Багдасарян Юрий Арамаисович ИСдо-42

**Дано:**

Мобильное клиент-серверное приложение "Частная пекарня" (приложение для продажи выпечки)

Задание 2.API методы

1. Создание заказа

{

"userId": "string",

"products": [

{

"productId": "string",

"quantity": "integer"

}

],

"deliveryAddress": "string",

"paymentMethod": "string"

}

Ответ:

{

"orderId": "string",

"status": "created"

}

1. Редактирование заказа

{

"products": [

{

"productId": "string",

"quantity": "integer"

}

]

}

Ответ:

{

"status": "updated"

}

1. Отмена заказа

Ответ:

{

"status": "canceled"

}

1. Изменение персональных данных

{

"name": "string",

"phone": "string",

"address": "string"

}

Ответ:

{

"status": "updated"

}

1. Оплата заказа

{

"paymentMethod": "string",

"amount": "number"

}

Ответ:

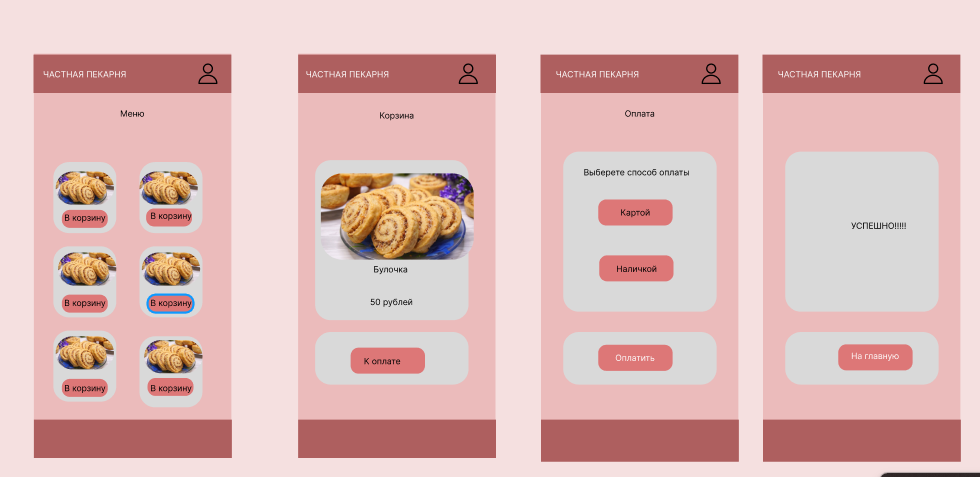
{

"status": "paid"

}

Задание 3.

https://www.figma.com/design/WzCW7m6SHacqwfRrwJbpxd/Untitled?node-id=0-1&p=f&t=6gQKBA4FgN9UgNmz-0



Задание 4.

Описание функции редактирования заказа

Цель функции

Функция редактирования заказа позволяет пользователю изменять детали своего заказа, такие как количество продуктов, адрес доставки и способ оплаты. Это обеспечивает гибкость и удобство для пользователей, позволяя им вносить изменения до момента оформления заказа.

Основные сценарии использования:

1. Пользователь открывает существующий заказ.

2. Пользователь редактирует количество продуктов.

3. Пользователь изменяет адрес доставки.

4. Пользователь выбирает другой способ оплаты.

5. Пользователь сохраняет изменения.

6. Пользователь отменяет редактирование и возвращается к предыдущему экрану.

Используемые API методы

1. Получение деталей заказа

Метод: GET /api/orders/{orderId}

Параметры:

orderId: идентификатор заказа

Возвращаемые параметры:

orderDetails: объект с деталями заказа (список продуктов, адрес, способ оплаты)

2. Обновление количества продукта

Метод: PATCH /api/orders/{orderId}/products

Параметры:

orderId: идентификатор заказа

productId: идентификатор продукта

quantity: новое количество продукта

Возвращаемые параметры:

success: статус выполнения операции (успех/неудача)

3. Обновление адреса доставки

Метод: PATCH /api/orders/{orderId}/address

Параметры:

orderId: идентификатор заказа

address: новый адрес доставки

Возвращаемые параметры:

success: статус выполнения операции (успех/неудача)

4. Обновление способа оплаты

Метод: PATCH /api/orders/{orderId}/payment

Параметры:

orderId: идентификатор заказа

paymentMethod: новый способ оплаты (наличными/картой)

Возвращаемые параметры:

success: статус выполнения операции (успех/неудача)

5. Сохранение изменений в заказе

Метод: PUT /api/orders/{orderId}

Параметры:

orderId: идентификатор заказа

updatedOrderDetails: объект с обновленными деталями заказа

Возвращаемые параметры:

success: статус выполнения операции (успех/неудача)

updatedOrder: обновленный объект заказа

Процесс хранения информации о покупках пользователя

1. Создание таблицы заказов в базе данных:

Таблица будет содержать следующие поля:

orderId: уникальный идентификатор заказа

userId: идентификатор пользователя, который сделал заказ

products: JSON-массив с продуктами и их количеством

address: адрес доставки

paymentMethod: способ оплаты

status: статус заказа (например, "в процессе", "доставлен", "отменен")

createdAt: дата и время создания заказа

updatedAt: дата и время последнего обновления заказа

2. Сохранение изменений:

При каждом изменении заказа (например, изменение количества продукта или адреса) данные будут обновляться в соответствующих полях таблицы.

