САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №1

Выполнил:

Пиотуховский Александр

K3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Необходимо спроектировать набор следующих диаграмм:

— общая архитектура решения (сервисы и их взаимосвязи, клиент-серверное взаимодействие);

— диаграмма компонентов;

— диаграммы БД по каждому сервису;

— диаграммы основных пользовательских сценариев (те сценарии, которые позволяют вашим приложением полноценно воспользоваться, пройти весь путь).

Ход работы

В данном отчете представлен технический дизайн сервиса библиотеки фильмов (FilmApp). Система обеспечивает полный цикл взаимодействия пользователя с медиа-контентом: от регистрации и управления безопасным доступом с использованием JWT-токенов до умного семантического поиска фильмов, ведения списков избранного, системы рейтингов и древовидных комментариев.

Архитектурно решение представляет собой распределенную систему с асинхронным взаимодействием. Основное ядро реализовано на Python и отвечает за бизнес-логику и работу с данными. Функционал отправки писем на почту вынесен в изолированный микросервис на Node.js, который взаимодействует с ядром через брокер сообщений Apache Kafka. Хранение данных организовано в реляционной базе PostgreSQL с использованием JSONB-структур для оптимизации сложных атрибутов контента.

Данный проект оптимально подходит для демонстрации технического дизайна, так как содержит все необходимые слои для построения требуемых диаграмм. Наличие событийно-ориентированного взаимодействия между сервисами (Python и Node.js), сложная схема базы данных и применение принципов чистой архитектуры (Service/Repository layers) позволяют детально проработать как компонентные схемы, так и сценарии пользовательского взаимодействия в распределенной среде.

Описание общей архитектуры решения

Диаграмма демонстрирует высокоуровневую архитектуру системы, декомпозицию на сервисы и интеграцию с внешней инфраструктурой. Она показывает границы системы и потоки данных между контейнерами. Система построена по гибридному принципу, сочетающему монолитное ядро и микросервисную обвязку для асинхронных задач.

Core API Service (Python/Starlette). Центральный компонент системы. Обрабатывает входящие HTTP-запросы от клиентов, реализует основную бизнес-логику и взаимодействует с базой данных.

Notification Service (Node.js). Специализированный микросервис-воркер. Его задача — изолировать логику отправки уведомлений от основного потока выполнения. Он не имеет публичного API и управляется исключительно событиями из очереди сообщений.

Apache Kafka. Выступает в роли брокера данных, обеспечивая асинхронную и надежную связь между Core API и Notification Service. Это позволяет основному сервису не блокироваться при отправке писем.

PostgreSQL. Единое хранилище реляционных данных для всех сущностей системы.

Внешние системы. TMDB API используется для обогащения контента (загрузка актуальных постеров и трейлеров); SMTP Server является внешним провайдером для доставки электронных писем.

На рисунке 1 изображена диаграмма общей архитектуры решения.

