1. Исходный код решения

   https://github.com/ellyzing/yaprofi-pi/tree/main/task_2

2. Описание технического решения

1) Соответствие принципам SOLID

   Принцип единственной ответственности (SRP): классы реализованы с разделением зон ответственности. Group, GroupWithPart, Participant отвечают за описание объектов, GroupService, PartService – за логику взаимодействия между объектами.

   Принцип открытости/закрытости (OCP): классы GroupService и PartService закрыты для изменения, но открыты для расширения. Новые методы и функциональность можно добавлять, не изменяя код самого класса.

   Принцип подстановки Барбары Лисков (LSP): UpdatedGroup и GroupWithPart являются подклассом класса Group, и его экземпляры могут использоваться везде, где ожидаются экземпляры класса Group, не нарушая корректности программы.

   Принцип разделения интерфейса (ISP): решение не нарушает данный принцип, так как клиент взаимодействует с сущностями группы и участников через отдельные методы. На более высоком уровне абстракции RESTAPI представляет один интерфейс для одной бизнес-задачи и данный принцип не применим.

   Принцип инверсии зависимостей (DIP): GroupService зависит от класса Group, что может рассматриваться как пример принципа инверсии зависимостей, поскольку класс GroupService зависит от абстракции Group, а не от ее реализации (то же самое с Participant).

2) Описание документации API в одной из общеиспользуемых спецификаций (рекомендуется OpenAPI (Swagger))

Так как используется фреймворк fastapi, документация доступна после запуска решения на http://localhost:8080/docs

3) Описание настройки среды выполнения и запуска приложения

Приложение реализовано как микроосервис. Хранение данных осуществляется с использованием массивов как структур данных. Реализованный REST API может легко интегрироваться с любой БД (например, с использованием sqlalchemy).

Для реализованного сервиса написан Dockerfile для сборки образа. Базовым образом выбран python:3.11-buster как легковесный и содержащий минимально необходимое ПО. Так как для реализации API был выбран фреймворк fast-api, команда запуска:

uvicorn main:app --reload --host 0.0.0.0 --port 8080

Для сервиса написан docker-compose.yaml. Сервис запускается на порту 8080.

Решение может запущено в любой среде при наличии установленного python, либо в контейнере при наличии установленного и запущенного демона Docker через docker run, либо в docker-compose при наличии установленного docker-compose.

Для реализации сервиса с использованием БД в docker-compose необходимо добавить второй сервис с указанием образа и порта. В код решения в этом случае необходимо будет импортировать sqlalchemy, создать движок БД и сессию:

```
engine = create_engine(
(f"postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/{DB_NAME}"), echo=True)

_SessionFactory = sessionmaker(bind=engine)
```