Реферат

Расчетно-пояснительная записка 60 с., 26 рис., 25 источников, 3 прил.

БАЗЫ ДАННЫХ, МОДЕЛИ БАЗ ДАННЫХ, РОЛЕВАЯ МОДЕЛЬ, ХАКАТОН, СЕРВЕР, API

Цель работы — разработка базы данных для хранения и обработки информации о хакатонах по направлению «Big Data». В процессе работы были изучены существующие модели баз данных, спроектирована собственная база данных, соответствующая поставленной цели, реализован сервер с API для доступа и работы с системой, проведено исследование времени обработки запроса в индексированной и неиндексированной таблице.

Содержание

Введение

Хакатон (от англ. hacker и marathon, «хакерский марафон») — соревнование для разработчиков, в котором команды на время создают прототип продукта, решающего ту или иную бизнес-задачу [1].

Хакатоны в современном мире IT являются, массовым мероприятием, в котором принимают участие десятки, а порой и сотни людей [2]. В связи с этим возникает необходимость наличия системы для хранения данных о мероприятии и его участниках, а также некоторой автоматизации их обслуживания.

В данной работе ставится цель создания базы данных для хакатонов по направлению «Big Data».

Для достижения поставленной цели, необходимо решить следующие задачи:

- проанализировать предметную область, выделить сущности и связи, которые должны моделироваться базой данных;
- проанализировать существующие модели данных;
- выбрать подходящую модель данных;
- формализовать задачу и определить необходимый функционал;
- описать структуру объектов БД, а также сделать выбор СУБД для ее хранения и взаимодействия;
- реализовать спроектированную БД;
- спроектировать и реализовать интерфейс для доступа к БД.

Итогом работы станет база данных с разработанным интерфейсом взаимодействия с ней с использованием различных ролевых моделей.

1 Аналитический раздел

1.1 Анализ существующих решений

Среди уже существующих проектов, решающих поставленную задачу, были выделены 3 аналога (таблица 1.1) – сервисы «HYPE Hackathon Solution» [3], «EventTornado» [4] и «Hackathon Manager» [5]. Данные приложения, позволяют создавать хакатоны, добавлять в них участников, а также получать подробную инфографику по их результатам. Сравнение (приведено в таблице 1.1) проводилось по ряду критериев: типу приложения, доступности в РФ, стоимости и наличию API.

Таблица 1.1 – Существующие решения поставленной задачи

Название проекта	**Тип приложения**	**Доступность в РФ**	**Стоимость**	**Наличие API**
HYPE Hackathon Solution	Desktop/Web	Нет	До 50 000$ в год в зависимости от конфигурации и числа пользователей	Да
Eventornado	Web	Нет	от 8$ за пользователя	Нет
Hackathon Manager	Web	Да	Бесплатно	Нет

Из сравнения видно, что «Eventornado» и «Hackathon Manager» не имеют API и не могут взаимодействовать с другими приложениями. «HYPE Hackathon Solution» же обладает относительно высокой стоимостью и недоступна на территории РФ. В данной работе была произведена попытка нивелировать недостатки конкурентов и в то же время совместить их преимущества.

1.2 Анализ предметной области

Как уже было сказано, хакатоны - это мероприятия, в которых программисты соревнуются в решении задач за ограниченное время. Каждое событие сфокусировано на конкретной области, такой как разработка приложения, API либо анализ данных. Основная часть хакатона - это время, которое участники проводят над своими проектами. Обычно это от 24 до 72 часов, в зависимости от конкретного хакатона. В это время участники работают в командах, создавая и развивая свои проекты в соответствии с заданием хакатона или своими собственными идеями [6].

Главной особенностью данного типа мероприятий является то, что по окончании соревнования, команды должны представить законченный программный продукт. Жюри, как правило, состоит из экспертов в области разработки, бизнеса или другой отрасли, связанной с тематикой хакатона. Жюри оценивает проекты, созданные участниками, и выбирает лучшие из них на основе определенных критериев, таких как инновационность, полезность, применимость, дизайн и технические навыки. Часто жюри также задает вопросы участникам, чтобы лучше понять их проекты и принять более обоснованные решения. Кроме того, в ходе хакатона участники могут получать обратную связь от жюри и других участников, что помогает им улучшить свои навыки и подготовиться к будущим мероприятиям [7].

1.3 Формализация задачи

В ходе выполнения курсовой работы необходимо разработать базу данных для хранения и обработки данных о проводимых хакатонах, командах, принимающих в них участие, а также непосредственно участниках. Кроме того, необходимо реализовать возможность участникам присоединяться к командам, организаторам хакатонов - создавать мероприятия. Также нужно добавить возможность отслеживать рейтинг команд и отдельных участников. Для каждой категории пользователей необходимо реализовать различный функционал и предоставить им определённый набор прав.

1.4 Формализация данных

В ходе анализа предметной области были выделены следующие сущности:

- хакатон, обладает следующими полями: id, название, дата проведения, тематика, id организатора, длительность и id команд, занявших 1-е, 2-е и 3-е место;
- команда, имеет поля: id, id капитана, число участников, название, рейтинг;
- участник, обладает полями: id, логин, пароль, имя и рейтинг;
- организатор, обладает полями id, имя, логин, пароль;
- администратор, имеет поля id, имя, логин, пароль;
- заявка, имеет поля: id, тип, статус, id подавшего, тип учетной записи подавшего, название хакатона/команды (в зависимости от типа заявки), поле для комментария.

Кроме того, были выделены следующие вспомогательные таблицы, содержащие лишь одно поле id:

- тип заявки, может принимать следующие значения:
  - 1 – создание команды;
  - 2 – создание хакатона;
  - 3 – вступление в команду;
  - 4 – участие в событии.
- тип учетной записи подавшего: 1 - участник, 2 - организатор;
- статус заявки:
  - 1 – открыта;
  - 2 – в работе;
  - 3 – принята;

- 4 – отклонена.

- тип учетной записи пользователя.

ER-диаграмма сущностей предметной области представлена на рисунке 1.1.

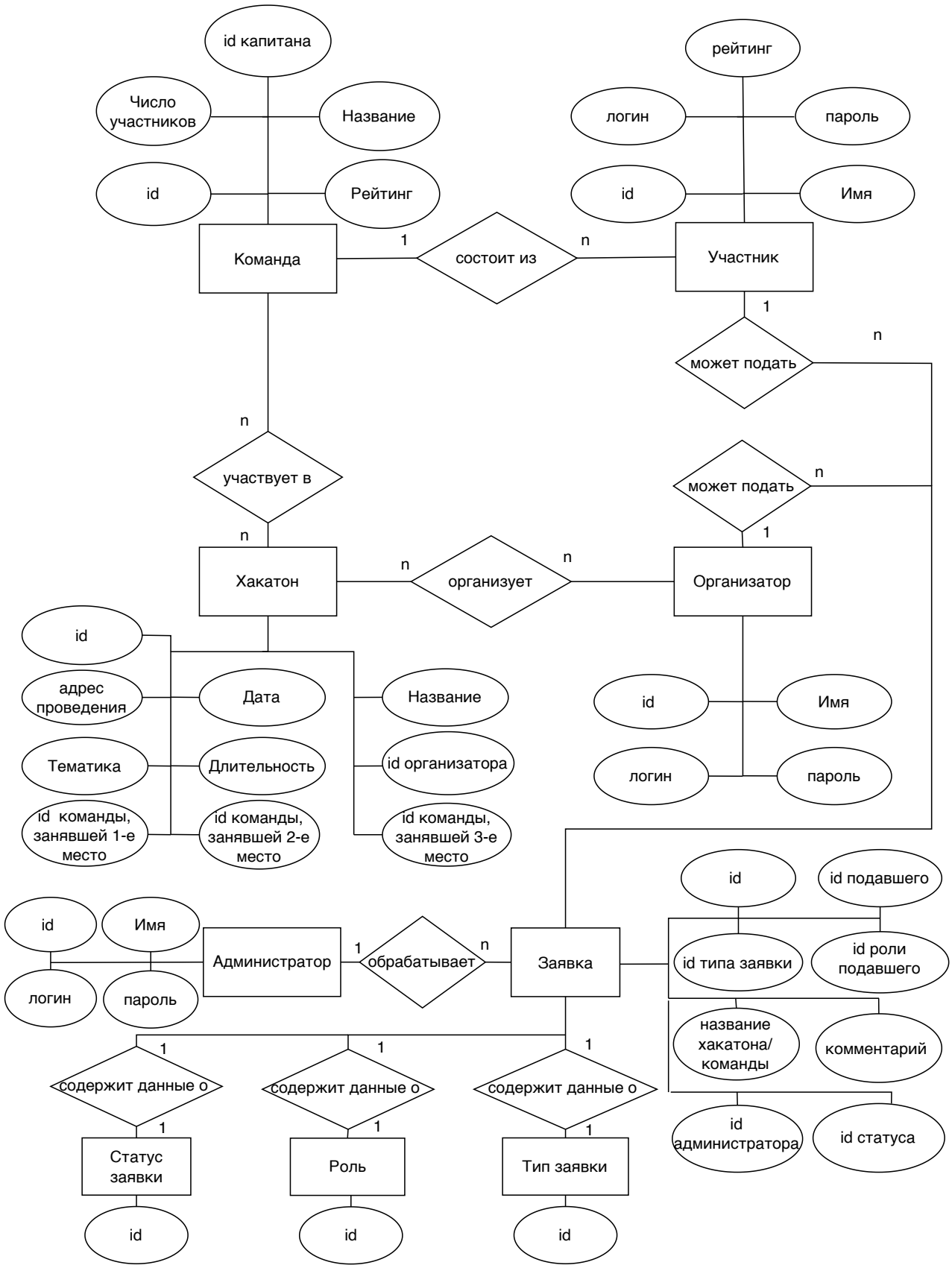

Рисунок 1.1 – ER-диаграмма сущностей предметной области

1.5 Функционал приложения

Приложение должно поддерживать следующий функционал:

- авторизация пользователей;
- создание новых заявок и изменение информации об уже существующих;
- создание организаторами хакатонов;
- создание участниками команд и добавление в неё других участников;
- просмотр пользователями списков команд и событий.

Для работы с системой обязательным этапом является прохождение авторизации. Пользователь может работать в системе под одной из следующих ролей:

1. Участник – пользователь, обладающий возможностями просмотра информации о событиях и командах, а также подачи заявок на создание команды или участие в команде.
2. Капитан команды – пользователь, обладающий возможностями добавления и удаления участников из команды и подачи заявки на участие команды в хакатоне.
3. Организатор – пользователь, обладающий возможностью создания хакатонов, приема заявок на участие команд в созданном им хакатоне, а также присвоения призовых мест командам.
4. Администратор – пользователь обладающий возможностью создавать и удалять заявки, удалять события, участников, а также просматривать информацию о событиях, участниках и командах.

На рисунках 1.2-1.5 представлена диаграмма использования приложения в соответствие с ролями пользователей.