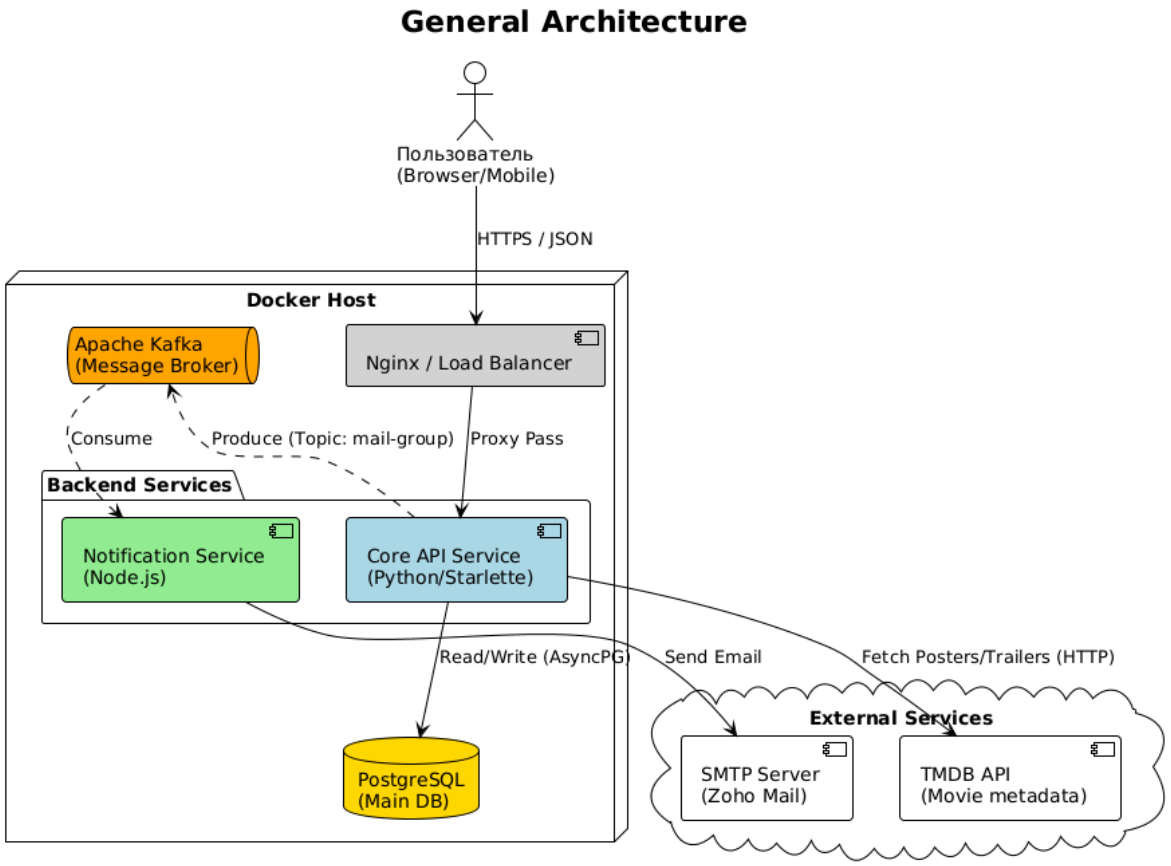

Рисунок 1 – Диаграмма общей архитектуры решения

Описание диаграммы компонентов

Диаграмма детализирует внутреннее устройство основного сервиса Core API. Она демонстрирует применение принципов «Чистой архитектуры» и инверсии зависимостей.

Диаграмма состоит из следующих структурных слоёв:

Уровень представления. Содержит контроллеры и Middleware, отвечает только за валидацию входных данных и формирование ответа. Не содержит бизнес-логики.

Сервисный уровень. Основная логика приложения, здесь сосредоточены алгоритмы работы системы. Сервисы не знают о том, откуда пришли данные (HTTP или CLI) и куда они сохраняются (SQL или файл), благодаря абстракциям.

Уровень доступа к данным. Отвечает за прямую работу с базой данных PostgreSQL. Преобразует объекты базы данных в DTO, используемые внутри приложения.

Инфраструктура и ядро. IoC контейнер управляет созданием объектов и внедрением зависимостей, связывая интерфейсы с их реализациями. Kafka Client инкапсулирует логику работы с брокером сообщений.

На рисунке 2 изображена диаграмма компонентов.

