Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное  
 образовательное учреждение высшего образования  
 «Самарский национальный исследовательский университет  
 имени академика С. П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

**Технологии сетевого программирования**

**Финальный отчет по лабораторным**

**«Трекер карьеры»**

Выполнили:

Фаюстова А.П.

Должиков Д.П.

группа 6303-010302D

Самара, 2025

СОДЕРЖАНИЕ

[1 Проектирование приложения 4](#_Toc199679186)

[2 Разработка базы данных 8](#_Toc199679187)

[3 Разработка API 9](#_Toc199679188)

[4 Авторизация 16](#_Toc199679189)

[5 Разработка клиентской части 18](#_Toc199679190)

[6 Контейнеризация приложения 24](#_Toc199679191)

[Заключение 26](#_Toc199679192)

[Приложение А Конфигурации 27](#_Toc199679193)

[А.1 Пример .env файла 27](#_Toc199679194)

[А.2 Фрагмент кода файла config.py 27](#_Toc199679195)

[А.3 Конфигурация Ruff 28](#_Toc199679196)

[Приложение Б Слой репозитория 29](#_Toc199679197)

[Б.1 Фрагмент кода определяющий структуру БД. Файл models.py 29](#_Toc199679198)

[Б.2 Фрагмент кода слоя репозитория, определяющего взаимодействие с БД 34](#_Toc199679199)

[Б.3 Фрагмент кода паттерна UnitOfWork 38](#_Toc199679200)

[Приложение В Сервисный слой 39](#_Toc199679201)

[В.1 Фрагмент кода файла users.py 39](#_Toc199679202)

[В.2 Фрагмент кода файла roadmaps.py 41](#_Toc199679203)

[В.3 Фрагмент кода файла user\_roadmaps.py 43](#_Toc199679204)

[В.4 Фрагмент кода файла tokens.py 48](#_Toc199679205)

[Приложение Г Слой API 49](#_Toc199679206)

[Г.1 Фрагмент кода файла auth.py 49](#_Toc199679207)

[Г.2 Фрагмент кода файла users.py 53](#_Toc199679208)

[Г.3 Фрагмент кода файла roadmaps.py 55](#_Toc199679209)

[Г.4 Фрагмент кода файла user\_roadmaps.py 57](#_Toc199679210)

[Приложение Д Зависимости 61](#_Toc199679211)

[Д.1 Фрагмент кода, определяющий UserDep 61](#_Toc199679212)

[Д.2 Фрагмент кода, определяющий RedisDep 62](#_Toc199679213)

[Д.3 Фрагмент кода, определяющий PaginationDep 62](#_Toc199679214)

[Приложение Е Frontend 62](#_Toc199679215)

[Е.1 Фрагмент кода из api.jsx 62](#_Toc199679216)

[Е.2 Код компонента карточки 64](#_Toc199679217)

[Приложение З Контейнеризация 67](#_Toc199679218)

[З.1 Код файла DockerFile для сервиса хранения изображений 67](#_Toc199679219)

[З.2 Код файла DockerFile для основного сервиса 67](#_Toc199679220)

[З.3 Код файла DockerFile для фронтенда 68](#_Toc199679221)

[3.4 Код файла docker-compose.yaml, определяющего все контейнеры 69](#_Toc199679222)

1. Проектирование приложения

Тема проекта: “Сервис для трекинга карьеры”. Проект позволяет пользователям эффективно планировать и отслеживать свое профессиональное развитие. Пользователи могут выбирать тематические роадмапы из предложенного каталога и добавлять их в свой персональный список, создавать собственные роадмапы с различным уровнем видимости. Каждый роадмап представляет собой последовательность шагов, оформленных в виде карточек, содержащих ссылки на образовательные ресурсы, необходимые для освоения темы.

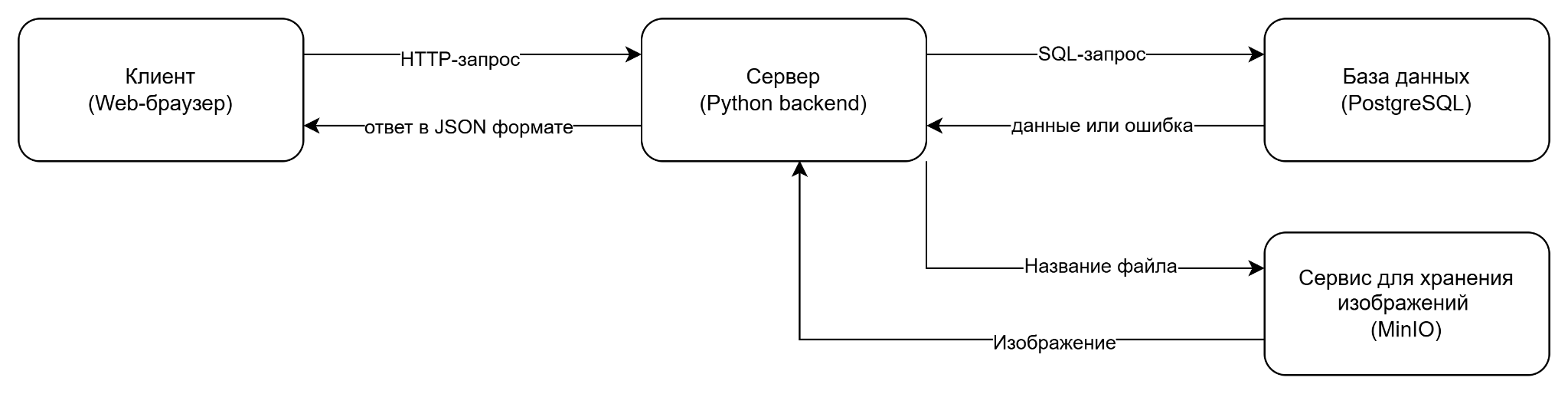


Рисунок 1 – Схема взаимодействия компонентов

Для реализации проекты был выбран следующий стек технологий:

Сервис хранения изображений:

Архитектура: MVC

Язык программирования: Java;

Фреймворк: Spring Boot;

S3: (MinIO).

Основной сервис:

Архитектура: слоистая (onion)

Язык программирования: Python.

Фреймворки и библиотеки: FastAPI, SQLAlchemy.

СУБД, с которыми взаимодействует сервис: PostgreSQL, Redis.

Миграции: Alembic.

Пакетный менеджер: UV.

Линтер и форматирование кода: Ruff.

Контейнеризация: Docker

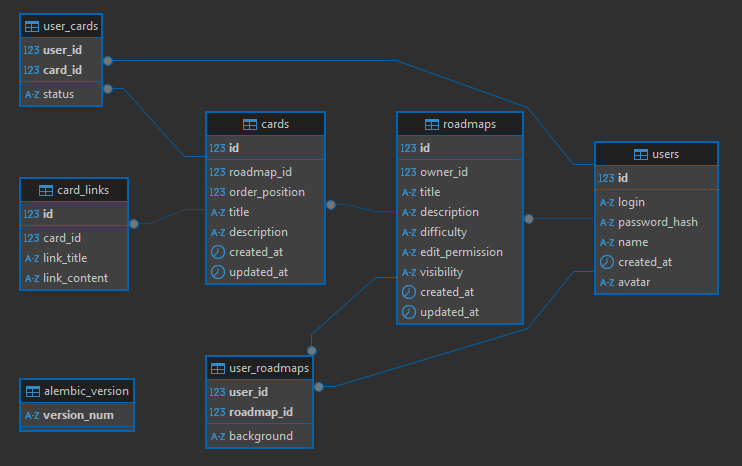


Рисунок 2– Схема базы данных

Объекты, представленные в базе данных:

**users -** представляет собой объект пользователя:

user\_id - идентификатор пользователя в системе;

login - логин пользователя для его аутентификации;

password\_hash - пароль пользователя;

name - имя пользователя;

created\_at - дата создания пользователя.

**roadmaps -** таблица с роадмапами:

roadmap\_id - идентификатор роадмапа в системе;

title - название роадмапа;

description - текстовое описание роадмапа;

difficulty - уровень сложности (начинающий, продвинутый, профи);

created\_at - дата создания роадмапа;

updated\_at - дата обновления роадмапа.

**cards -** таблица с карточками:

roadmap\_id - идентификатор роадмапа, в котором находится карточка;

order\_position - номер позиции по порядку, в котором находится карточка

card\_id - идентификатор карточки в системе;

title - название карточки;

description - текстовое описание карточки;

created\_at - дата создания карточки;

updated\_at - дата обновления карточки.

**card\_links** - внутреннее содержимое карточки:

card\_id - идентификатор карточки;

link\_title - название ресурса, на который ведет ссылка;

link\_content - URL ссылка на ресурс.

**user\_roadmaps:**

user\_id - идентификатор пользователя;

roadmap\_id - идентификатор роадмапа;

role - тип доступа к роадмапу для других пользователей (скрытый (доступный только создателю), доступный по ссылке, доступный всем пользователям).

**user\_cards:**

card\_id - идентификатор карточки;

user\_id - идентификатор пользователя в системе;

status - прогресс прохождения данной карточки.

Перед началом разработки необходимо было произвести конфигурацию линтера ruff, чтобы весь написанный код был в едином стиле. Данная конфигурация представлена в [приложении А.3](#_8zipm3n44pfu).

1. Разработка базы данных

В качестве СУБД был выбран PostgreSQL и поднят его образ в Docker контейнере, подробнее в разделе [6 Финализация приложения и упаковка в Docker](#_psmithoxhmtw).

Миграции в основном сервисе на Python настроены с помощью Alembic, код файла models.py, содержащий структуру БД находится в [приложении Б.1](#_9xc183196mah).

Взаимодействие с БД производилось через слой репозитория, код которого представлен в [приложении Б.2](#_a4dighglicpz). Для реализации транзакций при работе с БД был использован паттерн UnitOfWork, код которого представлен в [приложении Б.3](#_2yaslupyb6gn).

Слой сервисов представляет собой множество файлов, в которых реализована обработка бизнес-логики проекта. Фрагменты кода сервисного слоя представлен в [приложении В](#_uhs9anjl403k).

1. Разработка API

Фрагменты кода слоя API представлены в [приложении Г](#_t24govo7u92z). Для реализации многих эндпоинтов понадобилась инъекция зависимостей в параметры этих эндпоинтов. Данные инъекции позволили: корректно взаимодействовать с объектами UnitOfWork; защитить эндпоинты, т.е. предотвратить доступ для неавторизованных пользователей; корректное использование redis\_client; реализовать пагинацию. Код зависимостей находится в [приложении Д](#_6wjthur0ijs1).

Users api

| Метод | URL | Описание | Параметры | Формат ответа |
| --- | --- | --- | --- | --- |
| GET | /users | Получить данные текущего пользователя | — | 200 OK:  {id, login, name, created\_at, avatar} |
| DELETE | /users | Удалить аккаунт текущего пользователя | {access\_token, refresh\_token}  (cookie) | 204 No Content |
| PATCH | /users | Обновить данные текущего пользователя | {name, password\_hash, avatar} | 200 OK {user\_id, login, username} |
| GET | /users/avatar | Получить название файла аватара | — | 200 OK {avatar} |
| PUT | /users/avatar | Изменить название файла аватара | — | 200 OK {avatar} |
| DELETE | /users/avatar | Удалить аватар | — | 200 OK {null} |

Auth api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| GET | /auth/me | Получить информацию о текущем пользователе | — | 200 OK:  {id, login, username} |
| POST | /auth | Создать пользователя | {name, login, password\_hash} | 201 Created:  {user\_id} |
| POST | /auth/token | Войти в аккаунт | {username, pasword} | 200 OK:  {access\_token, token\_type,    id,    login,    username,    avatar} |
| POST | /auth/refresh | Обновить access token | {access\_token, refresh\_token}  (cookie) | 200 OK:  {access\_token, token\_type} |
| POST | /auth/logout | Выйти из профиля | {access\_token, refresh\_token}  (cookie) | 200 OK:  {detail:"Successfully logged out"} |

Roadmaps api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| GET | /roadmaps/public | Получить список публичных роадмапов | {search, difficulty, limit, page} | 200 OK:  {roadmaps:[], total\_pages} |
| POST | /roadmaps | Создать роадмап | {owner\_id, title, description, difficulty, edit\_permission, visibility} | 201 Created:  {roadmap\_id, user\_id} |
| GET | /roadmaps/{roadmap\_id} | Получить данные роадмапа | — | 200 OK:  {owner\_id, id, title, description, difficulty, edit\_permission, visibility, created\_at, updated\_at, cards:[]} |
| PATCH | /roadmaps/{roadmap\_id} | Обновить данные роадмапа | {title, description, difficulty, edit\_permission, visibility} | 200 OK: {roadmap\_id} |
| DELETE | /roadmaps/{roadmap\_id} | Удалить роадмап | — | 204 No Content |

