**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**
**РОССИЙСКОЙ ФЕДЕРАЦИИ**
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**
**(национальный исследовательский университет)» (МАИ)**
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

# Курсовой проект
# по дисциплине «Базы данных»
# Тема: «Видеоигровой портал»

Студент: Тарасов Е.Д.
Группа: М80-309Б-23
Преподаватель: Грубенко М.Д.
Оценка: _____
Дата: _____
Подпись: _____

Москва 2025

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

В современном мире видеоигры являются одним из наиболее популярных видов развлечений, занимающим важное место на рынке. Миллионы людей по всему миру проводят множество часов за играми, отслеживают свой прогресс, пишут отзывы и делятся впечатлениями с другими игроками. Для реализации таких процессов и удобного хранения и обработки информации о них требуются специализированные системы — игровые порталы.

Актуальность разработки подобной системы заключается в том, что наличие сервиса с удобными инструментами для учёта игр, прогресса, статистики и взаимодействия пользователей не только упрощает жизнь игрокам, но и способствует росту популярности и продаж видеоигр. Пользовательские отзывы и возможность отслеживать прогресс повышают вовлеченность аудитории: положительные оценки формируют доверие потенциальных покупателей, а статистика мотивирует игроков проводить больше времени в играх и делиться своими впечатлениями, что приводит к распространению информации об играх.

В качестве предметной области выбрана система учета игрового прогресса пользователей, включающая информацию об играх, разработчиках, жанрах, платформах, личных списках пользователей, отзывах и статистике.

Цель данной курсовой работы: проектирование и реализация системы игрового портала, включающая полную реализацию реляционной СУБД PostgreSQL и backend-приложения на языке Python с использованием фреймворка FastAPI.

# 1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

## 1.1 Обзор предметной области

Предметная область курсовой работы – разработка информационной системы для учета игрового прогресса и взаимодействия пользователей (игровой портал). Система должна позволять игрокам вести личный список игр, отмечать статус прохождения (играю, пройдено, планирую, брошено), фиксировать количество часов, проводимых пользователями в той или иной игре, писать и читать отзывы, а также просматривать статистику (как свою, так и других игроков).

Существующие аналоги подтверждают востребованность подобных систем. Например:

1) Steam – крупнейшая платформа, где пользователи видят часы, проведённые в играх, сами игры, достижения, пишут отзывы и видят средний рейтинг. система рекомендаций и статистика сильно повышают вовлеченность;
2) HowLongToBeat – специализированное веб-приложение, позволяющее отмечать статус игр, часы прохождения, делиться отзывами с другими игроками и сравнивать статистику;
3) PlayStation Network –- онлайн сервис компании Sony для владельцев консоли PlayStation. пользователи могут отслеживать трофеи, которые отражают прогресс в играх, просматривать свою и чужую статистику, писать отзывы, делиться впечатлениями с друзьями.

Популярность и спрос таких сервисов показывает, что пользователям необходимы инструменты для организации своей игровой библиотеки и анализа прогресса.

### 1.2 Постановка задачи

На основе анализа технического задания и предметной области можно сформулировать следующие задачи при разработке сервиса:

1) спроектировать структуру реляционной базы данных, включающую не менее 9–10 таблиц с отношениями типов 1:1, 1:N и N:M, соответствующую третьей нормальной форме. обеспечить наличие ключевых сущностей (пользователи, игры, отзывы) и журнала аудита изменений;
2) реализовать ограничения целостности: первичные и внешние ключи, уникальность, проверки (check, not null), каскадное обновление и удаление записей;
3) наполнить базу данных тестовыми данными в необходимом объеме;
4) разработать SQL-функциональность (триггеры для автоматического обновления агрегатных полей, таких как средний рейтинг и количество часов, и ведения журнала аудита; скалярные и табличные функции для расчета рейтингов, статистики и отчётов; не менее трех представлений с агрегированными данными; сложные запросы);
5) оптимизировать время работы запросов, функций и триггеров при помощи индексов и провести анализ с использованием explain analyze;
6) интегрировать базу данных с backend-приложением на Python (с использованием фреймворка FastAPI), реализовать REST API для основных операций и получения аналитики;
7) обеспечить контейнеризацию проекта при помощи Docker.

Разработанная система полностью соответствует требованиям технического задания и демонстрирует практические навыки работы с СУБД PostgreSQL и ее интеграцией с веб-приложением.

