**2. Основные функции приложения (в порядке убывания важности)**

1. **Создание и редактирование списков покупок**: Пользователь может создавать новые списки, добавлять, редактировать и удалять элементы из них.
2. **Синхронизация списков с сервером**: Обеспечение актуальности данных при изменениях на разных устройствах.
3. **Отметка покупок**: Пользователь может отмечать продукты как «купленные» или «не купленные», чтобы отслеживать, что было приобретено.
4. **Удаление списков**: Возможность полностью удалить списки покупок, которые больше не нужны.
5. **Категоризация продуктов**: Пользователь может сортировать продукты по категориям (фрукты, овощи, молочные продукты и т.д.).
6. **Поиск по списку**: Функция поиска позволяет быстро находить продукты в больших списках.
7. **Персонализация**: Добавление возможности пользователю устанавливать напоминающие уведомления о необходимых покупках или предпочтениях.
8. **Общий доступ к спискам**: Пользователь может делиться своими списками с другими пользователями.

**3. Процесс синхронизации данных между клиентом и сервером**

Синхронизация данных будет происходить через API, обеспечивая взаимодействие между клиентом и сервером. Основные операции включают:

**3.1 Основные API методы**

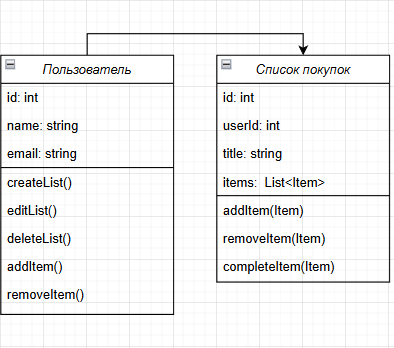
* **POST /api/lists**: создание нового списка покупок.
* **GET /api/lists**: получение списка всех списков пользователем.
* **GET /api/lists/{id}**: получение конкретного списка по ID.
* **PUT /api/lists/{id}**: редактирование существующего списка.
* **DELETE /api/lists/{id}**: удаление списка по ID.
* **POST /api/lists/{id}/items**: добавление новых товаров в список.
* **PUT /api/lists/{list\_id}/items/{item\_id}**: редактирование товара в списке.
* **DELETE /api/lists/{list\_id}/items/{item\_id}**: удаление товара из списка.
* **POST /api/items/{id}/purchase**: отметка товара как «купленного».
* **POST /api/items/{id}/unpurchase**: снятие отметки о покупке.

**3.2 Процесс синхронизации данных**

1. **Создание нового списка**:
   * Пользователь создает список на клиенте.
   * Клиент отправляет **POST /api/lists** с данными списка.
   * Сервер сохраняет данные и возвращает созданный список.
2. **Редактирование списка**:
   * Пользователь редактирует существующий список на клиенте.
   * Клиент отправляет **PUT /api/lists/{id}** с изменениями.
   * Сервер обновляет данные.
3. **Добавление/удаление товаров**:
   * Для добавления товара пользователь отправляет **POST /api/lists/{id}/items**.
   * Для удаления отправляется **DELETE /api/lists/{list\_id}/items/{item\_id}**.
   * Сервер обновляет соответствующий список.
4. **Отметка о покупке**:
   * Пользователь отмечает продукт как купленный – отправляется **POST /api/items/{id}/purchase**.
   * Сервер обновляет статус товара.
5. **Синхронизация изменений**:
   * При каждом открытии приложения клиент синхронизирует данные с сервером, чтобы получить актуальный список.
   * Используется **GET /api/lists** для получения данных.

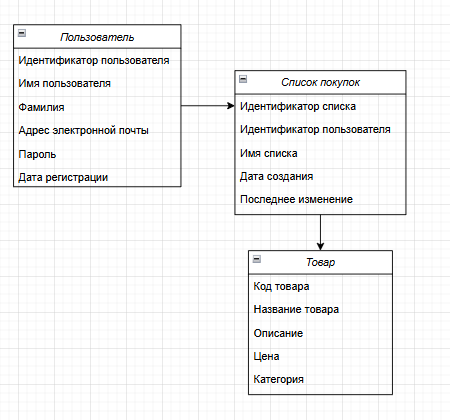
3.3 **UML диаграммы**

3.3.1 Диаграмма классов



**Рисунок - Приложение к заданию3.PNG**

3.3.2 ER-диаграмма



**Рисунок - Приложение к заданию3.2.PNG**

**4. Прототип экрана: Список покупок**

**Описание пользовательского интерфейса**

Общая структура экрана

Экран "Список покупок" представляет собой основной интерфейс, позволяющий пользователю добавлять и управлять товарами, которые он планирует купить. Он выполнен в минималистичном и интуитивно понятном дизайне. Основные элементы интерфейса представлены ниже.