User\_roadmaps api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| GET | /user\_roadmaps | Получить список роадмапов у пользвователя | {search, difficulty, limit, page} | 200 OK:  {roadmaps: [], total\_pages} |
| POST | /user\_roadmaps/{roadmap\_id} | Привязать пользователя к роадмапу | — | 200 OK:  {roadmap\_id, user\_ id, card\_ids:[]} |
| DELETE | /user\_roadmaps/{roadmap\_id} | Удалить привязку пользователя к роадмапу | — | 204 No Content |
| PUT | /user\_roadmaps/{roadmap\_id}/background | Поменять фон у роадмапа | roadmap\_id | 200 OK {background} |
| GET | /user\_roadmaps/in\_progress | Роадмапы, где есть карточки со статусом “в процессе” | {search, difficulty, limit, page} | 200 OK:  {roadmaps: [], total\_pages} |
| GET | /user\_roadmaps/{roadmap\_id}/background | Фон роадмапа | — | 200 OK {background} |

Cards api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| GET | /cards/{card\_id} | Получить данные карточки | — | 200 OK:  {id, order\_position,  title, description, created\_at, updated\_at, status,  links:[]} |
| POST | /cards | Добавить карточку в роадмап | {roadmap\_id, title, description, order\_position} | 200 OK:  {id} |
| PATCH | /cards/{card\_id} | Обновить карточку | {title, description, links, order\_position} | 200 OK  {card\_id} |
| DELETE | /cards/{card\_id} | Удалить карточку | — | 204 No Content |

Card\_links api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| DELETE | /card\_links/{card\_lonk\_id} | Удалить ссылку из карточки | — | 204 No Content |
| PATCH | /card\_links/{card\_lonk\_id} | Изменить ссылку у карточки | {link\_title, link\_content} | 200 OK  {card\_link\_id} |
| POST | /card\_links | Добавить ссылку в карточку | {card\_id, link\_title, link\_content} | 200 OK  {card\_link\_id} |

 user\_cards api

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| PUT | /user\_cards/status |  | {status, card\_id} | 200 OK |

MinIO API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | URL | Описание | Параметры | Формат ответа |
| GET | files/{bucket}/{filename} | Получить файл | — | 200 OK:  Content(mediaType = "image/jpeg") |
| POST | files/{bucket} | Загрузить файл | — | 200 OK:  {filename} |
| DELETE | files/{bucket}/{filename} | Удалить файл | — | 200 OK:  {“message”} |

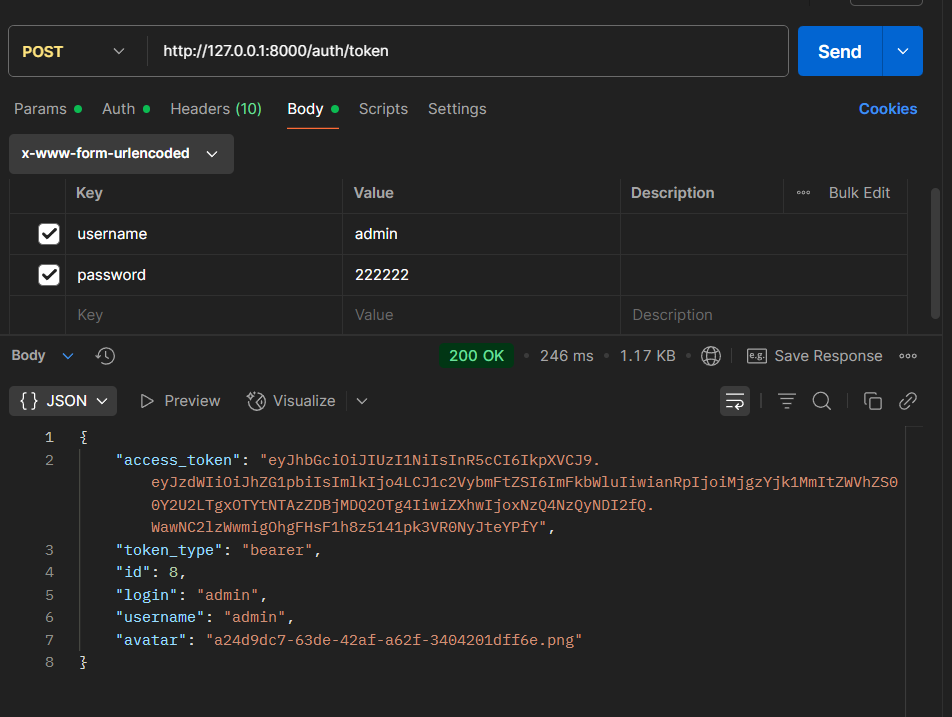
**Тестирование API с помощью Postman**

Рисунок 3 – Пример POST запроса на авторизацию

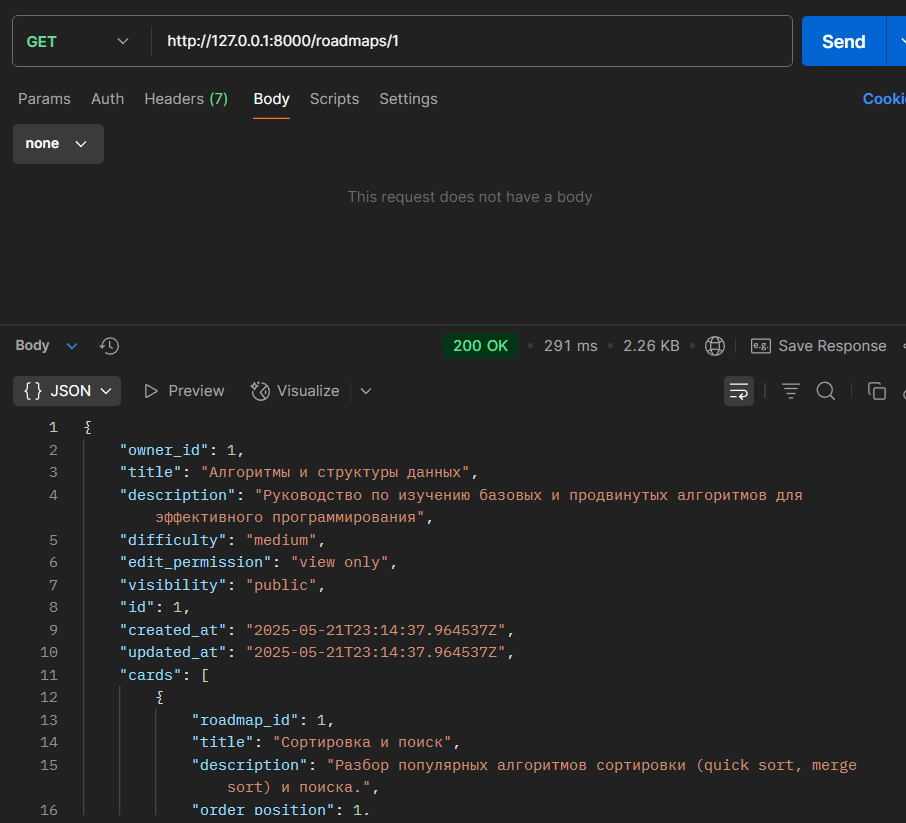


Рисунок 4 – Пример GET запроса на информацию о роадмапе

1. Авторизация

Была настроена аутентификация с помощью JWT, а также защищены необходимые эндпоинты с помощью UserDep. Код эндпоинтов, ответственных за аутентификацию и авторизацию представлен в [приложении Г.1](#_suvtmj3436wl).

За регистрацию пользователя отвечает эндпоинт /auth на него необходимо отправить POST запрос с JSON объектом следующего вида:

{

"name": "string",

"login": "string",

"password\_hash": "string"

},

где name — имя пользователя, login его логин, а password\_hash пароль, который после будет захеширован.

Ответы эндпоинта:

* статус 201: операция успешно выполнена. В качестве ответа возвращается идентификатор пользователя;
* статус 422: структура JSON неверная;
* статус 409: пользователь с данным логином уже существует.

Чтобы пользователю войти в аккаунт, необходимо воспользоваться эндпоинтом /auth/token, отправив логин и пароль. Тип запроса POST.

Ответы эндпоинта:

* статус 200: пользователь успешно аутентифицирован и авторизован. В ответ возвращаются токен авторизации, тип токена, идентификатор, имя, логин и аватар пользователя. Также в куки записываются access и refresh токены со статусом httponly и secure;
* статус 422: структура JSON неверная;
* статус 401: пользователь не прошел аутентификацию.

Для обновления access токена служит эндпоинт /auth/refresh с типом запроса POST. Данный эндпоинт использует refresh токен для обновления токена доступа. Используется Redis для проверки наличия данных токенов в списке отозванных, после прохождения проверки создаются новые токены, а старые добавляются в список отозванных.

Ответы эндпоинта:

* статус 200: токен успешно обновлен. В ответ возвращается токен авторизации и тип токена. Также в куки записываются access и refresh токены со статусом httponly и secure;
* статус 422: структура JSON неверная;
* статус 401: отсутствует refresh токен, имеет некорректную структуру, либо находится в списке отозванных.

Выход из аккаунта производится в эндпоинте /auth/logout запросом типа POST. Эндпоинт заносит access и refresh токены в список отозванных в Redis и удаляет их из куки.

Ответы эндпоинта:

* статус 200: пользователь успешно вышел из аккаунта, токены отозваны;
* статус 422: структура JSON неверная;
* статус 401: отсутствуют токены, имеют некорректную структуру, либо находятся в списке отозванных.

1. Разработка клиентской части

Для разработки клиентской части была использована библиотека React, библиотеки react-bootstrap для использования готовых компонентов и Bootstrap-icons для добавления иконок на кнопки. Для сборки фронтенда использовался Vite. Для доработки дизайна использовался css.

Отправка запросов на сервер была реализована через axios в отдельном файле api.jsx ([приложение Е.1](#_hduors52vvqi)).

Структура фронтенда:

Api - для отправки запросов на сервер

api.jsx

Components - компоненты

Header

RoadmapView

CardView

RoadmapRedact

CardRedact

Pages - страницы

AuthPage (/) - страница авторизации

RegisterPage (/register) - страница регистрации

EditProfilePage (/editprofile) - страница редактирования аккаунта

MyRoadmapsPage (/myroadmaps) - страница “Мои роадмапы”

RoadmapSearchPage (/roadmapsearch) - страница поиска публичных роадмапов

CardsPage (/myroadmaps/:id или /roadmapsearch/:id) - страница с карточками внутри роадмапа

Страницы:

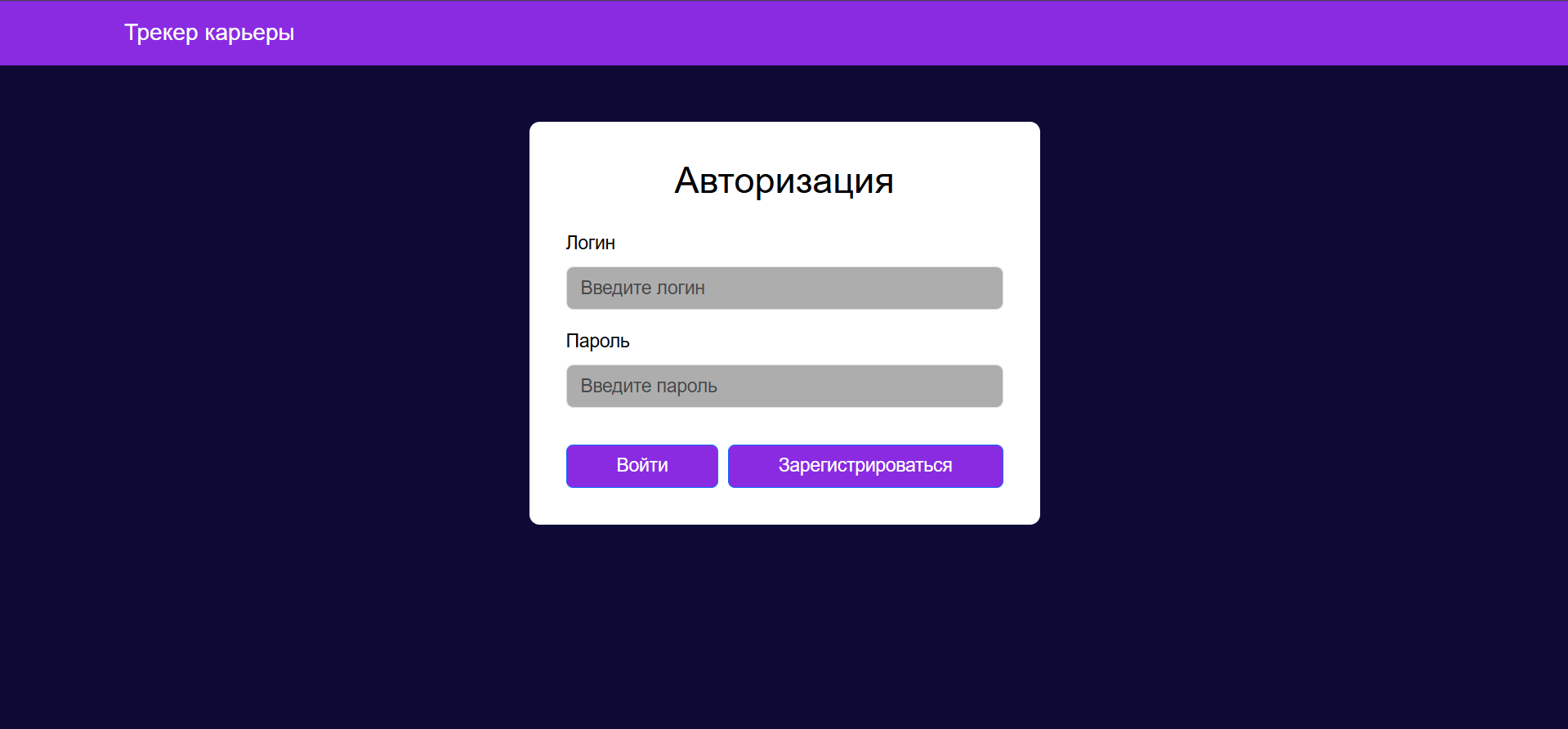


Рисунок 5 – Страница авторизации

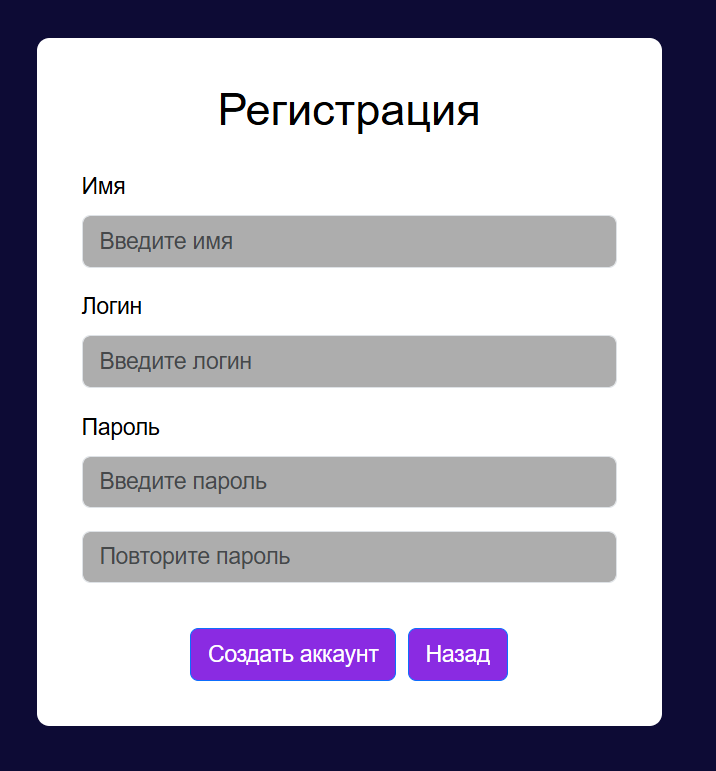


Рисунок 6 – Страница регистрации

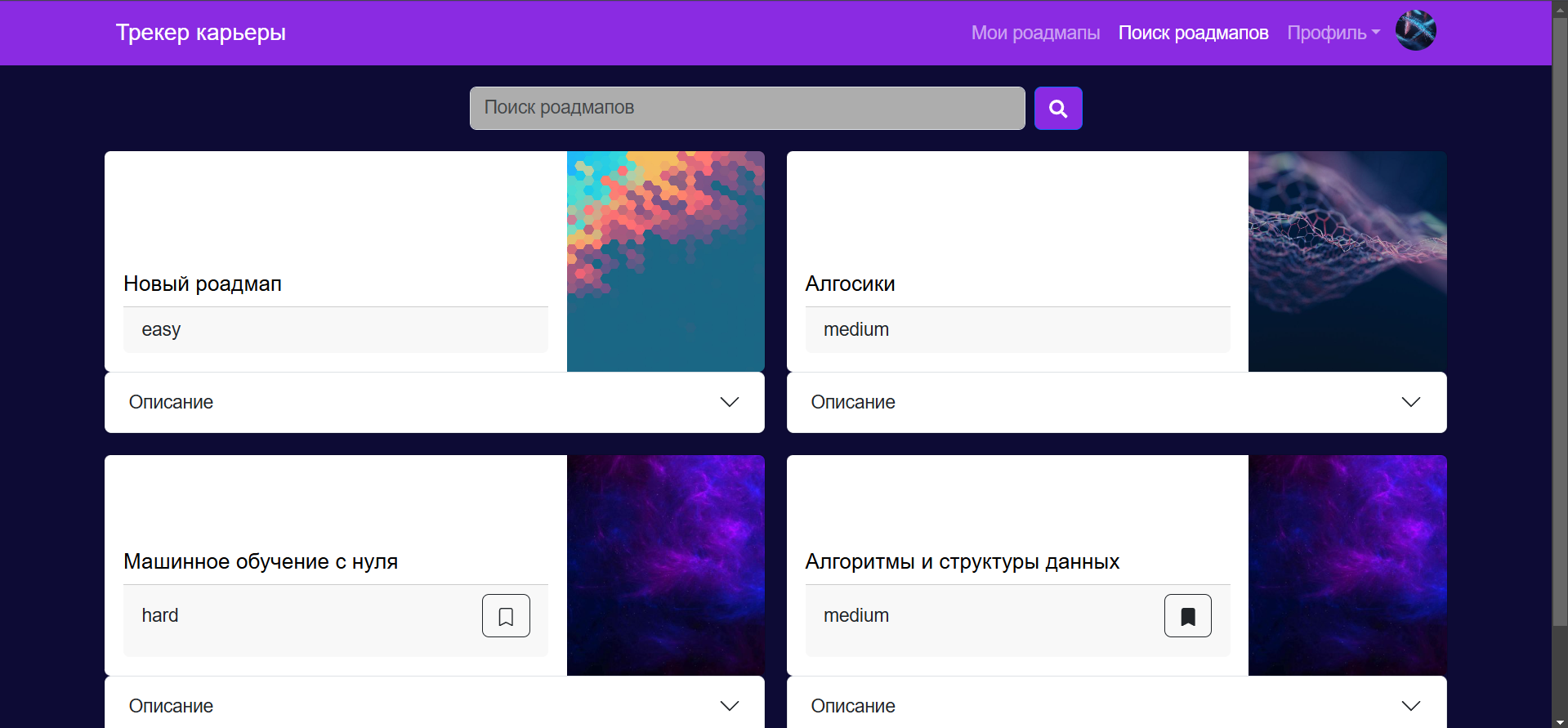


Рисунок 7 – Страница поиска публичных роадмапов

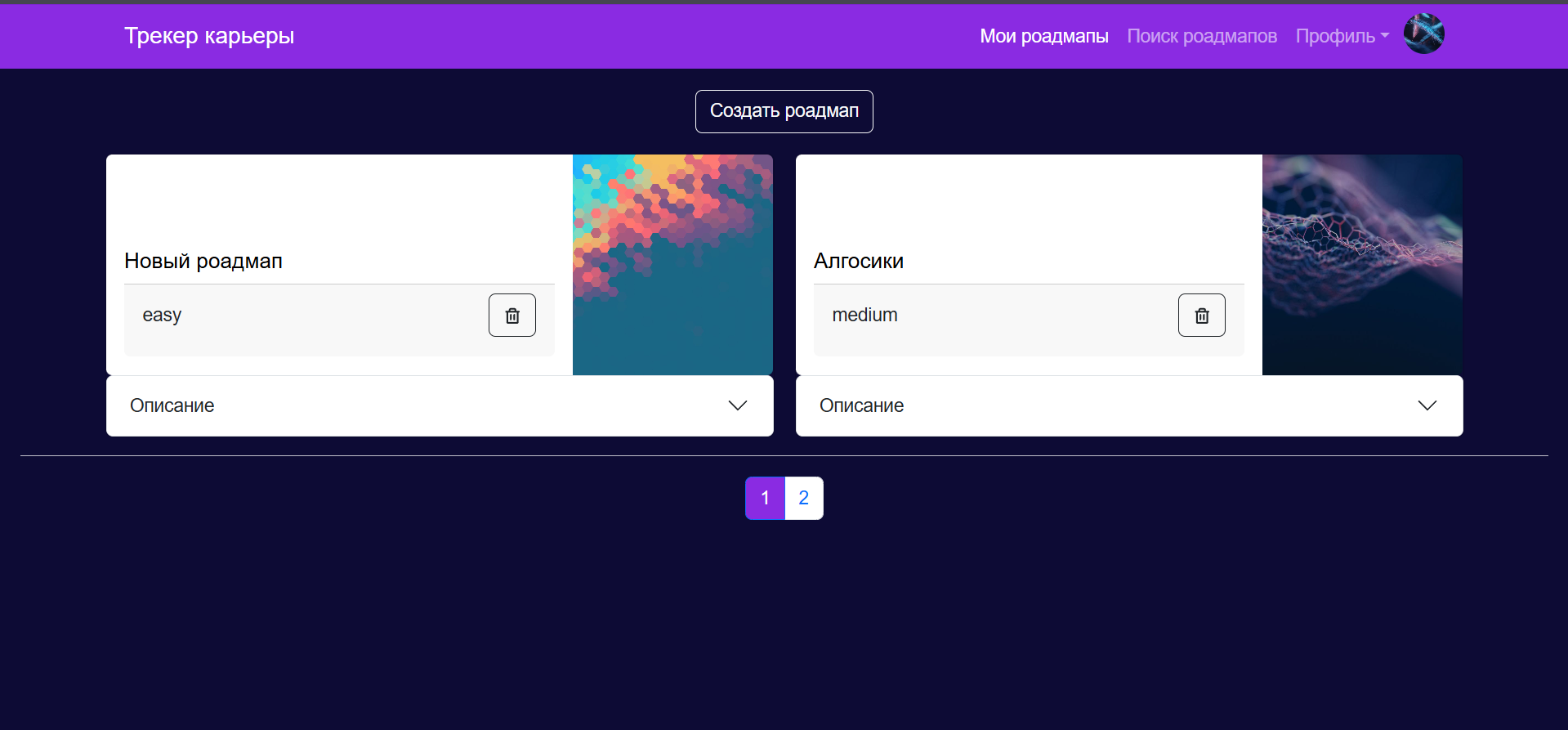


Рисунок 8 – Страница “Мои роадмапы”