# 2 ПРОЕКТНАЯ ЧАСТЬ

## 2.1 Архитектура системы

Разработанная информационная система «Игровой портал» построена по трехуровневой клиент-серверной архитектуре и полностью контейнеризована с использованием Docker и docker-compose. Это обеспечивает простоту развертывания, воспроизводимость окружения и независимость от операционной системы хоста. Общая схема взаимодействия уровней системы представлена на рисунке 2.1.

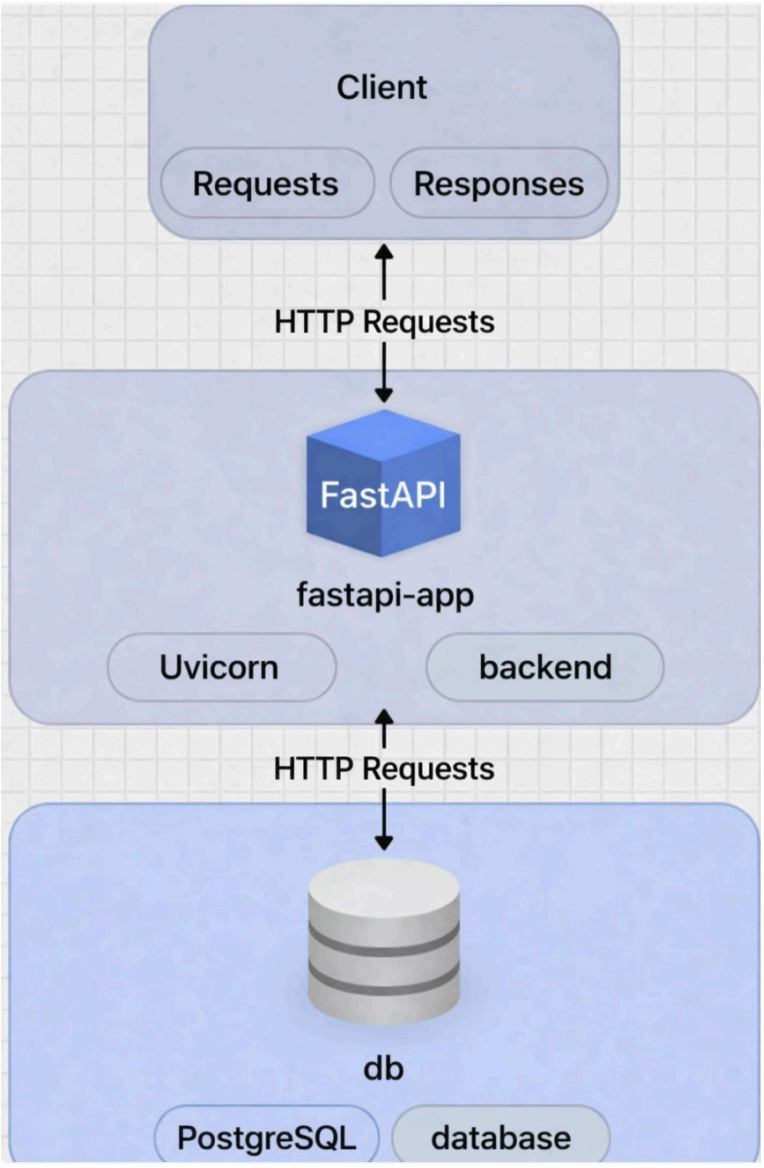


Рисунок 2.1 — Архитектура системы

Архитектура включает следующие уровни:

1) уровень базы данных: реализован на СУБД PostgreSQL версии 16. база данных хранит все сущности системы. для инициализации структуры используется сборка каталога database, содержащего полную реализацию базы данных при помощи SQL-скрипта, включающего создание таблиц, триггеров, функций, представлений и индексов;
2) уровень приложения (backend): веб-сервер, реализованный на языке Python 3.12 с использованием фреймворка FastAPI. приложение предоставляет RESTful API для выполнения CRUD-операций всех типов, получения аналитических данных с использованием API функций и представлений, а также батчевой загрузки данных. для взаимодействия с базой данных применяется ORM SQLAlchemy. для инициализации веб-приложения используется сборка каталога backend, содержащего инициализацию сервера, связь с базой данных, роутеры, а также модели SQLAlchemy и схемы Pydantic;
3) уровень клиента: любое устройство или приложение, способное выполнять HTTP-запросы. взаимодействие с системой осуществляется с помощью API backend-сервера. фреймворк FastAPI автоматически генерирует интерактивную документацию SwaggerUI для удобного взаимодействия пользователя с системой..