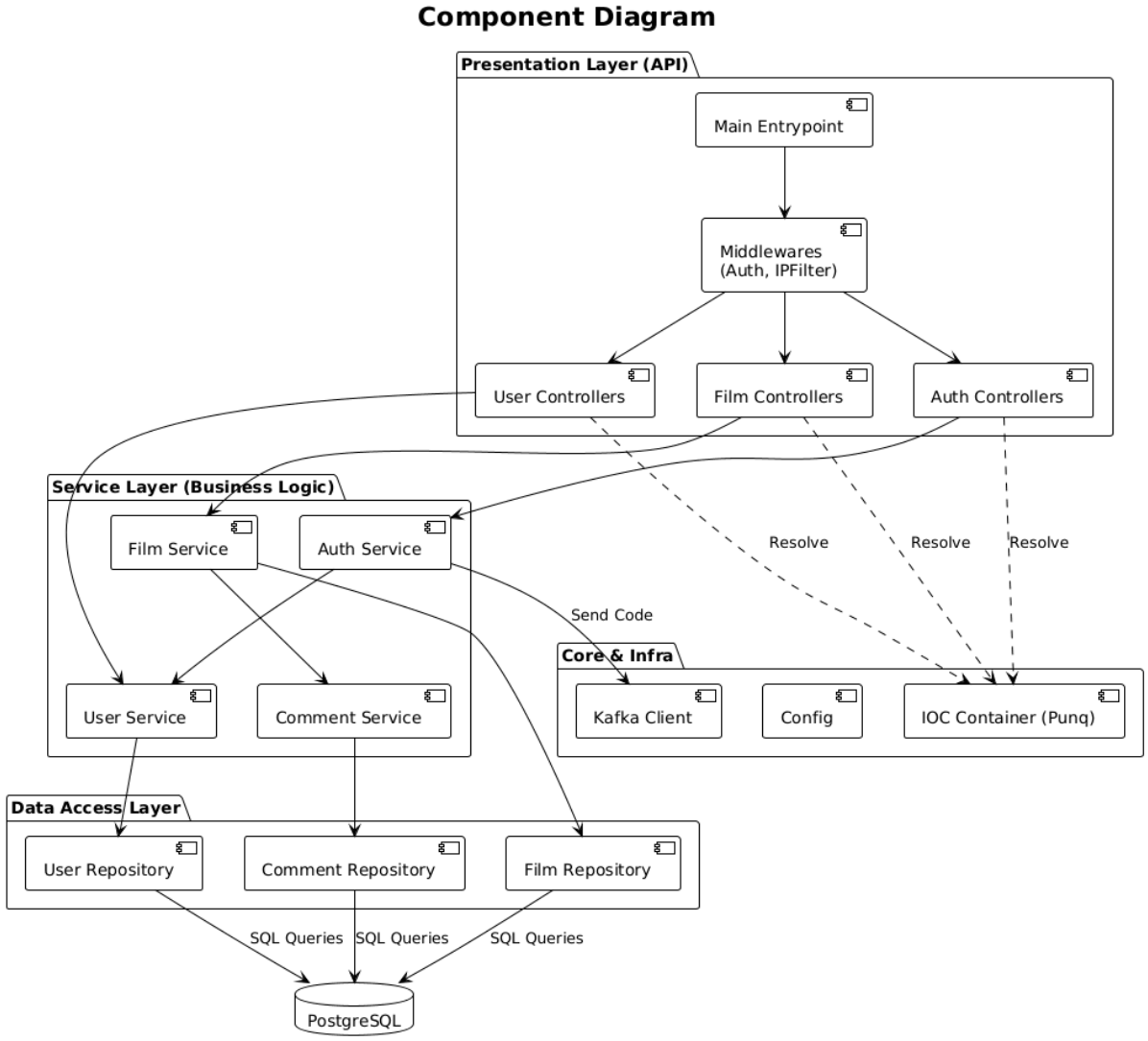

Рисунок 2 – Диаграмма компонентов

Описание схемы базы данных

Схема отражает логическую структуру хранения данных, связи между сущностями и ключевые атрибуты.

На рисунке 3 изображена схема базы данных.

