Курс - Системный аналитик

US

▼ Выполнение задания

User story (пользовательская история) — это простой и короткий текст, который описывает потребности и ожидания заказчика от продукта или услуги. Основная цель — понять, что именно нужно создать, чтобы удовлетворить потребности конечного пользователя.

Шаблон: "Я, как {роль}, хочу иметь {возможность} для того чтобы {цель}".

User Story	Я, как пользователь приложения [Telegram], хочу иметь возможность редактировать информацию на странице профиля, для того что бы я мог вносить актуальные данные о себе.

Макеты

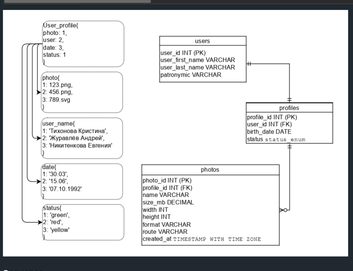
▼ Выполнение задания

Макеты или прототипы системы нужны для того, чтобы дизайнер мог понять, что именно ему необходимо изобразить, а также для того, чтобы понять пользовательский путь. В макете должно быть схематично отображен будущий интерфейс со всеми основными элементами

- отрисовка в draw.io
- не более 5 макетов
- только основной сценарий

1
2
3
4
5
5.1
5.2
Профиль
Фото
Имя
Дата рождения
Статус
Сохранить

```
drop TABLE users,profiles, photos;
drop TYPE status_enum
```



Sequence

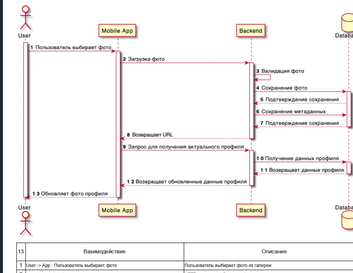
✅ Выполнение задания

1. Построить sequence-диаграмму с использованием PlantUML.
2. Составить сопроводительную таблицу, поясняющую взаимодействия/вызовы/стрелки, указанные на диаграмме. Пример брать из шаблона ФТ.

Требования

1. Только основной положительный сценарий (никаких альтернативных). Остановиться на ОДНОЙ END2END цепи вызовов.
2. Только синхронные запросы.
3. Только трехзвенная архитектура.

```
@startuml

skin rose

autonumber

actor User
participant "Mobile App" as App
participant "Backend" as Backend
database "Database" as DB

activate User
User --> App : Пользователь выбирает фото

activate App
App --> Backend : Загрузка фото

activate Backend
Backend --> Backend : Валидация фото
Backend --> DB : Сохранение фото

activate DB
DB --> Backend : Подтверждение сохранения
Backend --> DB : Сохранение метаданных
DB --> Backend : Подтверждение сохранения

deactivate DB
Backend --> App : Возвращает URL
deactivate Backend

App --> Backend : Запрос для получения актуального профиля
activate Backend
Backend --> DB : Получение данных профиля
activate DB
DB --> Backend : Возвращает данные профиля
deactivate DB
Backend --> App : Возвращает обновленные данные профиля
deactivate Backend

App --> User : Обновляет фото профиля
deactivate App
deactivate User
@enduml
```



	Взаимодействие	Описание
1	User -> App: Пользователь выбирает фото	Пользователь выбирает фото из галереи
2	App -> Backend : Загрузка фото	HTTP-запрос с выбранным фото
3	Backend -> Backend : Валидация фото	Проверка формата, максимального размера, кол-во загруженных фото
4	Backend -> DB : Сохранение фото	Бэкенд сохраняет фото в базе данных, выполняя INSERT в таблицу с использованием BLOB.
5	DB -> Backend : Подтверждение сохранения	Бэкенд получает ответ от базы данных о успешном сохранении
6	Backend -> DB : Сохранение метаданных	Бэкенд сохраняет метаданные (имя файла, дата загрузки и т.д.)
7	DB -> Backend : Подтверждение сохранения	Бэкенд получает ответ от базы данных о успешном сохранении
8	Backend -> App : Возвращает URL	Бэкенд возвращает URL для доступа к загруженному фото в мобильное приложение
9	App -> Backend : запрос для получения актуального профиля	Мобильное приложение отправляет дополнительный запрос на бэкенд для получения актуальных данных профиля
10	Backend -> DB : Получение данных профиля	Бэкенд запрашивает данные профиля из базы данных
11	DB -> Backend : Возвращает данные профиля	База данных возвращает данные профиля, включая обновленное фото.
12	Backend -> App : Возвращает обновленные данные профиля	Приложение обновляет UI, чтобы отобразить новый аватар, используя полученный URL.
13	App -> User : Обновляет фото профиля	Мобильное приложение обновляет отображение фото профиля на основе полученных данных.

