Аналитика

**1.Необходимо описать бизнес-процесс создания заказа (использовать любую удобную нотацию моделирования бизнес-процессов).**

[](https://www.figma.com/board/QJnvZMfZrh8LKc8XK5BiBn/%D0%91%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0-%D0%BE%D0%BF%D0%B8%D1%81%D1%8B%D0%B2%D0%B0%D1%8E%D1%89%D0%B0%D1%8F-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81?node-id=0-1&t=jYQm9bV9KHI3vBLK-1)

**2.Опишите процесс синхронизации данных между клиентом и сервером(создание, редактирование и отмена заказа, изменение персональных данных, оплата заказа и т.д.). Представить все в диаграммах UML, API методах и других представлениях, также составить ER-диаграмму сущностей.**

ER-диаграмма

CUSTOMER {

int customer\_id PK

varchar name

varchar email

varchar phone

varchar address

}

PRODUCT {

int product\_id PK

varchar name

text description

decimal price

varchar image\_url

int category\_id FK

}

CATEGORY {

int category\_id PK

varchar name

}

ORDER {

int order\_id PK

int customer\_id FK

datetime order\_date

varchar delivery\_address

varchar delivery\_type

varchar payment\_method

varchar status

decimal total\_price

}

ORDER\_ITEM {

int order\_item\_id PK

int order\_id FK

int product\_id FK

int quantity

decimal price

}

CUSTOMER ||--o{ ORDER : places

PRODUCT ||--o{ ORDER\_ITEM : includes

ORDER ||--o{ ORDER\_ITEM : contains

CATEGORY ||--o{ PRODUCT: includes

Uml диаграмма компонентов

componentDiagram

ClientApp [Мобильное приложение]

Backend [Серверное приложение]

Database [База данных]

API [API]

ClientApp -down-> API : Запросы

API -right-> Backend : Запросы

Backend -right-> Database : Чтение/Запись

Database -left-> Backend : Данные

Backend -left-> API : Ответы

API -up-> ClientApp : Ответы

mermaid

Описание компонентов:

* ClientApp: Мобильное приложение на устройстве пользователя.
* API: RESTful API, через которое общается мобильное приложение с сервером.
* Backend: Серверная часть приложения, обрабатывает запросы и взаимодействует с базой данных.
* Database: База данных, где хранятся все данные приложения.

UML диаграммы различных процессов для пекарни

1.Составление заказа

sequenceDiagram

participant Client as ClientApp

participant API

participant Backend

participant DB as Database

Client->>API: POST /orders {order\_data}

API->>Backend: {order\_data}

Backend->>DB: Запись заказа и order\_items

DB-->>Backend: order\_id

Backend->>API: {status:"success", order\_id}

API-->>Client: {status:"success", order\_id}

Описание:

1. Клиент отправляет POST запрос на /orders c данными заказа (order\_data)
2. API передает запрос на сервер
3. Сервер обрабатывает данные и сохраняет заказ в базу данных
4. Сервер возвращает ответ API с статусом успеха и order\_id
5. API передает ответ на клиент
6. Клиент получает подтверждение

2.Редактирование заказа

sequenceDiagram

participant Client as ClientApp

participant API

participant Backend

participant DB as Database

Client->>API: PUT /orders/{order\_id} {order\_data}

API->>Backend: {order\_data}

Backend->>DB: Обновление заказа и order\_items

DB-->>Backend:

Backend->>API: {status:"success"}

API-->>Client: {status:"success"}

Описание:

1. Клиент отправляет PUT запрос на /orders/{order\_id} с обновленными данными заказа.
2. API передает запрос на сервер.
3. Сервер обрабатывает данные и обновляет заказ в базе данных.
4. Сервер возвращает API ответ с статусом успеха.
5. API передает ответ на клиент
6. Клиент получает подтверждение

3. Отмена заказа:

sequenceDiagram

participant Client as ClientApp

participant API

participant Backend

participant DB as Database

Client->>API: DELETE /orders/{order\_id}

API->>Backend:

Backend->>DB: Обновление заказа (статус отменен)

DB-->>Backend:

Backend->>API: {status:"success"}

API-->>Client: {status:"success"}

Описание:

1. Клиент отправляет DELETE запрос на /orders/{order\_id}.
2. API передает запрос на сервер.
3. Сервер обновляет статус заказа в базе данных на “отменен”.
4. Сервер возвращает API ответ с статусом успеха.
5. API передает ответ на клиент
6. Клиент получает подтверждение

4. Изменение персональных данных:

sequenceDiagram

participant Client as ClientApp

participant API

participant Backend

participant DB as Database

Client->>API: PUT /customers/{customer\_id} {customer\_data}

API->>Backend: {customer\_data}

Backend->>DB: Обновление данных клиента

DB-->>Backend:

Backend->>API: {status:"success"}

API-->>Client: {status:"success"}

Описание:

1. Клиент отправляет PUT запрос на /customers/{customer\_id} с новыми данными клиента.
2. API передает запрос на сервер.
3. Сервер обрабатывает данные и обновляет данные клиента в базе данных.
4. Сервер возвращает API ответ с статусом успеха.
5. API передает ответ на клиент
6. Клиент получает подтверждение

5. Оплата заказа (представление для одного метода оплаты):

sequenceDiagram

participant Client as ClientApp

participant API

participant Backend

participant PaymentGateway

participant DB as Database

Client->>API: POST /payments/ {payment\_data, order\_id}

API->>Backend: {payment\_data, order\_id}

Backend->>PaymentGateway: {payment\_data}

PaymentGateway-->>Backend: {status:"success", payment\_id}

Backend->>DB: Обновление заказа (статус оплачен, payment\_id)

DB-->>Backend:

Backend->>API: {status:"success", payment\_id}

API-->>Client: {status:"success", payment\_id}

Описание:

1. Клиент отправляет POST запрос на /payments/ с данными оплаты.
2. API передает запрос на сервер.
3. Сервер инициирует оплату через платежный шлюз.
4. Платежный шлюз возвращает ответ на сервер.
5. Сервер обновляет данные заказа в базе данных и отправляет данные клиенту.
6. Клиент получает ответ с подтверждением

API Методы (RESTful API)

Вот примеры:

* GET /products: Получить список всех продуктов.
* GET /products/{product\_id}: Получить информацию о конкретном продукте.
* GET /categories: Получить список всех категорий.
* POST /orders: Создать новый заказ (принимает данные заказа).
* GET /orders/{order\_id}: Получить информацию о конкретном заказе.
* PUT /orders/{order\_id}: Обновить информацию о конкретном заказе.
* DELETE /orders/{order\_id}: Отменить заказ.
* GET /customers/{customer\_id}: Получить информацию о клиенте.
* PUT /customers/{customer\_id}: Обновить информацию о клиенте.
* POST /payments: Создать новый платеж (принимает данные платежа)
* GET /orders/customer/{customer\_id}: Получить список заказов конкретного клиента
* GET /orders/status/{status}: Получить список заказов конкретного статуса

**3. Подготовить прототип одного из экранов данного мобильного приложения и описать пользовательский интерфейс для данного экрана (например, создание заказа).**

[](https://www.figma.com/design/aL4d9b3Rtlz154YOWIgnib/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF?node-id=0-1&t=klzcuMEia9pukeTx-1)

В данном прототипе есть возможность взаимодействовать с:

1. Кнопка “Корзина” она ведет на страницу, которую вы видете(т.е. на страницу формирования заказа)
2. Кнопка “Бургер”, которая выводит меню на другие ссылки
3. Кнопка “Лого”, которая выводит пользователя на главную страницу
4. Дальше идет информация о товаре(кол-во штук, название)
5. Поля ввода,которые необходимо ввести для заказа(Фио,Телефон, Адрес и E-mail)
6. Итог(сколько выходит стоимость товара)
7. Кнопка “Отправить“ , которая отправляет всю информацию пекарям для готовки выпечки

**4. Подготовить подробное описание функции редактирования заказа , которую можно было бы использовать в качестве постановки задачи для разработки (помимо текстового описания, использовать UML диаграммы, указать используемые API методы, передаваемые и получаемые параметры, описать процесс хранения информации о покупках пользователя).**

1. Цель

Разработать функциональность редактирования заказов в мобильном клиент-серверном приложении “Частная пекарня”. Функция должна позволять пользователям изменять состав заказа (добавлять, удалять и менять количество товаров) после его создания, но до момента обработки заказа пекарней.

2. Функциональные требования

* Пользователь должен иметь возможность просматривать детали своего созданного, но не обработанного заказа.
* Пользователь должен иметь возможность добавлять новые товары в заказ.
* Пользователь должен иметь возможность удалять товары из заказа.
* Пользователь должен иметь возможность изменять количество каждого товара в заказе.
* После внесения изменений, пользователь должен иметь возможность сохранить изменения.
* Приложение должно отображать обновлённую стоимость заказа после каждого изменения.
* Приложение должно уведомлять пользователя об успешном сохранении изменений или об ошибках.
* Система должна предотвращать редактирование заказов, обработка которых уже началась в пекарне (заказы со статусом “в обработке”, “выполняется” и т.д.).

3. Нефункциональные требования

* Функция должна быть интуитивно понятна и удобна в использовании.
* Функция должна работать быстро и эффективно.
* Функция должна быть надежной и устойчивой к ошибкам.
* Функция должна быть защищена от несанкционированного доступа.

