Цель задания

Разработать веб-приложение для управления списком книг, используя следующие технологии:

- Фреймворк: Django или FastAPI (на ваш выбор)
- База данных: PostgreSQL или MySQL
- gRPC-сервис
- Брокер сообщений: RabbitMQ или Kafka

Описание задания

1. Веб-API

- Реализуйте REST API с использованием Django или FastAPI для следующих операций над книгами:
    - Создание новой книги
    - Получение списка всех книг
    - Получение деталей конкретной книги по `id`
    - Обновление информации о книге
    - Удаление книги
- Модель книги должна содержать следующие поля:
    - `id` (автоматически генерируется)
    - `название` (строка)
    - `автор` (строка)
    - `дата публикации` (дата)

2. Аутентификация

- Реализуйте аутентификацию пользователей с помощью JWT или OAuth2.
- Только аутентифицированные пользователи могут выполнять CRUD операции.

3. База данных

- Используйте PostgreSQL или MySQL для хранения данных.
- Настройте миграции базы данных с помощью `alembic` (для FastAPI) или встроенных средств Django.

4. gRPC-сервис

- Создайте отдельный gRPC-сервис, который предоставляет следующие методы:
    - Получение информации о книге по `id`.
    - Получение списка всех книг.
- gRPC-сервис должен обращаться к той же базе данных.

5. Брокер сообщений

- Настройте брокер сообщений (RabbitMQ или Kafka) для взаимодействия между веб-приложением и gRPC-сервисом.
- При создании, обновлении или удалении книги через REST API:
    - Отправляйте соответствующее сообщение в брокер.
- gRPC-сервис должен:
    - Слушать эти сообщения.
    - Записывать лог о пришедшем сообщении и выводить его в консоль

6. Docker

- Подготовьте файлы для контейнеризации проекта:
    - `Dockerfile` для каждого сервиса (веб-приложение, gRPC-сервис).
    - `docker-compose.yml` для оркестрации сервисов:
        - Веб-приложение
        - gRPC-сервис
        - База данных
        - Брокер сообщений

7. Тестирование

- Напишите unit-тесты для основных функций веб-приложения и gRPC-сервиса.

8. Документация

- Для веб-API:
    - Используйте Swagger/OpenAPI для автоматической генерации документации.
- Для gRPC-сервиса:
    - Предоставьте `.proto` файлы.
    - Опишите, как использовать сервис.

Требования к решению

- Код должен быть чистым, читабельным и соответствовать стандартам PEP 8.
- Используйте виртуальное окружение и предоставьте файл `requirements.txt` для управления зависимостями.
- Реализуйте корректную обработку ошибок и исключений.
- Предоставьте README.md с инструкциями по запуску проекта.

Дополнительные задания (необязательно, но будет плюсом)

1. Реализуйте декоратор `@retry`
    - Описание: Декоратор должен повторять вызов функции при возникновении определенных исключений. Параметры:
        - `times`: количество повторов (по умолчанию 3).
        - `delay`: задержка между повторами в секундах (по умолчанию 1).
        - `exceptions`: кортеж исключений, при возникновении которых следует повторить вызов (по умолчанию `Exception`).
    - Применение: Используйте этот декоратор для функций взаимодействия с базой данных или внешними сервисами в вашем приложении.
2. Создайте асинхронную версию API
    - Описание: Реализуйте один из эндпоинтов вашего API с использованием асинхронной функции.
    - Требование: Используйте `asyncio` и убедитесь, что асинхронная версия работает корректно в общем контексте приложения.
3. Напишите сложные тесты
    - Описание: Добавьте параметризованные тесты для ваших функций, покрывая как можно больше граничных случаев.
    - Требование: Используйте `pytest` и включите в тесты проверки на исключения и ошибки.
4. Кэширование: Реализуйте кэширование с помощью Redis для ускорения получения данных.

Ожидаемый результат

- Репозиторий с исходным кодом (GitHub, GitLab и т.д.).
- Подробные инструкции по запуску в `README.md`.
- Краткое описание архитектуры и принятых решений.

Как будут оцениваться работы

- Глубина понимания и правильность использования выбранного фреймворка (Django или FastAPI).
- Умение работать с базами данных.
- Реализация и интеграция gRPC-сервиса.
- Использование брокера сообщений для коммуникации между сервисами.
- Качество кода и структура проекта.
- Наличие тестов и документации.
- Выполнение дополнительных заданий будет преимуществом.