4.1. Заголовок экрана

В верхней части экрана расположен заголовок "Список покупок". Он написан крупным шрифтом, чтобы привлекать внимание пользователя. Справа от заголовка можно добавить кнопку "Редактировать", позволяющую перейти в режим редактирования списка.

4.2. Поле для добавления товара

Под заголовком находится текстовое поле с иконкой "плюс", где пользователь может ввести название товара. Рядом с полем также могут быть предложены популярные категории товаров (например, "Фрукты", "Овощи", "Молоко"), чтобы облегчить поиск.

4.3. Кнопка "Добавить"

Рядом с текстовым полем находится кнопка "Добавить", которая активируется после ввода текста. При нажатии на нее товар добавляется в список, представленный ниже.

4.4 Список товаров

В центральной части экрана находится список добавленных товаров. Каждый товар отображается в виде карточки с следующими элементами:

- Название товара (с возможностью редактирования)

- Количество (в виде увеличиваемого и уменьшаемого числа с кнопками + и -)

- Чекбокс для отметки о том, что товар был куплен

- Кнопка "Удалить" — иконка в виде корзины, которая удаляет товар из списка.

4.5. Фильтры и сортировка

В верхней части списка можно добавить небольшие переключатели для фильтрации продуктов по категориям (например, "Все", "Купленные", "Некупленные") и сортировки по алфавиту или по дате добавления.

4.6. Кнопка "Очистить список"

Внизу списка располагается кнопка "Очистить список", позволяющая удалить все товары из списка. При нажатии появляется всплывающее окно с подтверждением ("Вы уверены, что хотите очистить список?").

4.7. Нижняя панель навигации

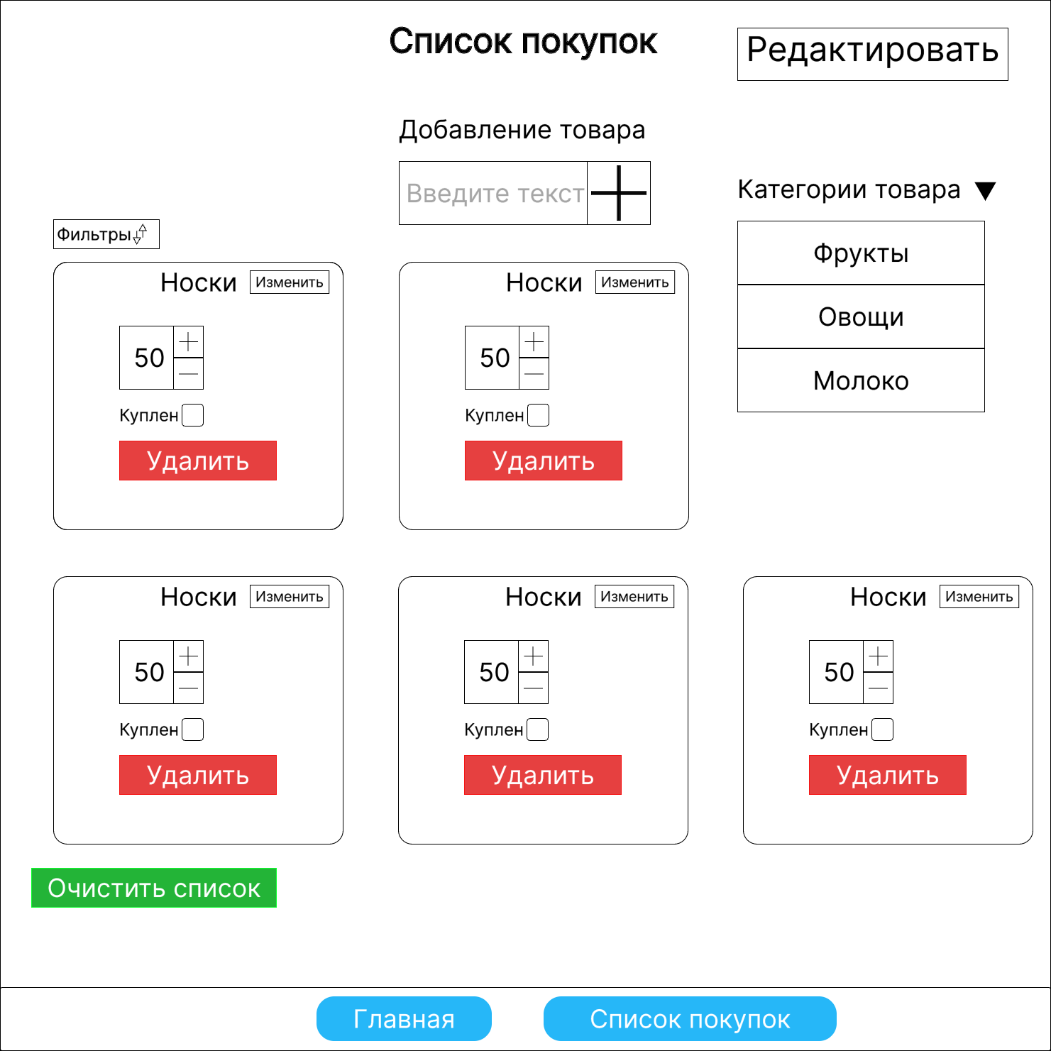
Внизу экрана расположена навигационная панель с несколькими кнопками:

- Главная (иконка домика)

- Список покупок (иконка списока)

- История покупок (иконка корзины)

- Настройки (иконка шестеренки)



**Рисунок - Приложение к заданию4.PNG**

Ссылка на макет [https://pixso.net/app/editor/4r-Egvhx6uJ1EdD7c6KtKg?icon\_type=1&page-id=0%3A1 Приглашаем вас присоединиться к Pixso Файл дизайна "Файл дизайна"](https://pixso.net/app/editor/4r-Egvhx6uJ1EdD7c6KtKg?icon_type=1&page-id=0%3A1%20Приглашаем%20вас%20присоединиться%20к%20Pixso%20Файл%20дизайна%20%20%22Файл%20дизайна%22)

**5. Описание функции покупки товара в приложении "Список покупок в магазинах"**

Общая задача

Разработать функционал покупки товара, который позволит пользователю создавать, редактировать и удалять покупки в рамках мобильного клиент-серверного приложения "Список покупок в магазинах". Пользователь должен иметь возможность добавлять товары и указывать количество, а также отмечать товары как "купленные".

Функциональные особенности

1. Добавление товара в список покупок:

- Пользователь может выбрать товар из заранее заданного списка или добавить новый товар с его названием и ценой.

- Указать количество товара.

2. Редактирование информации о товаре:

- Изменение названия, цены, количества или статуса ("Куплен" / "Не куплен").

3. Удаление товара из списка:

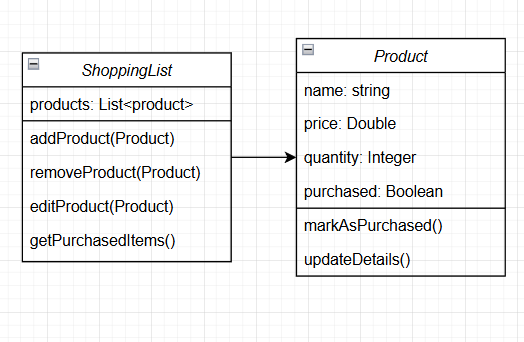
- Возможность удаления товара из списка покупок.

4. Отмечание товара как купленного:

- Пользователь может отметить товар как купленный, после чего он перемещается в раздел "Купленные товары".