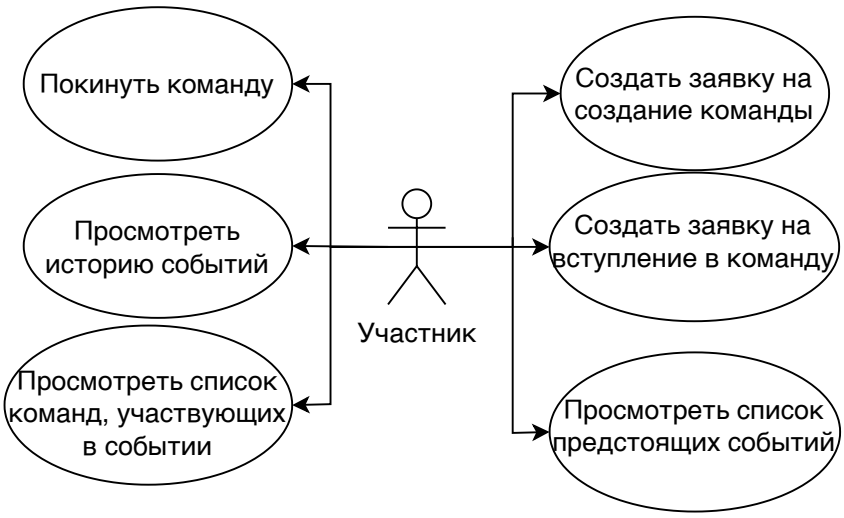

Рисунок 1.2 – Use-case диаграмма участника

Рисунок 1.3 – Use-case диаграмма капитана

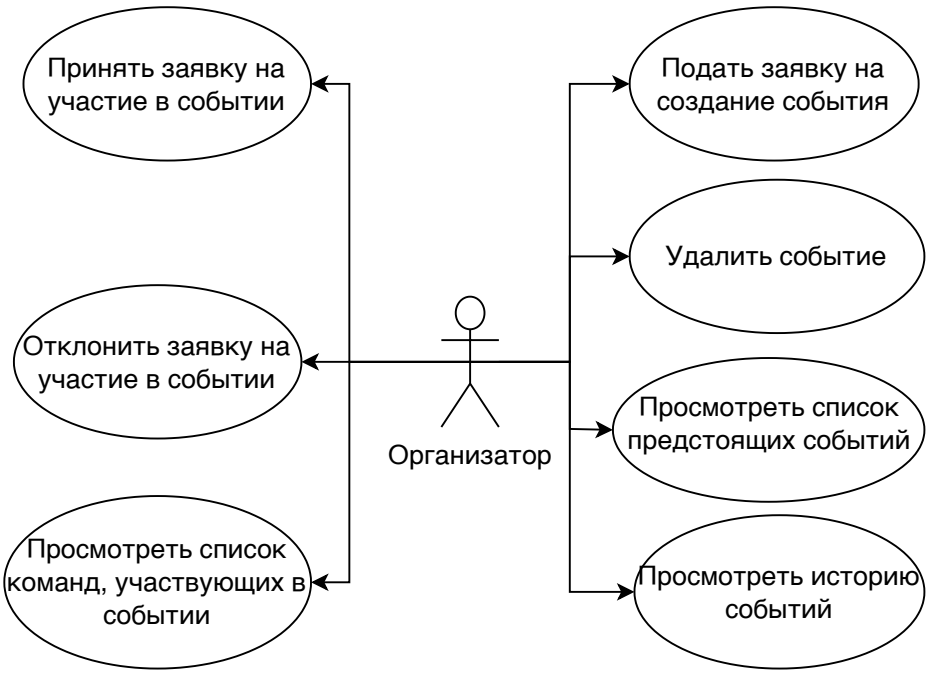

Рисунок 1.4 – Use-case диаграмма организатора

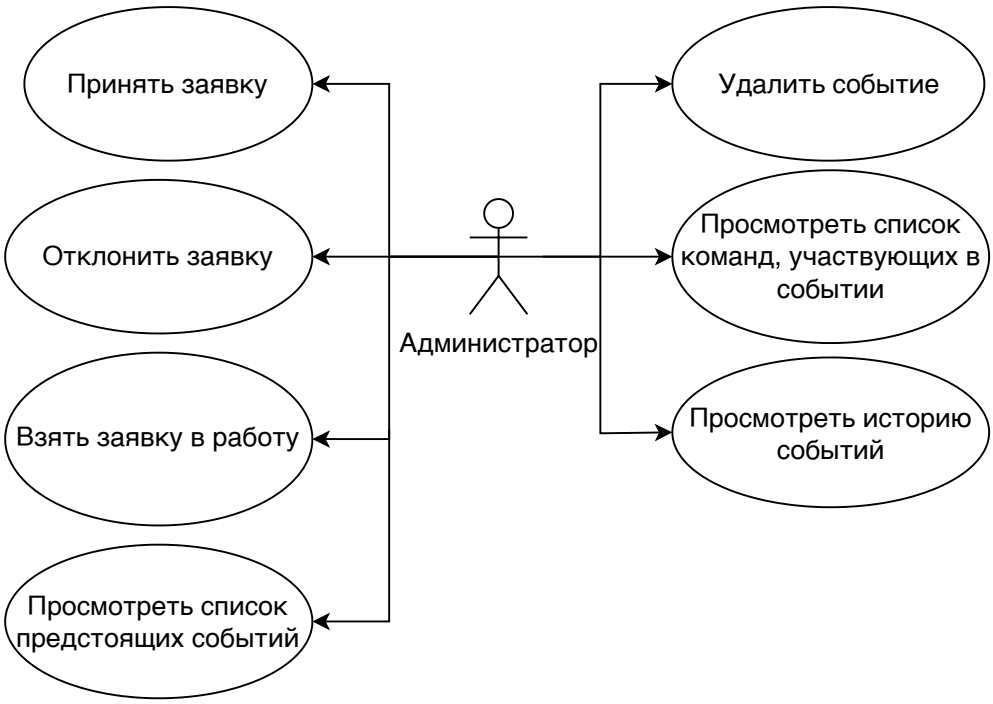

Рисунок 1.5 – Use-case диаграмма администратора

1.6 Анализ моделей баз данных

1.6.1 Иерархическая модель данных

Иерархическая модель данных подразумевает что элементы, организованные в структуры, объединены иерархической или древовидной связью. В таком представлении родительский элемент может иметь несколько дочерних, а дочерний – только один родительский.

Каждая вершина дерева соответствует типу сущности ПО. Конечные вершины, то есть вершины, из которых не выходит ни одной дуги, называются листьями дерева. Каждая некорневая вершина связана с родительской вершиной иерархическим групповым отношением. Тип сущности характеризуется произвольным количеством атрибутов, связанных с ней отношением 1:1. Атрибуты, связанные с сущностью отношением 1:n, образуют отдельную сущность (сегмент) и переносятся на следующий уровень иерархии. Реализация связей типа n:m не поддерживается.

Существенным недостатком такой модели является то, что каждая сущность может относится только к одному родительскому сегменту. Таким образом при необходимости множественного отношения происходит дублирование данных, что может привести к нарушению логической целостности БД [8].

1.6.2 Сетевая модель данных

Сетевая модель базы данных подразумевает, что у родительского элемента может быть несколько потомков, а у дочернего элемента – несколько предков. Структура такой модели представлена в виде графа, причем каждая вершина графа хранит экземпляры сущностей (записи одного типа) и сведения о групповых отношениях с сущностями других типов. Каждая запись может хранить произвольное количество значений атрибутов (элементов данных и агрегатов), характеризующих экземпляр сущности. Для каждого типа записи выделяется первичный ключ – атрибут, значение которого позволяет однозначно идентифицировать запись среди экземпляров записей данного типа.

Связи между записями выполняются в виде указателей, т.е. каждая запись хранит ссылку на другую однотипную запись (или признак конца списка) и ссылки на списки подчинённых записей, связанных с ней групповыми отношениями. Таким образом, в каждой вершине записи хранятся в виде связного списка.

В этой модели связь 1:n между сущностями реализуется с помощью групповых отношений, а связи 1:n между атрибутами сущности – в рамках записи. Для реализации связей типа n:m вводится вспомогательный тип записи и две связи 1:n.

Физическая независимость не обеспечивается в сетевой модели данных, потому что наборы организованы с помощью физических ссылок. Помимо этого данная модель не обеспечивает независимость данных от программ. Эти недостатки помешали обрести ей широкое распространение из-за сложного проектирования и поддержки [8].

1.6.3 Реляционная модель данных

Реляционная модель данных является самой распространенной в использовании. В отличие от вышеописанных, в данной модели не существует физических отношений между сущностями. Хранение информации осуществляется в виде таблиц (отношений), состоящих из рядов и столбцов. Отношение имеет имя, которое отличает его от имён всех других отношений. Атрибутам реляционного отношения назначаются имена, уникальные в рамках отношения. Обращение к отношению происходит по его имени, а к атрибуту – по именам отношения и атрибута.

В реляционных моделях данных нет необходимости просматривать все указатели, что облегчает выполнение запросов на выборку информации по сравнению с сетевыми и иерархическими моделями. Это означает, что порядок записей не является важным, также как и порядок столбцов.

Каждая запись идентифицируется уникальной комбинацией атрибутов – первичным ключом. Для связи между отношениями используется внешний ключ – копия первичного или уникального ключа атрибута сторонней сущности. Таким образом можно реализовывать связь 1:n. Для обеспечения связи

n:m вводится дополнительная сущность, атрибутами которой являются внешние ключи соответствующих сущностей.

Обработка данных осуществляется с помощью декларативного языка запросов SQL [9].

1.6.4 Многомерная модель

Многомерная модель представляет данные как многомерные массивы (гиперкубы), использование которых позволяет получать различные срезы при аналитической обработке данных. Осями многомерной системы координат служат основные атрибуты анализируемого бизнес-процесса. На пересечениях осей измерений находятся данные, количественно характеризующие процесс, — меры [10].

1.6.5 Объектно-ориентированная модель

Данная модель представляет из себя структуру, которую графически можно изобразить в виде дерева, узлами которого являются объекты. Базовыми понятиями являются: объекты, классы, методы, наследование, полиморфизм, инкапсуляция.

Спецификой объектно-ориентированной модели является то, что она поддерживает множественные группы, называемые ассоциированными множественными полями, а совокупность объединенных множественных полей называется ассоциацией. В объектно-ориентированной модели не накладываются требования на длину и количество полей в записях, что делает структуру таблиц более наглядной. Таким образом, основным достоинством данной модели является возможность представления совокупности связанных реляционных таблиц в виде одной объектно-ориентированной таблицы. А недостатком является сложность обеспечения целостности и непротиворечивости данных, хранимых в ней [10].

1.7 Выбор модели базы данных

В данном проекте будет использоваться реляционная модель данных, потому что в рамках проекта она обладает следующими преимуществами:

- изложение информации осуществляется с помощью простых и понятных форм (таблиц);
- позволяет работать со структурированными данными, структура которых не подвергается частым изменениям;
- имеет возможность произвольного доступа к записям сущностей;
- исключает дублирование, реализуя связь между отношениями посредством внешнего ключа.

1.8 Вывод из раздела

В данном разделе был проведён анализ предметной области, построена её модель в виде ER-диаграммы, выделены ролевые модели системы, конкретизированы хранимые данные и их связь между собой, построены соответствующие диаграммы. Также был проведен анализ существующих на рынке решений, который позволил понять, какие особенности стоит добавить в разрабатываемый проект. Был осуществлен выбор модели базы данных.

2 Конструкторский раздел

2.1 Формализация сущностей системы

На основе выделенных ранее сущностей спроектированы следующие таблицы базы данных.

Таблица Users содержит данные о пользователях системы. Включает следующие поля:

- id – целое число, идентификатор пользователя;
- name – строка, имя пользователя;
- login, password – строки, логин и пароль пользователя;
- role – целое число, роль пользователя ссылается на таблицу Roles.

Таблица Ratings содержит информацию, присущую лишь участникам и капитанам команд. Включает в себя поля part_id – целое число, идентификатор пользователя из таблицы Users, и rating – целое число, рейтинг пользователя.

Таблица Hackathons – содержит информацию о хакатонах. Включает следующие поля:

- id – целое число, идентификатор хакатона;
- name – строка, название хакатона;
- address – строка, адрес места проведения хакатона;
- date – тип «Дата и время», дата проведения хакатона;
- theme – строка, тематика хакатона;
- duration – вещественное число, продолжительность хакатона в часах.
- id_first, id_second, id_third – целые числа, id команд занявших первое, второе и третье место соответственно.

Таблица Requests – содержит информацию о заявках системы. Включает следующие поля:

– id – целое число, идентификатор заявки;

– type – целое число, тип заявки;

– requester_id – id пользователя, подавшего заявку;

– name – название команды / мероприятия, опциональное поле;

– comment – комментарий заявки;

– adm_id – id администратора, закрывшего заявку;

– status – статус заявки.

Таблица Teams – содержит информацию о командах. Включает следующие поля:

– id – целое число, идентификатор команды;

– name – строка, название команды;

– rating – целое число, рейтинг команды - является суммой рейтингов всех участников команды;

– cap_id – целое число, id капитана команды.

Кроме того, были спроектированы следующие вспомогательные таблицы:

– таблица Statuses, содержащая идентификаторы возможных статусов заявок, в которой записи полями id, принимающими значения 1, 2, 3, 4 обозначают статусы «Открыта», «В работе», «Принята» и «Отклонена» соответственно;

– таблица RequestTypes, содержащая идентификаторы типов заявок, содержит записи со следующими id 1, 2, 3, 4, обозначающими "Создание команды "Создание события "Присоединение к команде "Присоединение к событию";

– таблица Roles, содержащая идентификаторы возможных ролей пользователей.

Соответствующая диаграмма по описанным выше данным представлена на рисунке 2.1.