LLM-Gigachat

✅ Выполнение задания

На языке Python с использованием любых библиотек написать скрипт, который шлет в эти гигачата системный и пользовательский промпты (на свободную тему). Результат ответа от гигачата необходимо вывести в консоль (терминал).

LLM.py

Запуск скрипта с использованием GigaChat API

Шаг 1: Установка необходимых библиотек

Для работы скрипта вам потребуется установить несколько Python-библиотек. Если у вас еще нет установленных `langchain` и `getpass`, выполните следующую команду в терминале:

```
pip install langchain
pip install getpass
```

Шаг 2: Получение токена доступа

Чтобы использовать GigaChat API, вам понадобится токен доступа (`Authorization key`). Если вы еще не получили его, обратитесь к документации платформы, где вы используете GigaChat, чтобы узнать, как получить этот ключ.

Шаг 3: Сохранение токена в переменной окружения

После получения токена сохраните его в переменную окружения `GIGACHAT_TOKEN`. Это можно сделать следующим образом:

1. Откройте терминал и введите следующую команду:

   ```
   export GIGACHAT_TOKEN='ваш_токен'
   ```

   Убедитесь, что вместо `ваш_токен` подставили свой реальный токен.

2. Для проверки, что токен был сохранён правильно, выполните команду:

   ```
   echo $GIGACHAT_TOKEN
   ```

   Вы должны увидеть ваш токен в выводе команды.

Шаг 4: Запуск скрипта

Теперь, когда все готово, запустите скрипт:

Запустите скрипт следующей командой:

```
python LLM.py
```

Шаг 5: Ввод данных

Когда скрипт запустится, он запросит у вас ввод Use case. Введите интересующий вас сценарий и нажмите Enter. Скрипт отправит запрос в GigaChat и выведет ответ.

Документация REST API в табличном виде

✅ Выполнение задания

1. Ознакомится с типами, использующимися в json объектах тут: https://www.json.org/json-en.html
2. Создать PDF файл по описанию своего/своих спроектированных методов. Пример брать из Шаблона ФТ. В приложенном файле текстом описать какую бизнес логику реализует вызов (например, создает сущность "юзер" в системе).
3. Минимум один REST вызов, максимум два REST вызова.

Требования

1. Только положительный успешный вызов/вызовы. Никаких ждк кейсов. В вызове всё идёт хорошо, вызов не ломается.
2. Можно использовать GET ИЛИ/И POST. Брать другие типы вызовов нельзя.

REST. Табличный вид

Загрузка фотографии на сервер

POST /api/v1/users/:userId/avatar

Request

Название параметра	Принадлежность	Тип данных	Описание	Обязательность параметра
avatar	body	image/png	Файл фотографии	да

Response 201 Created

Название параметра	Принадлежность	Тип данных	Описание	Обязательность параметра
message	body	string	Сообщение о результате выполнения операции	да
userId	body	integer	Идентификатор пользователя, для которого был загружен аватар	да
avatarUrl	body	string	Ссылка на загруженную фотографию	да
timestamp	body	string	Время загрузки аватара в формате ISO 8601	да
filename	body	string	Исходное название файла	да
sizeMb	body	number	Размер загруженного файла	да
mimeType	body	string	MIME-тип загруженного файла	да

Swagger

✅ Выполнение задания

Взять свои методы, спроектированные в табличной форме и положить их в формат сваггера. - больше 2х методов делать нельзя. - методы только GET или POST. - только положительный успешный вызов/вызовы. Никаких ждк кейсов. В вызове всё идёт хорошо, вызов не ломается

```
openapi: 3.0.3
info:
  title: Swagger
    description: Описание методов API для загрузки фотографии на сервер
    version: 1.0.13
servers:
  - url: https://telegram.com/api/v1
tags:
  - name: avatar
    description: Метод для работы с фотографиями
paths:
  /users/{userId}/avatar:
    post:
      tags:
        - avatar
      summary: Загрузить аватар пользователя
      description: Загружает аватар для указанного пользователя
      parameters:
        - in: path
          name: userId
          required: true
          schema:
            type: integer
          description: Идентификатор пользователя
      requestBody:
        content:
          multipart/form-data:
            schema:
              $ref: '#/components/schemas/AvatarFile'
      responses:
        '201':
          description: Аватар успешно загружен
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AvatarUploadResponse'
        '400':
          description: Неправильный формат файла
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '413':
          description: Превышен допустимый размер файла
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '415':
          description: Неподдерживаемый MIME-тип файла
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Внутренняя ошибка сервера
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
```

```
    - avatar
  AvatarUploadResponse:
    type: object
    properties:
      message:
        type: string
        description: Сообщение о результате выполнения операции
        example: "Аватар успешно загружен"
      userId:
        type: integer
        description: Идентификатор пользователя, для которого был загружен аватар
        example: 42
      avatarUrl:
        type: string
        description: Ссылка на загруженный аватар
        example: "https://example.com/avatars/user_42.png"
      timestamp:
        type: string
        description: Время загрузки аватара в формате ISO 8601
        example: "2023-09-01T15:30:00Z"
      filename:
        type: string
        description: Исходное имя файла
        example: "profile_pic.jpg"
      sizeMb:
        type: number
        format: double
        description: Размер загруженного файла в мегабайтах
        example: 2.5
      mimeType:
        type: string
        description: MIME-тип загруженного файла
        example: "image/jpeg"
    required:
      - message
      - userId
      - avatarUrl
      - timestamp
      - filename
      - sizeMb
      - mimeType
  ErrorResponse:
    type: object
    properties:
      Code:
        type: number
        description: Код ошибки
        example: 400
      Message:
        type: string
        description: Подробное сообщение об ошибке
        example: "Неправильный формат файла"
    required:
      - Code
      - Message
```