Контейнеризация реализована с помощью docker-compose версии 3.8 и содержит следующие контейнеры: postgres – контейнер базы данных и api – контейнер с FastAPI приложением. Секреты хранятся в файле .env, который исключен из репозитория проекта.

С помощью такого подхода обеспечивается безопасность и удобство запуска проекта на различных операционных системах.

## 2.2 Проектирование структуры базы данных

Структура базы данных разработана в соответствии с требованиями предметной области и технического задания. База данных содержит 11 таблиц (10 таблиц для сущностей и 1 для аудита), обеспечивающих хранение информации о пользователях, играх, разработчиках, жанрах, платформах, прогрессе пользователей, отзывах и аудите изменений. Все таблицы приведены к третьей нормальной форме, что исключает избыточность данных и возникновение аномалий при операциях вставки, обновления и удаления.

Для визуализации логической структуры базы данных построена ER-диаграмма при помощи DBeaver (приведена на рисунке 2.2). На диаграмме отображены сущности, их атрибуты, ключи и связи.

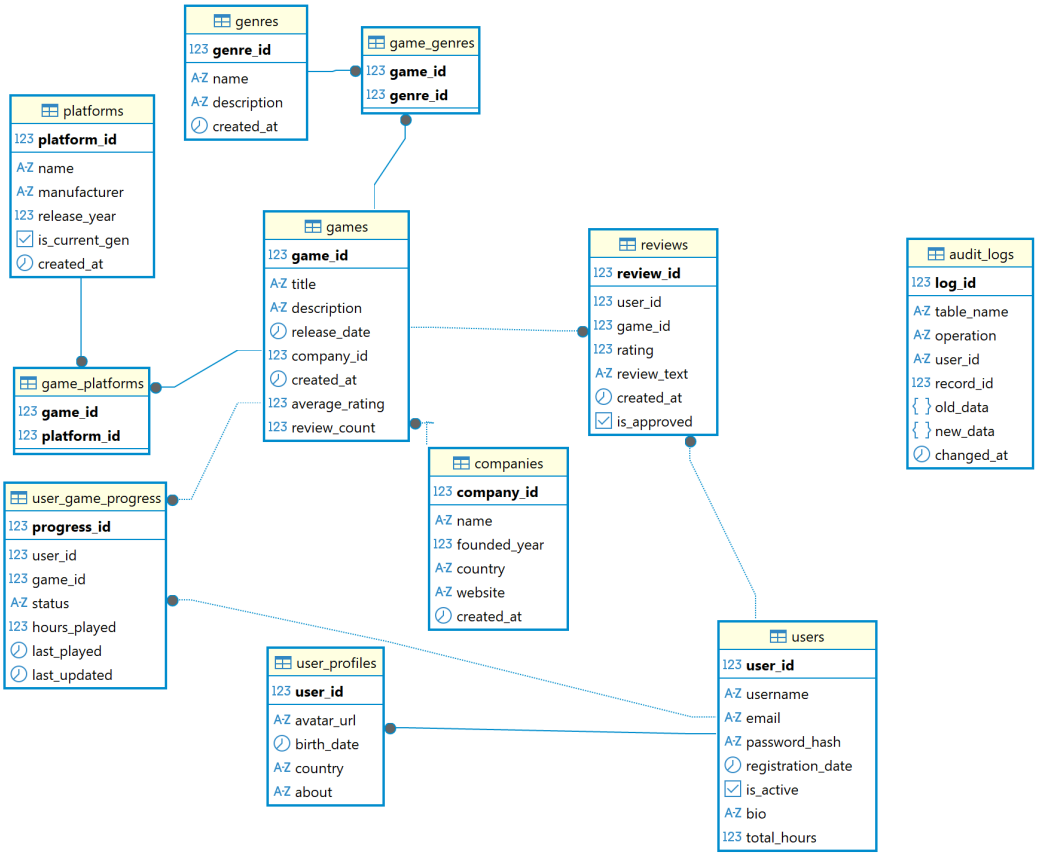


Рисунок 2.2 — ER-диаграмма

Как и требовалось в техническом задании, в моей базе данных реализованы все виды типов связей между сущностями:

1) 1:N, например, одна компания может разработать множество игр, но каждая игра может принадлежать только одной компании;
2) N:M, например, одна игра может относится к нескольким жанрам, а жанр также может относится к множеству игр;
3) 1:1, например, каждый пользователь может иметь свой уникальный профиль, который содержит дополнительную информацию об игроку.