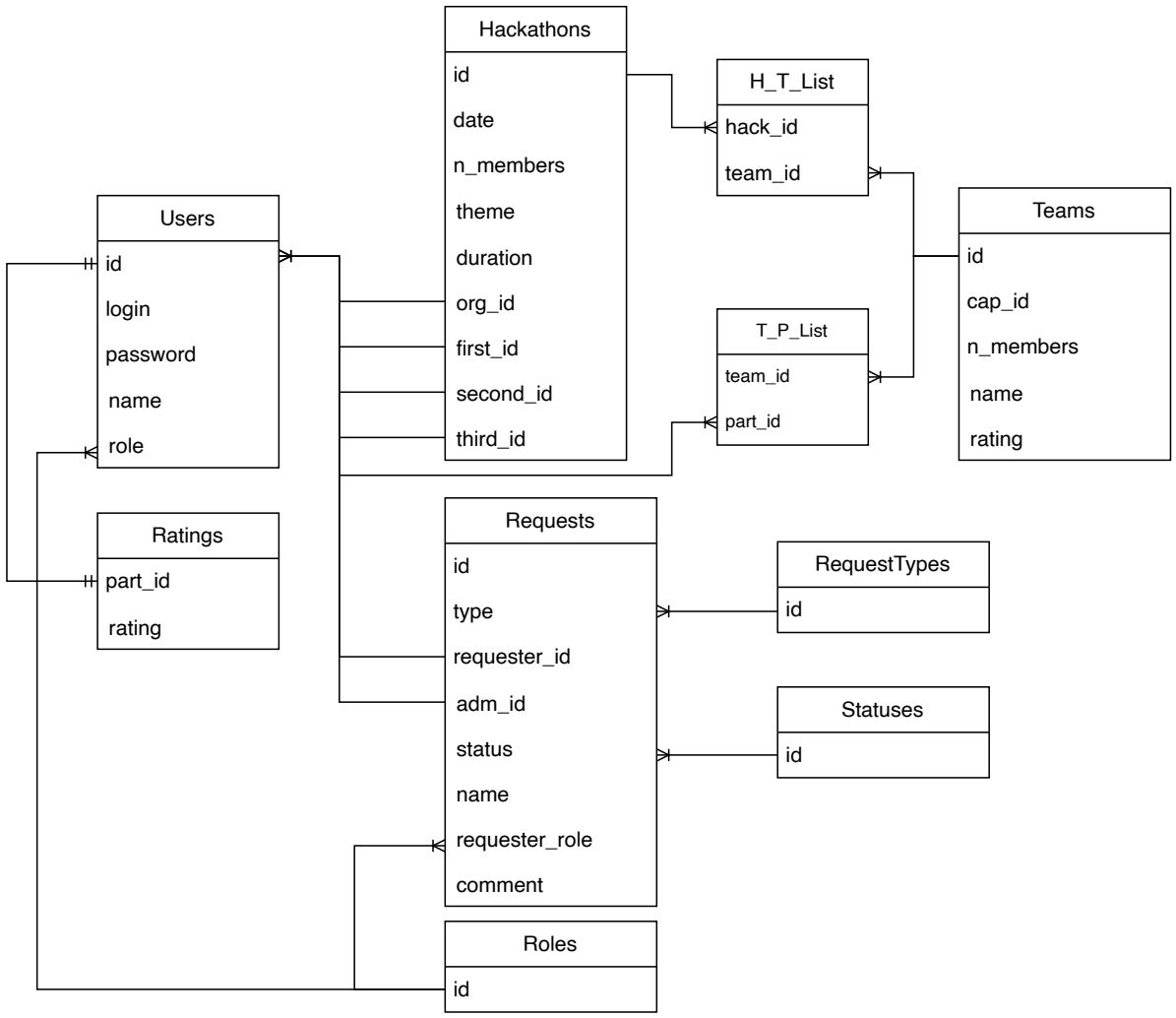

Рисунок 2.1 – Схема базы данных

2.2 Ролевая модель

Для обеспечения корректной работы пользователей с системой на уровне базы данных выделена следующая ролевая модель.

1. Participant – участник хакатона. Обладает правом SELECT над таблицами Users, Teams, Hackathons, H_T_List и T_P_List, Ratings, а также правом INSERT/UPDATE над таблицей Requests.

2. Captain – капитан команды, обладает всеми правами пользователя Participant и правами UPDATE над таблицей Teams.

3. Organizer – организатор хакатона. Обладает всеми правами пользователя Participant, а также правами UPDATE/DELETE над таблицей Hackathons.

4. Administrator – администратор. обладает всеми правами над всеми таблицами.

Использование ролевой модели на уровне базы данных гарантирует безопасность доступа к объектам.

2.3 Триггер

В системе предусмотрен механизм обновления рейтинга участников и команд, в которых они состоят. Он реализован также триггером AFTER на действие UPDATE в таблицу Hackathons. Данный триггер при обновлении записи хакатона, увеличивает рейтинг участников команд, занявших первое, второе и третье место на 3, 2 и 1 соответственно. Кроме того, обновляется рейтинг команд, в которых состоят данные участнике

На рисунке 2.2 представлена схема алгоритма работы выполняемой триггером функции updateRating().

Рисунок 2.2 – Триггер AFTER для обновления рейтинга

2.4 Ограничения целостности базы данных

В ходе проектирования базы данных были выделены следующие ограничения:

- логин и пароль пользователя должны содержать не больше 35 символов;
- рейтинг пользователя/команды - не может быть отрицательным;
- каждый участник может состоять только в одной команде;
- дата проведения добавляемого хакатона - не ранее завтрашнего дня от текущей даты;
- число участников в команде - не более пяти человек, включая капитана;
- число записей во вспомогательных таблицах Statuses, RequestTypes и Roles не может быть больше числа соответствующих сущностей.

2.5 Вывод из раздела

В данном разделе была произведена формализация сущностей системы с последующим составлением ролевой модели и выделением пользовательских прав, спроектированы основные и вспомогательные таблицы, построена ER-диаграмма базы данных, спроектирован триггер AFTER, обновляющий рейтинг участников и команд после обновления призовых мест хакатона, а также выделены ограничения целостности базы данных.

3 Технологический раздел

3.1 Средства реализации

При реализации проекта будет использована PostgreSQL[11], поскольку эта СУБД является реляционной и обладает достаточным набором инструментов для решения поставленной задачи.

В качестве используемого языка программирования выбран Python[12], по причине его тесной интеграции с PostgreSQL, простоте синтаксиса и высокой скорости развёртывания REST-приложений на нём.

Для реализации REST API был выбран фреймворк Flask[13], так как он достаточно прост в освоении и позволяет быстро создать полноценное REST-приложение.

Для реализации ORM[14] использовался инструментарий SQLAlchemy [15] по причине его тесной интеграции с Flask. Для тестирования приложения была выбрана утилита Postman[16], потому что она обладает графическим интерфейсом и в то же время предоставляет широкий набор возможностей для тестирования API приложения.

3.2 Создание базы данных

В соответствии с выбранной СУБД и спроектированной базой данных было осуществлено создание БД и ее сущностей. Реализация представлена в приложении А, листинг 4.1.

В предыдущем разделе был спроектирован триггер AFTER на создание новой заявки в системе. Код его создания представлен в листинге 3.1

Листинг 3.1 – Реализация триггера AFTER

```
1 create trigger updateRatingTrigger
2     after update of id_first, id_second, id_third
3     on hackathons
4     for each row
5 execute function updateRating();
```

Для этого триггера была написана соответствующая функция с помощью PL/pgSQL [17] – процедурного расширения языка SQL, используемого в СУБД PostgreSQL. Код функции представлен в листинге 3.2.

Листинг 3.2 – Реализация функции updateRating()

```
1
2  create or replace function updateRating()
3      returns trigger as
4  $$
5  begin
6      call updateTeamParticipantsRating(new.id_first, 3);
7      call updateTeamParticipantsRating(new.id_second, 2);
8      call updateTeamParticipantsRating(new.id_third, 1);
9
10     call updateTeamRating(new.id_first);
11     call updateTeamRating(new.id_second);
12     call updateTeamRating(new.id_third);
13
14 end
15 $$
16     language plpgsql;
17
18 create function updateTeamParticipantsRating(arg_team_id integer,
19 rating_change integer) returns void
20     language plpgsql
21 as
22 $$
23 begin
24     update ratings
25     set rating = rating + rating_change
26     where id = (select id from t_p_list where team_id = arg_team_id);
27 end
28 $$;
29
30 create function updateteamrating(arg_team_id integer) returns void
31     language plpgsql
32 as
33 $$
34 begin
35     update teams
36     set rating = (select sum(rating)
37                   from ratings
38                        join t_p_list on ratings.part_id = t_p_list.part_id
39                   where t_p_list.team_id = arg_team_id)
40     where id = arg_team_id;
41 end;
42 $$;
```

3.3 Создание ролей и выделение им прав

В конструкторском разделе была разработана ролевая модель, в которой выделены следующие роли:

– Participant – участник;

– Captain – капитан;

– Organizer – организатор;

– Administrator – администратор.

Соответствующий этой ролевой модели сценарий создания ролей и выделения им прав представлен на листинге 3.3.

Листинг 3.3 – Создание ролей и выделение им прав

```
1 create user "admin" password 'admin';
2 create user "organizer" password 'organizer';
3 create user "captain" password 'captain';
4 create user "participant" password 'participant';
5
6 grant select on teams, hackathons, ratings,
7 h_t_list, t_p_list to "participant";
8 grant insert on requests to "participant";
9
10 grant select on teams, hackathons, ratings,
11 h_t_list, t_p_list to "captain";
12 grant insert on requests to "captain";
13 grant update on Teams to "captain";
14
15 grant select on teams, hackathons, ratings,
16 h_t_list, t_p_list to "organizer";
17 grant insert on requests to "organizer";
18 grant update, delete on Hackathons to "organizer";
19
20 alter role "admin" superuser;
```

3.4 Наполнение базы данных

В ходе генерации данных было создано 5 пользователей типа "Администратор 100 типа "Организатор"и 1000 типа "Участник". Было создано 200

команд, по которым случайным образом были распределены участники и назначен капитан. Кроме того, было создано 100 хакатонов, команды-участники были назначены случайно. Значения полей таблиц были получены случайным образом в соответствии с описанными типами данных.

Выгрузка значений из электронных таблиц в базу данных представлен в приложении А, листинг 4.4.

3.5 Описание интерфейса приложения

Приложение разрабатывалось как микросервис, доступ к которому происходит при помощи RestAPI [18]. Схема приложения представлена на рисунке 3.1.

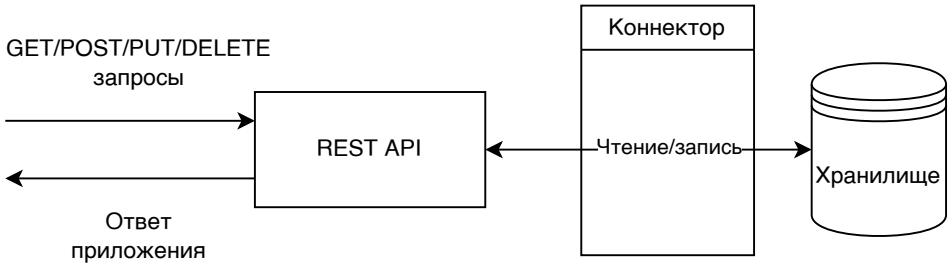

Рисунок 3.1 – Схема приложения

Для доступа к приложению был создан ряд HTTP-запросы, полный список которых содержится в приложении Б. В данном разделе приведены некоторые из них:

- POST («/login») – авторизация в приложении. При этом логин и пароль передаются в заголовке к запросу согласно методу BasicAuth[19];
- GET («/hackathons») – получить данные о всех хакатонах;
- GET («/hackathon/<id> ») – получить данные о хакатоне с идентификатором id;
- GET («/hackathons/<id>/teams») – получить данные о командах, участвующих в хакатоне;

– DELETE («/hackathons/<id> ») – удалить хакатон. Доступно только администратору и создателю хакатона;

– GET («/teams/places_available») – получить список команд со свободными местами;

– GET («/teams/<id>/participants») – получить список участников команды;

– POST («/teams») – создать команду. Доступно только администратору;

– POST («/requests») – подать заявку.

POST запросы требуют наличия у них тела в формате JSON[20]. Ответ возвращается также в данном формате.

3.6 Примеры работы

В качестве демонстрации примеров работы были выбраны следующие пользовательские сценарии:

– регистрация пользователя;

– авторизация пользователя;

– создание заявки на создание команды;

– одобрение заявки администратором;

– получение данных о будущих хакатонах.

