# Команда 1
# Электронная библиотека

Участники:

- Егор Калугин - **Тимлид-Разработчик**
- Алина Лукьянова - **Тестировщик QA-Разработчик**
- Игорь Барбашов - **Аналитик-Разработчик**
- Денис Еременко - **Разработчик**

# Электронная библиотека

## Основная информация о проекте

Приложение для поиска, чтения и добавления книг.

Основные функции:
•Поиск книг по названию, автору и категории.
•Страница книги с описанием, отзывами и рейтингом.
•Добавление книг в «Избранное».
•Личный кабинет пользователя для отслеживания прочитанных книг.

Дополнительные функции:
•Возможность оставлять отзывы и выставлять оценки.
•Интеграция с внешними API, например, Google Books API

# Электронная библиотека

## Требования к бэкенду

1. Разработать модель данных для реализаци и основных функций проекта. Минимальный набор таблиц:

    •автор: идентификатор, ФИО, год рождения, жанры, категории и подобное;

    •категория: идентификатор, наименование категории;

    •книга: идентификатор, название, год издания, ссылка на автора;

    •избранное: ссылка на пользователя, ссылка на книгу, статус прочтения и подобное.

2. Разработать модель данных для работы сервиса регистрации, авторизации и аутентификации пользователей: идентификатор, имя, фамилия, логин, пароль.

3. Разработать API для регистрации, авторизации и аутентификации. Эндпойнты: регистрация, аутентификация, смена пароля, обновление токена.

4. Разработать API, обеспечивающий работу основных функций для работы с книгами:

5. создание, удаление, редактирование, чтение (пагинация, фильтры по категории, дате, автору).

6. Разработать API, обеспечивающий работу основных функций для работы с избранным: добавление в избранное, удаление и избранного, просмотр и подобное.

# Электронная библиотека

## Цели проекта

1. Разработка учебного серверного приложения на основе FastAPI с использованием SQLAlchemy и PostgreSQL, демонстрирующего принципы построения REST API и работы с реляционными базами данных.

2. Освоение методов тестирования серверных приложений, включая написание модульных и интеграционных тестов для API и слоя работы с базой данных с использованием pytest.

3. Получение практического опыта командной разработки программного обеспечения с использованием системы контроля версий (gitlab) и распределения задач между участниками проекта.

# Электронная библиотека

## Архитектура

Проект разработан в соответствии с принципами Layered Architecture, обеспечивающими разделение ответственности между слоями приложения. Архитектура включает слой API, слой бизнес-логики, слой доступа к данным и слой моделей.

Для повышения масштабируемости и удобства сопровождения использован доменно-ориентированный подход, при котором каждая предметная область реализована в виде отдельного модуля.

| Папка / Файл | Назначение |
|---|---|
| `/api/v1/init.py` | инициализация всех `API` (подключение роутеров к `app`) |
| `/db/db/py` | инициализация подключения к БД, создание и контроль работы с сессиями БД |
| `/src/<domain_name>` | директория домена |
| `/src/common` | базовые `orm`-модели и доменные схемы, от которых наследуются все остальные |
| `/src/common/association` | содержит файлы для связующих таблиц, реализующих отношения `many-to-many` |
| `tests` | `unit`-тесты |

Каждая `<domain_name>` папка содержит:

| Файл | Назначение |
|---|---|
| `api.py` | контролеры (эндпоинты) |
| `models.py` | `orm`-модели |
| `repository.py` | имплементация работы с БД средствами `ORM`. Должны наследоваться от протокола |
| `schema.py` | доменные модели, `pydantic`-схемы объектов `dto` и `response` |
| `services.py` | бизнес-логика |

# Электронная библиотека

## Ключевые решения

- В рамках проекта применялась командная модель разработки с использованием системы контроля версий Git и платформы GitLab. Для повышения качества кода использовалось кросс-ревью. Все участники команды придерживались единых соглашений об именовании и структуры проекта, что обеспечило согласованность и читаемость кода. Процессы сборки и запуска были автоматизированы с использованием Docker и Makefile.
- Код приложения организован в каталоге **src** и структурирован по доменам. Каждый домен включает контроллеры, ORM-модели, репозитории, Pydantic-схемы и сервисы. Общие модели и связующие таблицы вынесены в отдельный модуль **common**. Для обеспечения обратной совместимости реализовано версионирование API (**api/v1**). Тесты вынесены в отдельный каталог и структурированы по доменам.
- В качестве СУБД используется PostgreSQL, доступ к данным реализован через ORM SQLAlchemy. Для изоляции SQL-логики применён паттерн Repository. В качестве первичных ключей используются UUID. Связи many-to-many реализованы через отдельные ассоциативные таблицы.
- Для управления изменениями схемы базы данных используется Alembic. Миграции применяются автоматически при запуске приложения. Проект разворачивается в Docker-контейнерах и поддерживает режимы разработки и production. Управление сборкой, запуском и остановкой сервиса осуществляется с помощью Makefile.
- В проекте применяются линтеры и форматтеры, выполняется код-ревью и реализованы unit-тесты для проверки. Документация и ER-диаграмма поддерживаются в актуальном состоянии в файле README.md**.**

Project
FastAPI project
Pinned
Issues 2
Manage
Plan
Issues 2
Issue boards
Milestones
Wiki
Code
Build
Secure
Deploy
Operate
Monitor
Analyze
Settings
What's new 3
Help



