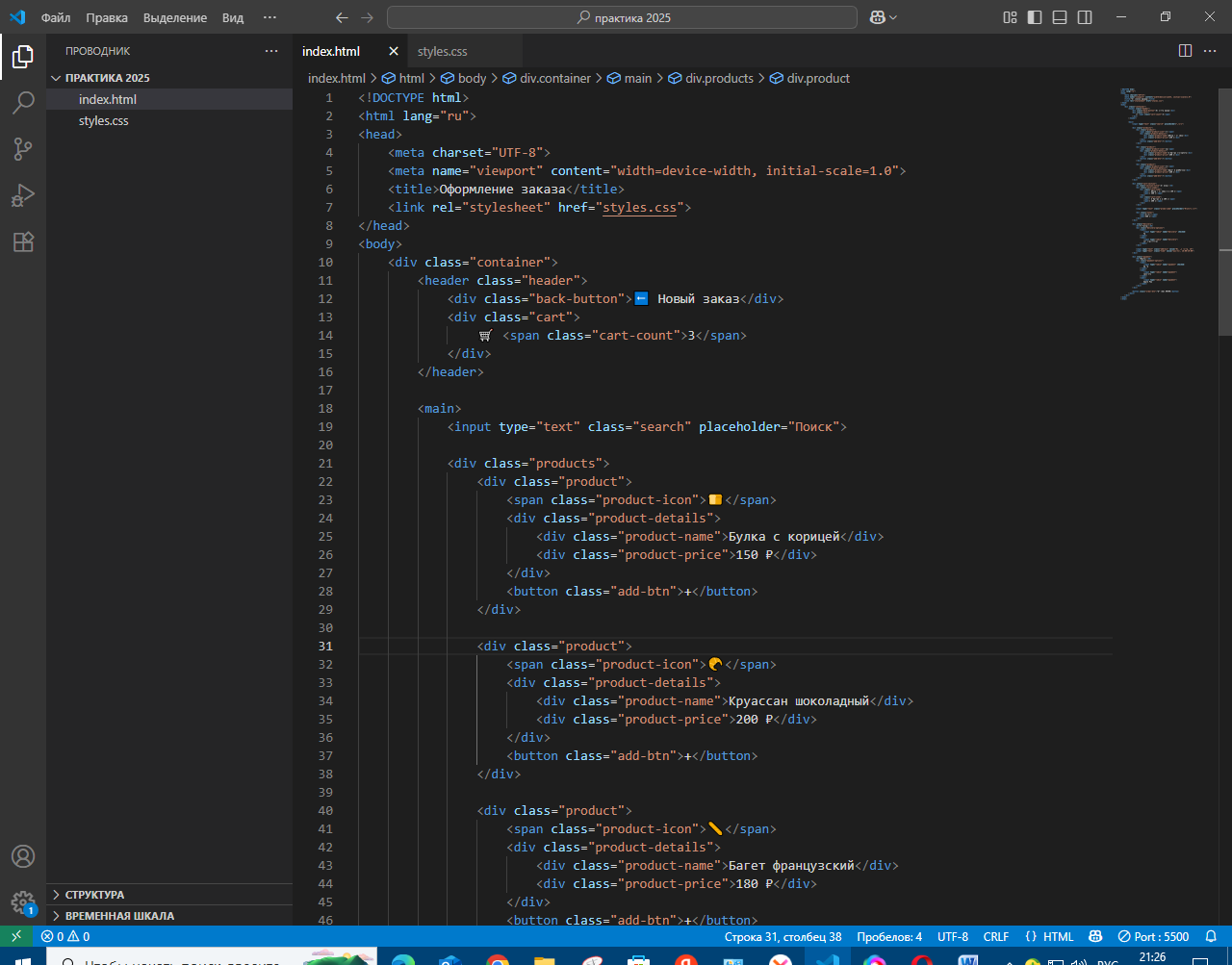


Рис.23.Html-код

В файл styles.css добавлены стили для всех элементов интерфейса:

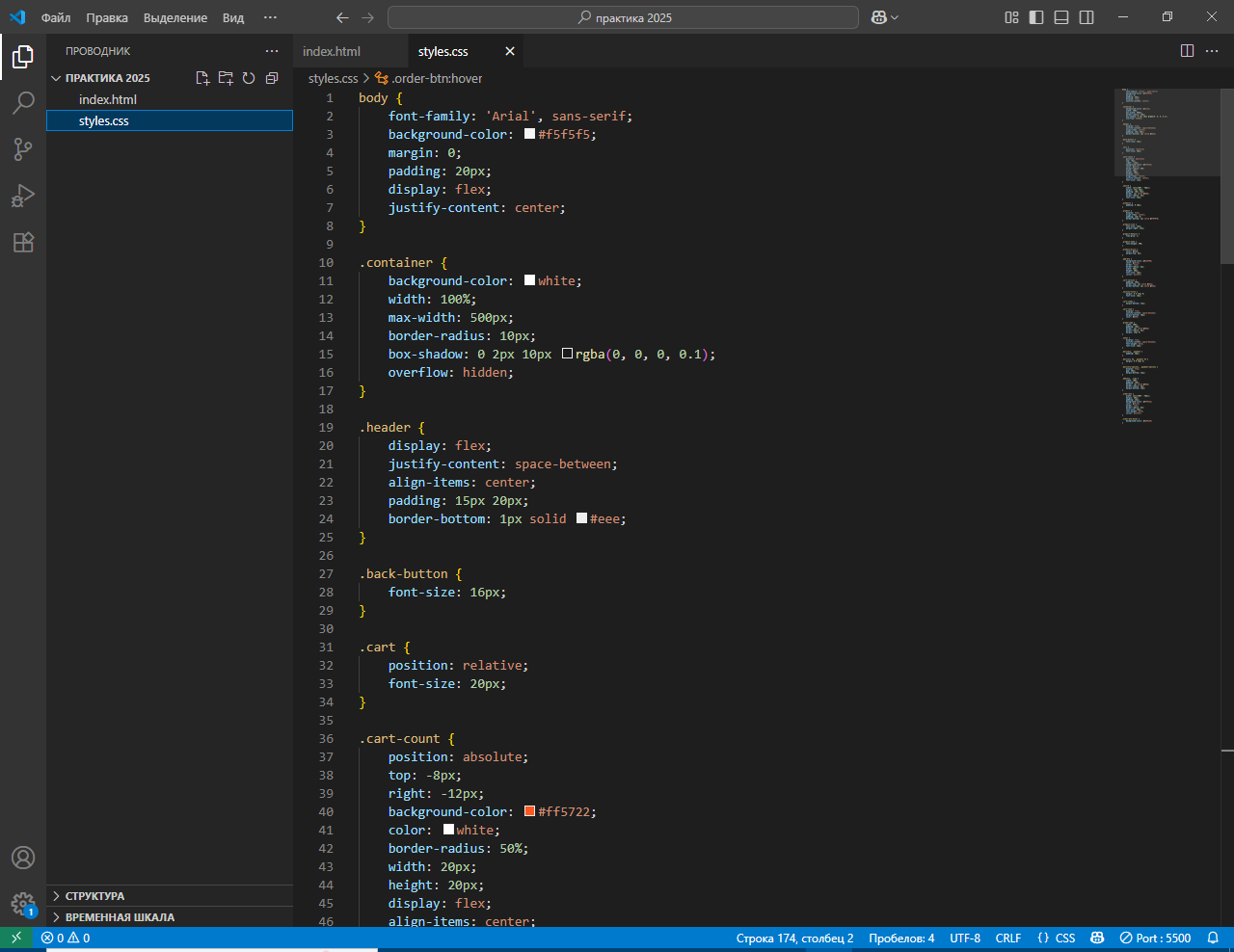


Рис.24.Css-код

После сохранения файлов проект был открыт в браузере Google Chrome. В итоге получен полностью рабочий прототип интерфейса для оформления заказа еды (см. ниже):

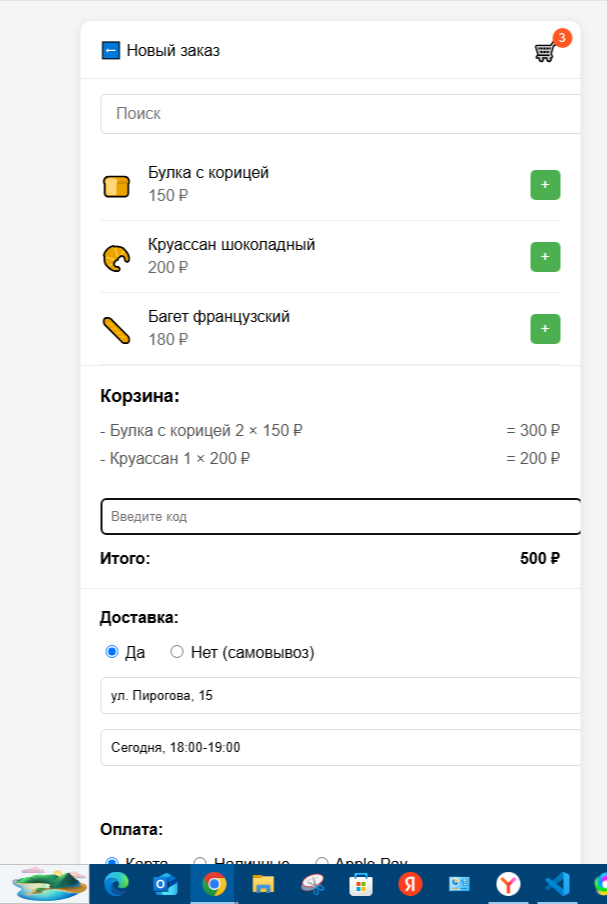


Рис.25.Прототип экрана «Создания заказа»