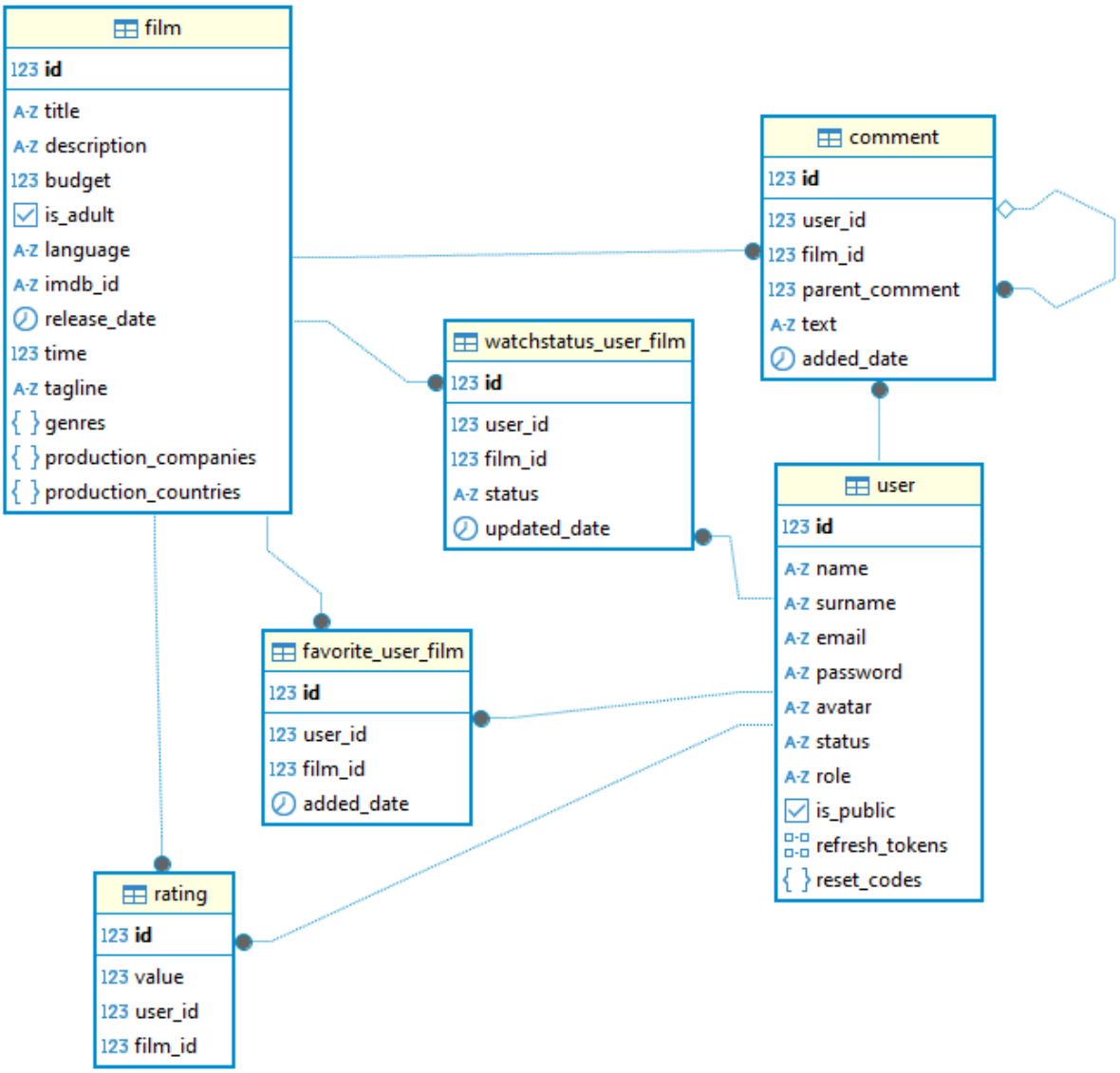

Рисунок 3 – Схема базы данных

Описание пользовательских сценариев

Сценарий 1. Регистрация пользователя и подтверждение Email.

Диаграмма последовательности демонстрирует событийно-ориентированное взаимодействие между микросервисами.