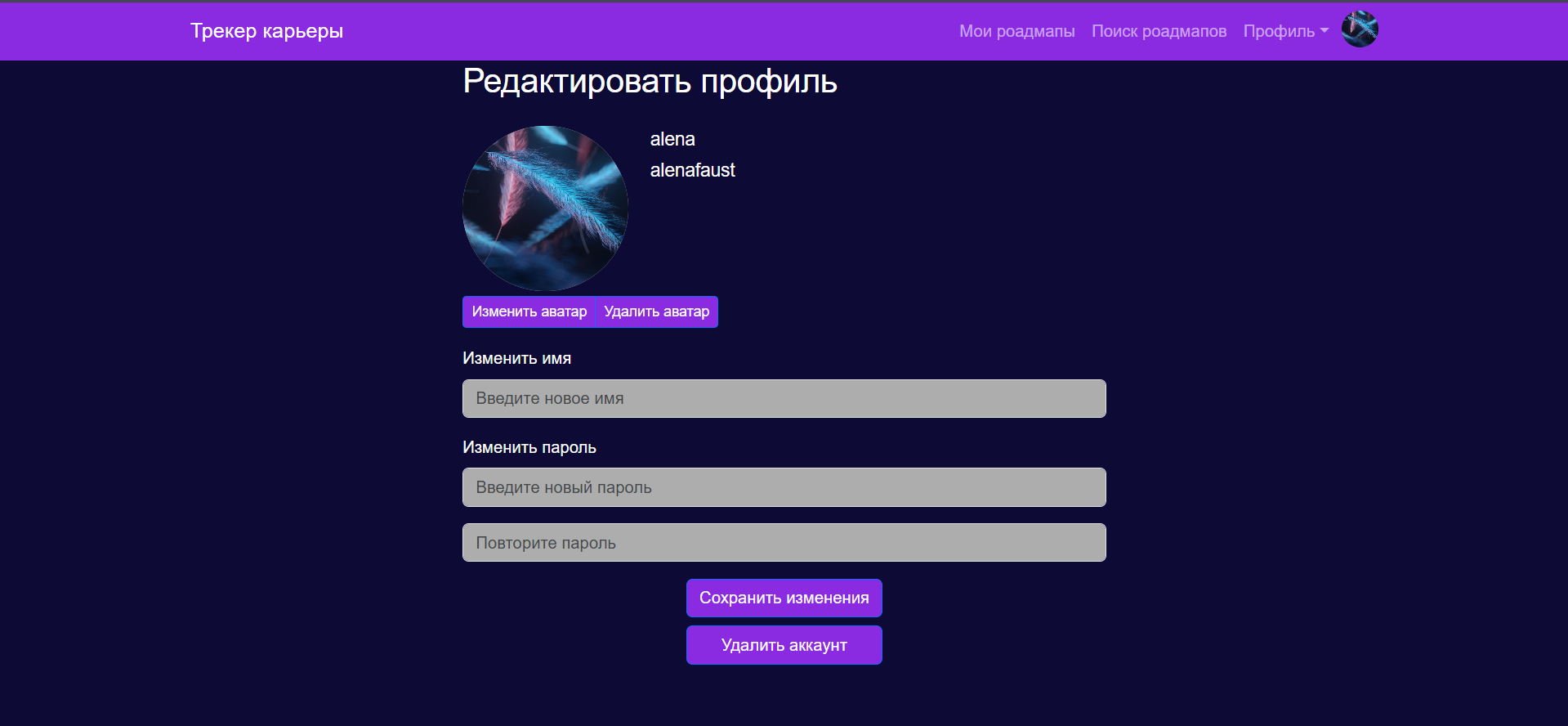


Рисунок 9 – Страница редактирования аккаунта

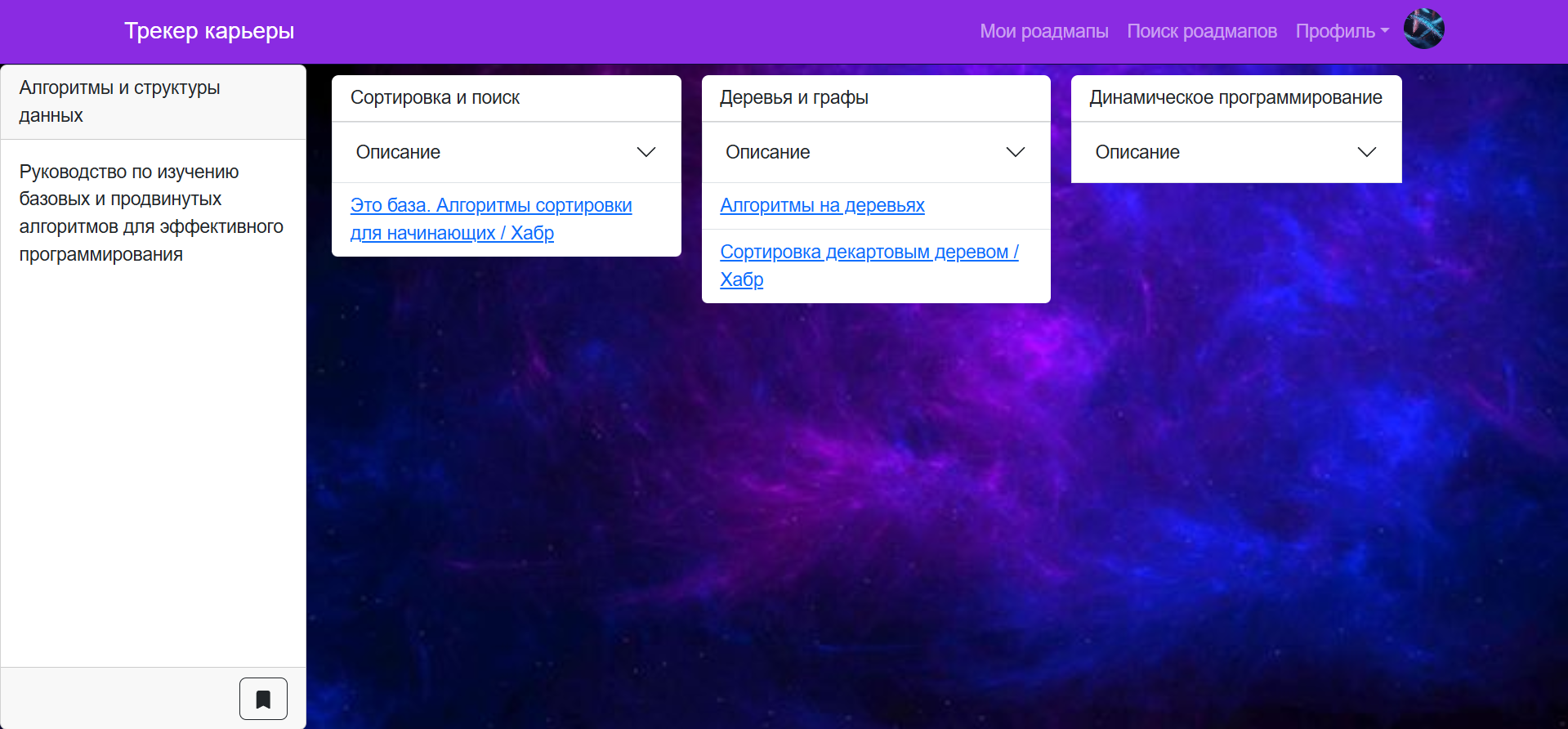


Рисунок 10 – Страница с карточками внутри роадмапа

Компоненты:

**Компонент карточки (CardView)** - представляет собой внешний вид карточки с названием, описанием и ссылками ([приложение Е.2](#_zb52ppt6djud)).

**Компонент роадмапа (RoadmapView)**- реализует внешний вид роадмапа, на котором указано название, описание, уровень сложности и присутствует кнопка добавления роадмапа в свою коллекцию или удаления из нее.

**Компонент создания/редактирования роадмапа (RoadmapRedact)** - реализован на основе модального окна, которое имеет 2 режима, меняющиеся через пропсы.

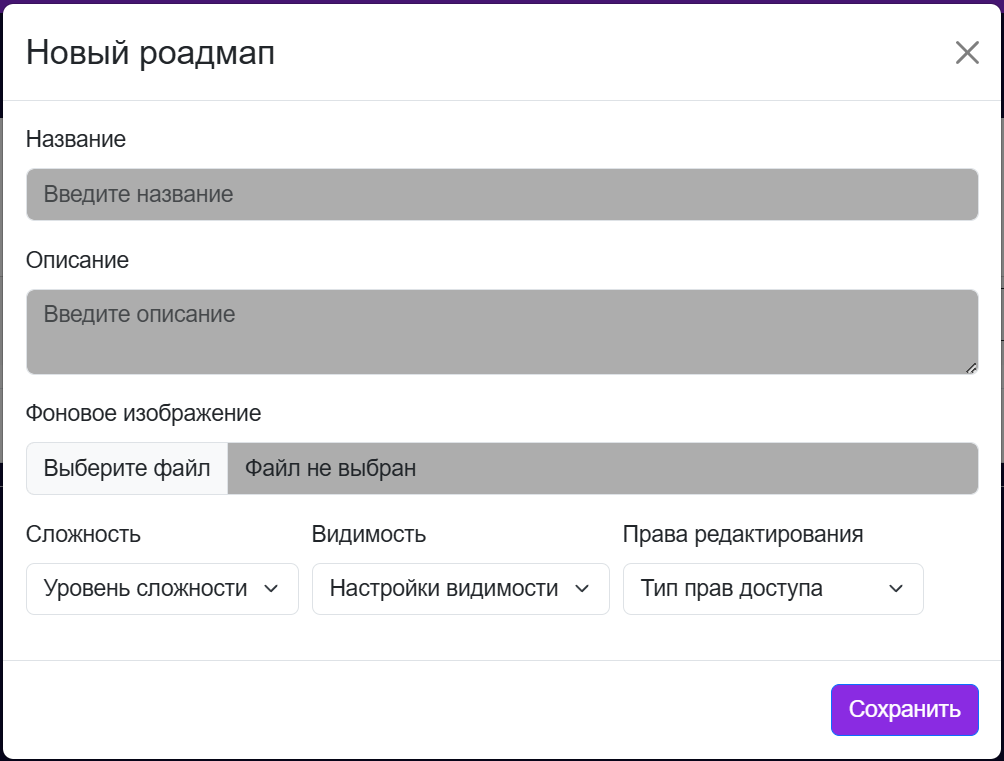


Рисунок 11 – Окно редактирования роадмапа

**Компонент создания/редактирования карточки внутри роадмапа (CardRedact)** - так же представляет собой модальное окно с 2 режимами. Позволяет добавлять несколько полей для ссылок, а также удалять их.

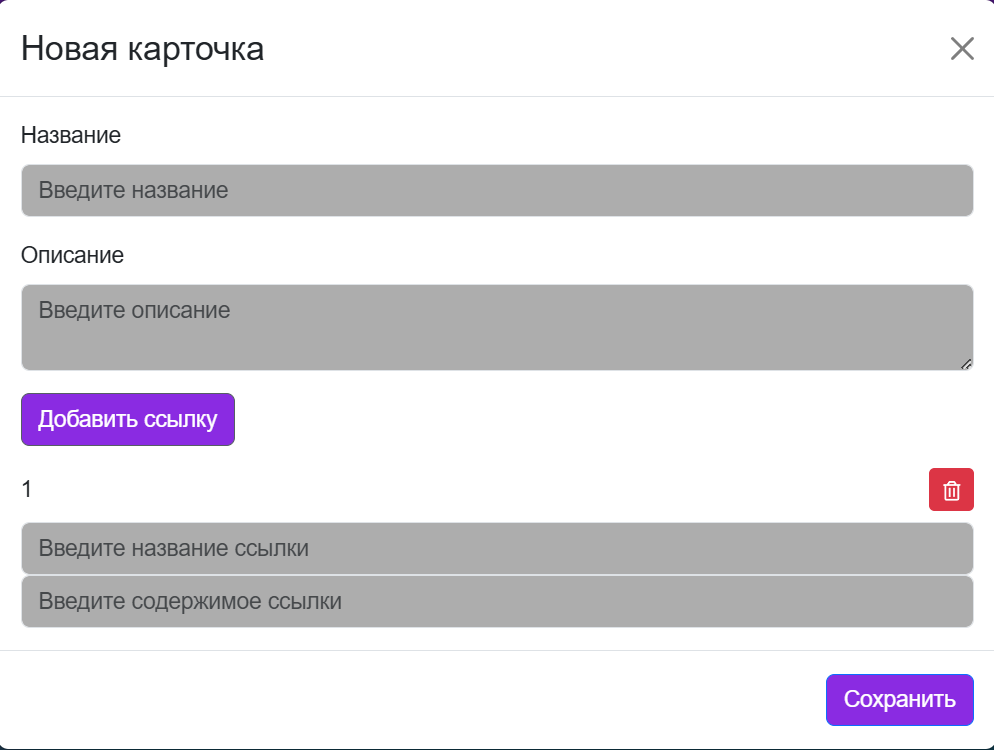


Рисунок 12 – Окно редактирования карточки

**Шапка (Header) -**  находится в верхней части всех страниц. Нужна для перехода между страницами. Так же содержит выпадающее меню и аватар пользователя.

1. Контейнеризация приложения

В финале приложение было упаковано в Docker контейнеры. Код от файлов DockerFile и docker-compose.yml находятся в [приложении З](#_fi4rr2dowfxz). Почти у каждого контейнера присутствуют переменные окружения, которые берутся из .env файла, пример которого представлен в [приложении А.1](#_z8s340rdoqwp).

Давайте теперь рассмотрим подробнее устройство docker-compose.yml.

Для фронтенда понадобился всего 1 контейнер. Работа фронтенда реализована на порту 5050. В docker-compose файле указан путь до DockerFile и необходимые аргументы сборки.

Для сервиса хранения изображений используется 2 контейнера:

* контейнер с образом MinIO. Данный контейнер работает на портах 9000 и 9001. Порт 9000 служит для обращения к самому хранилищу S3, а порт 9001 необходим для работы интерфейса админ панели в браузере;
* контейнер с программой на Java, реализующей взаимодействие с MinIO. Контейнер работает на порту 8080, стандартном для приложений на Spring Boot. Для сборки контейнера используется DockerFile.

Для работы основного сервиса необходимы 3 контейнера:

* контейнер с образом PostgreSQL. Работает на порту 5438. Служит БД для хранения всех данных пользователей;
* контейнер с образом Redis. Использует порт 6379. Используется для хранения отозванных токенов;
* контейнер с основным сервисом, который реализует все возможности приложения (за исключением хранения изображений). Работает на 8000 порту. Для сборки контейнера используется DockerFile.

Теперь давайте подробнее рассмотрим каждый из DockerFile`ов и начнем с файла для фронтенда. Как указано в [приложении З.3](#_gb1kpncxyetw) в качестве образа используется alpine версия nodejs, для уменьшения веса итогового контейнера, после подгружаются аргументы сборки и выгружаются в переменные окружения. Далее происходит установка зависимостей, сборка проекта, открытие порта и, наконец, запуск фронтенд части.

Перейдем к DockerFile для сервиса хранения изображений. Здесь используется двухэтапная сборка контейнера, сначала мы используем образ, содержащий jdk для сборки, собираем проект, после используем образ с jre, который запускает результаты сборки с прошлого этапа.

И наконец, рассмотрим DockerFile основного сервиса. В качестве образа используется python версии 3.13. В файле прописаны: отключение буферизации для отображения логов в реальном времени, установка UV как пакетного менеджера, настройка окружения, включение компиляции Python файлов в байт код для улучшения запуска приложения, устанавливается режим копирования файлов для UV, что необходимо для корректного использования кэша Docker. После идет установка зависимостей, копирование остальных файлов проекта, установка проекта с использованием кэша, открытие порта и запуск приложения через uvicorn.

Заключение

В ходе выполнения лабораторных работ по курсу "Технологии сетевого программирования" был разработан полнофункциональный сервис "Трекер карьеры" с современной микросервисной архитектурой. Основной сервис построен на слоистой архитектуре (onion), что обеспечивает модульность и масштабируемость системы.

