Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

О Т Ч Е Т

по индивидуальному заданию

Название: Разработка микросервиса авторизации и аутентификации

Дисциплина: Проектирование информационных систем

Студент	ИУ6-75Б		Т.А. Кудрявцев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			А.Ф. Прутик
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2025

Цель работы

Целью данной работы является разработка микросервиса авторизации и регистрации пользователей для распределённой системы онлайн-кинотеатра.

Разрабатываемый сервис обеспечивает:

- управление пользовательскими аккаунтами;
- безопасную регистрацию и аутентификацию;
- выдачу и проверку JWT-токенов;
- взаимодействие с клиентской частью через REST API.

Сервис реализован как отдельный backend-компонент и не зависит от других микросервисов системы.

Требования к системе

В рамках работы были реализованы следующие функциональные и нефункциональные требования:

- **Регистрация и вход пользователя**: реализованы сценарии регистрации (signup) и аутентификации (login) с выдачей пары JWT-токенов (*access* и *refresh*);
- **Безопасность хранения данных**: пароли пользователей хранятся в базе данных **исключительно в виде хешей** с использованием надёжных алгоритмов хеширования;
- **Доступ к защищённым маршрутам**: доступ к пользовательским данным (например, профиль текущего пользователя) осуществляется только при наличии валидного JWT-токена;
- **Обновление сессии**: реализован механизм обновления access-токена с использованием refresh-токена;
- **Выход из системы**: реализован logout с использованием blacklist refresh-токенов;
- **Ролевая модель пользователей**: поддерживаются роли user, admin;
- **Клиентская часть (Frontend)**: разработан простой SPA-интерфейс, взаимодействующий с backend-API без перезагрузки страницы.

В учебной конфигурации сервиса отправка email-сообщений отключена, а пользователь считается подтверждённым сразу после регистрации.

1. Описание решения и используемые технологии

Для реализации сервиса выбрана микросервисная архитектура с использованием контейнеризации, что обеспечивает изоляцию компонентов, масштабируемость и удобство развертывания.

Используемый технологический стек

Backend:

- Python 3.11;
- FastAPI — выбран за высокую производительность, асинхронную модель работы и автоматическую генерацию документации Swagger / OpenAPI;
- Uvicorn — ASGI-сервер для запуска приложения;
- JWT (JSON Web Token) — для реализации аутентификации и авторизации;
- OAuth2PasswordBearer — для интеграции JWT с FastAPI.

База данных:

- PostgreSQL;
- SQLAlchemy 2.x (асинхронный режим);
- asyncpg — драйвер для асинхронного доступа к БД;
- миграции базы данных реализованы через Alembic.

Frontend:

- HTML5;
- CSS3;
- Vanilla JavaScript (ES6+);
- клиентская часть реализована как SPA и взаимодействует с backend через Fetch API.

Инфраструктура:

- Docker;
- Docker Compose;
- Mailpit.

2 Архитектурные схемы

На рисунке 1 показана диаграмма контекста, которая показывает систему в окружении, выделяя основные внешние системы и пользователей, которые взаимодействуют с ней. На рисунке 2 – диаграмма контейнеров, показывающая взаимодействие микросервисов и сторонних систем. На рисунке 3 отображена диаграмма компонентов реализованного микросервиса аутентификации, регистрации, базового функционала профиля пользователя. Система состоит из веб-приложения (SPA), API-сервера, базы данных и почтового сервера. На 4 рисунке показана диаграмма классов и структура кода в целом.