Пользователь отправляет данные регистрации. Сервис сохраняет пользователя в БД со статусом `not_verified` и генерирует код подтверждения. Вместо синхронной отправки письма, сервис публикует событие в топик брокера и сразу возвращает ответ пользователю. Сервис нотификаций, подписанный на топик, получает сообщение, формирует письмо и отправляет его через SMTP-сервер. Пользователь вводит код, и сервис подтверждает активацию аккаунта.

Преимуществом подхода является высокая отзывчивость интерфейса (пользователь не ждёт отправки письма) и отказоустойчивость (если почтовый сервис недоступен, сообщение сохранится в Kafka и будет обработано позже).

На рисунке 4 изображён алгоритм регистрации пользователя.

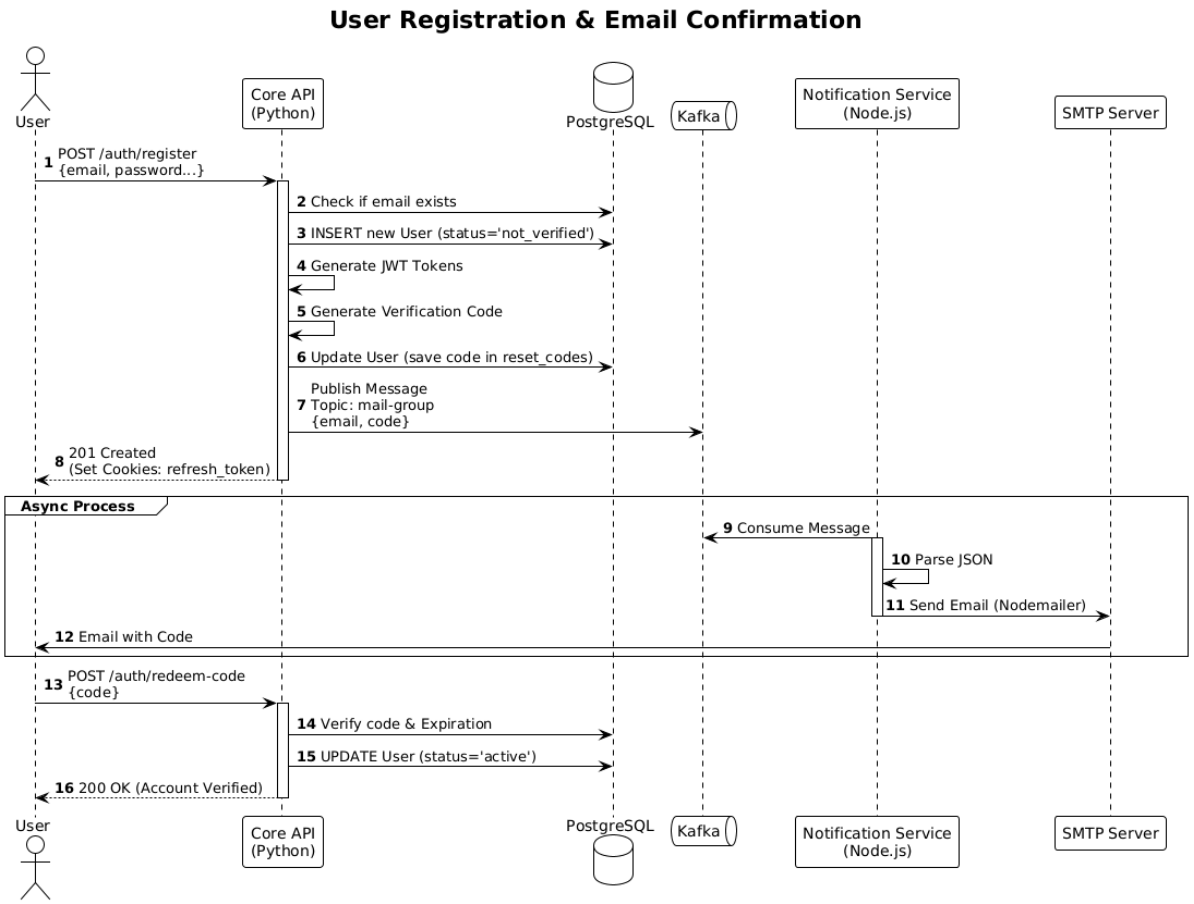

Рисунок 4 – Алгоритм регистрации пользователя

Сценарий 2. Поиск и фильтрация фильмов.