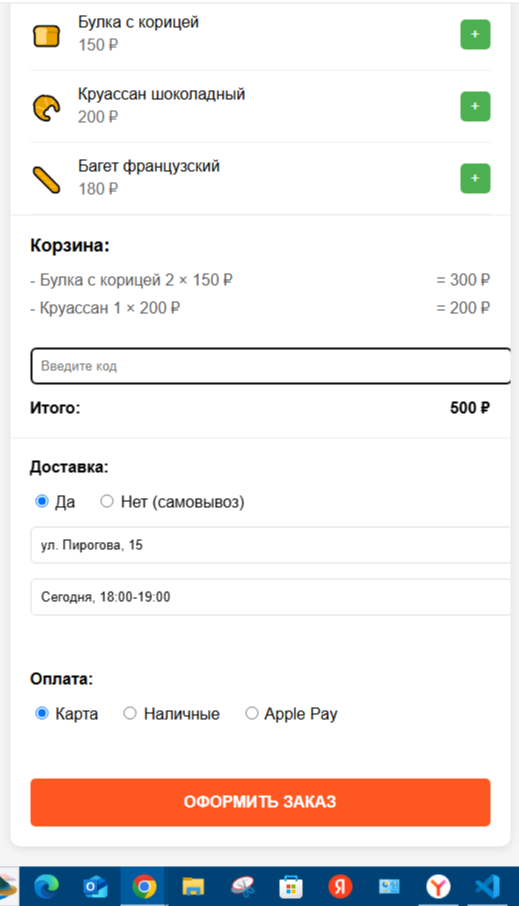


Рис.26.Прототип экрана «Создания заказа»

**Структура и функциональные блоки:**

**Прототип состоит из 4 ключевых секций:**

1. Каталог товаров (верхняя часть):

* Список доступной выпечки с ценами
* Каждая позиция содержит:
* Название товара
* Цену
* Кнопку добавления (+)

2. Корзина (средняя часть):

* Динамически обновляемый список выбранных товаров
* Указание количества и суммы по каждой позиции
* Поле для ввода промокода
* Итоговая стоимость

3. Способ получения:

* Переключатель между доставкой и самовывозом
* Соответствующие поля для адреса/времени

4. Оплата и подтверждение:

* Выбор способа оплаты
* Основная кнопка действия

**Пользовательский сценарий:**

1. Пользователь открывает приложение и переходит в раздел "Новый заказ".
2. Выбирает товары, нажимая "+".
3. Видит добавленные товары в корзине, может изменить количество.
4. Выбирает способ получения (доставка/самовывоз) и вводит данные.
5. Выбирает способ оплаты.
6. Нажимает "Оформить заказ" → переход на экран подтверждения.

**Задание №4**

**Тема: Функция редактирования заказа в мобильном приложении "Частная пекарня".**

**Общее описание функциональности**

**Цель функции:**

Функция редактирования заказа предназначена для:

1. Изменения количества товаров в заказе
2. Удаления позиций из заказа
3. Корректировки состава заказа перед окончательным подтверждением

**Условия доступности:**

Функция доступна пользователю при следующих условиях:

1. Статус "В корзине":

* Заказ не был отправлен на подтверждение
* Пользователь может свободно редактировать все параметры

2. Статус "Ожидает подтверждения":

* Заказ отправлен, но еще не принят пекарней
* Редактирование возможно в течение ограниченного времени (15 минут)
* Некоторые изменения могут быть ограничены (например, нельзя добавить товары с долгим сроком приготовления)

**Основные сценарии использования:**

1. Пользователь добавляет неправильное количество товара и хочет исправить
2. Пользователь передумал покупать определенный товар
3. Пользователь хочет добавить дополнительные товары к существующему заказу
4. Пользователь получает уведомление о недоступности товара и хочет изменить заказ

**UML Диаграммы**

**1. Диаграмма последовательности редактирования заказа:**

Для наглядного представления архитектуры и процессов системы при разработке функции редактирования заказа в приложении "Частная пекарня" были созданы UML-диаграммы с помощью PlantUML — специализированного инструмента, преобразующего текстовые описания в графические схемы.

Использованный код PlantUML представлен ниже на скриншоте:

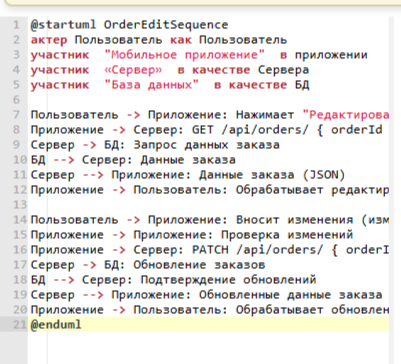
****

Рис.27. PlantUML-код диаграммы последовательности редактирования заказа

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма последовательности редактирования заказа.