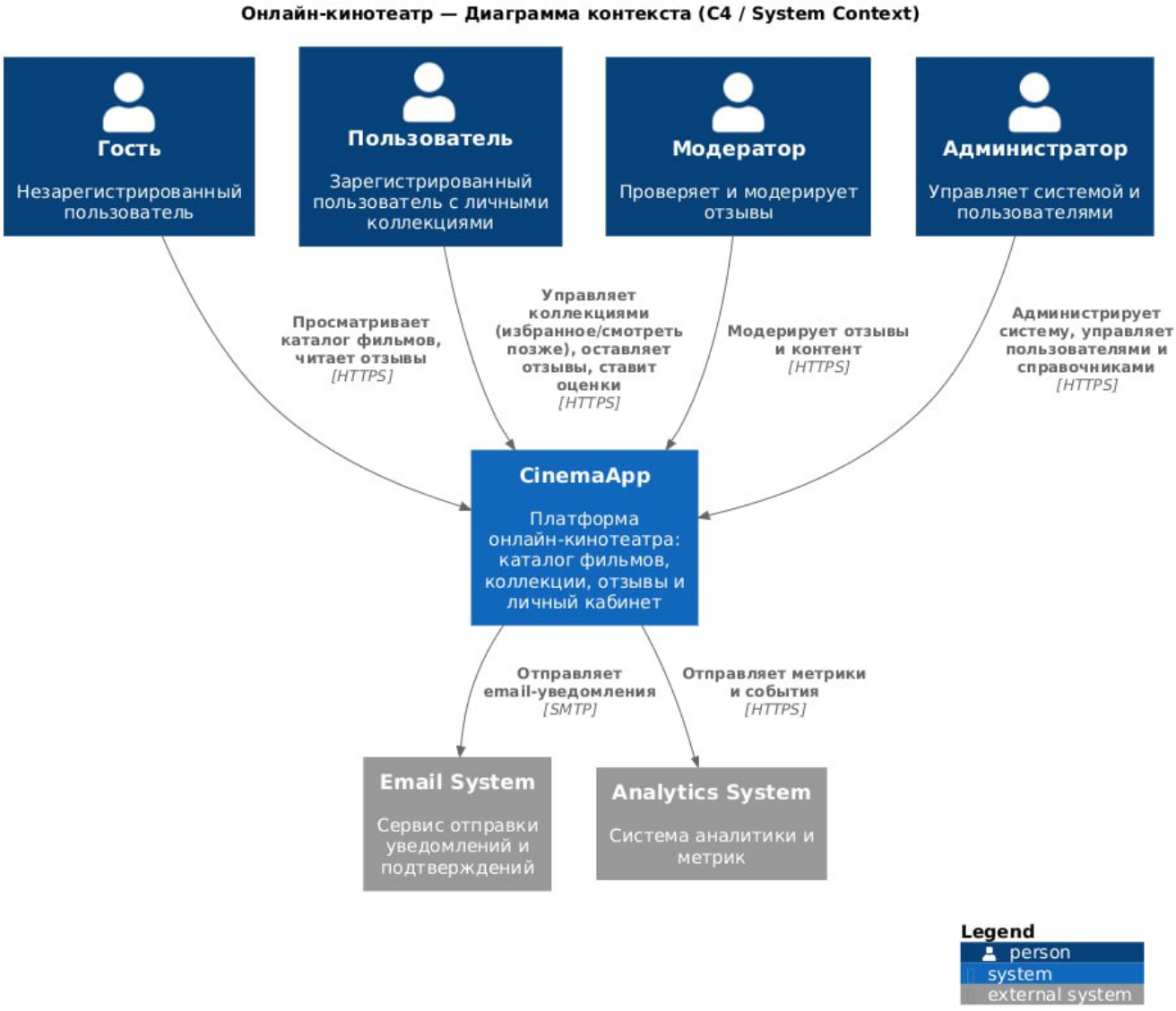

Рисунок 1 – Диаграмма контекста

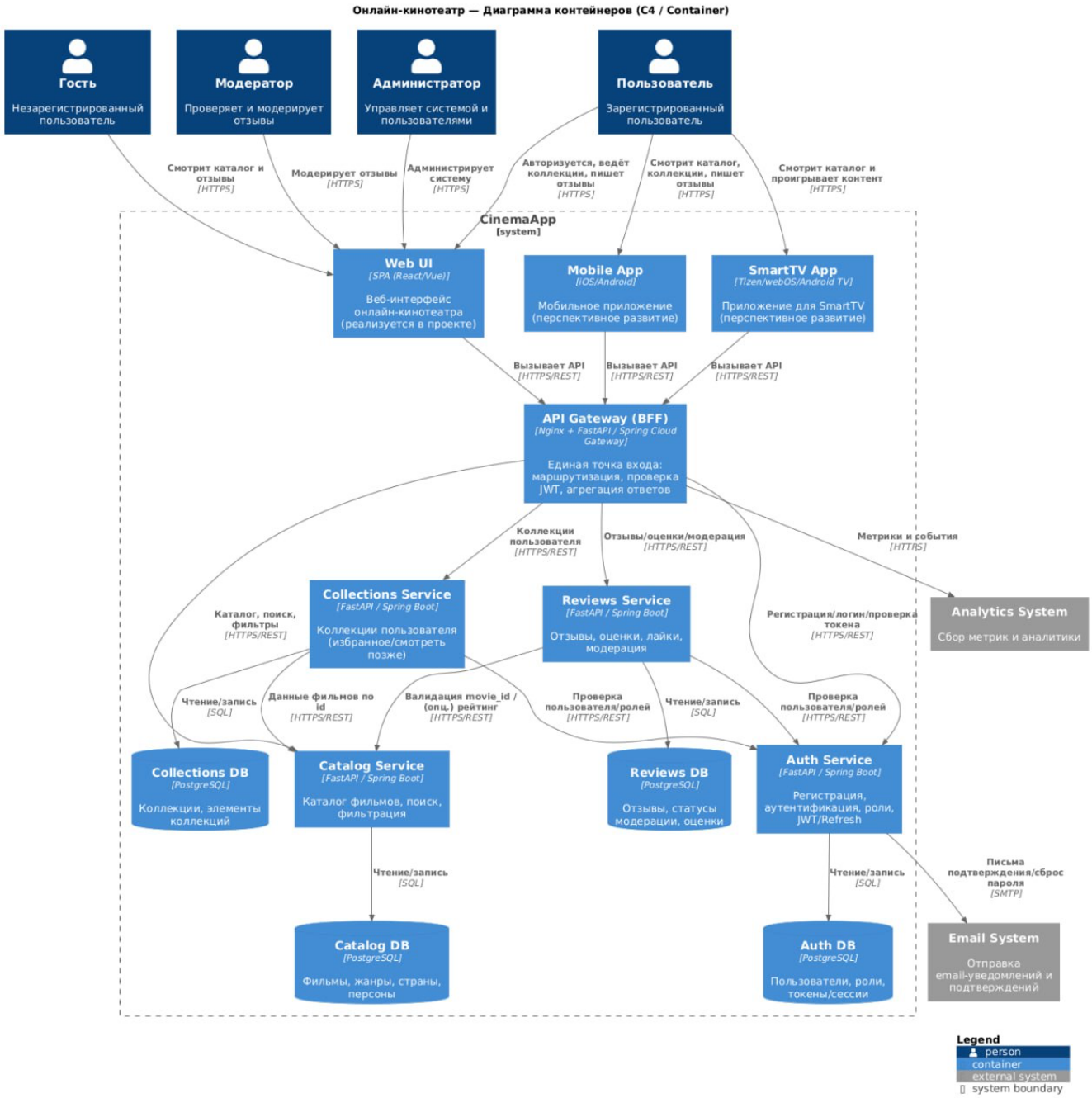

Рисунок 2 – Диаграмма контейнеров

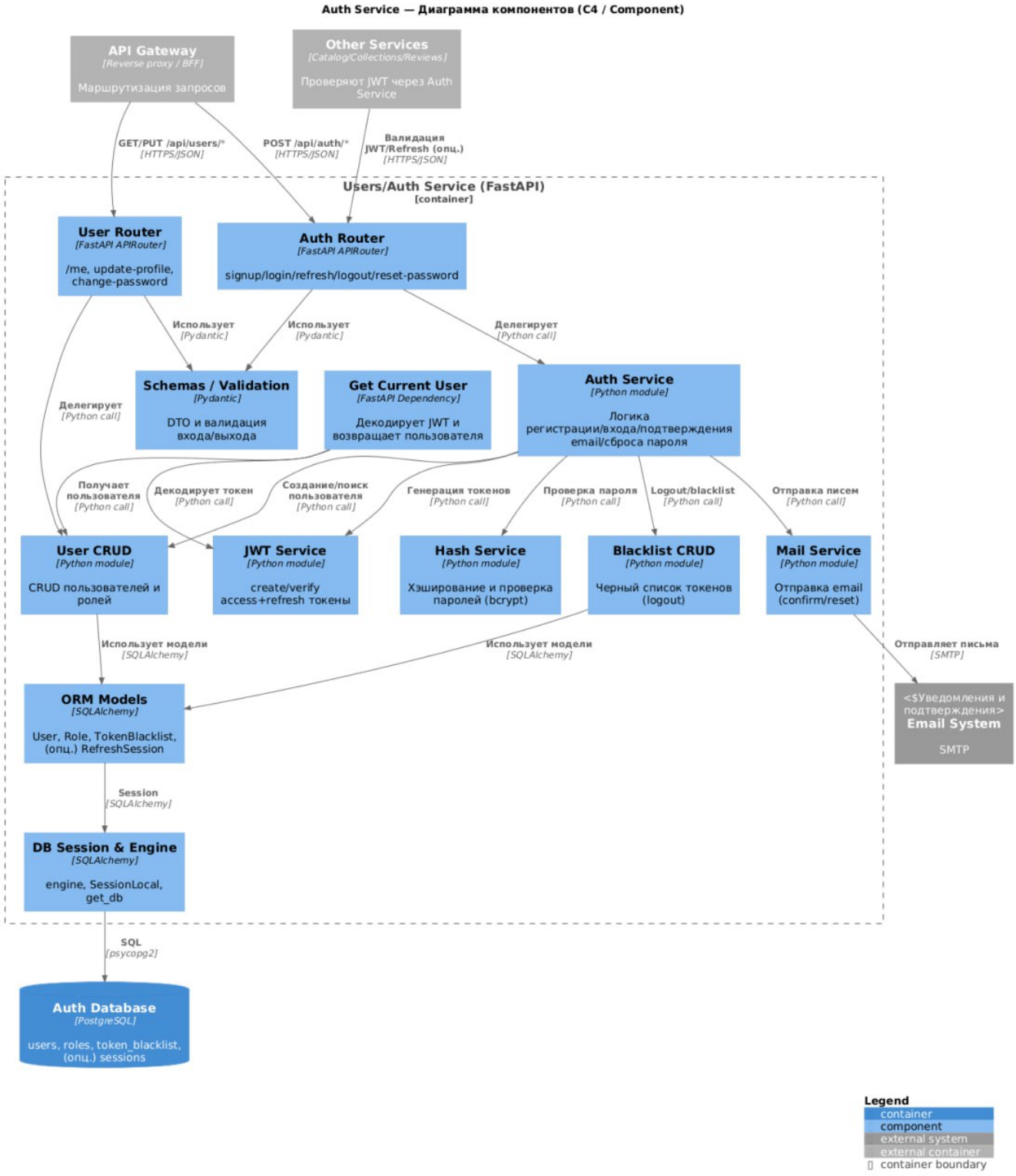

Рисунок 3 – Диаграмма компонентов

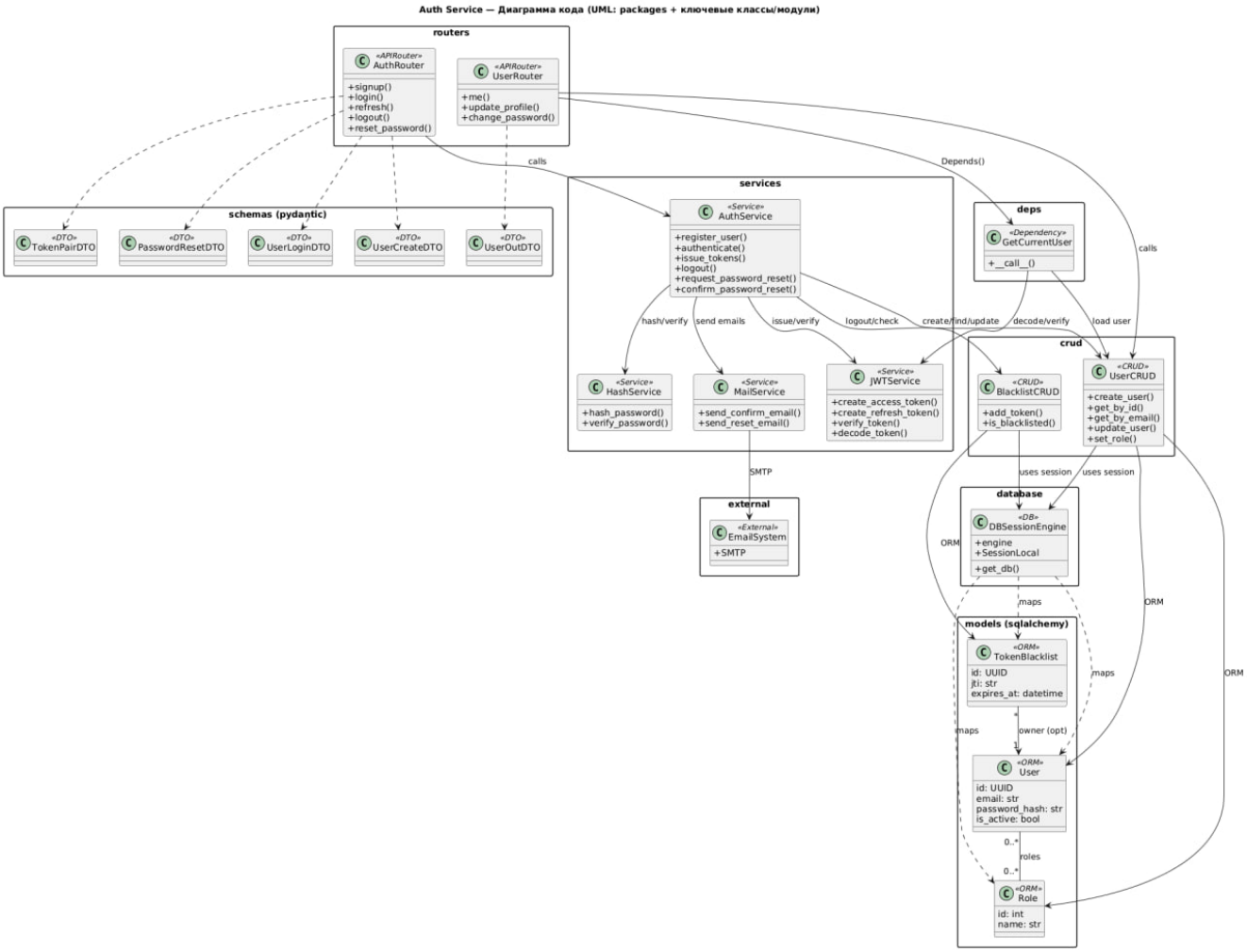

Рисунок 4 – Диаграмма кода

3 Детали реализации:

В ходе работы были реализованы следующие эндпоинты.

Эндпоинты аутентификации и авторизации

POST /auth/signup **— Регистрация пользователя**

Эндпоинт предназначен для создания нового пользовательского аккаунта.

Принимает email, username и пароль пользователя.

Пароль перед сохранением хешируется с использованием надёжного алгоритма.