Примеры запросов по данным сценариям и результаты их обработки представлены на рисунках 3.2 - 3.6

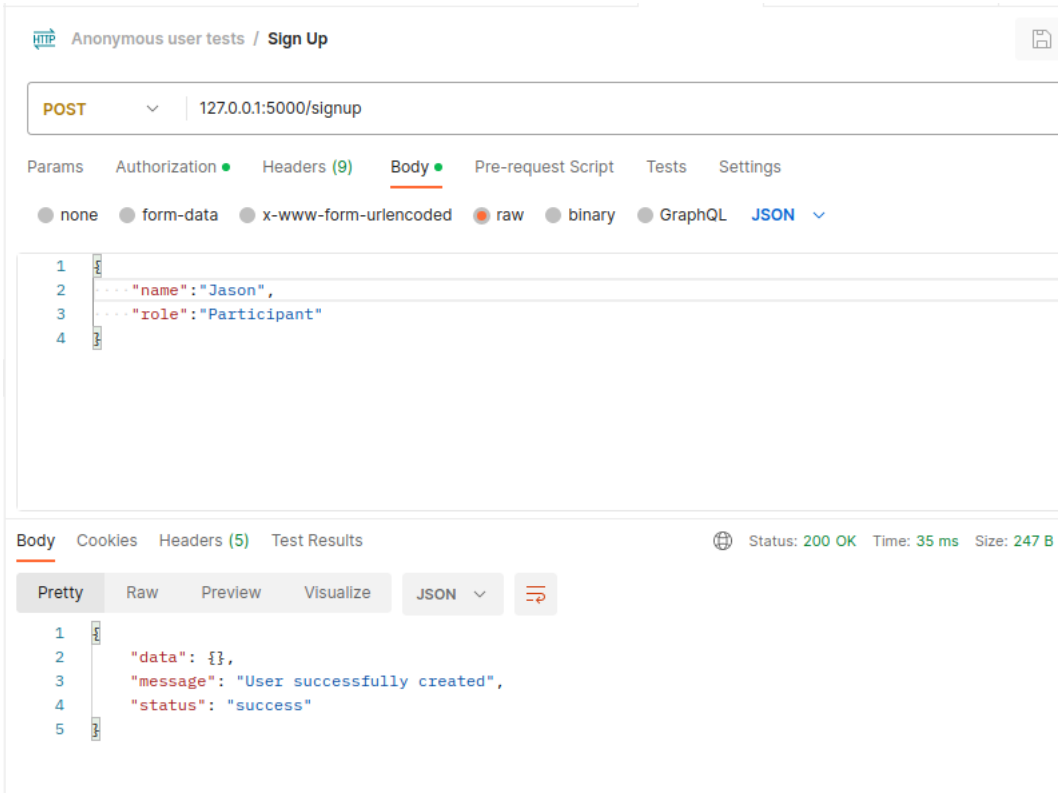

Рисунок 3.2 – Регистрация пользователя

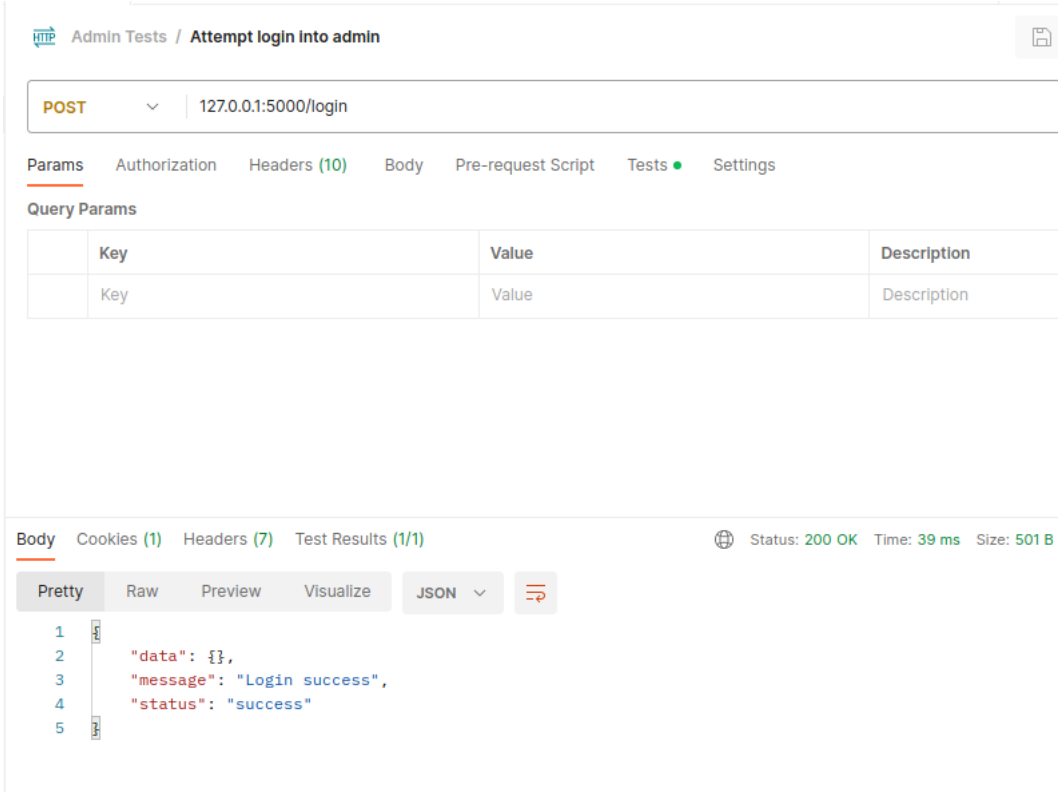

Рисунок 3.3 – Авторизация пользователя

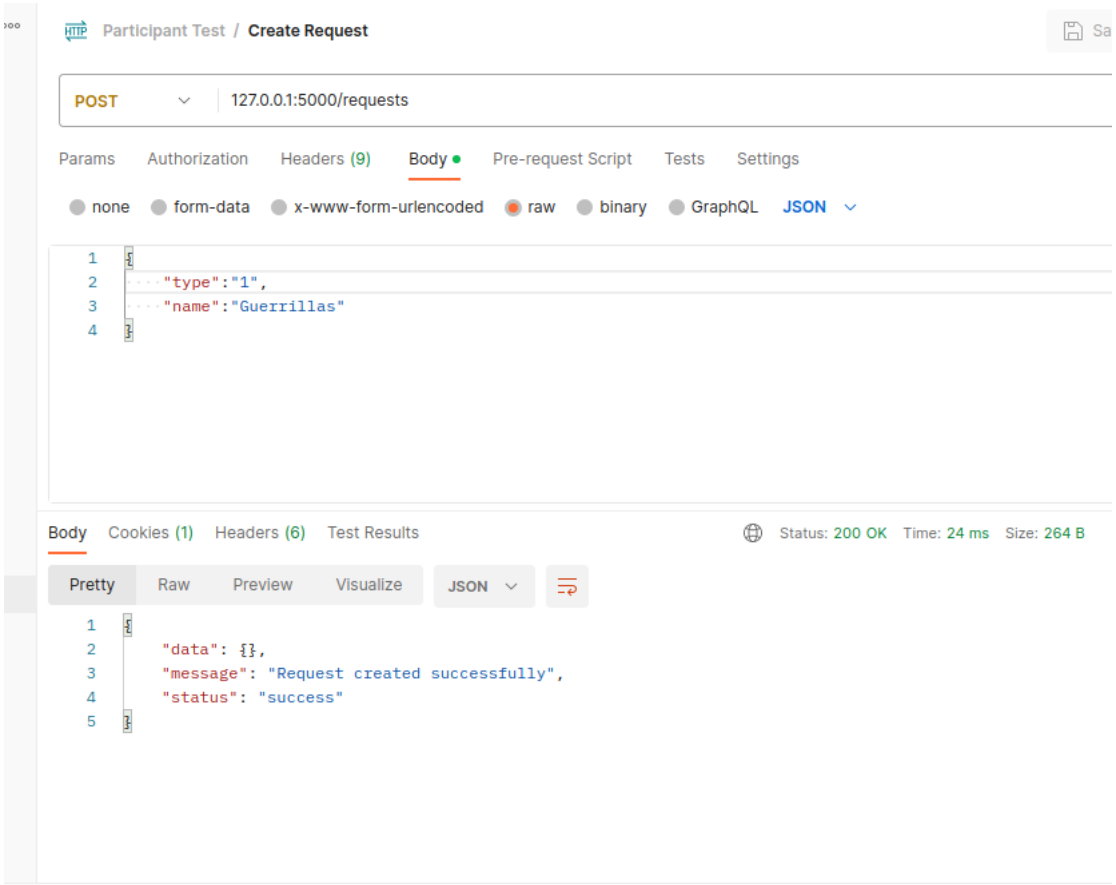

Рисунок 3.4 – Создание заявки на создание команды

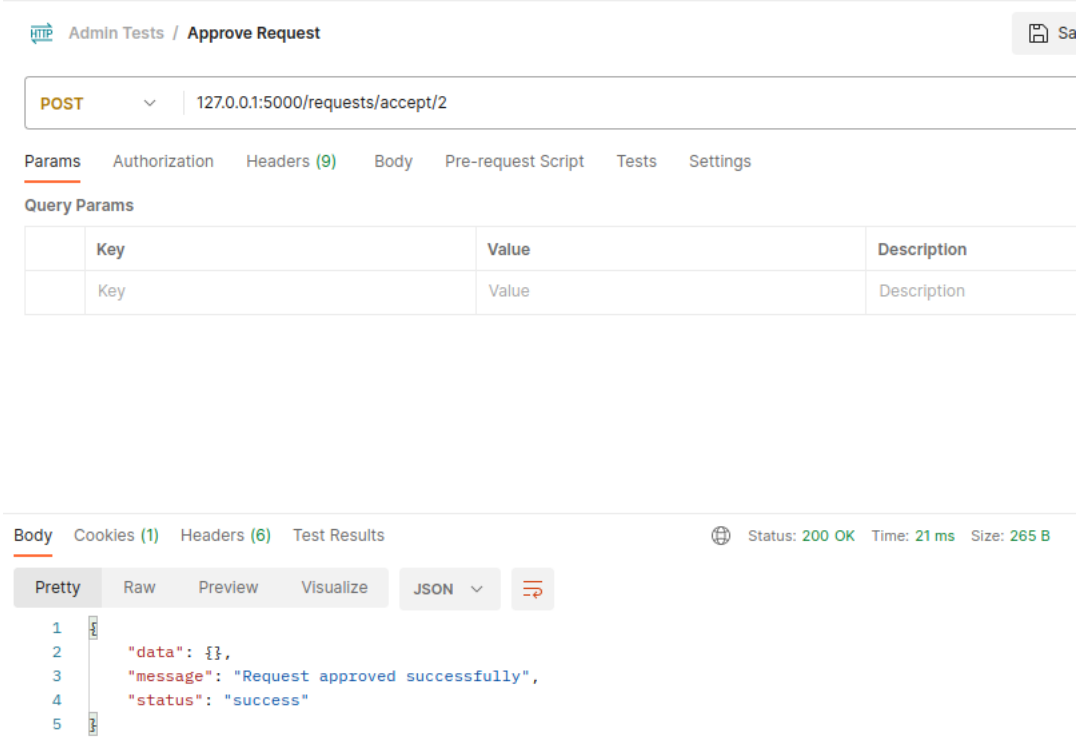

Рисунок 3.5 – Одобрение заявки администратором

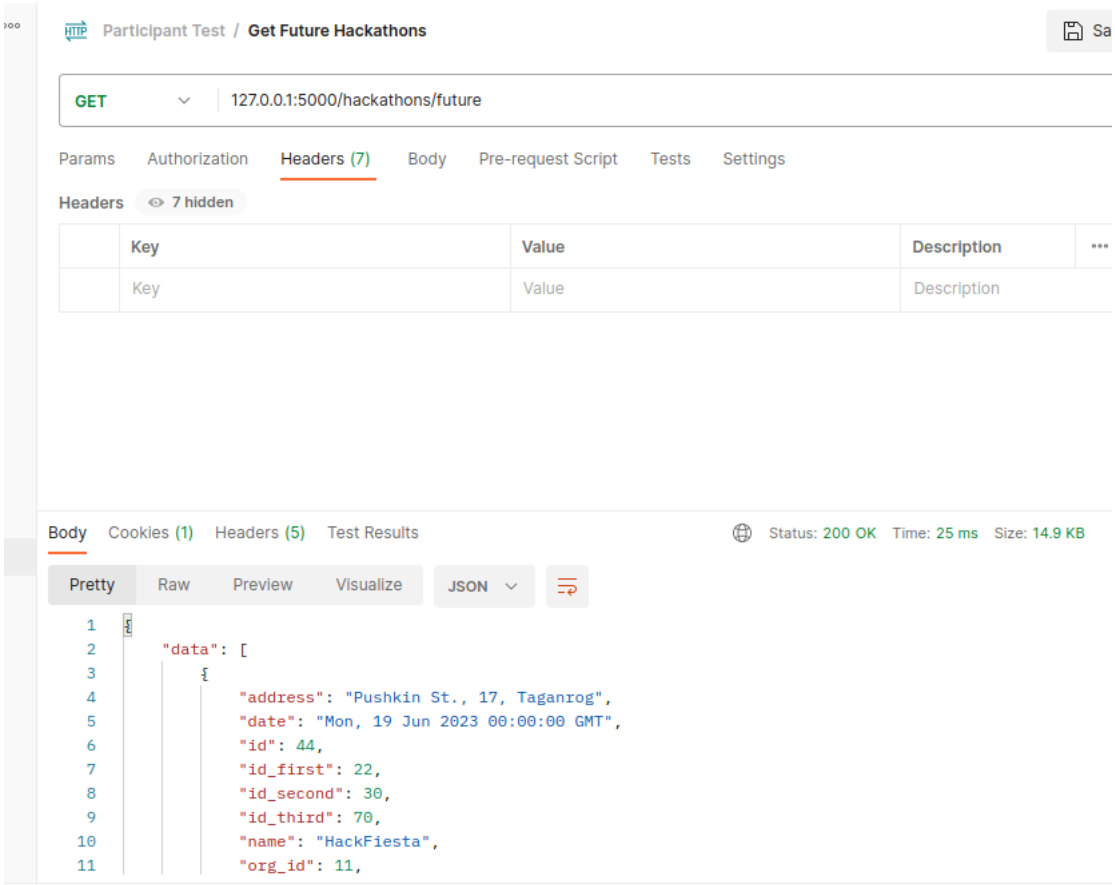

Рисунок 3.6 – Получение данных о будущих хакатонах

3.7 Нагрузочное тестирование

В данном подразделе было выполнено нагрузочное тестирование для оценки быстродействия разработанного приложения при различном числе одновременно посылающих запросы пользователей.

Были исследованы следующие пользовательские сценарии:

1) получение данных о будущих хакатонах;
2) получение данных о всех хакатонах;
3) получение списка команд со свободными местами, данных о будущих хакатонах и отправка заявки на вступление в команду.

Нагрузочное тестирование проводилось следующим образом: на протяжении 400 секунд число пользователей равномерно увеличивалось от 1 до 400

(то есть, каждую секунду подключался один дополнительный пользователь). Пользователи отправляли сервер

Сервер во время обработки запросов был запущен на ПК со следующими характеристиками:

- RAM: 8 Гбайт DDR4;
- ЦПУ: AMD Ryzen 5 3500U [21];
- ОС: Ubuntu 20.04 [22].

При проведении тестирования на ПК были запущены только сервисы ОС, сервер приложения и ПО для проведения тестирования – Apache JMeter 5.5 [23]. Во время тестирования ПК был подключен к сети электропитания.

3.7.1 Сценарий №1

Данный сценарий состоит из следующих запросов:

1) POST («/login»);
2) GET («/hackathons/future»);
3) POST («/logout»).

На графике 3.11 показаны результаты тестирования по сценарию №1 на количестве пользователей от 1 до 100, а на 3.12 – от 1 до 400.

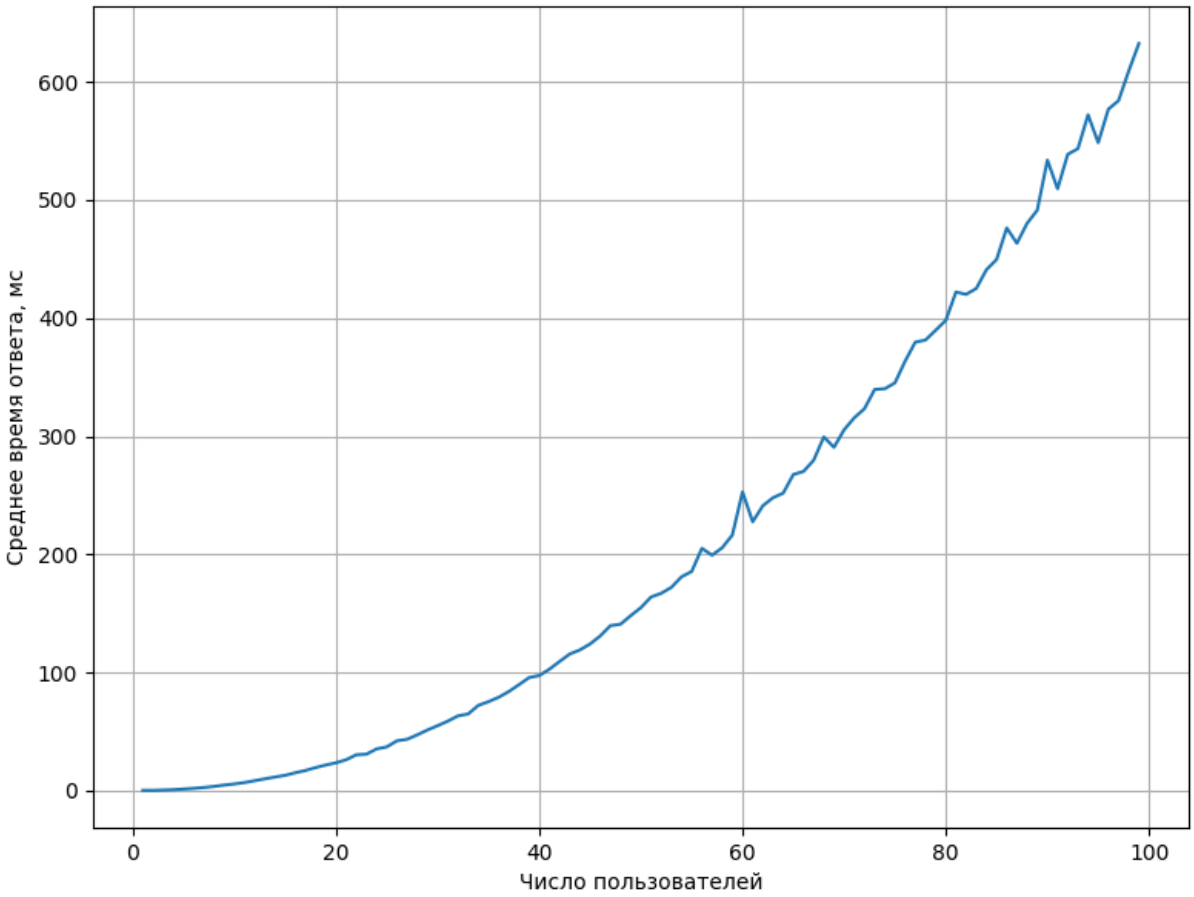

Рисунок 3.7 – Тестирование по сценарию №1, 100 пользователей

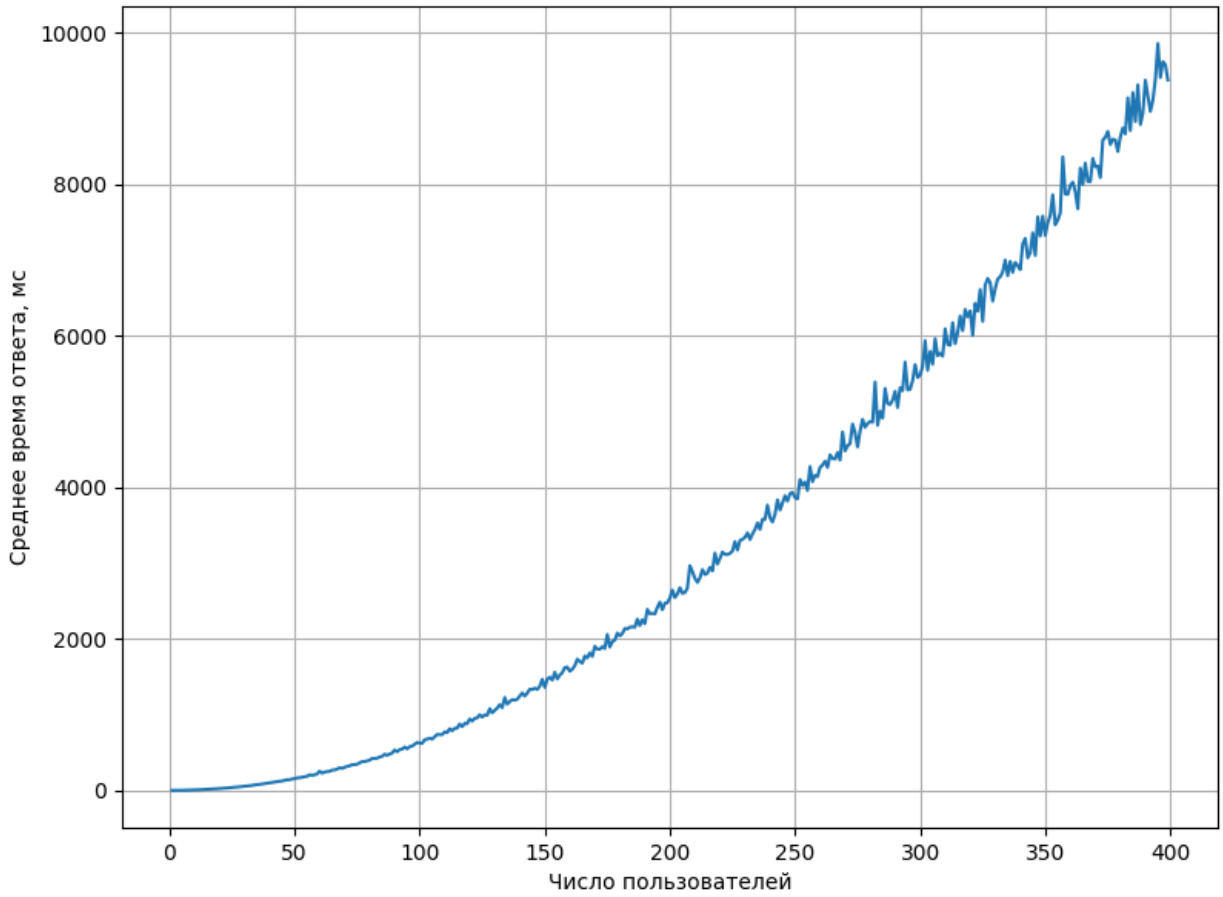

Рисунок 3.8 – Тестирование по сценарию №1, 400 пользователей

Из графиков видно, что при числе пользователей до 100 среднее время выполнения сценария не превышает 700 мс, однако уже при 170 пользователях время ожидания ответа приближается к 2 секундам и увеличивается вплоть до 10 секунд при 400 пользователях.

3.7.2 Сценарий №2

В ходе пользовательского сценария №2 выполняются следующие запросы:

1) POST («/login»);
2) GET («/hackathons>);
3) GET («/hackathons/future>);
4) GET («/teams»);

5) POST («/logout»).

Результаты тестирования представлены на рисунках 3.9 и 3.10.