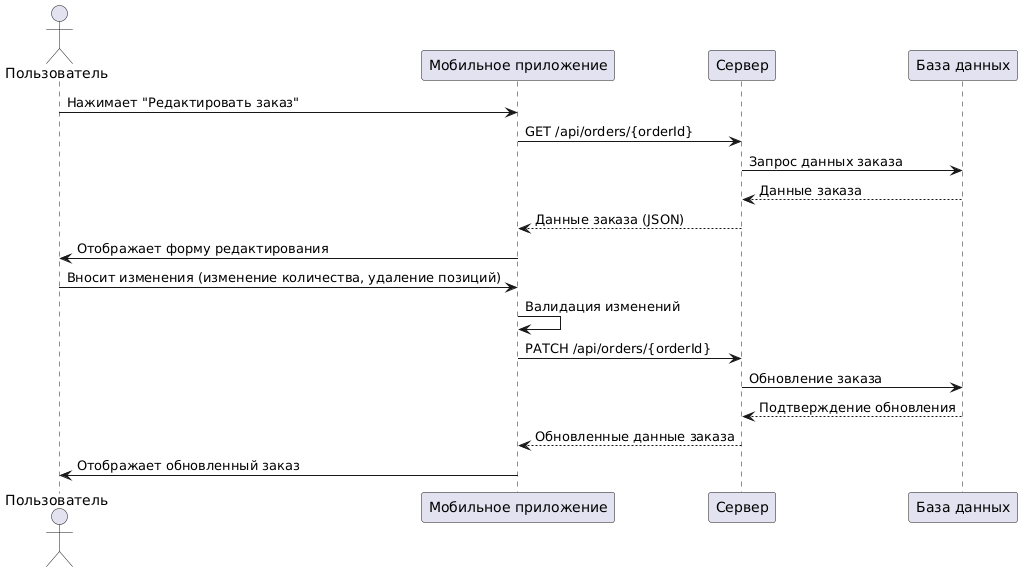


Рис.28. Диаграмма последовательности редактирования заказа

**Описание диаграммы последовательности редактирования:**

**Ключевые шаги:**

1) Инициация редактирования

* Пользователь нажимает кнопку "Редактировать заказ" в мобильном приложении.
* Приложение отправляет GET-запрос на сервер для получения текущих данных заказа (/api/orders/{orderId}).

2)Получение данных

* Сервер запрашивает информацию из базы данных и возвращает её в формате JSON.
* Приложение отображает форму редактирования с актуальными данными.

3) Внесение изменений

* Пользователь изменяет состав заказа (количество товаров, удаляет позиции).
* Приложение выполняет локальную валидацию изменений.

4) Сохранение изменений

* Приложение отправляет PATCH-запрос на сервер (/api/orders/{orderId}) с внесёнными правками.
* Сервер обновляет данные в БД и возвращает подтверждение.
* Пользователь видит обновлённый заказ.

**Роли компонентов:**

* Пользователь — инициирует действия через UI.
* Мобильное приложение — обрабатывает запросы, валидирует данные.
* Сервер — управляет бизнес-логикой и взаимодействует с БД.
* База данных — хранит актуальную информацию о заказах.

**2. Диаграмма состояний заказа:**

Использованный код PlantUML представлен ниже на скриншоте:

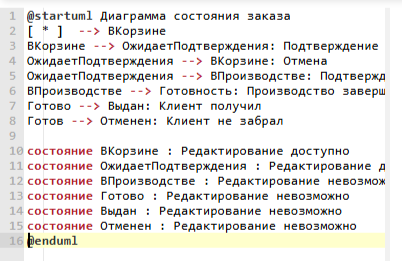


Рис.29. PlantUML-код диаграммы состояний заказа

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма состояний заказа.

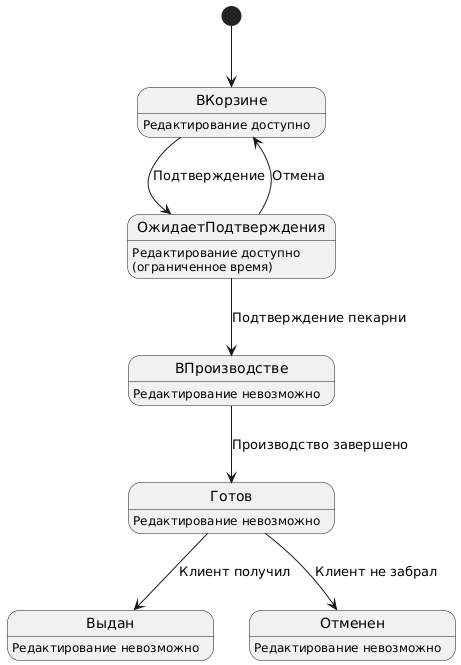


Рис.30. Диаграмма состояний заказа

**Описание диаграммы последовательности редактирования:**

**Ключевые состояния заказа:**

1) "В корзине"

* Статус: Заказ не оформлен, находится в корзине.
* Редактирование: Полностью доступно (можно менять состав, количество, удалять позиции).