Основные сущности базы данных:

1) users – пользователи системы. содержит идентификатор, имя пользователя, email, хэш пароля, дату регистрации, статус активности, статус, общее количество часов игры. имеет связь 1:1 с user_profiles, 1:N с user_game_progress и 1:N с reviews;
2) user_profiles – дополнительная информация о пользователе. содержит аватар, дату рождения, страну, текст «о себе»). имеет 1:1 связь с users;
3) companies – компании-разработчики игр. содержит идентификатор, название, год основания, страну, сайт. имеет связь 1:N с games;
4) genres – жанры игр. содержит идентификатор, название, описание. используется для связи N:M с games с game_genres;
5) platforms – игровые платформы. содержит идентификатор, название, производителя, год выпуска, признак текущего поколения. используется для связи N:M с games через game_platforms;
6) games – игры. содержит идентификатор, название, описание, дату выхода, идентификатор компании-разработчика, средний рейтинг, количество отзывов. имеет связь N:1 с companies, N:M с genres через game_genres, N:M с platforms через game_platforms, 1:N с

user_game_progress и 1:N с reviews;

7) game_genres – связующая таблица для реализации связи многие-ко-многим между games и genres. содержит идентификаторы игры и жанра (составной первичный ключ). имеет связи N:1 с games и genres;
8) game_platforms – связующая таблица для реализации связи многие-ко-многим между games и platforms. содержит идентификаторы игры и платформы (составной первичный ключ). имеет связи N:1 с games и platforms;
9) user_game_progress – прогресс пользователя по игре. содержит идентификатор, идентификатор пользователя, идентификатор игры, статус прохождения, количество часов, дату последнего запуска. имеет связи N:1 с users и games;
10) reviews – отзывы пользователей на игры. содержит идентификатор, идентификатор пользователя, идентификатор игры, оценку, текст отзыва, статус одобрения. имеет связи N:1 с users и games;
11) audit_logs – журнал аудита изменений. содержит идентификатор, название таблицы, тип операции, идентификатор записи, старые и новые данные в формате JSONB.

В таблицах games (average_rating, review_count) и user_game_progress (total_hours) была применена осознанная денормализация для повышения производительности чтения часто запрашиваемых агрегатных данных.

### 2.3 Описание триггеров, функций, представлений, индексов

В моей базе данных активно используются триггеры, SQL-функции, представления и индексы для обеспечения согласованности данных, выполнения сложных вычислений, предоставления аналитических отчетов и оптимизации производительности запросов.

Реализовано пять триггеров, реализующие ведение аудита и обновление агрегированных данных:

1) trig_update_game_aggregates — срабатывает после insert, delete или update в таблице reviews. автоматически обновляет агрегированные поля average_rating и review_count в таблице games, учитывая только одобренные отзывы;
2) trig_update_user_total_hours — срабатывает после insert, delete или update в таблице user_game_progress. автоматически обновляет агрегированное поле total_hours в таблице users;
3) audit_users, audit_progress, audit_reviews, audit_games — триггеры аудита, срабатывающие после insert, delete или update на соответствующих таблицах (users, user_game_progress, reviews, games). записывают информацию об изменениях в таблицу audit_logs.

Разработаны четыре функции для предоставления основных аналитических данных:

1) get_game_rating — скалярная функция, возвращающая средний рейтинг одобренных отзывов для указанной игры;
2) get_user_total_hours — скалярная функция, возвращающая общее количество часов, наигранных пользователем;
3) get_top_players_by_genre — табличная функция, возвращающая топ-10 пользователей по количеству часов в играх указанного жанра (с сортировкой по убыванию);
4) get_user_activity — табличная функция, возвращающая ежедневную активность всех пользователей за указанный период (часы игры и количество написанных отзывов).

Создано три представления для доступа к агрегированным данным:

1) game_ratings_view — содержит идентификатор игры, название, дату выхода, средний рейтинг и количество одобренных отзывов;

2) user_stats_view — статистика пользователей: количество игр в списке, количество завершенных, общее количество часов;
3) popular_games_view — топ-10 самых популярных игр по количеству игроков с указанием среднего рейтинга.