Проект реализован с использованием актуальных технологий: Python с FastAPI для backend, React для frontend, PostgreSQL в качестве базы данных, Redis для кэширования и управления сессиями, MinIO для хранения файлов. Реализованы ключевые функции: создание и управление роадмапами, система карточек с образовательными ресурсами, отслеживание прогресса, система прав доступа и персонализация интерфейса. Внедрена надежная система аутентификации на основе JWT-токенов с механизмом refresh-токенов.

Выполнение проекта позволило освоить принципы микросервисной архитектуры и получить практический опыт работы с современными технологиями. Изучены аспекты безопасности веб-приложений, включая JWT-аутентификацию и управление сессиями. Освоены современные практики разработки: использование линтеров (Ruff), миграции баз данных (Alembic) и контейнеризация приложений с Docker. Развиты навыки проектирования пользовательского интерфейса и интеграции frontend с backend через REST API.

Разработанный сервис является современным, масштабируемым решением, готовым к практическому использованию. Проект обеспечил получение ценных практических навыков и знаний в области веб-разработки, способствуя профессиональному росту.

# Приложение А Конфигурации

А.1 Пример .env файла

# Redis config

REDIS\_HOST=localhost

REDIS\_PORT=6379

REDIS\_DB=0

# PostgreSQL config

DB\_USER=postgres

DB\_PASSWORD=your\_password\_here

DB\_HOST=localhost

DB\_PORT=5432

DB\_NAME=your\_database\_name

# default images config

DEFAULT\_AVATAR=your\_default\_user\_avatar\_name\_from\_MinIo

DEFAULT\_BACKGROUND=your\_default\_roadmap\_background\_from\_MinIo

# jwt config

SECRET\_KEY=your\_secret\_key

ALGORITHM=HS256

ACCESS\_TOKEN\_EXPIRE\_MINUTES=15

REFRESH\_TOKEN\_EXPIRE\_DAYS=14

# MinIO config

MINIO\_ROOT\_USER=admin

MINIO\_ROOT\_PASSWORD=password

А.2 Фрагмент кода файла config.py

class Settings(BaseSettings):

REDIS\_HOST: str

REDIS\_PORT: int

REDIS\_DB: int

DB\_HOST: str

DB\_PORT: int

DB\_USER: str

DB\_PASSWORD: str

DB\_NAME: str

DEFAULT\_AVATAR: str

DEFAULT\_BACKGROUND: str

SECRET\_KEY: str

ALGORITHM: str

ACCESS\_TOKEN\_EXPIRE\_MINUTES: int

REFRESH\_TOKEN\_EXPIRE\_DAYS: int

@property

def DATABASE\_URL\_asyncpg(self):

return f"postgresql+asyncpg://{self.DB\_USER}:{self.DB\_PASSWORD}@{self.DB\_HOST}:{self.DB\_PORT}/{self.DB\_NAME}"

model\_config = SettingsConfigDict(env\_file=".env", extra="ignore")

А.3 Конфигурация Ruff

[tool.ruff]

line-length = 90

show-fixes = true

exclude = ["migrations"]

src = ["src"]

[tool.ruff.lint]

select = [

"E", # pycodestyle errors

"W", # pycodestyle warnings

"F", # pyflakes

"I", # isort

"B", # flake8-bugbear

"C4", # flake8-comprehensions

"UP", # pyupgrade

"ARG001", # unused arguments in functions

]

ignore = [

"E501", # line too long, handled by black

"B008", # do not perform function calls in argument defaults

"W191", # indentation contains tabs

"B904", # Allow raising exceptions without from e, for HTTPException

]

[tool.ruff.lint.pyupgrade]

# Preserve types, even if a file imports `from \_\_future\_\_ import annotations`.

keep-runtime-typing = true

# Приложение Б Слой репозитория

Б.1 Фрагмент кода определяющий структуру БД. Файл models.py

class Base(DeclarativeBase):

pass

class User(Base):

\_\_tablename\_\_ = "users"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

login: Mapped[str] = mapped\_column(String(256), unique=True, nullable=False)

password\_hash: Mapped[str] = mapped\_column(String, nullable=False)

name: Mapped[str] = mapped\_column(String(256), nullable=False)

created\_at: Mapped[DateTime] = mapped\_column(

DateTime(timezone=True), server\_default=func.now()

)

# Set your default UUID to avatar

avatar: Mapped[str] = mapped\_column(

String,

nullable=True,

server\_default=text(f"'{settings.DEFAULT\_AVATAR}'"),

)

owned\_roadmaps: Mapped[list["Roadmap"]] = relationship(

back\_populates="owner", passive\_deletes=True

)

roadmaps: Mapped[list["UserRoadmap"]] = relationship(

back\_populates="user", passive\_deletes=True

)

cards: Mapped[list["UserCard"]] = relationship(

back\_populates="user", passive\_deletes=True

)

def to\_read\_model(self) -> UserDTO:

return UserDTO.model\_validate(self, from\_attributes=True)

class Roadmap(Base):

\_\_tablename\_\_ = "roadmaps"

class DifficultyEnum(enum.Enum):

easy = "easy"

medium = "medium"

hard = "hard"

class VisibilityEnum(enum.Enum):

public = "public"

link\_only = "link only"

private = "private"

class EditPermissionEnum(enum.Enum):

view\_only = "view only"

can\_edit = "can edit"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

owner\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("users.id", ondelete="CASCADE"), nullable=False

)

title: Mapped[str] = mapped\_column(String(256), nullable=False)

description: Mapped[str | None] = mapped\_column(Text)

difficulty: Mapped[DifficultyEnum] = mapped\_column(

Enum(DifficultyEnum), nullable=False

)

edit\_permission: Mapped[EditPermissionEnum] = mapped\_column(

Enum(EditPermissionEnum), nullable=False

)

visibility: Mapped[VisibilityEnum] = mapped\_column(

Enum(VisibilityEnum), nullable=False

)

created\_at: Mapped[DateTime] = mapped\_column(

DateTime(timezone=True), server\_default=func.now()

)

updated\_at: Mapped[DateTime] = mapped\_column(

DateTime(timezone=True), server\_default=func.now(), onupdate=func.now()

)

owner: Mapped["User"] = relationship(

back\_populates="owned\_roadmaps", passive\_deletes=True

)

cards: Mapped[list["Card"]] = relationship(

back\_populates="roadmap", passive\_deletes=True

)

users: Mapped[list["UserRoadmap"]] = relationship(

back\_populates="roadmap", passive\_deletes=True

)

def to\_read\_model(self) -> RoadmapDTO:

return RoadmapDTO.model\_validate(self, from\_attributes=True)

class Card(Base):

\_\_tablename\_\_ = "cards"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

roadmap\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("roadmaps.id", ondelete="CASCADE"), nullable=False

)

order\_position: Mapped[int] = mapped\_column(Integer, nullable=False)

title: Mapped[str] = mapped\_column(String(256), nullable=False)

description: Mapped[str | None] = mapped\_column(Text)

created\_at: Mapped[DateTime] = mapped\_column(

DateTime(timezone=True), server\_default=func.now()

)

updated\_at: Mapped[DateTime] = mapped\_column(

DateTime(timezone=True), server\_default=func.now(), onupdate=func.now()

)

roadmap: Mapped["Roadmap"] = relationship(

back\_populates="cards", passive\_deletes=True

)

users: Mapped[list["UserCard"]] = relationship(

back\_populates="card", passive\_deletes=True

)

links: Mapped[list["CardLink"]] = relationship(

back\_populates="card", passive\_deletes=True

)

\_\_table\_args\_\_ = (

# order\_pos will be unique in the roadmap

UniqueConstraint("roadmap\_id", "order\_position", name="uq\_roadmap\_card\_position"),

)

def to\_read\_model(self) -> CardDTO:

return CardDTO.model\_validate(self, from\_attributes=True)

class UserRoadmap(Base):

\_\_tablename\_\_ = "user\_roadmaps"

user\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("users.id", ondelete="CASCADE"), primary\_key=True

)

roadmap\_id: Mapped[int] = mapped\_column(

Integer,

ForeignKey("roadmaps.id", ondelete="CASCADE"),

primary\_key=True,

)

background: Mapped[str] = mapped\_column(

String,

nullable=False,

server\_default=text(f"'{settings.DEFAULT\_BACKGROUND}'"),

)

user: Mapped["User"] = relationship(back\_populates="roadmaps", passive\_deletes=True)

roadmap: Mapped["Roadmap"] = relationship(

back\_populates="users", passive\_deletes=True

)

def to\_read\_model(self) -> UserRoadmapDTO:

return UserRoadmapDTO.model\_validate(self, from\_attributes=True)

class UserCard(Base):

\_\_tablename\_\_ = "user\_cards"

class StatusEnum(enum.Enum):

in\_progress = "in progress"

done = "done"

to\_do = "to do"

user\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("users.id", ondelete="CASCADE"), primary\_key=True

)

card\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("cards.id", ondelete="CASCADE"), primary\_key=True

)

status: Mapped[StatusEnum] = mapped\_column(

Enum(StatusEnum), nullable=False, server\_default=text("'to\_do'")

)

user: Mapped["User"] = relationship(back\_populates="cards", passive\_deletes=True)

card: Mapped["Card"] = relationship(back\_populates="users", passive\_deletes=True)

def to\_read\_model(self) -> UserCardDTO:

return UserCardDTO.model\_validate(self, from\_attributes=True)

class CardLink(Base):

\_\_tablename\_\_ = "card\_links"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

card\_id: Mapped[int] = mapped\_column(

Integer, ForeignKey("cards.id", ondelete="CASCADE"), nullable=False

)

link\_title: Mapped[str | None] = mapped\_column(String(256))

link\_content: Mapped[str | None] = mapped\_column(String)

card: Mapped["Card"] = relationship(back\_populates="links", passive\_deletes=True)

def to\_read\_model(self) -> CardLinkDTO:

return CardLinkDTO.model\_validate(self, from\_attributes=True)

Б.2 Фрагмент кода слоя репозитория, определяющего взаимодействие с БД

class SQLAlchemyRepository(AbstractRepository):

model = None

def \_\_init\_\_(self, session: AsyncSession):

self.session = session

def \_get\_primary\_key\_columns(self) -> list:

"""Get primary key columns for the model"""

if hasattr(self.model, "\_\_table\_\_") and hasattr(

self.model.\_\_table\_\_, "primary\_key"

):

return [

getattr(self.model, c.name)

for c in self.model.\_\_table\_\_.primary\_key.columns

]

return [self.model.id]

async def add\_one(self, data: dict[str, Any]) -> dict[str, Any] | int:

"""

Add a single record and return its primary key(s)

Args:

data: Dictionary with data to insert

Returns:

For single primary key: the scalar value

For composite keys: dictionary of primary key fields and values

"""

pk\_columns = self.\_get\_primary\_key\_columns()

stmt = insert(self.model).values(\*\*data).returning(\*pk\_columns)

res = await self.session.execute(stmt)

if len(pk\_columns) == 1:

return res.scalar\_one()

else:

return res.mappings().one()

async def edit\_one(

self, id\_or\_filter: int | dict[str, Any], data: dict[str, Any]

) -> dict[str, Any] | int | None:

"""

Update a record using either a single ID or composite key filters

Args:

id\_or\_filter: Either an integer ID or a dictionary with filter conditions

data: Dictionary with data to update

Returns:

For single primary key: the scalar value

For composite keys: dictionary of primary key fields and values

None: if no record found

"""

pk\_columns = self.\_get\_primary\_key\_columns()

if isinstance(id\_or\_filter, dict):

stmt = (

update(self.model)

.values(\*\*data)

.filter\_by(\*\*id\_or\_filter)

.returning(\*pk\_columns)

)

else:

stmt = (

update(self.model)

.values(\*\*data)

.filter\_by(id=id\_or\_filter)

.returning(\*pk\_columns)

)

res = await self.session.execute(stmt)

if len(pk\_columns) == 1:

return res.scalar\_one\_or\_none()

else:

result = res.mappings().one\_or\_none()

return result

async def find\_all(self, filter\_by: dict[str, Any] | None = None) -> list[Any]:

"""

Find all records with optional filtering

Args:

filter\_by: Optional dictionary with filter conditions

Returns:

List of model instances

"""

stmt = select(self.model)

if filter\_by:

stmt = stmt.filter\_by(\*\*filter\_by)

res = await self.session.execute(stmt)

results = res.scalars().all()

if results and hasattr(results[0], "to\_read\_model"):

return [item.to\_read\_model() for item in results]

return results

async def find\_one(self, \*\*filter\_by) -> Any | None:

"""

Find a single record by filter criteria

Args:

\*\*filter\_by: Filter conditions as keyword arguments

Returns:

Single model instance or None if not found

"""

stmt = select(self.model).filter\_by(\*\*filter\_by)

res = await self.session.execute(stmt)

result = res.scalar\_one\_or\_none()

if result and hasattr(result, "to\_read\_model"):

return result.to\_read\_model()

return result

async def delete\_one(

self, id\_or\_filter: int | dict[str, Any]

