Техническое задание на разработку сервиса «Библиотека»

Ласкин Павел

1. Цели и задачи

Разработка сервиса для централизованного управления каталогом книг и авторов, а также бронирования книг для читателей. Основные задачи:

- создание API для управления книгами и авторами;
- реализация бронирования книг с указанием читателя и срока;

2. Функциональные требования

Сервис должен предоставлять следующие операции:

Управление книгами

- **Добавление книги:** название + список идентификаторов авторов.
- **Изменение книги:** обновление названия и/или списка авторов.
- **Получение информации о книге:** по ID возвращаются название, авторы, статус брони, данные о брони (кто и на какой срок).
- **Бронирование книги:** установка статуса `booked = true`, сохранение имени читателя, дат начала и окончания брони (срок — 14 дней от текущего момента).
- **Снятие брони:** сброс статуса и обнуление связанных полей.

Управление авторами

- **Регистрация автора:** имя (латиница, цифры, пробелы, длина 1–512).
- **Изменение имени автора:** по ID.
- **Получение информации об авторе:** по ID возвращается имя.

Связь книг и авторов

- **Получение книг автора:** поток (server-side stream) всех книг, написанных указанным автором, с полной информацией (включая статус брони).

3. Требования к API

API реализуется по протоколу gRPC с автоматической генерацией REST-эндпоинтов через gRPC-gateway. Спецификация описывается в формате Protocol Buffers (v3).

Основные методы (REST-соответствие):

- `POST /v1/library/book` — добавить книгу;
- `PUT /v1/library/book` — изменить книгу;
- `GET /v1/library/book/{id}` — получить книгу;
- `POST /v1/library/book/{id}/reserve` — забронировать (тело: {"booked_by": "имя"});
- `POST /v1/library/book/{id}/release` — освободить книгу;
- `POST /v1/library/author` — зарегистрировать автора;
- `PUT /v1/library/author` — изменить автора;
- `GET /v1/library/author/{id}` — получить автора;
- `GET /v1/library/author_books/{author_id}` — получить книги автора (stream).

Валидация входных данных осуществляется с использованием `protoc-gen-validate`. Все идентификаторы должны быть валидными UUID; имя автора — по шаблону $^[A-Za-z0-9]+([A-Za-z0-9]+)*\$, 1\check{}512; \bar{}1\check{}512;$ `booked_by`$\bar{}1\check{}256$.

4. Технологический стек

- `Язык реализации: Go 1.22+, архитектура - чистая (Clean Architecture).`
- `Протоколы: gRPC, REST (gRPC-gateway).`
- `База данных: PostgreSQL 15, миграции - goose, драйвер - pgx.`
- `Логирование: zap.`
- `Конфигурация: переменные окружения (12-factor).`
- `Контейнеризация: Docker, оркестрация - Kubernetes (на последующих этапах).`
- `CI/CD: GitHub Actions (сборка, тестирование, публикация образа).`

5. Модель данных

```
author (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT now(),
    updated_at TIMESTAMPTZ DEFAULT now()
)

book (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
    name TEXT NOT NULL,
    booked BOOLEAN DEFAULT FALSE,
    booked_by TEXT,
    reservation_start TIMESTAMPTZ,
    reservation_end TIMESTAMPTZ,
    created_at TIMESTAMPTZ DEFAULT now(),
    updated_at TIMESTAMPTZ DEFAULT now()
)

author_book (
    author_id UUID REFERENCES author ON DELETE CASCADE,
    book_id UUID REFERENCES book ON DELETE CASCADE,
    PRIMARY KEY (author_id, book_id)
)
```

Индексы: на author(name), book(name), author_book(book_id), book(booked_by), book(reserv
Для автообновления updated_at используются триггеры.