Для оптимизации часто выполняемых запросов созданы индексы на внешние ключи (например, на company_id в games), покрывающие индексы с include (например, idx_user_progress_user_hours для суммирования часов), индексы по полям сортировки (например, на last_updated) и частичные индексы (is_approved = true в reviews).

### 2.4 Описание API и взаимодействия с базой данных

Backend-приложение разработано на языке Python 3.12 с использованием фреймворка FastAPI, который обеспечивает высокую производительность, автоматическую генерацию документации API и простоту разработки. Для взаимодействия с базой данных применяется ORM SQLAlchemy, позволяющая работать с таблицами PostgreSQL как с объектами Python. Код моего backend-приложение состоит из следующих файлов:

4) main.py — точка входа, создание ядра FastAPI, подключение роутеров и создание таблиц при запуске;
5) database.py — конфигурация подключения к PostgreSQL через SQLAlchemy;
6) schemas.py — Pydantic-модели для ввода/вывода данных в API;
7) routers – отдельные роутеры для логической группировки эндпоинтов.

Моё приложение предоставляет следующие группы эндпоинтов:

1) /users — создание, получение списка и отдельного пользователя;
2) /games — создание, получение списка, детальной информации (с рейтингом и количеством отзывов), обновление и удаление игр;

3) /reviews — создание отзыва и получение отзывов по игре;
4) /views — доступ к агрегированным данным через представления (рейтинги игр, статистика пользователей, популярные игры);
5) /stats — аналитические эндпоинты на основе SQL-функций (рейтинг игры, часы пользователя, топ игроков по жанру, активность пользователей за период).

Реализованное backend-взаимодействие с базой данных сочетает преимущества ORM (удобство CRUD) и прямого SQL (удобно при работе с API функций), что полностью соответствует требованиям технического задания по интеграции БД с backend-приложением.

### 2.5 Анализ производительности

В качестве примера для анализа производительности будет использоваться запрос получения среднего рейтинга игры по дате и названию, так как в нем хорошо заметен прирост в скорости, который дает использование индексов. На рисунке 2.3 показан результат выполнения данного запроса до создания индексов в моей базе данных.

```
11                  ->  Nested Loop Left Join  (cost=0.56..1309.50 rows=1 width=129) (actual time=6.625..6.627 rows=0 loops=1)
12                      ->  Nested Loop  (cost=0.28..1101.25 rows=1 width=121) (actual time=6.623..6.625 rows=0 loops=1)
13                          ->  Seq Scan on games g  (cost=0.00..1092.93 rows=1 width=86) (actual time=6.622..6.622 rows=0 loops
14                                Filter: ((release_date > '2020-01-01'::date) AND ((title)::text ~~* '%game%'::text))
15                                Rows Removed by Filter: 7929
16                          ->  Index Scan using companies_pkey on companies c  (cost=0.28..8.29 rows=1 width=43) (never execut
17                                Index Cond: (company_id = g.developer_id)
18                      ->  Index Scan using reviews_user_id_game_id_key on reviews r  (cost=0.28..208.24 rows=1 width=12) (never
19                            Index Cond: (game_id = g.game_id)
20                            Filter: is_approved
21  Planning Time: 3.878 ms
22  Execution Time: 6.874 ms
```

Рисунок 2.3 — Результат выполнения запроса до создания индекса

Далее на рисунке 2.4 изображен результат работы запроса после создания индекса. Время выполнения запроса составило 1.1936 миллисекунд, что в 5.76 раз быстрее чем запрос без индекса. .


Рисунок 2.4 — Результат выполнения запроса после создания индекса

Из данного примера заметно, что индексы, используемые в проекте, могут ускорять выполнение основных операций в разы.

# 3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

## 3.1 Запуск проекта

Система поддерживает два варианта запуска, что делает её гибкой для разных сценариев разработки и демонстрации.

Первый тип запуска – полностью контейнеризированный и выполняется одной командой: docker compose up -d --build api. В этом режиме запускаются сразу оба контейнера: postgres и api. Поэтапно запуск происходит следующим образом:

1) сначала docker-compose собирает и запускает контейнер базы данных (postgres). Происходит быстрая сборка каталога ./database, при которой автоматически выполняются все SQL-скрипты инициализации таблиц, функций, триггеров, индексов и представлений;
2) после того, как healthcheck подтвердит готовность PostgreSQL базы данных к работе, запускается контейнер api (папка backend);
3) приложение становится доступным по адресу http://localhost:8000, а интерактивная документация FastAPI по адресу http://localhost:8000/docs.