Для этого будет использоваться метод PUT, который заменяет старые данные на новые.

3. История заказов:

Пользователи смогут просматривать свою историю заказов, где будут отображаться все заказы с их статусами и деталями.

Это может быть реализовано через отдельный API-метод, который возвращает список заказов для конкретного пользователя.

UML последовательности:



Задание 5.

1. Вывести всех покупателей с количеством заказов

SELECT

c.customer\_id,

c.customer\_name,

c.customer\_surname,

COUNT(o.order\_id) AS order\_count

FROM

Customers c

LEFT JOIN

Orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id, c.customer\_name, c.customer\_surname

ORDER BY

order\_count DESC;

2. Вывести общую сумму покупок для каждого покупателя

SELECT

c.customer\_id,

c.customer\_name,

c.customer\_surname,

SUM(oi.price) AS total\_spent

FROM

Customers c

LEFT JOIN

Orders o ON c.customer\_id = o.customer\_id

LEFT JOIN

Order\_Items oi ON o.order\_id = oi.order\_id

GROUP BY

c.customer\_id, c.customer\_name, c.customer\_surname

ORDER BY

total\_spent DESC;

3. Найти покупателей, которые сделали только один заказ

SELECT

c.customer\_id,

c.customer\_name,

c.customer\_surname

FROM

Customers c

JOIN

Orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id, c.customer\_name, c.customer\_surname

HAVING

COUNT(o.order\_id) = 1;

4. Получить список товаров, купленных каждым покупателем

SELECT

c.customer\_id,

c.customer\_name,

c.customer\_surname,

oi.product,

SUM(oi.price) AS total\_price

FROM

Customers c

JOIN

Orders o ON c.customer\_id = o.customer\_id

JOIN

Order\_Items oi ON o.order\_id = oi.order\_id

GROUP BY

c.customer\_id, c.customer\_name, c.customer\_surname, oi.product

ORDER BY

c.customer\_id, oi.product;

5. Вывести заказы с общей стоимостью каждого заказа

SELECT

o.order\_id,

o.customer\_id,

SUM(oi.price) AS order\_total

FROM

Orders o

JOIN

Order\_Items oi ON o.order\_id = oi.order\_id

GROUP BY

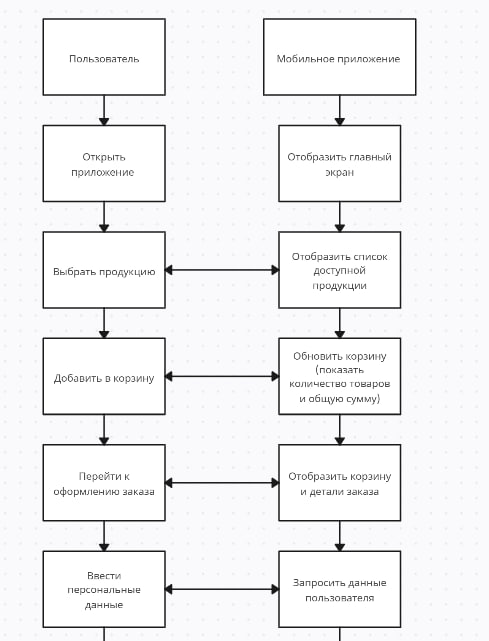
o.order\_id, o.customer\_id

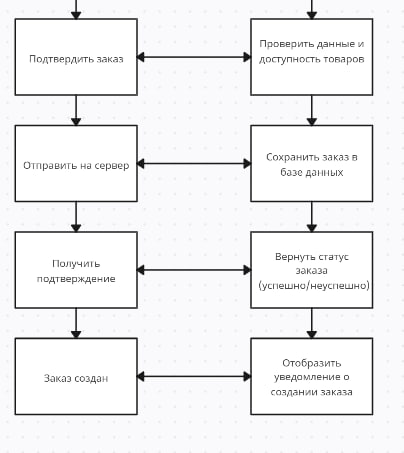
ORDER BY

order\_total DESC;

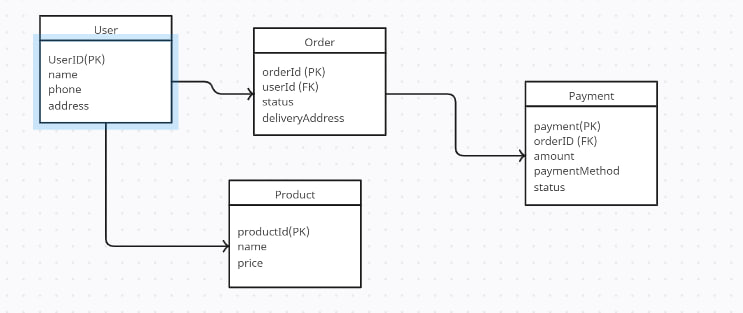
Диаграммы:

Задание 1 Создание заказа





Задание 2 ER-диаграмма



Задание 2

UML диаграмма