) -> dict[str, Any] | int | None:

"""

Delete a record using either a single ID or composite key filters

Args:

id\_or\_filter: Either an integer ID or a dictionary with filter conditions

Returns:

For single primary key: the scalar value

For composite keys: dictionary of primary key fields and values

None: if no record found

"""

pk\_columns = self.\_get\_primary\_key\_columns()

if isinstance(id\_or\_filter, dict):

stmt = delete(self.model).filter\_by(\*\*id\_or\_filter).returning(\*pk\_columns)

else:

stmt = delete(self.model).filter\_by(id=id\_or\_filter).returning(\*pk\_columns)

res = await self.session.execute(stmt)

if len(pk\_columns) == 1:

return res.scalar\_one\_or\_none()

else:

result = res.mappings().one\_or\_none()

return result

Б.3 Фрагмент кода паттерна UnitOfWork

class UnitOfWork:

def \_\_init\_\_(self):

self.session\_factory = async\_session\_factory

async def \_\_aenter\_\_(self):

self.session = self.session\_factory()

self.users = UserRepository(self.session)

self.roadmaps = RoadmapRepository(self.session)

self.cards = CardRepository(self.session)

self.user\_cards = UserCardRepository(self.session)

self.user\_roadmaps = UserRoadmapRepository(self.session)

self.card\_links = CardLinkRepository(self.session)

async def \_\_aexit\_\_(self, \*args):

await self.rollback()

await self.session.close()

async def commit(self):

await self.session.commit()

async def rollback(self):

await self.session.rollback()

# Приложение В Сервисный слой

В.1 Фрагмент кода файла users.py

class UsersService:

@staticmethod

async def authenticate\_user(

uow: IUnitOfWork, login: str, password: str

) -> UserAuthDTO | None:

async with uow:

user: UserAuthDTO = await uow.users.find\_auth\_info(login)

if not user:

return None

if not verify\_password(password, user.password\_hash):

return None

return user

@staticmethod

async def add\_user(uow: IUnitOfWork, user: UserAddDTO):

user\_dict = user.model\_dump()

user\_dict["password\_hash"] = hash\_password(user\_dict["password\_hash"])

async with uow:

user\_id = await uow.users.add\_one(user\_dict)

await uow.commit()

return user\_id

@staticmethod

async def edit\_user(uow: IUnitOfWork, user\_id: int, user: UserEditDTO):

user\_dict = user.model\_dump(exclude\_unset=True)

async with uow:

await uow.users.edit\_one(user\_id, user\_dict)

await uow.commit()

@staticmethod

async def get\_user(uow: IUnitOfWork, \*\*filter\_by) -> UserDTO:

async with uow:

users = await uow.users.find\_one(\*\*filter\_by)

return users

async def get\_user\_by\_login(uow: IUnitOfWork, login: str) -> UserDTO:

async with uow:

user = await uow.users.find\_one(login=login)

return user

@staticmethod

async def get\_users(

uow: IUnitOfWork, filter\_by: dict[str, Any] | None = None

) -> list[UserDTO]:

async with uow:

users = await uow.users.find\_all({"id": filter\_by})

return users

@staticmethod

async def delete\_user(uow: IUnitOfWork, user\_id: int):

async with uow:

user = await uow.users.delete\_one(user\_id)

await uow.commit()

return user

@staticmethod

async def delete\_avatar(uow: IUnitOfWork, user\_id: int):

async with uow:

user: UserDTO = await uow.users.find\_one(id=user\_id)

if user.avatar == settings.DEFAULT\_AVATAR:

return False

await uow.users.edit\_one(user\_id, {"avatar": settings.DEFAULT\_AVATAR})

await uow.commit()

return True

@staticmethod

async def edit\_avatar(uow: IUnitOfWork, avatar: AvatarDTO, user\_id: int):

async with uow:

user = await uow.users.edit\_one(user\_id, avatar.model\_dump())

await uow.commit()

return user

@staticmethod

async def get\_avatar(uow: IUnitOfWork, user\_id: int):

async with uow:

user: UserDTO = await uow.users.find\_one(id=user\_id)

return AvatarDTO.model\_validate(user, from\_attributes=True)

В.2 Фрагмент кода файла roadmaps.py

class RoadmapsService:

@staticmethod

async def add\_roadmap(uow: IUnitOfWork, roadmap: RoadmapAddDTO):

roadmap\_dict = roadmap.model\_dump()

async with uow:

roadmap\_id = await uow.roadmaps.add\_one(roadmap\_dict)

await uow.commit()

owner\_id = (await RoadmapsService.get\_roadmap(uow, roadmap\_id)).owner\_id

return roadmap\_id, owner\_id

@staticmethod

async def edit\_roadmap(uow: IUnitOfWork, roadmap\_id: int, roadmap: RoadmapEditDTO):

roadmap\_dict = roadmap.model\_dump(exclude\_unset=True)

async with uow:

await uow.roadmaps.edit\_one(roadmap\_id, roadmap\_dict)

await uow.commit()

@staticmethod

async def get\_roadmap(uow: IUnitOfWork, roadmap\_id: int) -> RoadmapDTO | None:

async with uow:

roadmaps = await uow.roadmaps.find\_one(id=roadmap\_id)

return roadmaps

@staticmethod

async def get\_roadmaps(uow: IUnitOfWork, roadmap\_ids: list[int]):

async with uow:

roadmaps = await uow.roadmaps.find\_all({"id": roadmap\_ids})

return roadmaps

@staticmethod

async def get\_roadmap\_extended(uow: IUnitOfWork, roadmap\_id: int, user\_id: int):

async with uow:

roadmap = await uow.roadmaps.find\_one(id=roadmap\_id)

if roadmap is None:

return None

roadmap\_dict = roadmap.model\_dump()

cards = await uow.cards.find\_all({"roadmap\_id": roadmap\_id})

roadmap\_dict["cards"] = []

for card in cards:

roadmap\_dict["cards"].append(

await CardsService.get\_card\_extended(uow, card.id, user\_id)

)

roadmap\_dict["cards"].sort(key=lambda x: x.order\_position)

logger.debug("\n\n\n", roadmap\_dict, "\n\n\n")

extended\_roadmap = RoadmapExtendedDTO.model\_validate(

roadmap\_dict, from\_attributes=True

)

return extended\_roadmap

@staticmethod

async def get\_public\_roadmaps(

uow: IUnitOfWork,

pagination: PaginationParams,

search: str | None = None,

difficulty: str | None = None,

) -> PaginatedRoadmapsDTO:

"""

Gets public roadmaps with search and filtering capabilities

Args:

uow: Unit of Work instance

search: Optional str for searching by title

difficulty: Optional filter by roadmap difficulty can be 'easy', 'medium' or 'hard'

pagination: Pagination parameters

Returns:

PaginatedRoadmapsDTO object

"""

async with uow:

roadmaps, total\_count = await uow.roadmaps.find\_public\_roadmaps(

search=search, difficulty=difficulty, pagination=pagination

)

simplified\_roadmaps = [

SimplifiedRoadmapDTO.model\_validate(roadmap, from\_attributes=True)

for roadmap in roadmaps

]

total\_pages = (

math.ceil(total\_count / pagination.limit) if pagination.limit > 0 else 0

)

return PaginatedRoadmapsDTO(

roadmaps=simplified\_roadmaps, total\_pages=total\_pages

)

@staticmethod

async def delete\_roadmap(uow: IUnitOfWork, roadmap\_id: int):

async with uow:

roadmap = await uow.roadmaps.delete\_one(roadmap\_id)

await uow.commit()

return roadmap

В.3 Фрагмент кода файла user\_roadmaps.py

class UserRoadmapsService:

@staticmethod

async def link\_user\_to\_roadmap(

uow: IUnitOfWork, roadmap\_id: int, user\_id: int

) -> UserRoadmapAddDTO:

async with uow:

user\_roadmap\_id = await uow.user\_roadmaps.add\_one(

{"roadmap\_id": roadmap\_id, "user\_id": user\_id}

)

await uow.commit()

return UserRoadmapAddDTO.model\_validate(user\_roadmap\_id, from\_attributes=True)

@staticmethod

async def change\_background(

uow: IUnitOfWork, roadmap\_id: int, user\_id: int, background: UserRoadmapEditDTO

):

background\_dict = background.model\_dump(exclude\_unset=True)

async with uow:

await uow.user\_roadmaps.edit\_one(

id\_or\_filter={"roadmap\_id": roadmap\_id, "user\_id": user\_id},

data=background\_dict,

)

await uow.commit()

@staticmethod

async def get\_user\_roadmaps(uow: IUnitOfWork, user\_id: int):

async with uow:

user\_roadmaps: list[UserRoadmapDTO] = await uow.user\_roadmaps.find\_all(

{"user\_id": user\_id}

)

roadmaps = [roadmap.roadmap\_id for roadmap in user\_roadmaps]

return roadmaps

@staticmethod

async def delete\_user\_roadmap(uow: IUnitOfWork, user\_roadmap\_id: dict[str, int]):

async with uow:

user\_roadmap = await uow.user\_roadmaps.delete\_one(user\_roadmap\_id)

await uow.commit()

return user\_roadmap

@staticmethod

async def get\_background(

uow: IUnitOfWork, user\_roadmap\_id: dict[str, int]

) -> BackgroundDTO:

async with uow:

user\_roadmap = await uow.user\_roadmaps.find\_one(

user\_id=user\_roadmap\_id["user\_id"],

roadmap\_id=user\_roadmap\_id["roadmap\_id"],

)

if user\_roadmap is None:

return None

return BackgroundDTO.model\_validate(user\_roadmap, from\_attributes=True)

@staticmethod

async def get\_linked\_roadmaps(

uow: IUnitOfWork,

pagination: PaginationParams,

user\_id: int,

search: str | None = None,

difficulty: str | None = None,

) -> PaginatedRoadmapsDTO:

"""

Gets roadmaps linked to user\_id with search and filtering capabilities

Args:

uow: Unit of Work instance

user\_id: id of the user

search: Optional str for searching by title

difficulty: Optional filter by roadmap difficulty can be 'easy', 'medium' or 'hard'

pagination: Pagination parameters

Returns:

PaginatedRoadmapsDTO object

"""

async with uow:

user\_roadmaps\_list = await uow.user\_roadmaps.find\_all({"user\_id": user\_id})

roadmap\_ids = [ur.roadmap\_id for ur in user\_roadmaps\_list]

roadmaps, total\_count = await uow.user\_roadmaps.find\_user\_roadmaps(

roadmap\_ids=roadmap\_ids,

search=search,

difficulty=difficulty,

pagination=pagination,

)

simplified\_roadmaps = [

SimplifiedRoadmapDTO.model\_validate(roadmap, from\_attributes=True)

for roadmap in roadmaps

]

total\_pages = (

math.ceil(total\_count / pagination.limit) if pagination.limit > 0 else 0

)

return PaginatedRoadmapsDTO(

roadmaps=simplified\_roadmaps,

total\_pages=total\_pages,

)

@staticmethod

async def get\_roadmaps\_with\_in\_progress\_cards(

uow: IUnitOfWork,

pagination: PaginationParams,

user\_id: int,

search: str | None = None,

difficulty: str | None = None,

) -> PaginatedRoadmapsDTO:

"""

Gets roadmaps linked to user that have at least one card with 'in\_progress' status

Args:

uow: Unit of Work instance

user\_id: id of the user

search: Optional str for searching by title

difficulty: Optional filter by roadmap difficulty can be 'easy', 'medium' or 'hard'

pagination: Pagination parameters

Returns:

PaginatedRoadmapsDTO object

"""

async with uow:

user\_roadmaps\_list = await uow.user\_roadmaps.find\_all({"user\_id": user\_id})

roadmap\_ids = [ur.roadmap\_id for ur in user\_roadmaps\_list]

(

roadmaps,

total\_count,

) = await uow.user\_roadmaps.find\_roadmaps\_with\_in\_progress\_cards(

roadmap\_ids=roadmap\_ids,

user\_id=user\_id,

search=search,

difficulty=difficulty,

pagination=pagination,

)

simplified\_roadmaps = [

SimplifiedRoadmapDTO.model\_validate(roadmap, from\_attributes=True)

for roadmap in roadmaps

]

total\_pages = (

math.ceil(total\_count / pagination.limit) if pagination.limit > 0 else 0

)

return PaginatedRoadmapsDTO(

roadmaps=simplified\_roadmaps,

total\_pages=total\_pages,

)

В.4 Фрагмент кода файла tokens.py

class TokensService:

@staticmethod

async def is\_token\_revoked(jti: str, redis\_client: Redis):

key = f"token:{jti}"

token = await redis\_client.get(key)

logger.debug(f"redis: is\_token\_revoked: {token}")

return token is not None

@staticmethod

async def revoke\_token(jti: str, expires\_at: datetime, redis\_client: Redis):

key = f"token:{jti}"

remaining\_time = expires\_at - datetime.now(UTC)

if remaining\_time.total\_seconds() > 0:

await redis\_client.setex(key, int(remaining\_time.total\_seconds()), 1)

logger.debug(f"redis: revoke\_token {await redis\_client.get(key)}")

else:

logger.debug(f"token not revoked {remaining\_time.total\_seconds()}")