4. UML Диаграммы

4.1. Диаграмма классов:

@startuml

class Order {

- orderId : int

- customerId : int

- orderDate : DateTime

- status : string

- totalPrice : decimal

+ getOrderItems() : List<OrderItem>

+ updateOrderItem(orderItem : OrderItem) : void

+ addOrderItem(orderItem : OrderItem) : void

+ removeOrderItem(orderItemId : int) : void

+ calculateTotalPrice() : decimal

}

class OrderItem {

- orderItemId : int

- orderId : int

- productId : int

- quantity : int

- price : decimal

+ getProductName() : String

}

class Product {

- productId : int

- productName : string

- price : decimal

+getProductName():String

}

Order "1" \*-- "\*" OrderItem : contains

OrderItem "1" -- "1" Product : refers to

@enduml

4.2. Диаграмма последовательности:

@startuml

actor User

participant ClientApp

participant API

participant Backend

participant Database

User -> ClientApp: Запрос на редактирование заказа

activate ClientApp

ClientApp -> API: PUT /orders/{orderId} {updatedOrderData}

activate API

API -> Backend: {updatedOrderData}

activate Backend

Backend -> Database: Обновление информации о заказе

activate Database

Database --> Backend:

deactivate Database

Backend -> API: {success/error}

deactivate Backend

API --> ClientApp: {success/error}

deactivate API

ClientApp -> User: Ответ (успех/ошибка)

deactivate ClientApp

@enduml

**5. API Методы**

5.1. PUT /orders/{orderId}

* Метод: PUT
* URL: /orders/{orderId} (где {orderId} — ID редактируемого заказа)
* Заголовок: Authorization: Bearer {токен} (для авторизации)
* Тело запроса (updatedOrderData):
  + orderItems: Массив объектов OrderItem. Каждый объект содержит:
    - orderItemId: ID позиции в заказе (если позиция уже существует, иначе null).
    - productId: ID товара.
    - quantity: Новое количество товара.
* Ответ:
  + При успехе: {status: "success", updatedOrder: {orderId, totalPrice, orderItems}}
  + При ошибке: {status: "error", message: "Сообщение об ошибке"}

6. Передаваемые и получаемые параметры

Запрос (PUT /orders/{orderId}):

* orderId: ID редактируемого заказа (целое число).
* updatedOrderData: JSON-объект, содержащий обновлённую информацию о заказе (см. выше).

Ответ:

* status: Строка, указывающая на успех ("success") или ошибку ("error").
* updatedOrder: (только при status: "success") Обновлённые данные заказа, включая orderId, totalPrice и orderItems.
* message: (только при status: "error") Строка с описанием ошибки.

7. Хранение информации о покупках пользователя

Информация о покупках пользователя хранится в базе данных. Основная таблица — Order, связанная с таблицей OrderItem (для деталей о товарах в заказе). Таблица Order содержит информацию о самом заказе (дата, статус, общая стоимость, способ оплаты, адрес доставки и т.д.), а таблица OrderItem содержит информацию о конкретных товарах, включенных в заказ (ID товара, количество, цена). В таблице Customer хранится информация о пользователе. Связи между таблицами реализуются с помощью внешних ключей (customerId в Order, orderId в OrderItem).

8. Обработка ошибок

Функция должна обрабатывать следующие ошибки:

* Заказ не найден.
* Пользователь не имеет прав на редактирование этого заказа (например, заказ уже обработан пекарней).
* Ошибка обновления данных в базе данных.
* Некорректные данные в запросе.
* Товар недоступен.

5. SQL-запросы:

1. Вывести покупателей с количеством осуществленных покупок

**SELECT**

c.CustomerID,

c.FirstName,

c.LastName,

COUNT(p.PurchaseID) **AS** PurchaseCount

**FROM**

Customers c

**LEFT** **JOIN**

Purchases p **ON** c.CustomerID = p.CustomerID

**GROUP** **BY**

c.CustomerID, c.FirstName, c.LastName

**ORDER** **BY**

PurchaseCount **DESC**;

2. Общую стоимость товаров для каждого покупателя и отсортировать результат в порядке убывания

**SELECT**

c.CustomerID,

c.FirstName,

c.LastName,

SUM(pr.Price) **AS** TotalSpent

**FROM**

Customers c

**JOIN**

Purchases p **ON** c.CustomerID = p.CustomerID

**JOIN**

Products pr **ON** p.ProductID = pr.ProductID

**GROUP** **BY**

c.CustomerID, c.FirstName, c.LastName

**ORDER** **BY**

TotalSpent **DESC**;

3. Получить покупателей, купивших только один товар

**SELECT**

c.CustomerID,

c.FirstName,

c.LastName

**FROM**

Customers c

**JOIN**

Purchases p **ON** c.CustomerID = p.CustomerID

**GROUP** **BY**

c.CustomerID, c.FirstName, c.LastName

**HAVING**

COUNT(**DISTINCT** p.ProductID) = 1;