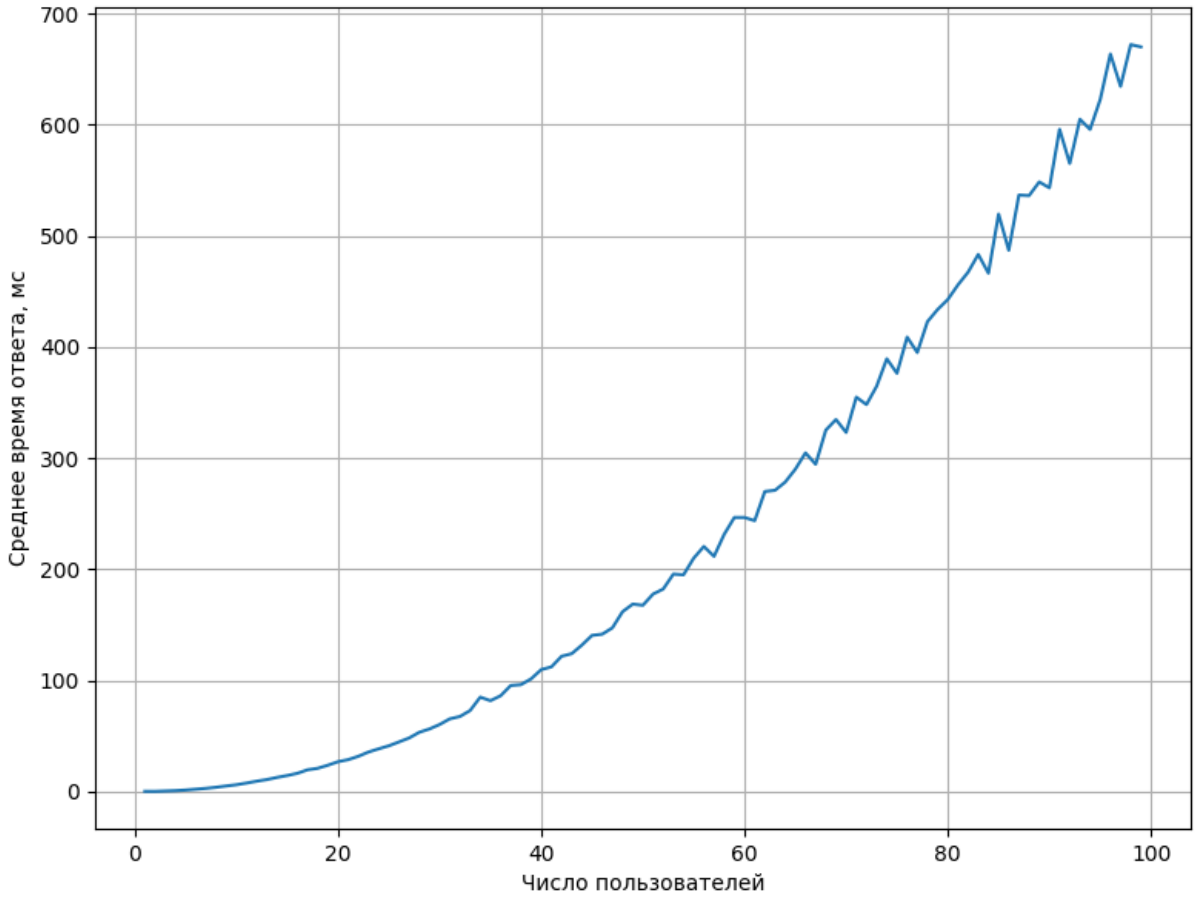

Рисунок 3.9 – Тестирование по сценарию №2, 100 пользователей

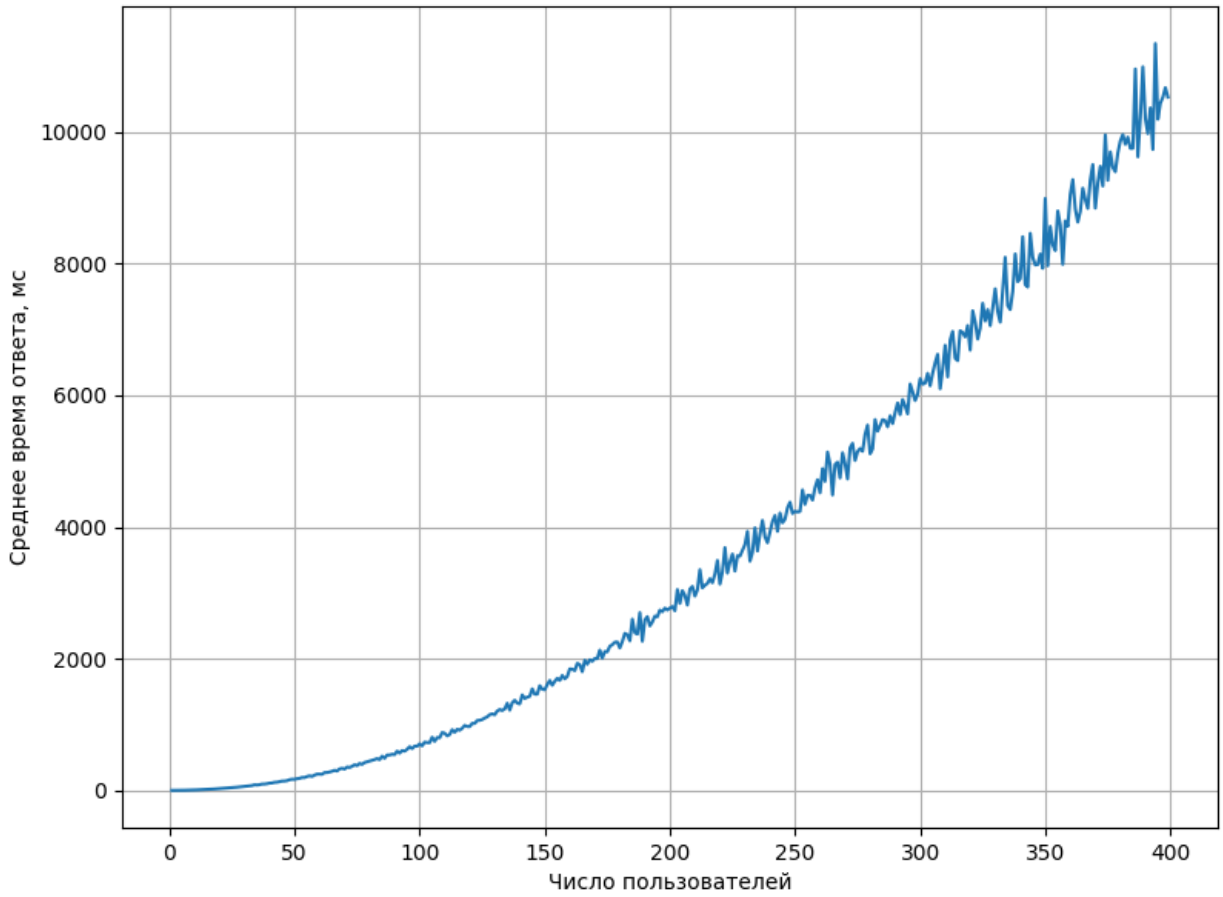

Рисунок 3.10 – Тестирование по сценарию №2, 400 пользователей

Из графиков видно, что добавление двух дополнительных GET запросов привело к увеличению времени ожидания ответа от сервера на 5-10% в зависимости от числа пользователей.

3.7.3 Сценарий №3

Данный сценарий состоит из следующих запросов:

1) POST («/login»);
2) GET («/hackathons>);
3) GET («/hackathons/future»);
4) GET («/teams»);
5) POST («/requests»);

6) POST («/logout»).

Сценарий №3 содержит запросы из Сценария №2, к которым был добавлен POST запрос. В ходе его выполнения каждый пользователь создаёт заявку, данные о которой сохраняются на сервере.

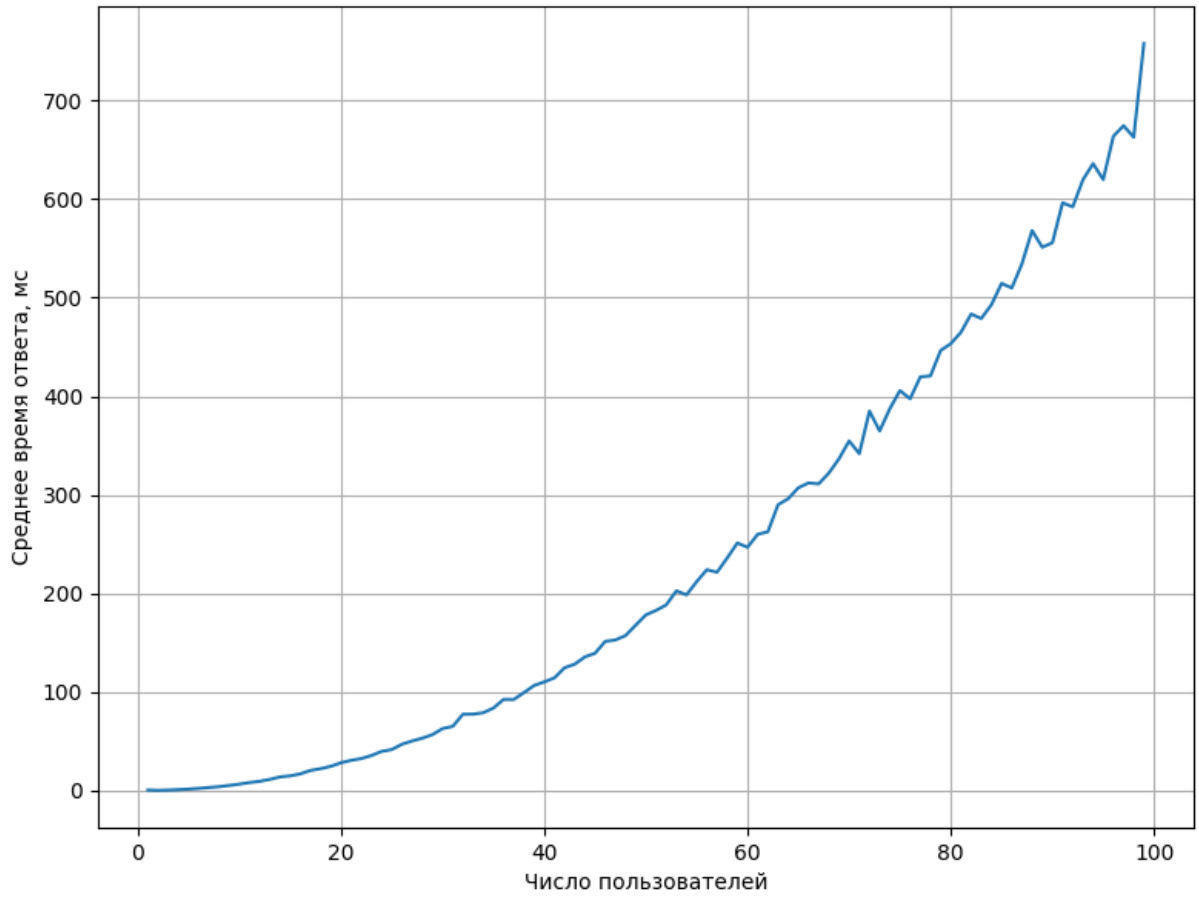

Рисунок 3.11 – Тестирование по сценарию №3, 100 пользователей

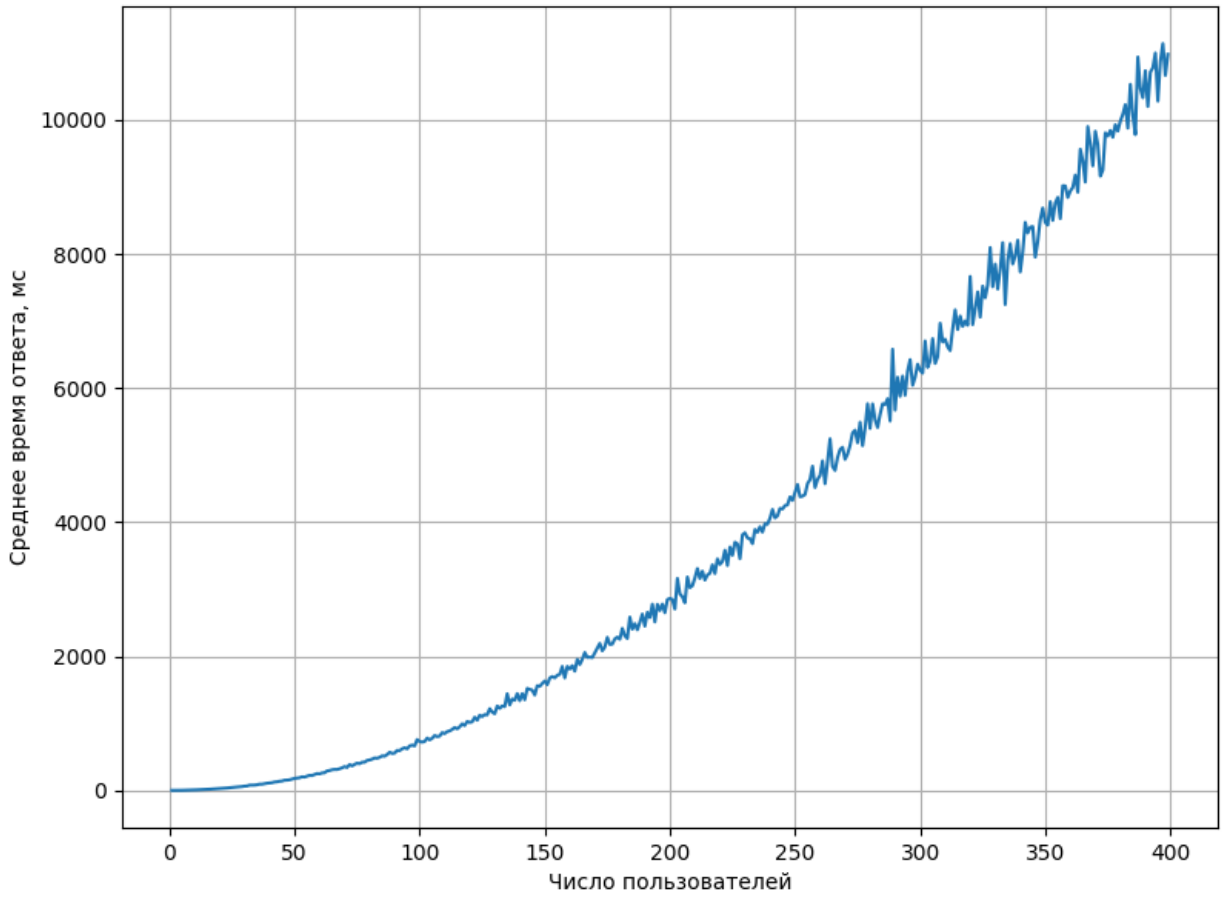

Рисунок 3.12 – Тестирование по сценарию №3, 400 пользователей

В ходе тестирования по данному сценарию было выявлено дальнейшее увеличение времени ответа на 5-10%.

