САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Пиотуховский Александр

K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Необходимо реализовать RESTful API, используя ранее написанный boilerplate.

Ход работы

Была использована архитектура, разработанная в ЛР1, включающая: слой моделей и репозиториев (для работы с данными), слой DTO (для валидации и сериализации), слой сервисов (для бизнес-логики), слой роутов (эндпоинты).

На основе этой архитектуры реализованы REST-эндпоинты для сущностей. Каждый из них поддерживает набор операций:

- GET /films – получить список,
- GET /films/{film_id} – получить сущность по ID,
- POST /films – создать новый объект,
- PUT /films/{film_id} – обновить объект,
- DELETE /films/{film_id} – удалить объект.

На рисунке 1 изображены маршруты REST API для сущности Film, реализованные на основе boilerplate. Маршруты включают работу с основными ресурсами: постеры, трейлеры, рейтинги, комментарии, а также фильтрацию и поиск.

```
1   import ...
4
5   routes = [
6       Route( path: "/", controllers.Film),
7       Route( path: "/{film_id:int}", controllers.FilmDetail),
8       Route( path: "/{film_id:int}/poster", controllers.Poster),
9       Route( path: "/{film_id:int}/trailer", controllers.Trailer),
10      Route( path: "/{film_id:int}/rating", controllers.FilmRating),
11      Route( path: "/{film_id:int}/comments", controllers.FilmComment),
12      Route( path: "/{film_id:int}/comments/{comment_id:int}", controllers.FilmCommentDetail),
13      Route( path: "/filters", controllers.FilmFilter),
14      Route( path: "/search", controllers.FilmSearch),
15      Route( path: "/gigasearch", controllers.FilmGigaSearch),
16  ]
17
```

Рисунок 1 – маршруты REST API для сущности Film

Реализация опирается на DI-контейнер, автоматически внедряющий нужные зависимости в контроллеры. Это обеспечивает изоляцию слоёв и гибкость подмены реализации. На рисунке 2 изображён контроллер для сущности Film, демонстрирующий обработку HTTP-запросов GET и POST. Контроллер использует внедрение зависимости IFilmService, обрабатывает запросы, вызывает бизнес-логику и возвращает структурированный ответ с нужным HTTP-статусом.

```python
class Film(HTTPEndpoint):  1 usage
    __service: IFilmService = container.resolve(IFilmService)

    async def get(self, request: Request):
        dto = GetFilmsDTO(**request.query_params)
        films = await self.__service.get_films_assortment(dto)

        return JSONResponse(status_code=status.HTTP_200_OK, content=films.dict())

    @requires( scopes: "authenticated", status_code=401)
    @requires( scopes: "admin", status_code=403)
    async def post(self, request: Request):
        data = await request.json()
        dto = CreateFilmDTO(**data)
        created_film = await self.__service.create_new_film(dto)

        return JSONResponse(status_code=status.HTTP_201_CREATED, content=created_film)
```

Рисунок 2 – контроллер для сущности Film

Валидация входных данных и формирование ответов осуществляются через схемы DTO.

Вывод

В ходе выполнения лабораторной работы было реализовано RESTful API на базе ранее разработанного boilerplate-приложения. API соответствует принципам REST и включает все необходимые CRUD-операции для ключевых сущностей. Проект легко расширяется и масштабируется благодаря использованию DI и принципов чистой архитектуры.