**Дано: Мобильное клиент-серверное приложение «Список покупок в магазинах» (Примечание: приложение без интеграции с магазинами).**

**Задание 1. Бизнес-логику мобильного приложения (использовать любую удобную нотацию моделирования бизнес-процессов).**

**Основные процессы:**

* **Управление списками (CRUD):** создание, просмотр, редактирование, удаление списков.
* **Управление элементами списка (CRUD):** добавление, просмотр, редактирование, удаление товаров. Отметка товара как купленного.
* **Управление категориями (упрощенный CRUD):** добавление, просмотр, (опционально) редактирование, удаление категорий.
* **Управление магазинами (упрощенный CRUD):** добавление, просмотр, (опционально) редактирование, удаление магазинов.

**Основа процесса (пример: Добавление товара):**

1. Пользователь открывает список и нажимает “Добавить товар”.
2. Приложение отображает форму для ввода названия, количества, категории.
3. Пользователь вводит данные и нажимает “Сохранить”.
4. Приложение сохраняет товар в списке.
5. Товар отображается в списке.

**Ключевые элементы логики:**

* Использование локальной базы данных для хранения данных.
* Возможность отметки товара как купленного.
* Валидация вводимых данных (например, проверка на пустые поля).

**Дополнительные возможности (улучшения):**

* Сортировка списков и товаров.
* Поиск товаров.
* Фильтры по категориям/статусу.
* Шаблоны списков.
* (Потенциально требует серверной части) Общий доступ к спискам.
* (Усложнение) Привязка товаров к магазинам.

**Создание BPMN-диаграммы:**

1. Используйте инструмент BPMN (draw.io, Lucidchart, Bizagi).
2. Событие начала -> Действие -> Последовательность -> Шлюз (ветвление, если нужно) -> Событие окончания.
3. Визуализируйте основные процессы, начиная с простого добавления товара.

**Задание** **2. Основные функции данного приложения (список должен быть составлен в порядке убывания важности функций для пользователя).**

1. **Создание и ведение списков покупок:** (Самая важная функция)
   * Позволяет пользователю создавать несколько списков покупок для разных магазинов, событий или категорий (например, «Покупки в продуктовом магазине», «Товары для дома», «Отпуск»).
2. **Добавление товаров в список:**
   * Возможность быстро и легко добавлять товары в выбранный список, указывая название товара.
3. **Редактирование товаров в списке:**
   * Возможность изменять название, количество и другие параметры товара, добавленного в список.
4. **Отметка товаров как купленных:**
   * Функция, позволяющая отмечать товары, которые уже куплены, чтобы видеть, что ещё осталось приобрести.
5. **Удаление товаров из списка:**
   * Возможность легко удалять товары, которые больше не нужны в списке.
6. **Управление категориями товаров:**
   * Опционально, но полезно: позволяет группировать товары по категориям (например, «Молочные продукты», «Фрукты», «Овощи») для лучшей организации списка. Может быть менее важным, если пользователь предпочитает сортировать вручную.
7. **Управление магазинами:**
   * Опционально: позволяет вести список магазинов, в которых пользователь обычно совершает покупки. Эта функция становится более важной, если товары привязаны к конкретным магазинам.
8. **Сортировка товаров в списке:**
   * Возможность сортировать товары по названию, категории или другим параметрам для удобства использования.
9. **Поиск товаров в списке:**
   * Функция поиска товара по названию внутри списка. Особенно полезна для больших списков.
10. **Удаление списков покупок:**
    * Возможность удалять старые или ненужные списки.
11. **Редактирование названия списка покупок:**
    * Позволяет переименовать список для лучшей организации.
12. **Создание шаблонов списков:**
    * Более продвинутая функция: возможность сохранять часто используемые списки в качестве шаблонов для быстрого создания новых списков.
13. **Общий доступ к спискам (синхронизация):**
    * (Требуется серверная часть). Функция совместного использования списка покупок с другими пользователями (например, членами семьи).
14. **Привязка товаров к магазинам:**
    * Более продвинутая функция: возможность указать, в каком магазине планируется купить тот или иной товар.