Диаграмма показывает синхронный поток обработки запроса внутри монолитного сервиса, проходящий через все архитектурные слои.

Контроллер принимает GET-запрос с параметрами. Сервис применяет бизнес-правила, такие как валидация параметров, и вызывает репозиторий. Репозиторий формирует SQL-запрос. Данные возвращаются по цепочке вверх, преобразуясь из сырых данных БД в Pydantic-модели для отправки клиенту.

На рисунке 5 изображена алгоритм поиска и фильтрации фильмов.

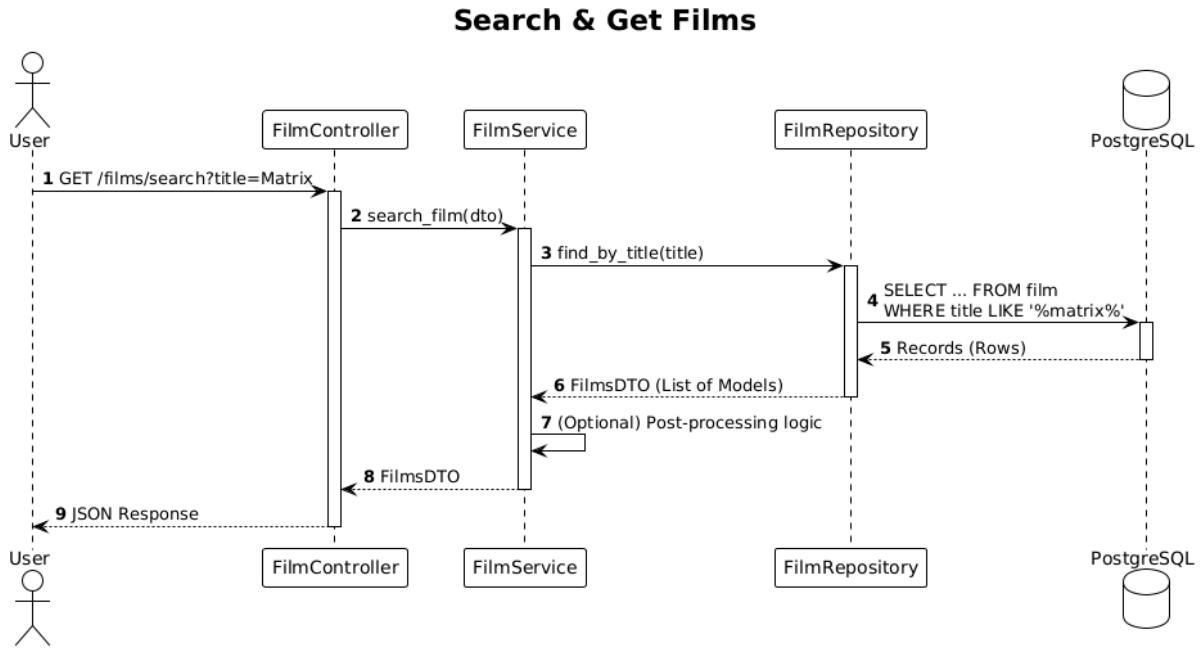

Рисунок 5 – алгоритм поиска и фильтрации фильмов

Вывод

В ходе выполнения домашней работы были спроектированы требуемые диаграммы и схемы. Также были обновлены знания о написанном проекте.