2) "Ожидает подтверждения"

* Статус: Пользователь подтвердил заказ, но пекарня ещё не начала его выполнение.
* Редактирование: Доступно в течение ограниченного времени (например, 15 минут).
* Переходы:
* Подтверждение пекарни → статус "В производстве".
* Отмена → возврат в "В корзине".

3) "В производстве"

* Статус: Пекарня готовит заказ.
* Редактирование: Недоступно (изменения запрещены).
* Переход: При завершении → статус "Готов".

4) "Готов"

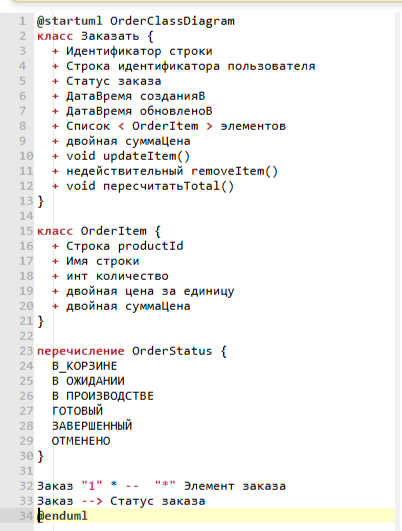
* Статус: Заказ приготовлен, ожидает выдачи.
* Редактирование: Недоступно.
* Переходы:
* Клиент получил → статус "Выдан".
* Клиент не забрал → статус "Отменен".

5) Финальные статусы ("Выдан", "Отменен")

* Редактирование: Недоступно (заказ завершён или отменён).

**3. Диаграмма классов:**

Использованный код PlantUML представлен ниже на скриншоте:

****Рис.31. PlantUML-код диаграммы классов

В результате преобразования PlantUML-кода была автоматически сгенерирована представленная ниже диаграмма классов.

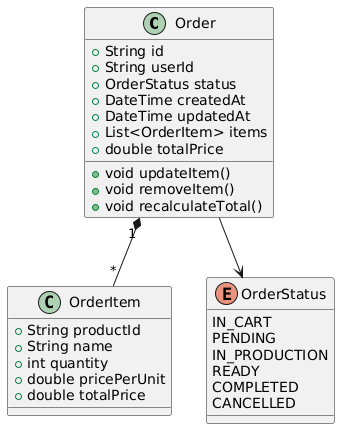


Рис.32. Диаграмма классов

**Описание диаграммы классов заказа:**

**Основные классы:**

1) Класс Order (Заказ)

Атрибуты:

* id - уникальный идентификатор заказа
* userId - идентификатор пользователя-владельца
* status - текущий статус (тип OrderStatus)
* createdAt/updatedAt - метки времени
* items - список позиций заказа
* totalPrice - общая сумма

Методы:

* updateItem() - изменение позиции
* removeItem() - удаление позиции
* recalculateTotal() - пересчет суммы

2) Класс OrderItem (Позиция заказа)

Атрибуты:

* productId - идентификатор товара
* name - название товара
* quantity - количество
* pricePerUnit - цена за единицу
* totalPrice - сумма по позиции

3) Перечисление OrderStatus (Статусы заказа)

Возможные значения:

* IN\_CART - в корзине
* PENDING - ожидает подтверждения
* IN\_PRODUCTION - в производстве
* READY - готов к выдаче
* COMPLETED - завершен
* CANCELLED - отменен

**Связи между классами:**

* Заказ (Order) содержит список позиций (OrderItem) - отношение композиции (1 ко многим)
* Заказ использует перечисление OrderStatus для хранения состояния

**API Спецификация**

**1. Получение данных заказа (Order Details Response)**

Метод: GET

Endpoint: /api/orders/{orderId}

Параметры: orderId (path) - идентификатор заказа

****

Рис.33. Успешный ответ (200 OK)

Коды ошибок:

* 403 Forbidden - нет прав доступа
* 404 Not Found - заказ не существует

**2. Обновление заказа (Order Update Response)**

Метод: PATCH

Endpoint: /api/orders/{orderId}

Параметры: orderId (path) - идентификатор заказа

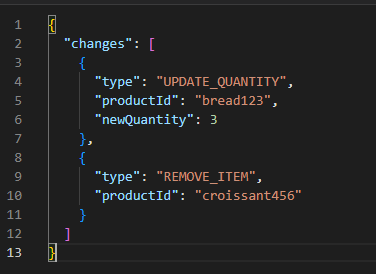


Рис.34. Тело запроса



Рис.35.Успешный ответ (200 OK)

Коды ошибок:

* 400 Bad Request - неверный формат запроса
* 403 Forbidden - редактирование запрещено
* 409 Conflict - конфликт при обновлении

**Бизнес-логика на сервере**

**1. Проверка прав доступа:**

* Только владелец заказа может его редактировать
* Заказы в статусах кроме "В корзине" и "Ожидает подтверждения" не могут быть изменены