@staticmethod

def prepare\_token\_for\_revocation(token: str):

payload = jwt.decode(token, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM])

jti = payload.get("jti")

exp = payload.get("exp")

if jti and exp:

expires\_at = datetime.fromtimestamp(exp, tz=UTC)

return jti, expires\_at

return jti, exp

# Приложение Г Слой API

Г.1 Фрагмент кода файла auth.py

@router.post("", status\_code=status.HTTP\_201\_CREATED)

async def create\_user(user: UserAddDTO, uow: UOWDep):

already\_exist = await UsersService.get\_user(uow, login=user.login)

if already\_exist is not None:

raise HTTPException(status\_code=409, detail="User with this email already exist")

user\_id = await UsersService.add\_user(uow, user)

return {"user\_id": user\_id}

@router.post("/token", response\_model=LoginDTO)

async def login\_for\_access\_token(

response: Response,

form\_data: Annotated[OAuth2PasswordRequestForm, Depends()],

uow: UOWDep,

):

user = await UsersService.authenticate\_user(

uow, form\_data.username, form\_data.password

)

if user is None:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate user.",

)

access\_token = create\_token(

user.id,

user.login,

user.name,

timedelta(minutes=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES),

)

refresh\_token = create\_token(user.id, user.login, user.name, timedelta(days=7))

response.set\_cookie(

key="access\_token",

value=access\_token,

httponly=True,

secure=True,

samesite="lax",

max\_age=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

expires=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

)

response.set\_cookie(

key="refresh\_token",

value=refresh\_token,

httponly=True,

secure=True,

samesite="lax",

max\_age=settings.REFRESH\_TOKEN\_EXPIRE\_DAYS \* 24 \* 60 \* 60,

expires=settings.REFRESH\_TOKEN\_EXPIRE\_DAYS \* 24 \* 60 \* 60,

)

return {

"access\_token": access\_token,

"token\_type": "bearer",

"id": user.id,

"username": user.name,

"login": user.login,

"avatar": user.avatar,

}

@router.post("/refresh", response\_model=TokenDTO)

async def refresh\_token(

redis\_dep: RedisDep,

response: Response,

access\_token: str | None = Cookie(None),

refresh\_token: str | None = Cookie(None),

):

if not refresh\_token:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Refresh token missing",

)

payload = jwt.decode(

refresh\_token, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM]

)

user\_id = payload.get("id")

user\_name = payload.get("username")

user\_login = payload.get("sub")

expires\_at = datetime.fromtimestamp(payload.get("exp"), tz=UTC)

jti = payload.get("jti")

if access\_token:

access\_token\_payload = jwt.decode(

access\_token, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM]

)

access\_token\_expires\_at = datetime.fromtimestamp(

access\_token\_payload.get("exp"), tz=UTC

)

access\_token\_jti = access\_token\_payload.get("jti")

if user\_id is None or jti is None:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Invalid refresh token",

)

if await TokensService.is\_token\_revoked(jti, redis\_dep):

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Refresh token revoked",

# headers={"WWW-Authenticate": "Bearer"},

)

new\_access\_token = create\_token(

user\_id,

user\_login,

user\_name,

timedelta(minutes=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES),

)

new\_refresh\_token = create\_token(

user\_id,

user\_login,

user\_name,

timedelta(days=settings.REFRESH\_TOKEN\_EXPIRE\_DAYS),

)

await TokensService.revoke\_token(jti, expires\_at, redis\_dep)

if access\_token:

await TokensService.revoke\_token(

access\_token\_jti, access\_token\_expires\_at, redis\_dep

)

response.set\_cookie(

key="access\_token",

value=new\_access\_token,

httponly=True,

secure=True,

samesite="lax",

max\_age=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

expires=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

)

response.set\_cookie(

key="refresh\_token",

value=new\_refresh\_token,

httponly=True,

secure=True,

samesite="lax",

max\_age=settings.REFRESH\_TOKEN\_EXPIRE\_DAYS \* 24 \* 60 \* 60,

expires=settings.REFRESH\_TOKEN\_EXPIRE\_DAYS \* 24 \* 60 \* 60,

)

return {"access\_token": new\_access\_token, "token\_type": "bearer"}

@router.post("/logout")

async def logout(

user\_dep: UserDep,

redis\_dep: RedisDep,

response: Response,

access\_token: str | None = Cookie(None),

refresh\_token: str | None = Cookie(None),

):

if access\_token:

jti, expires\_at = TokensService.prepare\_token\_for\_revocation(access\_token)

if jti and expires\_at:

await TokensService.revoke\_token(jti, expires\_at, redis\_dep)

if refresh\_token:

jti, expires\_at = TokensService.prepare\_token\_for\_revocation(refresh\_token)

if jti and expires\_at:

await TokensService.revoke\_token(jti, expires\_at, redis\_dep)

response.delete\_cookie(key="access\_token")

response.delete\_cookie(key="refresh\_token")

return {"detail": "Successfully logged out"}

@router.get("/me")

async def get\_curr\_user(user\_dep: UserDep):

return user\_dep

Г.2 Фрагмент кода файла users.py

@router.get("", response\_model=UserDTO)

async def get\_user\_info(user\_dep: UserDep, uow: UOWDep):

user = await UsersService.get\_user(uow, id=user\_dep["id"])

return user

@router.delete("", status\_code=204)

async def delete\_user(

user\_dep: UserDep,

uow: UOWDep,

redis\_dep: RedisDep,

response: Response,

access\_token: str | None = Cookie(None),

refresh\_token: str | None = Cookie(None),

):

await UsersService.delete\_user(uow, user\_dep["id"])

if access\_token:

jti, expires\_at = TokensService.prepare\_token\_for\_revocation(access\_token)

if jti and expires\_at:

await TokensService.revoke\_token(jti, expires\_at, redis\_dep)

if refresh\_token:

jti, expires\_at = TokensService.prepare\_token\_for\_revocation(refresh\_token)

if jti and expires\_at:

await TokensService.revoke\_token(jti, expires\_at, redis\_dep)

response.delete\_cookie(key="access\_token")

response.delete\_cookie(key="refresh\_token")

return {"detail": "Successfully deleted"}

@router.patch("")

async def edit\_user(user\_dep: UserDep, user: UserEditDTO, uow: UOWDep):

if user.password\_hash is not None:

user.password\_hash = hash\_password(user.password\_hash)

await UsersService.edit\_user(uow, user\_dep["id"], user)

user\_data = await UsersService.get\_user(uow, id=user\_dep["id"])

return {

"user\_id": user\_dep["id"],

"login": user\_data.login,

"username": user\_data.name,

}

@router.delete("/avatar")

async def delete\_user\_avatar(user\_dep: UserDep, uow: UOWDep):

await UsersService.delete\_avatar(uow, user\_dep["id"])

@router.put("/avatar")

async def change\_user\_avatar(user\_dep: UserDep, avatar: AvatarDTO, uow: UOWDep):

await UsersService.edit\_avatar(uow, avatar, user\_dep["id"])

return avatar

@router.get("/avatar")

async def get\_user\_avatar(user\_dep: UserDep, uow: UOWDep):

return await UsersService.get\_avatar(uow, user\_dep["id"])

Г.3 Фрагмент кода файла roadmaps.py

@router.get("/public", response\_model=PaginatedRoadmapsDTO)

async def get\_public\_roadmaps(

# user\_dep: UserDep,

uow: UOWDep,

pagination: PaginationDep,

search: str | None = None,

difficulty: str | None = None,

):

roadmaps = await RoadmapsService.get\_public\_roadmaps(

uow, pagination, search, difficulty

)

return roadmaps

@router.post("")

async def add\_roadmap(user\_dep: UserDep, roadmap: RoadmapAddDTO, uow: UOWDep):

roadmap.owner\_id = user\_dep["id"]

roadmap\_id, user\_id = await RoadmapsService.add\_roadmap(uow, roadmap)

await UserRoadmapsService.link\_user\_to\_roadmap(uow, roadmap\_id, user\_id)

return {"roadmap\_id": roadmap\_id, "user\_id": user\_id}

@router.get("/{roadmap\_id}", response\_model=RoadmapExtendedDTO)

async def get\_roadmap\_info(

user\_dep: UserDep, roadmap\_id: Annotated[int, Path(title="Roadmap id")], uow: UOWDep

):

extended\_roadmap = await RoadmapsService.get\_roadmap\_extended(

uow, roadmap\_id, user\_dep["id"]

)

if extended\_roadmap is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

if (

extended\_roadmap.owner\_id != user\_dep["id"]

and extended\_roadmap.visibility == "private"

):

raise HTTPException(status\_code=403, detail="Roadmap is private")

return extended\_roadmap

# TODO if owner of roadmap change roadmap visibilyty to private raise exeption

# TODO if owner of roadmap change roadmap edit\_permission to on

# TODO если владелец роадмапа меняет параметры доступа на недоступные для остальных пользователей, то нужно удалить привязки пользователей к роадмапу

@router.patch("/{roadmap\_id}")

async def edit\_roadmap(

user\_dep: UserDep,

roadmap\_id: Annotated[int, Path(title="Roadmap id")],

roadmap: RoadmapEditDTO,

uow: UOWDep,

):

roadmap\_info: RoadmapDTO | None = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap\_info is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

if (

roadmap\_info.owner\_id != user\_dep["id"]

and roadmap\_info.edit\_permission != "can edit"

):

raise HTTPException(

status\_code=403,

detail="The user does not have permission to edit this roadmap",

)

await RoadmapsService.edit\_roadmap(uow, roadmap\_id, roadmap)

return {"roadmap\_id": roadmap\_id}

@router.delete("/{roadmap\_id}", status\_code=204)

async def delete\_roadmap(

user\_dep: UserDep, roadmap\_id: Annotated[int, Path(title="Roadmap id")], uow: UOWDep

):

roadmap = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

if roadmap.owner\_id != user\_dep["id"]:

raise HTTPException(

status\_code=403,

detail="The user does not have permission to delete this roadmap",

)

await RoadmapsService.delete\_roadmap(uow, roadmap\_id)

Г.4 Фрагмент кода файла user\_roadmaps.py

@router.get("", response\_model=PaginatedRoadmapsDTO)

async def get\_linked\_roadmaps(

user\_dep: UserDep,

pagination: PaginationDep,

uow: UOWDep,

search: str | None = None,

difficulty: str | None = None,

):

roadmaps = await UserRoadmapsService.get\_linked\_roadmaps(

uow=uow,

user\_id=user\_dep["id"],

search=search,

difficulty=difficulty,

pagination=pagination,

)

return roadmaps

@router.post("/{roadmap\_id}", response\_model=UserRoadmapAddExtendedDTO)

async def link\_user\_to\_roadmap(

user\_dep: UserDep, roadmap\_id: Annotated[int, Path(title="Roadmap id")], uow: UOWDep

):

roadmap = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

user\_roadmap\_info = await UserRoadmapsService.get\_background(

uow, {"user\_id": user\_dep["id"], "roadmap\_id": roadmap\_id}

)

if user\_roadmap\_info is not None:

raise HTTPException(status\_code=409, detail="User roadmap relation already exist")

user\_roadmap\_id = await UserRoadmapsService.link\_user\_to\_roadmap(

uow, roadmap\_id, user\_dep["id"]

)

card\_ids = await UserCardService.link\_user\_to\_cards\_in\_roadmap(

uow, user\_dep["id"], roadmap\_id

)

user\_roadmap\_id\_with\_card\_ids = {

"user\_id": user\_roadmap\_id.user\_id,

"roadmap\_id": user\_roadmap\_id.roadmap\_id,

"card\_ids": card\_ids,

}

return UserRoadmapAddExtendedDTO.model\_validate(

user\_roadmap\_id\_with\_card\_ids, from\_attributes=True

)

@router.get("/in\_progress", response\_model=PaginatedRoadmapsDTO)

async def get\_roadmaps\_with\_status\_in\_progress(

user\_dep: UserDep,

pagination: PaginationDep,

uow: UOWDep,

search: str | None = None,

difficulty: str | None = None,

):

return await UserRoadmapsService.get\_roadmaps\_with\_in\_progress\_cards(

uow=uow,

pagination=pagination,

user\_id=user\_dep["id"],

search=search,

difficulty=difficulty,

)

@router.delete("/{roadmap\_id}", status\_code=204)

async def delete\_user\_roadmap\_link(

user\_dep: UserDep, roadmap\_id: Annotated[int, Path(title="Roadmap id")], uow: UOWDep

):

roadmap\_info = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap\_info is None:

raise HTTPException(

status\_code=404, detail=f"Roadmap with id: {roadmap\_id} not found"

)

if roadmap\_info.owner\_id == user\_dep["id"]:

await RoadmapsService.delete\_roadmap(uow, roadmap\_id)

else:

await UserRoadmapsService.delete\_user\_roadmap(

uow, {"roadmap\_id": roadmap\_id, "user\_id": user\_dep["id"]}

)

await UserCardService.delete\_user\_cards(uow, user\_dep["id"], roadmap\_id)

@router.get("/{roadmap\_id}/background", response\_model=BackgroundDTO)

async def get\_roadmap\_background(

user\_dep: UserDep, roadmap\_id: Annotated[int, Path(title="Roadmap id")], uow: UOWDep

):

roadmap\_info = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap\_info is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

background = await UserRoadmapsService.get\_background(

uow, {"user\_id": user\_dep["id"], "roadmap\_id": roadmap\_id}

)

if background is None:

return BackgroundDTO(background=settings.DEFAULT\_BACKGROUND)

return background

@router.put("/{roadmap\_id}/background")

async def edit\_roadmap\_background(

user\_dep: UserDep,

roadmap\_id: Annotated[int, Path(title="Roadmap id")],

background: BackgroundDTO,

uow: UOWDep,

):

roadmap = await RoadmapsService.get\_roadmap(uow, roadmap\_id)

if roadmap is None:

raise HTTPException(status\_code=404, detail="Roadmap not found")

user\_roadmap = await UserRoadmapsService.get\_background(

uow, {"user\_id": user\_dep["id"], "roadmap\_id": roadmap\_id}

)

if user\_roadmap is None:

raise HTTPException(status\_code=404, detail="User roadmap relation not found")

await UserRoadmapsService.change\_background(

uow,

roadmap\_id,

user\_dep["id"],

UserRoadmapEditDTO.model\_validate(background, from\_attributes=True),

)

# Приложение Д Зависимости

Д.1 Фрагмент кода, определяющий UserDep

async def get\_current\_user(token: Annotated[str, Depends(oauth2\_bearer)]):

try:

payload = jwt.decode(token, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM])

login: str | None = payload.get("sub")

user\_id: int | None = payload.get("id")

user\_name: str | None = payload.get("username")

if login is None or user\_id is None or user\_name is None:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate user.",

)

return {"login": login, "id": user\_id, "username": user\_name}

except JWTError:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Could not validate user."