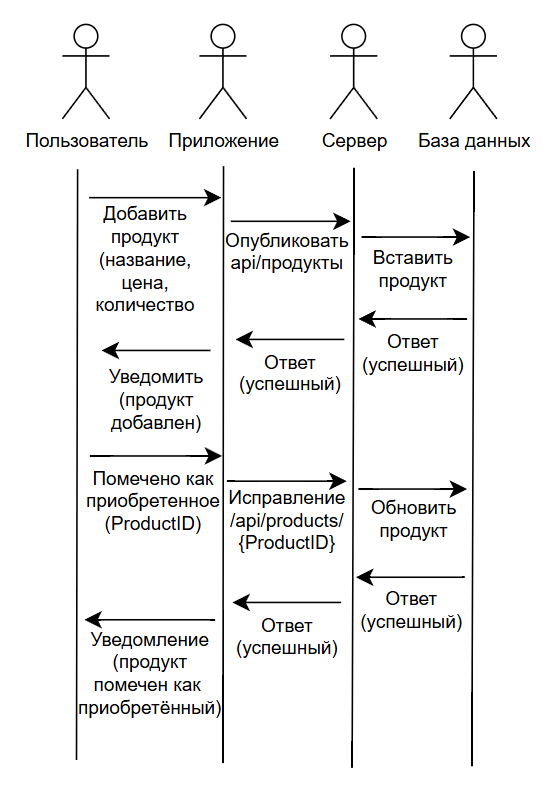
UML Диаграммы

1. Диаграмма классов:



**Рисунок - Приложение к заданию5.PNG**

2. Диаграмма последовательности:



**Рисунок - Приложение к заданию5.2.PNG**

Используемые API методы

1. Добавление товара:

- Метод: `POST /api/products`

- Передаваемые параметры:

- `name`: название товара (String)

- `price`: цена товара (Double)

- `quantity`: количество товара (Integer)

- Получаемые параметры:

- `status`: статус операции (String)

- `productId`: уникальный идентификатор созданного товара (Integer)

2. Редактирование товара:

- Метод: `PUT /api/products/{productId}`

- Передаваемые параметры:

- `productId`: уникальный идентификатор товара (Integer)

- `name`: новое название товара (String)

- `price`: новая цена товара (Double)

- `quantity`: новое количество товара (Integer)

- Получаемые параметры:

- `status`: статус операции (String)

3. Удаление товара:

- Метод: `DELETE /api/products/{productId}`

- Передаваемые параметры:

- `productId`: уникальный идентификатор товара (Integer)

- Получаемые параметры:

- `status`: статус операции (String)

4. Отметить товар как купленный:

- Метод: `PATCH /api/products/{productId}`

- Передаваемые параметры:

- `productId`: уникальный идентификатор товара (Integer)

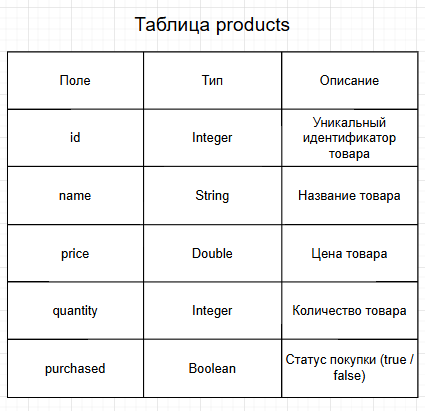
- `purchased`: статус товара (Boolean)

- Получаемые параметры:

- `status`: статус операции (String)

Процесс хранения информации о покупках

Информация о покупках пользователя будет храниться в базе данных. Структура базы данных может включать следующую таблицу:



**Рисунок - Приложение к заданию 5.3.PNG**

Каждый пользователь будет иметь свою собственную коллекцию товаров, привязанную к его учетной записи в системе.

Заключение

Таким образом, описанная функция покупки товара в приложении "Список покупок в магазинах" включает добавление, редактирование, удаление и отметку товара как купленного, а также использование структурированного подхода к проектированию через UML-диаграммы и подробное описание API. Это поможет в разработке функционала, соответствующего современным требованиям пользователей.

**6. Разработка мобильного клиент-серверного приложения «Список покупок в магазинах» может столкнуться с рядом сложностей и вопросов, которые следует обсудить с заказчиком для более точного понимания его потребностей и ожиданий. Ниже приведены основные сложности и вопросы, которые могут возникнуть в процессе разработки.**

Основные сложности разработки:

1. **Проектирование архитектуры приложения**:
   * Выбор между монолитной архитектурой и микросервисами. Определение, как будет организовано взаимодействие между клиентом и сервером.
2. **Управление данными**:
   * Необходимо разработать эффективную модель данных для хранения списков покупок, включая возможность организации по категориям, магазинам и статусу (куплено/не куплено).
3. **Аутентификация и безопасность**:
   * Реализация системы аутентификации пользователей (например, через почту или социальные сети) и обеспечение безопасности данных пользователя (шифрование, защита от несанкционированного доступа).
4. **Синхронизация данных**:
   * Создание механизма, который бы обеспечивал синхронизацию данных между устройствами (если пользователь использует приложение на нескольких устройствах).
5. **UI/UX дизайн**:
   * Разработка удобного и интуитивно понятного интерфейса, который будет привлекательным и простым в использовании. Учет разных платформ (iOS, Android) и экранов различных размеров.
6. **Тестирование и отладка**:
   * Обеспечение качества приложения через тестирование на различных устройствах и ОС, а также устройство с разными версиями, что требует дополнительных ресурсов и времени.

Вопросы к заказчику:

1. **Каковы основные функции приложения?**
   * Какие конкретные функции наиболее важны? Например, добавление/удаление товаров, возможность делиться списками, напоминания и т.д.
2. **Кто является целевой аудиторией приложения?**
   * Понимание целевой аудитории поможет в разработке интерфейса и функциональности.
3. **Нужна ли система регистрации/авторизации?**
   * Нужно ли хранить данные о пользователе на сервере и предоставлять возможность входа через социальные сети?
4. **Каковы требования к безопасности данных?**
   * Есть ли особые требования по защите личных данных пользователей или информации о покупках?
5. **Нужно ли интегрировать приложение с другими системами или сервисами?**
   * Планируется ли в будущем интеграция с онлайн-магазинами или системами управления запасами?
6. **Какова ожидаемая нагрузка на сервер?**
   * Сколько пользователей ожидается, и каковы предполагаемые объемы данных? Это поможет в выборе серверной архитектуры.
7. **Какой план по поддержке и обновлениям приложения?**
   * Как заказчик видит дальнейшую поддержку приложения? Какие новые функции и обновления планируются в будущем?
8. **Каковы сроки и бюджет проекта?**
   * Какие есть ограничения по времени и финансам на разработку?

Обсуждение этих вопросов с заказчиком поможет прояснить ожидания и упростит процесс разработки приложения, минимизируя потенциальные сложности.

**7. Вот запросы для каждой из указанных задач.**

**1. Общая стоимость книг для каждого автора и отсортировать результат в порядке убывания:**

SELECT

a.Id AS AuthorId,

a.AuthorName,

SUM(b.Price) AS TotalBookPrice

FROM

Authors a

JOIN

Books b ON a.Id = b.AuthorId

GROUP BY

a.Id, a.AuthorName

ORDER BY

TotalBookPrice DESC;

**2. Стоимость книг автора превышает 1500:**

SELECT

a.Id AS AuthorId,

a.AuthorName,

SUM(b.Price) AS TotalBookPrice

FROM

Authors a

JOIN

Books b ON a.Id = b.AuthorId

GROUP BY

a.Id, a.AuthorName

HAVING

SUM(b.Price) > 1500;

**3. Вывести авторов с количеством книг:**

SELECT

a.Id AS AuthorId,

a.AuthorName,

COUNT(b.Id) AS NumberOfBooks

FROM

Authors a

LEFT JOIN

Books b ON a.Id = b.AuthorId

GROUP BY

a.Id, a.AuthorName;

**4. Получить автора без книг:**

SELECT

a.Id AS AuthorId,

a.AuthorName

FROM

Authors a

LEFT JOIN

Books b ON a.Id = b.AuthorId

WHERE

b.Id IS NULL;

Пояснения:

1. Общая стоимость книг: Запрос объединяет таблицы `Authors` и `Books`, группирует их по авторам и суммирует стоимость книг для каждого автора. Результаты сортируются в порядке убывания общей стоимости.

2. Стоимость книг автора превышает 1500: Этот запрос аналогичен первому, но добавляется условие через `HAVING`, чтобы выбрать только тех авторов, у которых сумма стоимости книг превышает 1500.

3. Авторы с количеством книг: Здесь используется `LEFT JOIN`, чтобы вернуть всех авторов, даже если у них нет книг. С помощью `COUNT(b.Id)` учитывается количество книг у каждого автора.

4. Авторы без книг: В этом запросе, используя `LEFT JOIN`, выбираем только тех авторов, у которых нет связанных записей в таблице `Books`, что обозначается условием `WHERE b.Id IS NULL`.