**Задание 3. Процесс синхронизации данных между клиентом и сервером (добавление и удаление списка, наполнение и редактирование списка, покупка/«откупка продукта» и т.д.). Представить все в диаграммах UML, API методах и других представлениях, также составить ER-диаграмму сущностей.**

{

"/lists": {

"GET": {

"description": "Получение списков",

"response": [{ "list\_id": "...", "name": "..." }]

},

"POST": {

"description": "Добавление списка",

"request": { "name": "..." },

"response": { "list\_id": "...", "name": "..." }

}

},

"/lists/{list\_id}": {

"GET": {

"description": "Получение списка",

"response": { "list\_id": "...", "name": "..." }

},

"PUT": {

"description": "Редактирование списка",

"request": { "name": "..." },

"response": { "list\_id": "...", "name": "..." }

},

"DELETE": {

"description": "Удаление списка",

"response": { "message": "..." }

}

},

"/lists/{list\_id}/items": {

"GET": {

"description": "Получение товаров",

"response": [{ "item\_id": "...", "name": "...", "purchased": false }]

},

"POST": {

"description": "Добавление товара",

"request": { "name": "...", "quantity": 0 },

"response": { "item\_id": "...", "name": "...", "purchased": false }

}

},

"/lists/{list\_id}/items/{item\_id}": {

"PUT": {

"description": "Редактирование товара",

"request": { "name": "...", "quantity": 0, "purchased": true },

"response": { "item\_id": "...", "name": "...", "purchased": true }

},

"DELETE": {

"description": "Удаление товара",

"response": { "message": "..." }

}

}

}

**Задание 4. Подготовить прототип одного из экранов мобильного приложения и описать пользовательский интерфейс для данного экрана (например, покупка товара).**

<https://www.figma.com/design/G2XDxygtH6grw8JvLuDkY8/Untitled?node-id=0-1&p=f&t=qc0Mrfcw6uyHzw8y-0>

**Задание 5. Подготовить подробное описание функции покупки товара, 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).**

**{**

**"/orders/{order\_id}": {**

**"GET": {**

**"desc": "Получить заказ",**

**"res": {**

**"order\_id": "string",**

**"items": [{ "product\_id": "string", "name": "string", "qty": "int", "price": "number" }],**

**"total\_price": "number"**

**}**

**},**

**"PUT": {**

**"desc": "Обновить заказ",**

**"req": {**

**"items": [{ "product\_id": "string", "qty": "int" }]**

**},**

**"res": { "msg": "string" }**

**}**

**}**

**}**

**Задание 6. Основные, на ваш взгляд, сложности разработки такого приложения. Вопросы, возникшие при выполнении тестового задания, которые вы бы задали заказчику.**

1. **Синхронизация данных:**
   * **Сложность:** обеспечение надежной и быстрой синхронизации данных между мобильным клиентом и сервером (если требуется функция общего доступа к спискам). Необходимо учитывать возможные проблемы с сетью, конфликты при одновременном изменении данных разными пользователями и необходимость решения этих конфликтов.
   * **Решение:** использование стратегий разрешения конфликтов (например, «последний записавший»), оптимистическая блокировка, механизмы повторных попыток при сетевых ошибках, кэширование данных на клиенте.
2. **Производительность:**
   * **Сложность:** обеспечение быстрой работы приложения, особенно при больших списках покупок и большом количестве данных.
   * **Решение:** оптимизация запросов к базе данных, использование эффективных алгоритмов поиска и сортировки, кэширование данных, использование асинхронных операций для предотвращения блокировки пользовательского интерфейса.
3. **Пользовательский интерфейс (UX):**
   * **Сложность:** создание интуитивно понятного и удобного интерфейса, который будет прост в использовании для пользователей с разным уровнем технической грамотности.
   * **Решение:** проектирование простого и минималистичного интерфейса, использование общепринятых UX-паттернов, проведение юзабилити-тестирования с реальными пользователями, итеративная разработка с учетом обратной связи.
4. **Масштабируемость (если планируется большое количество пользователей):**
   * **Сложность:** серверная инфраструктура должна выдерживать большое количество одновременных пользователей и запросов.
   * **Решение:** использование масштабируемой архитектуры (например, микросервисы), балансировка нагрузки, оптимизация базы данных, использование CDN для статических ресурсов.
5. **Безопасность:**
   * **Сложность:** защита данных пользователей от несанкционированного доступа, обеспечение безопасной аутентификации и авторизации (если требуется функция общего доступа).
   * **Решение:** использование надежных алгоритмов шифрования для хранения паролей, защита API с помощью токенов (например, JWT), проверка данных на сервере, защита от SQL-инъекций и других распространенных веб-уязвимостей.
6. **Поддержка разных платформ:**
   * **Сложность:** разработка приложения, которое будет корректно работать на разных версиях iOS и Android с учетом различий в API, разрешениях экрана и других особенностях каждой платформы.
   * **Решение:** использование кроссплатформенных фреймворков (например, React Native, Flutter) или нативная разработка для каждой платформы.
7. **Управление состоянием (State Management):**
   * **Сложность:** эффективное управление состоянием приложения, особенно при сложной логике и асинхронных операциях.
   * **Решение:** использование библиотек для управления состоянием (например, Redux, MobX, Zustand) или встроенных механизмов платформы (например, Context API в React).
8. **Отсутствие интеграции с магазинами:**
   * **Сложность:** приходится полагаться на ручной ввод данных пользователем, что может быть утомительным и приводить к ошибкам.
   * **Решение:** Максимально упростить процесс добавления товаров (например, предлагать автозаполнение на основе истории покупок, использовать голосовой ввод).