**2. Валидация изменений:**

* Проверка наличия товаров в меню
* Проверка доступного количества (не превышает лимиты пекарни)
* Проверка минимальной суммы заказа (если есть)

**3. Пересчет суммы:**

* При изменении количества или состава пересчитывается общая сумма
* Применяются актуальные цены (на случай если они изменились)

**Хранение информации о покупках**

Структура базы данных:

1. Таблица orders:

* id (PK)
* user\_id (FK)
* status (enum)
* total\_price (decimal)
* created\_at (timestamp)
* updated\_at (timestamp)

2. Таблица order\_items:

* id (PK)
* order\_id (FK)
* product\_id (FK)
* quantity (integer)
* price\_per\_unit (decimal)
* total\_price (decimal)

**Ограничения и особые случаи**

1) Временное окно редактирования: После перехода в статус "Ожидает подтверждения" редактирование доступно только в течение 15 минут

2) Изменение цен: При редактировании применяются текущие цены, а не цены на момент создания заказа

3) Доступность товаров: Если товар больше не доступен, он автоматически удаляется из заказа с уведомлением пользователя

4) Минимальная сумма: После редактирования сумма заказа не должна быть меньше установленного минимума (например, 500 руб.)

**Задание №5**

**Тема: SQL-запросы для приложения "Частная пекарня".**

Цель задания: На основе предоставленной реляционной модели данных (см.ниже),составим SQL-запросы для выполнения поставленных задач.

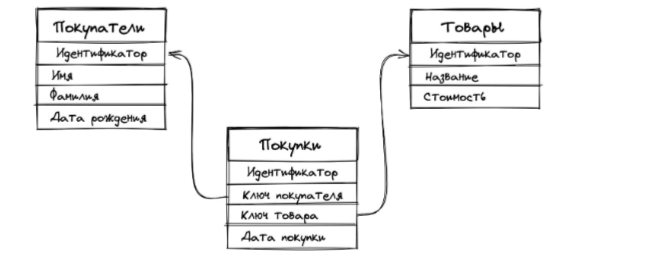


Рис.36. Реляционная модель данных

Для выполнения данного задания воспроизведем реляционную модель данных в Microsoft Access 2016. В ходе работы создадим три ключевые таблицы:

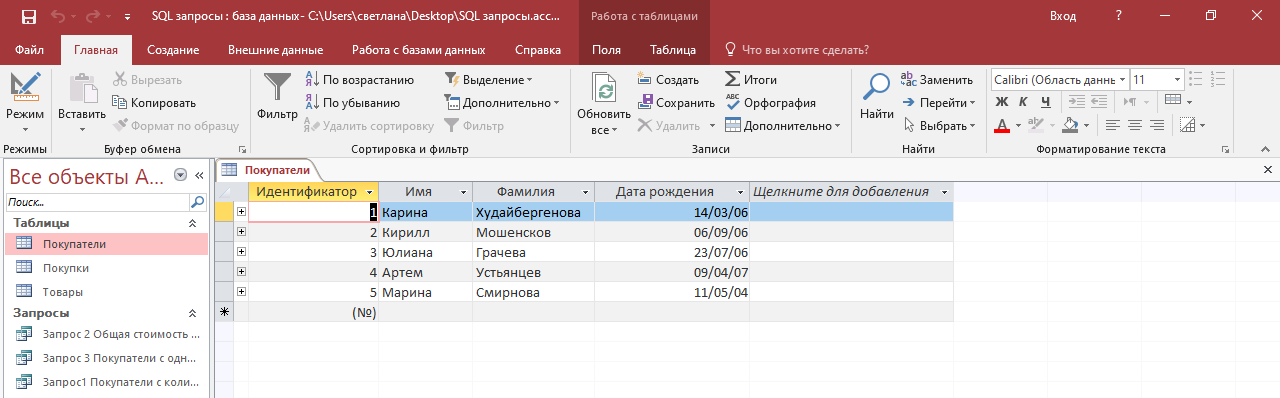


Рис.37.Таблица «Покупатели»

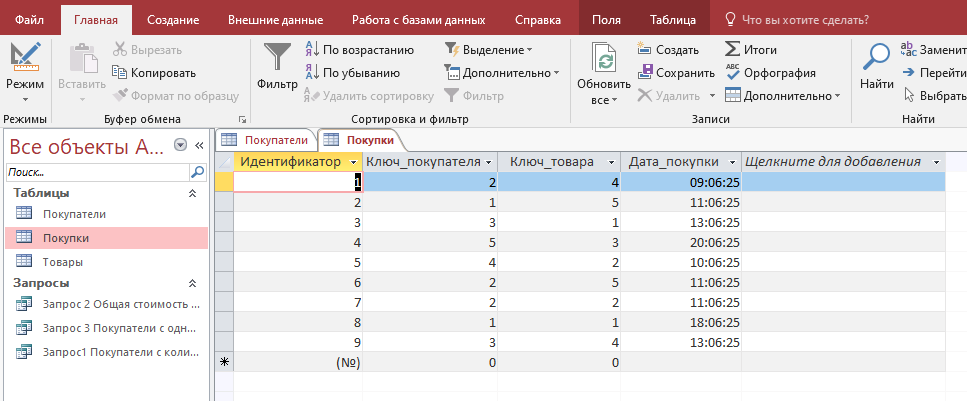


Рис.38. Таблица «Покупки»