3.8 Вывод из раздела

В данном разделе был сделан выбор СУБД и средств реализации, описано создание БД, триггера, ролей с выделением прав. Также были представлены описание генерации данных для наполнения базы и примеры работы ПО и проведено нагрузочное тестирование, в ходе которого было выяснено,что созданное приложение на вышеуказанном оборудовании достаточно плохо справляется с ситуацией, в которой достаточно большое (≥ 150) число пользователей одновременно посылает запросы. При этом уже начиная с 400 пользователей время ожидания ответа от приложения составляет порядка десяти секунд, что достаточно сильно усложняет работу пользователя [24].

4 Экспериментальный раздел

4.1 Описание эксперимента

Целью эксперимента является оценка изменения времени выполнения запросов на стороне БД в зависимости от наличия индексирования.

Эксперимент проводился над таблицами Users и Participants созданной базы данных.

В ходе эксперимента было исследовано влияние скорость выполнения следующих запросов к базе данных:

1. select * from Participants where rating $<$ 100.
2. select * from Users where login = 'culturedToucan5'.

В первом случае запрашивалась информация о всех пользователях с рейтингом меньше 100 (возвращалось около 1% пользователей из БД). Во втором – информация о пользователе с именем пользователя «$culturedToucan5$» В обоих случаях тестирование проводилось на таблицах с различным числом записей: от 10 000 до 1 000 000. Кроме того, было исследовано время индексирования таблицы и время вставки 10000 записей в проиндексированную и неиндексированную таблицы. Время выполнения каждого из запросов замерялось 20 раз, отбрасывались минимальное и максимальное значения, а затем бралось среднее арифметическое.

Построение индекса проводилось над полями rating и login для соответствующих таблиц. Индексирование проводилось при помощи алгоритма B-Tree, являющимся стандартным в PostgreSQL. Данный алгоритм позволяет ускорить выполнение запросов, использующих операции $\leq, <, =, >, \geq$ [25].

База данных была запущена на ПК со следующими характеристиками:

– RAM: 8 Гбайт DDR4;

– ЦПУ: AMD Ryzen 5 3500U [21];

– ОС: Ubuntu 20.04 [22].

При проведении тестирования на ПК были запущены только сервисы ОС, среда разработки и база данных. Во время тестирования ПК был подключен к сети электропитания.

Полученные данные продемонстрированы на рисунках 4.1 - 4.7.

4.1.1 Запрос №1

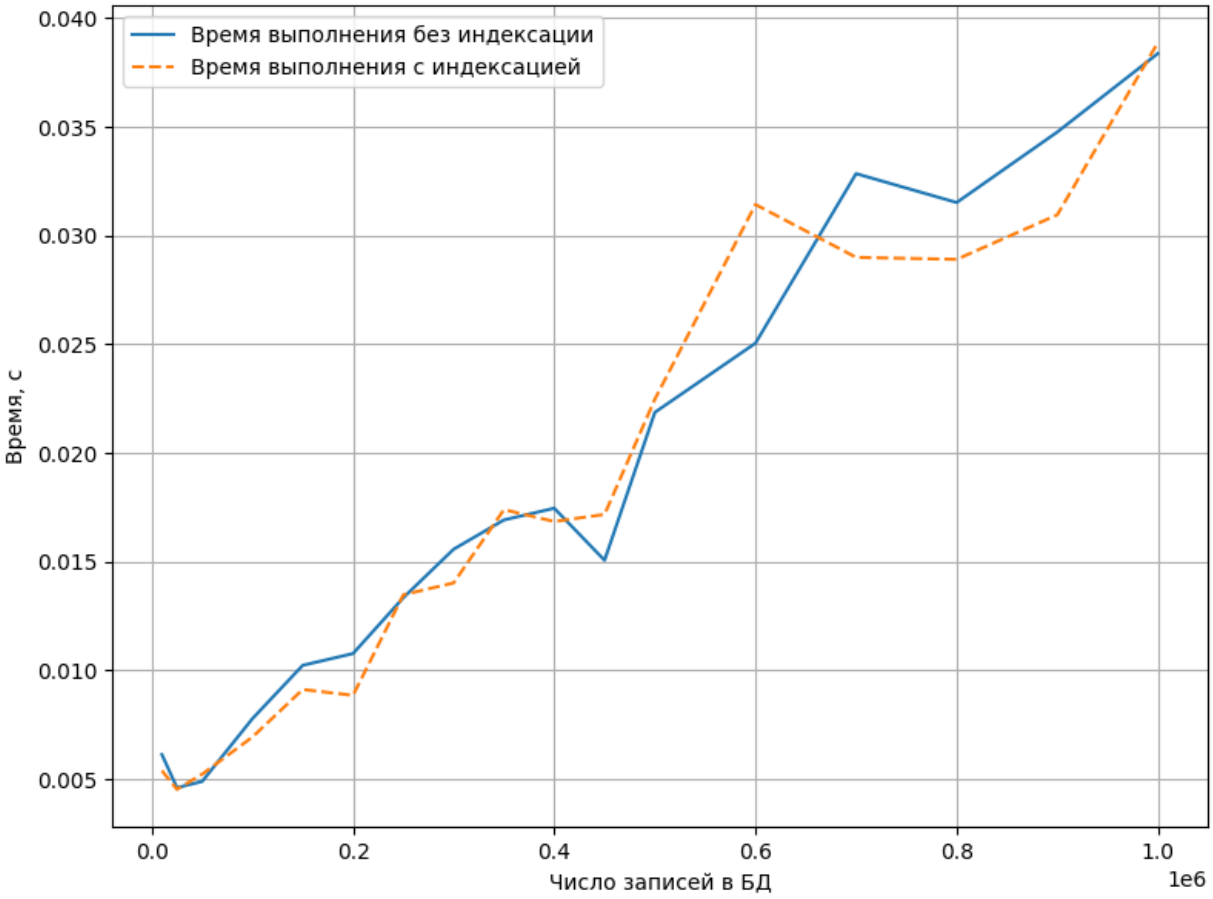

Рисунок 4.1 – Время выполнения запросов с индексацией и без

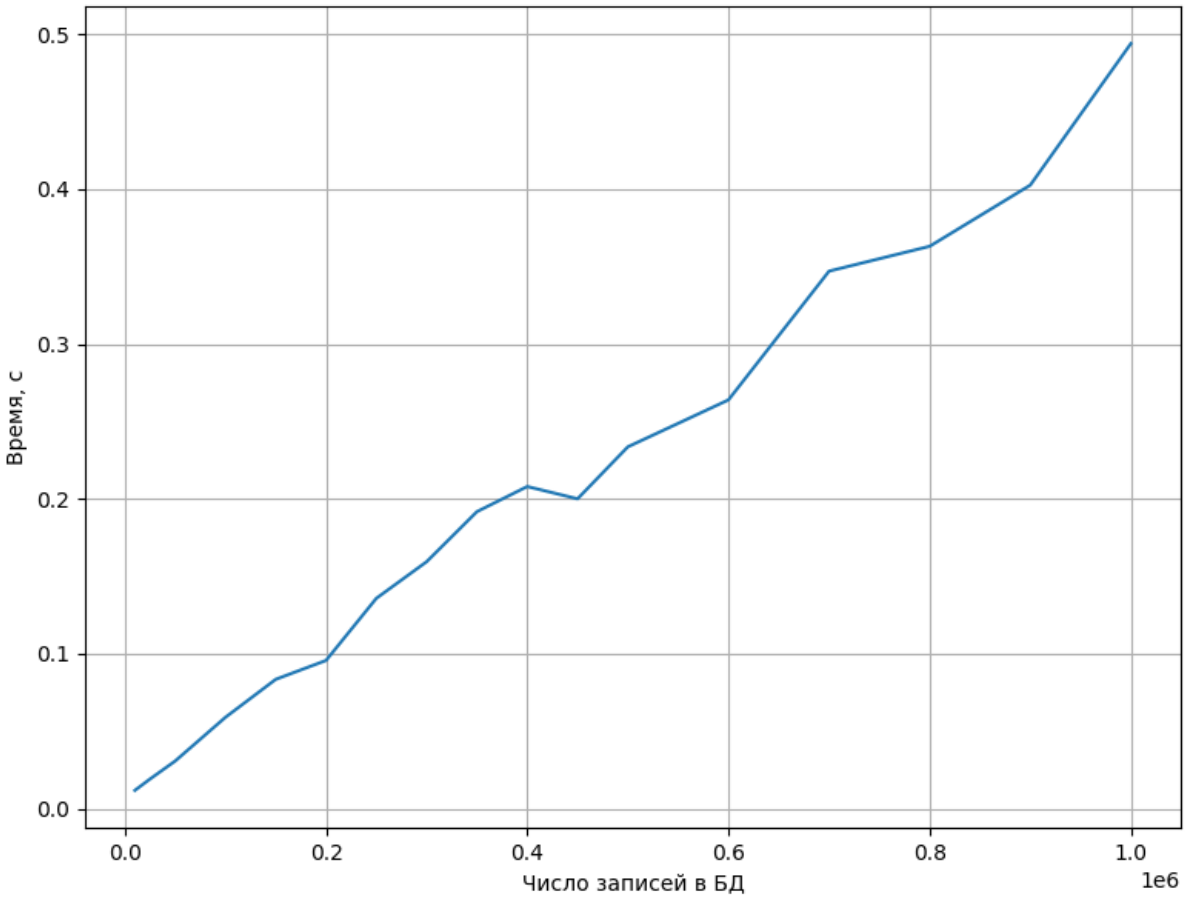

Рисунок 4.2 – Время выполнения индексации

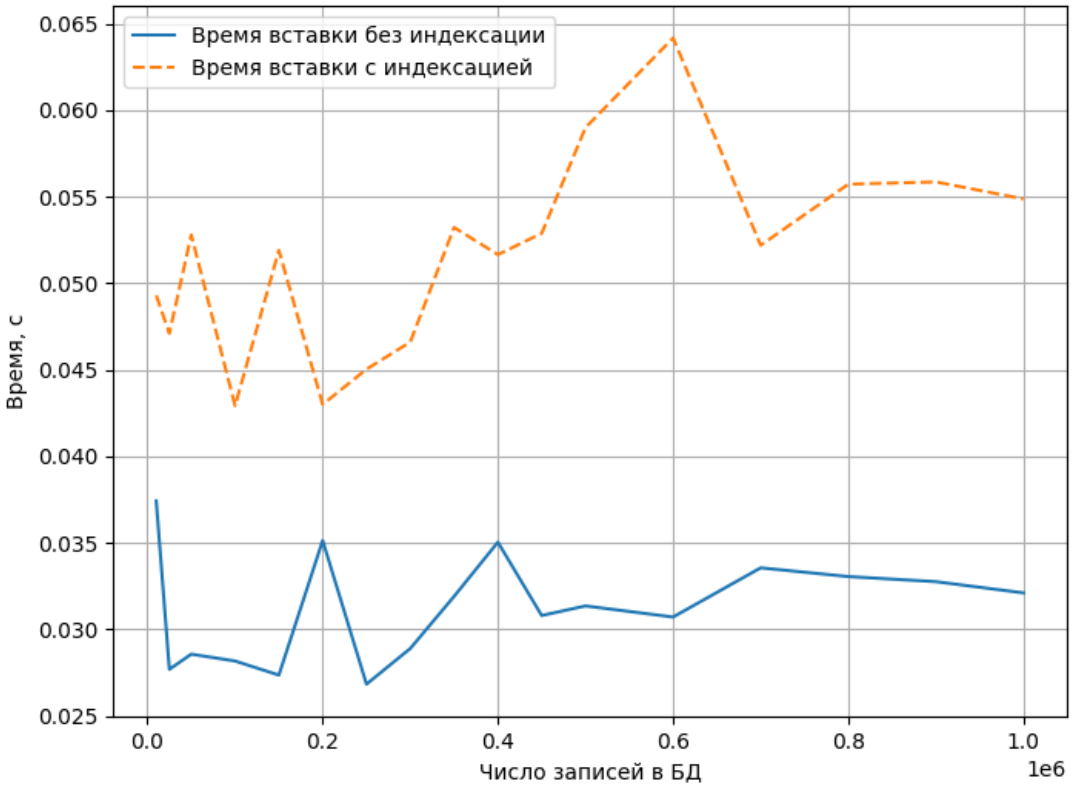

Рисунок 4.3 – Время вставки 10000 записей в таблицу

По полученным данным нельзя сказать, что индексация оказала заметное воздействие на время выполнения запросов. Время индексации таблицы составило до 0.5 секунд (при 1 000 000 записей). Более того, время вставки в проиндексированную таблицу возросло в 1.5 - 2 раза по сравнению с неиндексированной таблицей.