)

UserDep = Annotated[dict, Depends(get\_current\_user)]

Д.2 Фрагмент кода, определяющий RedisDep

async def get\_redis\_client(request: Request):

redis\_client = request.app.state.redis\_client

if not redis\_client:

raise HTTPException(

status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,

detail="Redis client not avialible.",

)

return redis\_client

RedisDep = Annotated[Redis, Depends(get\_redis\_client)]

Д.3 Фрагмент кода, определяющий PaginationDep

class PaginationParams(BaseModel):

limit: int = Field(

default=6, ge=0, le=100, description="Number of elements on a page"

)

page: int = Field(default=1, ge=1, description="Page for pagination")

PaginationDep = Annotated[PaginationParams, Depends(PaginationParams)]

# Приложение Е Frontend

Е.1 Фрагмент кода из api.jsx

import axios from "axios";

const apiClient = axios.create({

withCredentials: true,

headers: {

"Content-Type": "application/json",

},

});

// перехватчики для токенов авторизации

apiClient.interceptors.request.use(

(config) => {

const token = localStorage.getItem("accessToken");

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

},

(error) => Promise.reject(error)

);

apiClient.interceptors.response.use(

(response) => response,

async (error) => {

if (error.response?.status === 401) {

try {

const response = await axios.post("/api/core/auth/refresh");

localStorage.setItem("accessToken", response.data.access\_token);

// Повторяем исходный запрос с новым токеном

return apiClient(error.config);

} catch (refreshError) {

// Если не удалось обновить токен, перенаправляем на логин

localStorage.removeItem("accessToken");

localStorage.removeItem("userData");

window.location = "/";

}

}

return Promise.reject(error);

}

);

const handleError = (error, context) => {

console.error(`Error ${context}:`, error.response?.data || error.message);

throw error;

};

// --- Core Service (Python Backend - /api/core) ---

// --- Auth ---

export const authAPI = {

create: async (name, login, password) => {

try {

const userData = { name: name, login: login, password\_hash: password };

const response = await apiClient.post("/api/core/auth", userData);

return response.data;

} catch (error) {

handleError(error, "registering");

}

},

login: async (credentials) => {

try {

const response = await apiClient.post(

"/api/core/auth/token",

new URLSearchParams(credentials),

{ headers: { "Content-Type": "application/x-www-form-urlencoded" } }

);

localStorage.setItem("accessToken", response.access\_token);

const userData = {

id: response.data.id,

login: response.data.login,

username: response.data.username,

};

localStorage.setItem("userData", JSON.stringify(userData));

localStorage.setItem("avatar", response.data.avatar);

return response.data;

} catch (error) {

handleError(error, "logging in");

}

},

Е.2 Код компонента карточки

import React, { useState, useEffect} from "react";

import { Accordion, Button, ListGroup, Card, ButtonGroup, Form } from "react-bootstrap";

import { PiTrashBold } from "react-icons/pi";

import { PiNotePencilBold } from "react-icons/pi";

import "./CardView.css"

import { useLocation } from "react-router-dom";

import { EditCard } from "./CardRedact";

import { cardAPI, userCardAPI } from "../api/api";

function CardView({cardInfo, mayRedact, onDelete}) {

const location = useLocation();

const [modalShow, setModalShow] = React.useState(false);

const [selectedStatus, setSelectedStatus] = useState('');

useEffect(() => {

setSelectedStatus(cardInfo.status);

}, [cardInfo.id, cardInfo.status]);

useEffect(()=>{

const getCard = async()=>{

cardInfo = await cardAPI.getCard(cardInfo.id)

}

getCard()

},[modalShow, cardInfo.id])

const handleStatusChange = async (event) => {

const newStatus = event.target.value;

setSelectedStatus(newStatus);

await userCardAPI.editCardStatus(cardInfo.id, newStatus);

}

const handleDeleteCard = async() => {

await cardAPI.deleteCard(cardInfo.id);

onDelete(cardInfo.id);

};

return(

<Card className="cardcard">

<Card.Header className="cardhead">

<span>{cardInfo.title}</span>

{location.pathname === `/myroadmaps/${cardInfo.roadmap\_id}` && mayRedact &&

<div className="settingsgroup">

<Form.Select size="sm"

onChange={handleStatusChange}

value={selectedStatus}

>

<option value="to\_do">К выполнению</option>

<option value="in\_progress">В процессе</option>

<option value="done">Выполнено</option>

</Form.Select>

<ButtonGroup>

<Button size="sm" className="cardbutton" onClick={() => setModalShow(true)}><PiNotePencilBold/></Button>

<Button size="sm" className="cardbutton" onClick={handleDeleteCard}><PiTrashBold/></Button>

</ButtonGroup>

</div>}

</Card.Header>

<ListGroup variant="flush">

{/\* работает только если есть описание \*/}

{cardInfo.description &&

<Accordion className="cardaccordion">

<Accordion.Item eventKey="0" >

<Accordion.Header>{"Описание"}</Accordion.Header>

<Accordion.Body>

{cardInfo.description}

</Accordion.Body>

</Accordion.Item>

</Accordion>}

{cardInfo.links.map((link, index) => (

<ListGroup.Item key={index}><Card.Link href={link.link\_content}>{link.link\_title}</Card.Link></ListGroup.Item>

))}

</ListGroup>

<EditCard

show={modalShow}

onHide={() => setModalShow(false)}

initialData={cardInfo}

/>

</Card>

)

}

export default CardView

# Приложение З Контейнеризация

З.1 Код файла DockerFile для сервиса хранения изображений

# Этап сборки

FROM amazoncorretto:17 AS builder

WORKDIR /app

COPY . .

RUN chmod +x gradlew

RUN ./gradlew clean build -x test

# Финальный образ

FROM amazoncorretto:17

WORKDIR /app

COPY --from=builder /app/build/libs/\*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

З.2 Код файла DockerFile для основного сервиса

FROM python:3.13

ENV PYTHONUNBUFFERED=1

WORKDIR /app/src

# Install uv

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#installing-uv

COPY --from=ghcr.io/astral-sh/uv:0.5.11 /uv /uvx /bin/

# Place executables in the environment at the front of the path

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#using-the-environment

ENV PATH="/app/.venv/bin:$PATH"

# Compile bytecode

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#compiling-bytecode

ENV UV\_COMPILE\_BYTECODE=1

# uv Cache

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#caching

ENV UV\_LINK\_MODE=copy

# Install dependencies

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#intermediate-layers

RUN --mount=type=cache,target=/root/.cache/uv \

--mount=type=bind,source=uv.lock,target=uv.lock \

--mount=type=bind,source=pyproject.toml,target=pyproject.toml \

uv sync --frozen --no-install-project

COPY ./alembic.ini /app/alembic.ini

COPY ./pyproject.toml ./uv.lock /app/

COPY ./src /app/src

# Sync the project

# Ref: https://docs.astral.sh/uv/guides/integration/docker/#intermediate-layers

RUN --mount=type=cache,target=/root/.cache/uv \

/bin/sh -c "cd /app && uv sync"

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

З.3 Код файла DockerFile для фронтенда

FROM node:22-alpine

ARG VITE\_DEFAULT\_AVATAR

ARG VITE\_DEFAULT\_BACKGROUND

ENV VITE\_DEFAULT\_AVATAR=$VITE\_DEFAULT\_AVATAR

ENV VITE\_DEFAULT\_BACKGROUND=$VITE\_DEFAULT\_BACKGROUND

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 5050

# CMD ["npm", "run", "dev"]

CMD ["npm", "run", "preview", "--", "--port", "5050", "--host", "0.0.0.0"]

3.4 Код файла docker-compose.yaml, определяющего все контейнеры

services:

frontend:

build:

context: ./frontend

args:

VITE\_DEFAULT\_AVATAR: ${DEFAULT\_AVATAR}

VITE\_DEFAULT\_BACKGROUND: ${DEFAULT\_BACKGROUND}

container\_name: frontend

ports:

- "5050:5050"

networks:

- storage-net

restart: always

minio:

image: quay.io/minio/minio:latest

container\_name: minio

ports:

- "9000:9000"

- "9001:9001"

environment:

MINIO\_ROOT\_USER: ${MINIO\_ROOT\_USER}

MINIO\_ROOT\_PASSWORD: ${MINIO\_ROOT\_PASSWORD}

volumes:

- minio\_data:/data

- minio\_config:/root/.minio

command: server /data --console-address ":9001"

networks:

- storage-net

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:9000/minio/health/live"]

interval: 10s

timeout: 5s

retries: 3

minio-api:

build: ./minio\_img\_service

container\_name: minio\_img\_service

ports:

- "8080:8080"

environment:

MINIO\_URL: http://minio:9000

MINIO\_ROOT\_USER: ${MINIO\_ROOT\_USER}

MINIO\_ROOT\_PASSWORD: ${MINIO\_ROOT\_PASSWORD}

SECRET\_KEY: ${SECRET\_KEY}

DEFAULT\_AVATAR: ${DEFAULT\_AVATAR}

DEFAULT\_BACKGROUND: ${DEFAULT\_BACKGROUND}

depends\_on:

minio:

condition: service\_healthy

networks:

- storage-net

postgres:

image: postgres:latest

container\_name: tcp-labs-db

ports:

- "5438:5432"

environment:

POSTGRES\_USER: ${DB\_USER}

POSTGRES\_PASSWORD: ${DB\_PASSWORD}

POSTGRES\_DB: ${DB\_NAME}

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- postgres\_data:/var/lib/postgresql/data

restart: always

healthcheck:

test: ["CMD-SHELL", "pg\_isready"]

interval: 10s

timeout: 5s

retries: 5

networks:

- storage-net

redis:

image: redis:latest

container\_name: redis

restart: always

ports:

- "6379:6379"

volumes:

- redis\_data:/data

command: redis-server --save 20 1 --loglevel warning

networks:

- storage-net

fastapi\_app:

build: .

container\_name: fastapi\_app

ports:

- "8000:8000"

environment:

# Redis config

REDIS\_HOST: redis

REDIS\_PORT: 6379

REDIS\_DB: ${REDIS\_DB}

# PostgreSQL config

DB\_HOST: postgres

DB\_PORT: 5432

DB\_NAME: ${DB\_NAME}

DB\_USER: ${DB\_USER}

DB\_PASSWORD: ${DB\_PASSWORD}

# Default images config

DEFAULT\_AVATAR: ${DEFAULT\_AVATAR}

DEFAULT\_BACKGROUND: ${DEFAULT\_BACKGROUND}

# jwt config

SECRET\_KEY: ${SECRET\_KEY}

ALGORITHM: ${ALGORITHM}

ACCESS\_TOKEN\_EXPIRE\_MINUTES: ${ACCESS\_TOKEN\_EXPIRE\_MINUTES}

REFRESH\_TOKEN\_EXPIRE\_DAYS: ${REFRESH\_TOKEN\_EXPIRE\_DAYS}

depends\_on:

postgres:

condition: service\_healthy

redis:

condition: service\_started

networks:

- storage-net

restart: always

networks:

storage-net:

driver: bridge

volumes:

minio\_data:

driver: local

minio\_config:

driver: local

postgres\_data:

driver: local

redis\_data:

driver: local