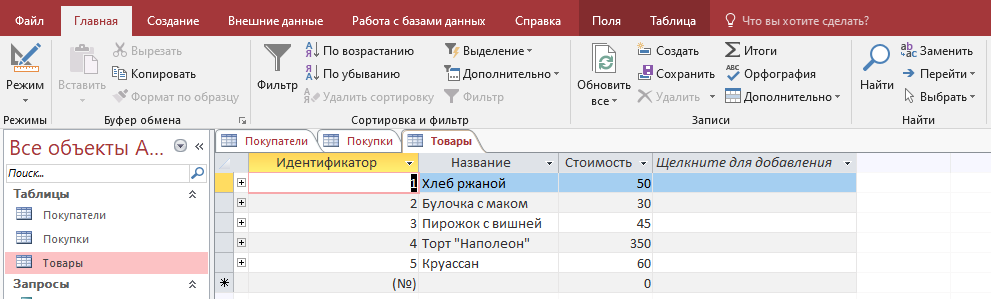


Рис.39. Таблица «Товары»

После создания таблиц между ними установим соответствующие связи в схеме данных.

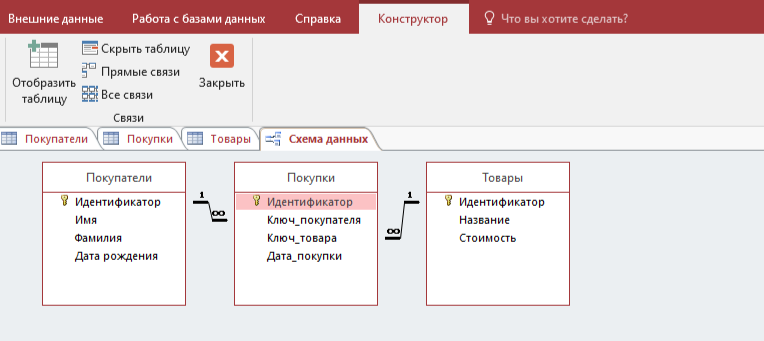


Рис.40. Схема данных

Такой подход обеспечит сохранение целостности информации и корректное выполнение SQL-запросов.

**SQL-запросы**

На основе созданной структуры базы данных разработаем три специализированных запроса через конструктор запросов в режиме SQL:

**1. Вывести покупателей с количеством осуществленных покупок:**

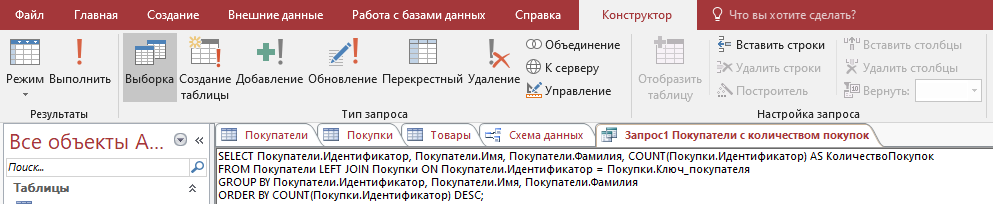
****

Рис.41.Запрос 1: Покупатели с количеством покупок

После ввода SQL-кода нажимаем "Выполнить" (!) для проверки:

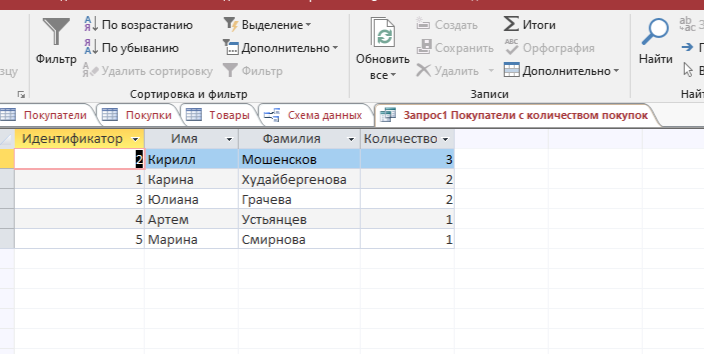


Рис.42.Результат запроса 1

Что делает этот запрос:

* Выводит список всех покупателей с подсчетом их покупок
* Использует LEFT JOIN для включения покупателей без покупок (им будет показано 0)
* Группирует данные по покупателям (GROUP BY)
* Считает количество записей в таблице Покупки для каждого покупателя (COUNT)
* Сортирует результат по количеству покупок в порядке убывания (DESC)