Этот вариант запуска полностью изолирован: не требуется установка Python, pip или зависимостей на хостовой машине, кроме того, всё окружение будет работать одинаково на любой машине с любой операционной системой.

Второй тип запуска выполняется последовательностью команд: docker compose up -d postgres -> cd backend -> pip install -r requirements.txt -> cd ../ -> uvicorn backend.main:app --reload. В этом режиме база данных PostgreSQL запускается так же, как и в первом варианте запуска, однако backend-приложение запускается напрямую на хостовой машине с использованием uvicorn.

Данный вариант запуска позволяет использовать перезапуск сервера при помощи отладчика Python (--reload) без необходимости перезапуска контейнера api. Кроме того, такой вариант удобнее, если есть необходимость отладки кода, ведь при таком запуске все ошибки, происходящие не сервере будут сразу выводиться в консоль, без необходимости прописывания дополнительной команды.

### 3.2 Процесс заполнения данными

Для подготовки системы к тестированию и демонстрации после её сборки разработан скрипт populate_db.py. Скрипт использует библиотеку Faker для генерации реалистичных тестовых данных на русском языке. Процесс заполнения таблиц данными при помощи скрипта происходит в следующем порядке:

1) очистка всех таблиц для обеспечения воспроизводимости результатов работы API при повторных запусках;
2) отключение всех триггеров для ускорения заполнения таблиц;
3) последовательная генерация и вставка данных в таблицы с учетом зависимостей;
4) включение триггеров обратно.

Таким образом, скрипт обеспечивает требуемый объем реалистичных данных, а также позволяет быстро подготовить систему к тестированию API и аналитических функций,

### 3.3 Тестирование

Для тестирования работы веб-приложения используется пользовательский интерфейс под названием Swagger (рисунок 3.1). Разработка на FastAPI позволяет не задумываться о написании Swagger-документации, ведь этот фреймворк генерирует ее автоматически.