Development
Search

## Open 2

**Авторизация - этап 2 (refresh_token)**
#28

**[EPIC] Интеграция с внешними API, например, Google Books API.**
#17

## in progress 0

## Closed 29

**Разработать API, обеспечивающий работу основных функций для работы с избранным: добавление в избранное, удаление из избранного, просмотр и подобное.**
in progress
#14

**Написать тесты на api Genre**
#32

**Страница книги с описанием, отзывами и рейтингом.**
#15

**Написание отчета по работе**
#31

**Тестирование - настроить тестовую среду**
#30

**Личный кабинет пользователя для отслеживания прочитанных книг.**

# FastAPI 0.1.0 OAS 3.1

/api/openapi.json

Authorize

## author

## genre

## category

GET /api/v1/category/ Получить список категорий

POST /api/v1/category/ Создать категорию

GET /api/v1/category/{category_id} Получить категорию по id

PATCH /api/v1/category/{category_id} Обновить категорию

DELETE /api/v1/category/{category_id} Удалить категорию

## favorites

GET /api/v1/favorites/ Получить список избранного для текущего пользователя

POST /api/v1/favorites/ Добавить книгу в избранное

DELETE /api/v1/favorites/{book_id} Удалить книгу из избранного

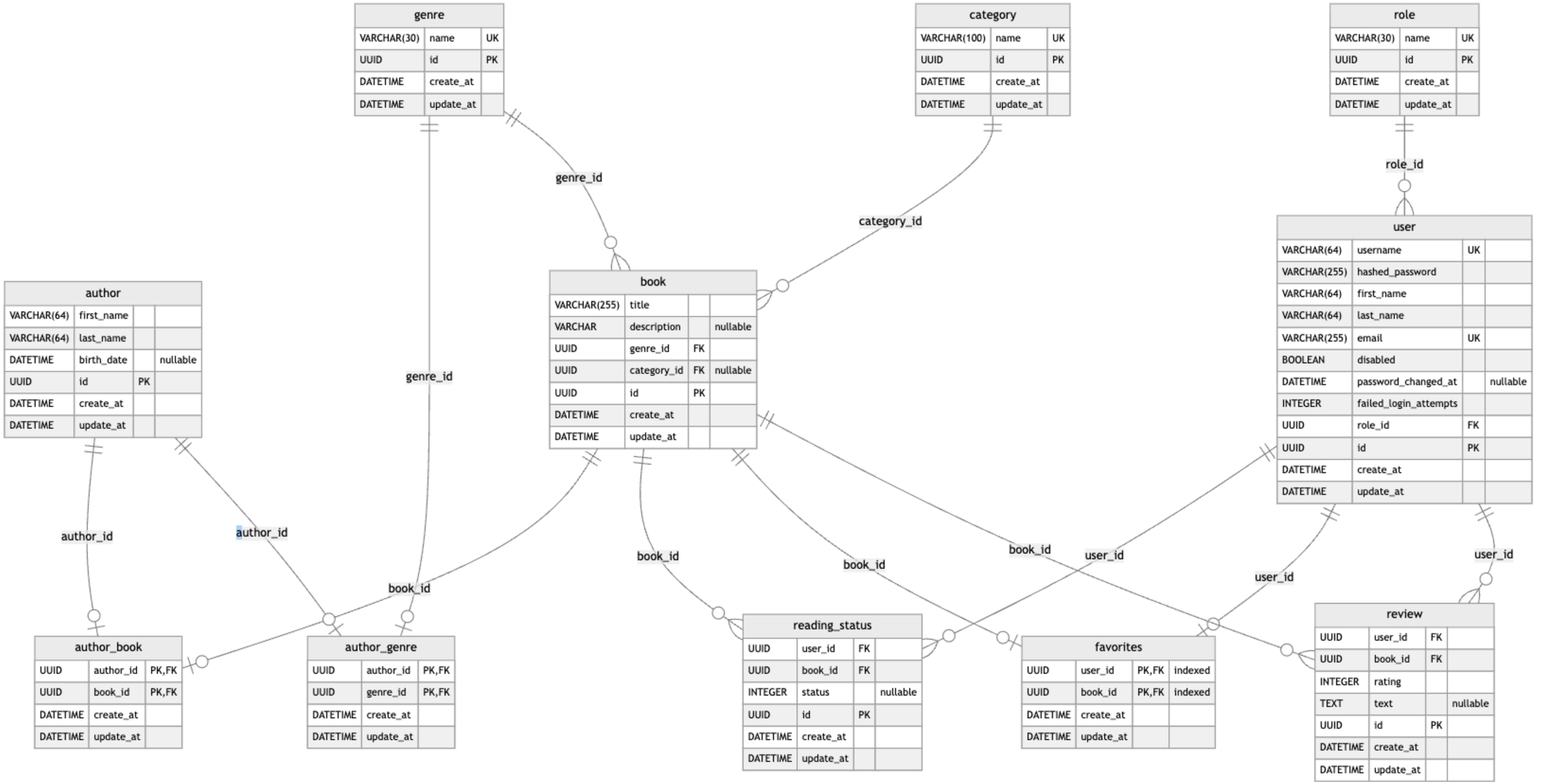
## review

## user

## book

## auth

# Схема данных



# Спасибо за внимание!