**2. Общая стоимость товаров для каждого покупателя и отсортировать результат в порядке убывания:**

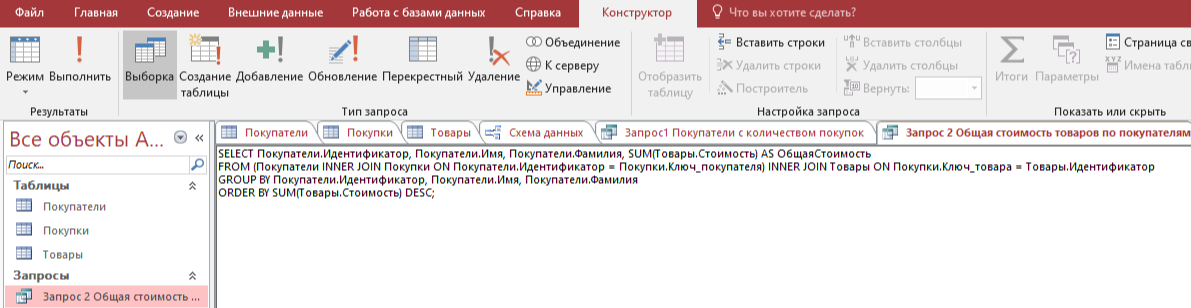
****

Рис.43. Запрос 2: Общая стоимость товаров для каждого покупателя (сортировка по убыванию)

После ввода SQL-кода нажимаем "Выполнить" (!) для проверки:

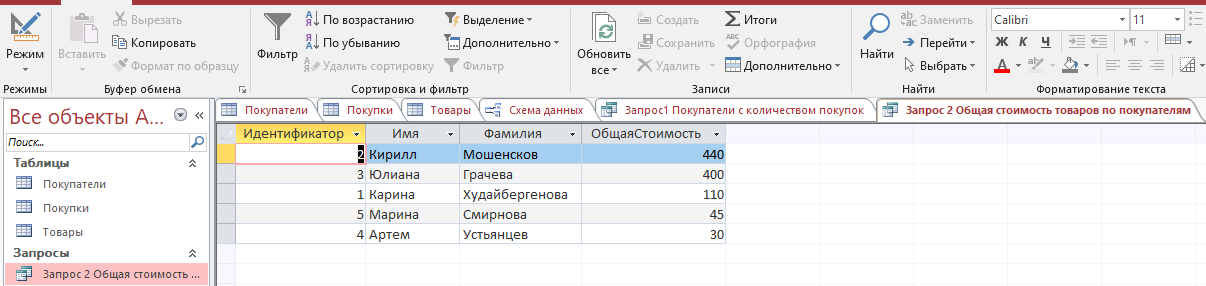


Рис.44.Результат запроса 2

Что делает этот запрос:

* Вычисляет общую сумму всех покупок для каждого покупателя
* Соединяет три таблицы: Покупатели → Покупки → Товары
* Использует INNER JOIN - показываются только покупатели с покупками
* Суммирует стоимость товаров (SUM) для каждого покупателя
* Сортирует по сумме расходов от большего к меньшему

**3. Получить покупателей, купивших только один товар:**

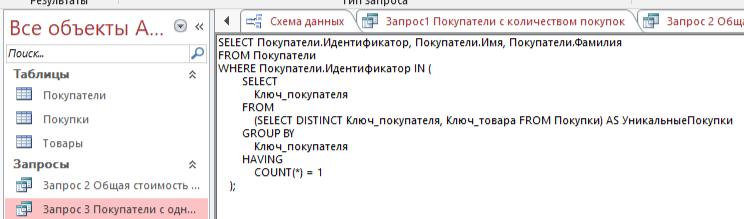
****

Рис.45. Запрос 3: Покупатели, купившие только один товар

После ввода SQL-кода нажимаем "Выполнить" (!) для проверки:

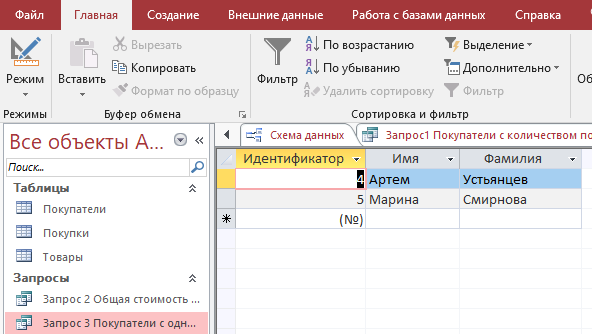
****

Рис.46.Результат запроса 3

Что делает этот запрос:

* Находит покупателей, которые приобретали только один тип товара (может несколько раз)
* Сначала выбирает уникальные комбинации покупатель-товар (DISTINCT)
* Затем группирует по покупателям и оставляет только тех, у кого 1 уникальный товар (HAVING COUNT(\*) = 1)
* Возвращает данные только этих покупателей