Вопросы заказчику:

1. **Функциональные требования:**
   * Какие основные сценарии использования приложения? Кто является целевой аудиторией?
   * Насколько важна функция общего доступа к спискам? Каким образом будет реализовано управление правами доступа?
   * Какие дополнительные функции планируются в будущем (например, сканирование штрихкодов, интеграция с сервисами доставки)?
   * Какие форматы данных поддерживаются для импорта/экспорта списков (если планируется)?
2. **Нефункциональные требования:**
   * Какие требования к производительности (время отклика, скорость синхронизации)?
   * Какие требования к безопасности (какие данные нужно особенно защищать)?
   * Какие требования к масштабируемости (сколько пользователей должно поддерживать приложение)?
   * На каких платформах должно работать приложение (iOS, Android, веб)?
3. **Дизайн и UX:**
   * Есть ли готовый дизайн или wireframes?
   * Какие требования к доступности (accessibility)?
   * Какие принципы дизайна должны быть соблюдены?
4. **Интеграция:**
   * Планируется ли в будущем интеграция с другими сервисами (например, с приложениями лояльности магазинов)?
5. **Бюджет и сроки:**
   * Какой бюджет выделен на разработку приложения?
   * Какие сроки разработки?
6. **Поддержка и сопровождение:**
   * Кто будет заниматься поддержкой и сопровождением приложения после запуска?
   * Какие планы на будущее развитие приложения?
7. **Относительно “отсутствия интеграции с магазинами”:**
   * Почему было принято решение об отсутствии интеграции? Это временное ограничение или принципиальная позиция?
   * Рассматривались ли альтернативные варианты, которые могли бы упростить ввод данных (например, использование базы данных товаров, предоставляемой сторонним сервисом)?

**Задание 7. Есть таблицы Books и Authors, где AuthorId табл. Books равно Id табл. Authors. Необходимо написать SQL-запрос, чтобы найти: 1. Общую стоимость книг для каждого автора и отсортировать результат в порядке убывания; 2. Стоимость книг автора превышает 1500; 3. Вывести авторов с количеством книг; 4. Получить автора без книг.**

**1. Общая стоимость книг для каждого автора и сортировка:**

**SELECT Authors.Id, Authors.Name, SUM(Books.Price) AS TotalBookPrice**

**FROM Authors**

**JOIN Books ON Authors.Id = Books.AuthorId**

**GROUP BY Authors.Id, Authors.Name**

**ORDER BY TotalBookPrice DESC;**

**2. Авторы, у которых общая стоимость книг превышает 1500:**

**SELECT Authors.Id, Authors.Name, SUM(Books.Price) AS TotalBookPrice**

**FROM Authors**

**JOIN Books ON Authors.Id = Books.AuthorId**

**GROUP BY Authors.Id, Authors.Name**

**HAVING TotalBookPrice > 1500;**

**3. Авторы с количеством книг:**

**SELECT Authors.Id, Authors.Name, COUNT(Books.Id) AS BookCount**

**FROM Authors**

**LEFT JOIN Books ON Authors.Id = Books.AuthorId**

**GROUP BY Authors.Id, Authors.Name;**

**4. Авторы без книг:**

**SELECT Authors.Id, Authors.Name**

**FROM Authors**

**LEFT JOIN Books ON Authors.Id = Books.AuthorId**

**WHERE Books.Id IS NULL;**