В учебной конфигурации подтверждение email отключено, и пользователь считается подтверждённым сразу после регистрации (is_verified = true).

POST /auth/login **— Аутентификация пользователя**

Принимает email и пароль пользователя.

В случае успешной аутентификации возвращает пару JWT-токенов:

- access token — для доступа к защищённым маршрутам;
- refresh token — для обновления access-токена.

POST /auth/refresh **— Обновление access-токена**

Принимает валидный refresh-токен и возвращает новый access-токен. Используется для продления пользовательской сессии без повторного ввода логина и пароля.

POST /auth/logout **— Выход из системы**

Принимает refresh-токен пользователя. Токен добавляется в blacklist, что предотвращает его повторное использование.

Эндпоинты управления пользователем

GET /user/me **— Получение данных текущего пользователя**

Возвращает информацию о текущем аутентифицированном пользователе.
Является защищённым маршрутом и требует передачи access-токена в заголовке Authorization: Bearer.

PATCH /user/update-username **— Изменение имени пользователя**
Позволяет изменить имя пользователя.
Доступен только для аутентифицированных пользователей и требует валидного access-токена.

PATCH /user/update-password **— Смена пароля пользователя**

Позволяет изменить пароль из профиля пользователя.
Для выполнения операции необходимо:

- передать старый пароль;
- указать новый пароль;
- иметь валидный access-токен.

Часть эндпоинтов, связанных с email-подтверждением и восстановлением пароля, в текущей учебной конфигурации не активны и могут быть расширены в дальнейшем без изменения общей архитектуры сервиса.

Написаны интеграционные и unit-тесты с использованием pytest. Они запускаются в отдельной тестовой базе данных (auth_test_db), которая создается и очищается для каждого прогона.

Вывод: был успешно спроектирован и реализован микросервис авторизации. Система обеспечивает безопасное хранение данных пользователей и поддерживает полный цикл управления доступом. Использование Docker позволяет легко развернуть сервис в любой среде.