AC + NFR

▼ Выполнение задания

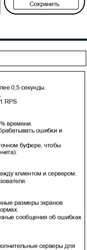
КП

- Кейсы в рамках ваших фичей, связаны с US, UC и макетами
- Кейсы не должны включать в себя основной сценарий
- Описать от 2 до 4 кейсов включительно

Критерии приемки

User Story	Я, как пользователь приложения [Telegram], хочу иметь возможность редактировать информацию на странице профиля, для того что бы я мог вносить актуальные данные о себе
Основной сценарий	1. Система отображает форму выбора редактирования аватара, имени, даты рождения, статуса (экран номер 1) 2. Клиент нажимает на иконку плюса (экран номер 1) 3. Система отображает экран разрешения доступа к галерее (экран 5.2) 4. Клиент нажимает на кнопку "Да" 5. Клиент выбирает фото которое устанавливается на аватар (экран 5) 6. Клиент нажимает область с изменением имени (экран 1) 7. Система отображает окно ввода имени и фамилии (экран 4) 8. Клиент вводит данные и нажимает кнопку сохранить (экран 4) 9. Клиент нажимает область с изменением дня рождения (экран 1) 10. Система выводит всплывающее окно с выбором дня, месяца, года (экран 3) 11. Клиент свайпом вверх/вниз выбирает день рождения и нажимает кнопку сохранить(экран 3) 12. Клиент нажимает область с изменением статуса (экран 1) 13. Система выводит окно с выбором статуса, по умолчанию статус "в сети" (экран 2) 14. Клиент выбирает статус "Занят" (экран 2) **Критерий успеха:** Изменение аватар, заполнена графа Ф.И.О, заполнен День рождения, изменен статус

Номер кейса 1	
Функциональность: Клиент находится на экране галереи и выбирает фото. **Дано:** Клиент выбирает фото которое устанавливается на аватар (экран 5) **Когда:** Клиент пытается загрузить файл, который превышает допустимый размер для аватара. **Тогда:** Система выводит сообщение об ошибке "Размер файла слишком велик. Пожалуйста, выберите другой файл."	

Номер кейса 2	
Функциональность: Клиент начал процесс редактирования ФИО. **Дано:** Клиент вводит данные и нажимает кнопку сохранить (экран 4) **Когда:** Клиент вводит недопустимые символы (например, специальные знаки или цифры) в поле имени. **Тогда:** Система блокирует сохранение и выводит предупреждение "Имя может содержать только буквы."	

Нефункциональные требования

Производительности:
Время отклика интерфейса при выборе поля для редактирования должно составлять не более 0,5 секунды.
Загрузка страницы с формой редактирования профиля должна занимать не более 2 секунд.
Запросы на обновление профиля PUT /api/v1/users/profile должны выдерживать нагрузку 1 RPS

Надежности:
Система должна обеспечивать доступность функции редактирования профиля минимум 99% времени.
При возникновении сбоев в процессе редактирования данных система должна корректно обрабатывать ошибки и информировать пользователя о необходимости повторной попытки.
Все изменения, сделанные пользователем, должны автоматически сохраняться в промежуточном буфере, чтобы избежать потери данных при внезапном прерывании процесса (например, отключении интернета).

Безопасности:
Личные данные пользователя (ФИО, дата рождения) должны шифроваться при передаче между клиентом и сервером.
Доступ к редактированию профиля возможен только после успешной аутентификации пользователя.

Удобства использования:
Поддержка различных устройств: Интерфейс должен адаптивно подстраиваться под различные размеры экранов (мобильные устройства, планшеты, десктопы), обеспечивая удобство работы на всех платформах.
Предупреждения и подсказки: Система должна предоставлять пользователям ясные и полезные сообщения об ошибках и подсказки для успешного завершения процесса редактирования.

Масштабируемости:
Система должна поддерживать горизонтальное масштабирование, позволяя добавлять дополнительные серверы для обработки увеличивающейся нагрузки.
Данные пользователей должны храниться в распределенной базе данных, способной справиться с большим количеством записей и запросов