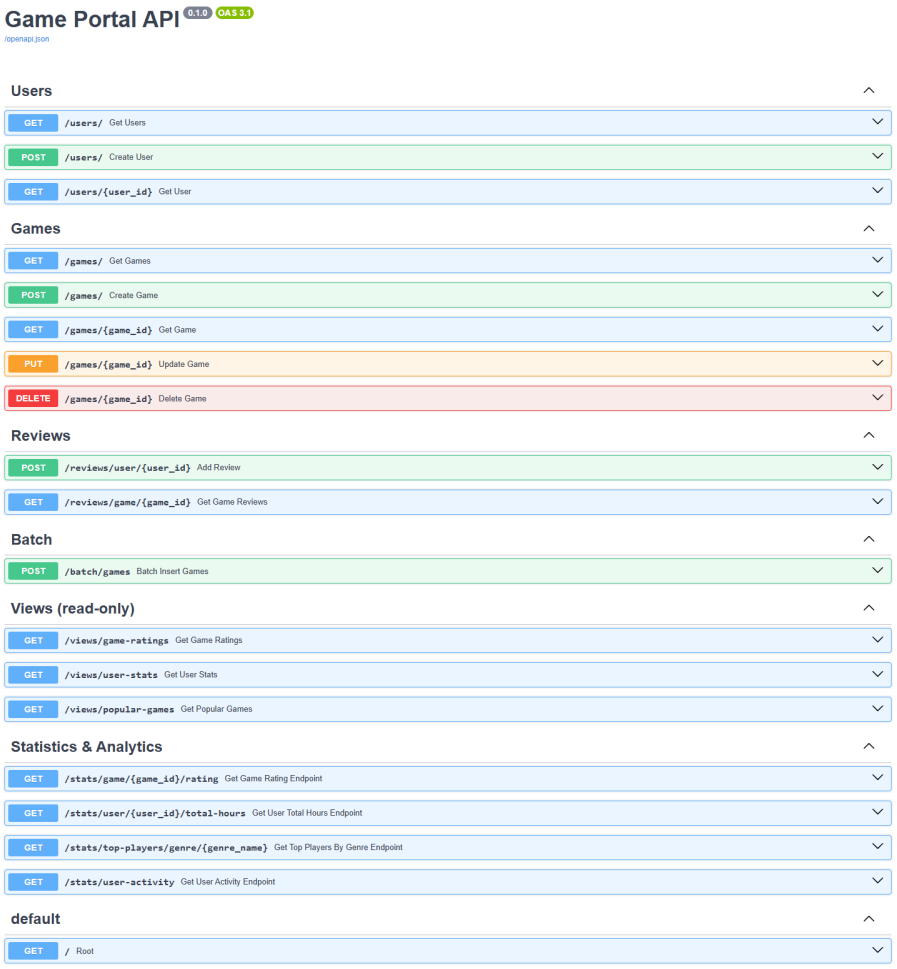


Рисунок 3.1 — Пример интерфейса Swagger

В качестве примера работы возьмем API для создания пользователя, проверив выполнение верно заданного запроса (рисунок 3.2). Результат имеет код ответа, а также текст ответа в формате JSON.


Рисунок 3.2 — Результат вызова метода из API с верными входными данными

Теперь попробуем выполнить тот же запрос, но с вводом неверного email пользователя (рисунок 3.3). Как видим, запрос не был выполнен в связи с ошибкой.

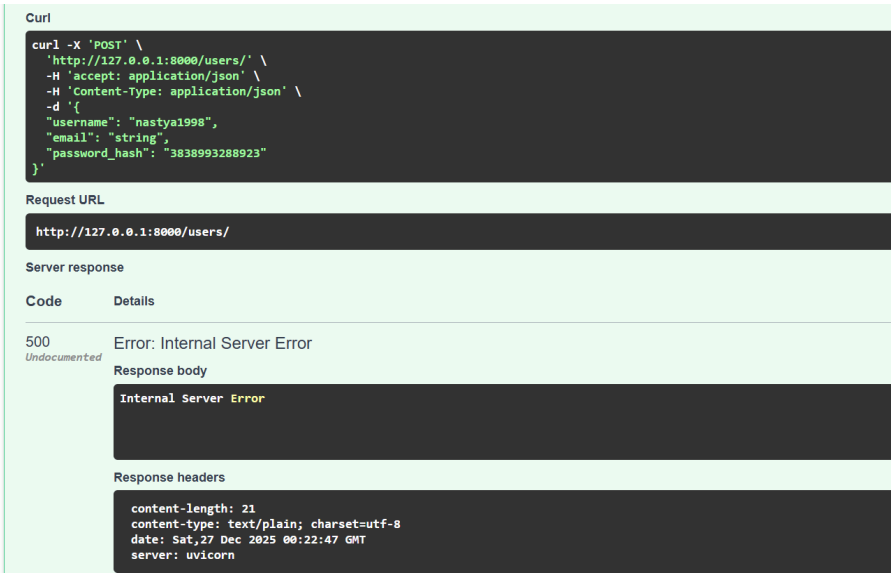


Рисунок 3.3 — Результат вызова метода из API с неверными входными данными

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была спроектирована и реализована информационная система «Игровой портал», представляющая собой функциональный backend-сервис с интеграцией базы данных PostgreSQL для учета игрового прогресса пользователей, хранения отзывов и предоставления аналитической статистики.

Все поставленные задачи были решены:

1) спроектирована структура реляционной базы данных, включающая 11 таблиц с настроенными ограничениями целостности и с использованием всех видов связи между таблицами. все таблицы находятся в третьей нормальной форме (за исключением небольших обоснованных денормализаций);
2) разработаны триггеры, для автоматического обновления агрегированных данных (средней оценки игр, количества отзывов по игре и часов, проведенных игроком в игре) и ведения журнала аудита;
3) созданы четыре SQL-функции для ведения основных аналитических расчетов, а также три представление для более точного восстановления агрегированных данных;
4) проведена оптимизация производительности системы за счет использования индексов и аналитики explain analyze;
5) реализован RESTful API при помощи фреймворка FastAPI с реализацией всех CRUD-операций, доступом к аналитике и батчевой загрузкой данных; API интегрирован с базой данных через ORM SQLAlchemy с использованием прямого SQL для сложных запросов;
6) обеспечена контейнеризация проекта с помощью Docker и docker-compose, включая два удобных варианта запуска.

Разработанная система демонстрирует высокую эффективность:

агрегатные данные доступны мгновенно благодаря денормализации и триггерам, аналитические запросы выполняются быстро за счёт индексов и представлений, а FastAPI предоставляет удобный и документированный интерфейс для взаимодействия в виде Swagger-документации.

Таким образом, цели курсовой работы достигнуты в полном объёме, а разработанная система полностью соответствует техническому заданию и может служить основой для дальнейшего расширения.