4.1.2 Запрос №2

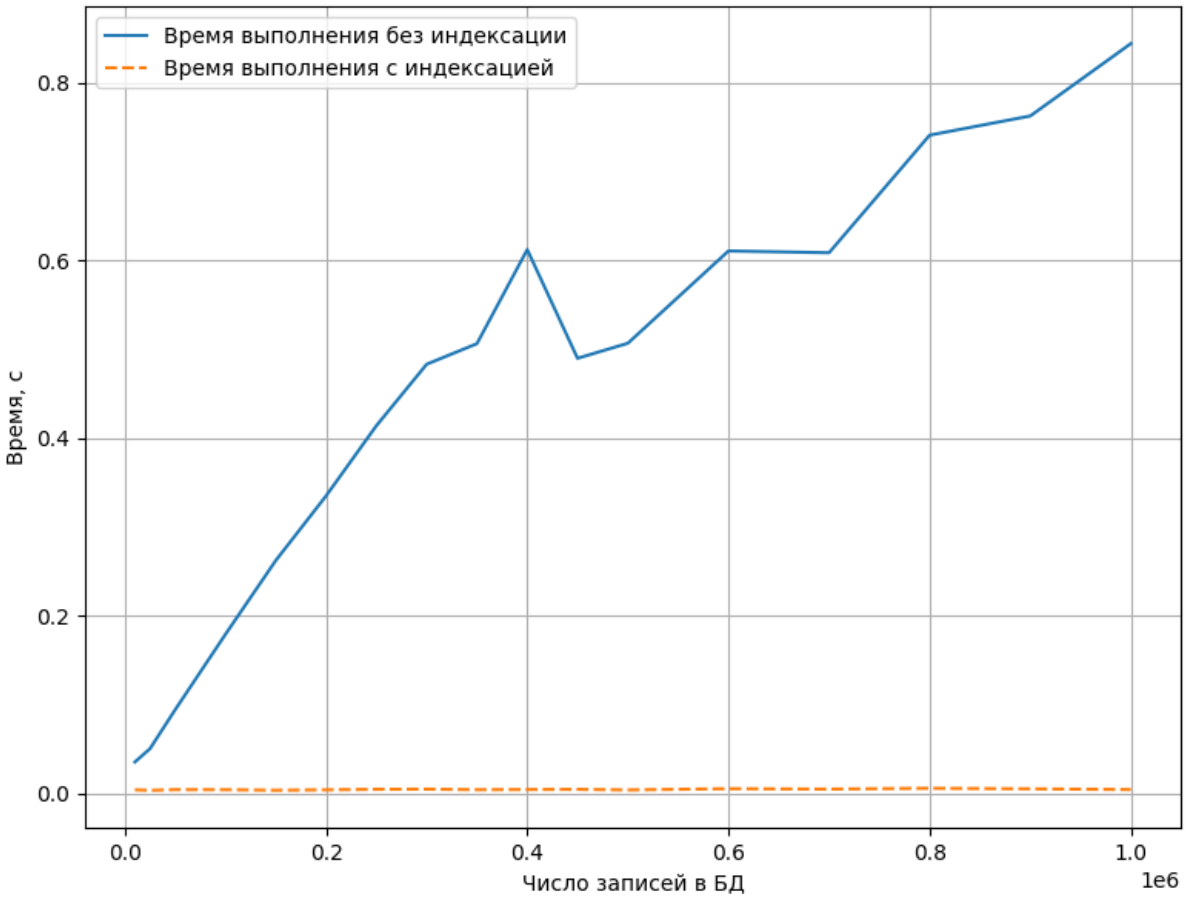

Рисунок 4.4 – Время выполнения запросов с индексацией и без

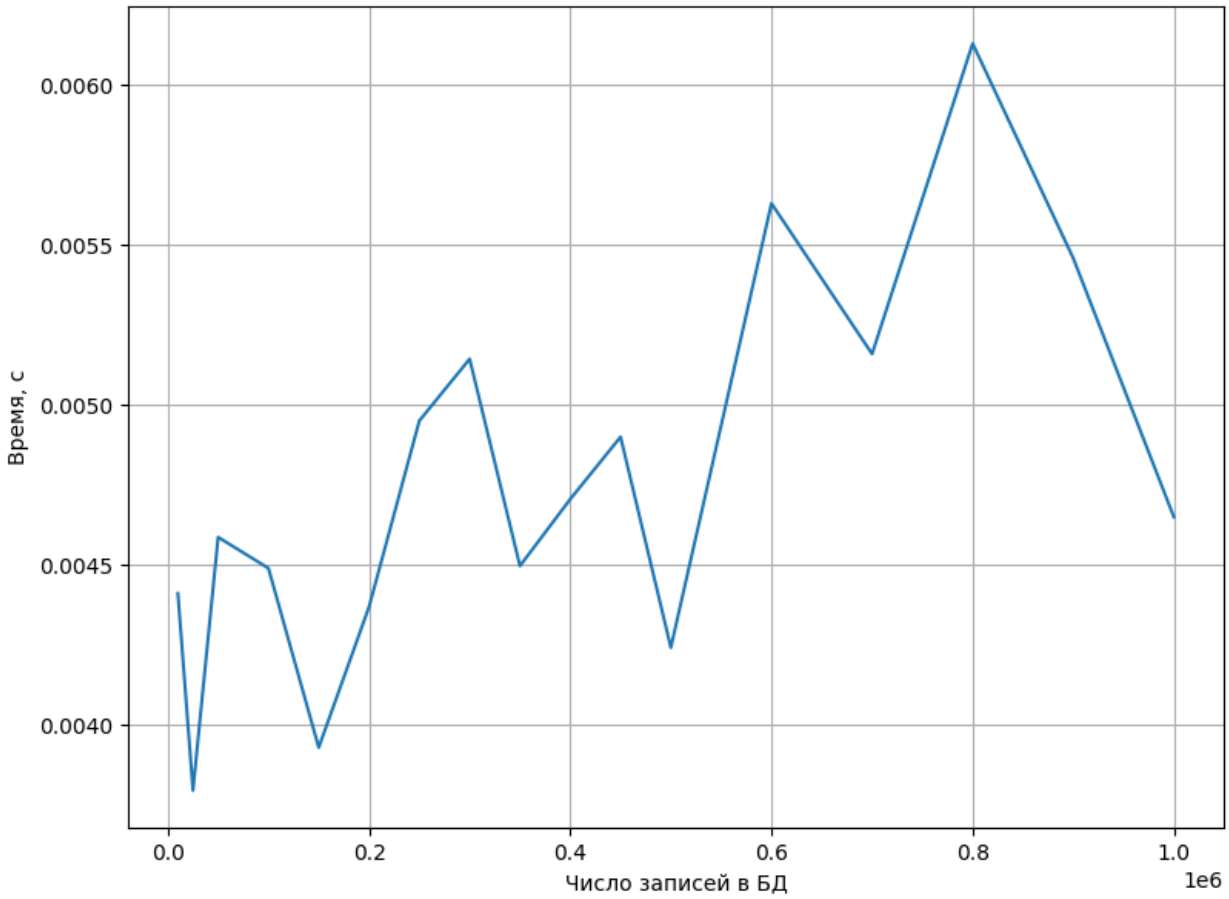

Рисунок 4.5 – Время выполнения запросов с индексацией

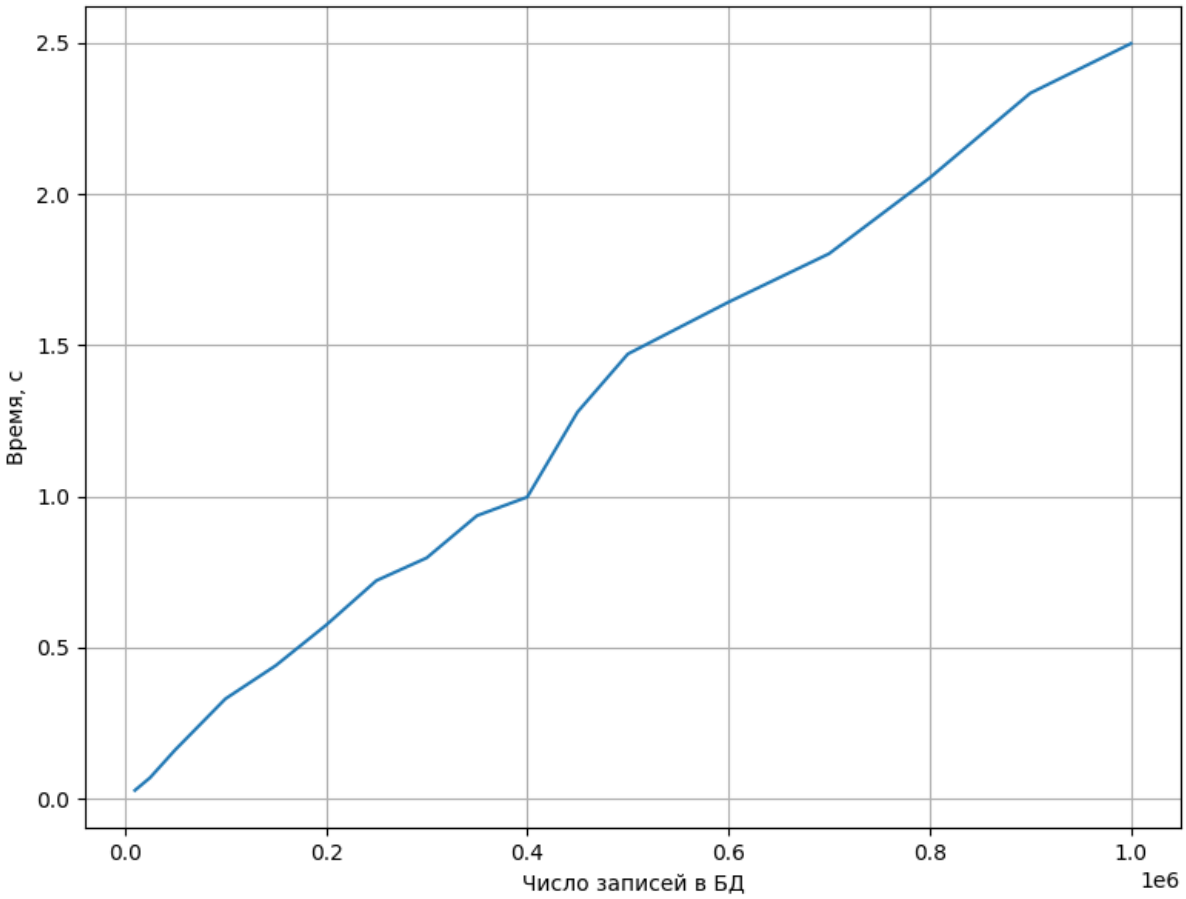

Рисунок 4.6 – Время выполнения индексации

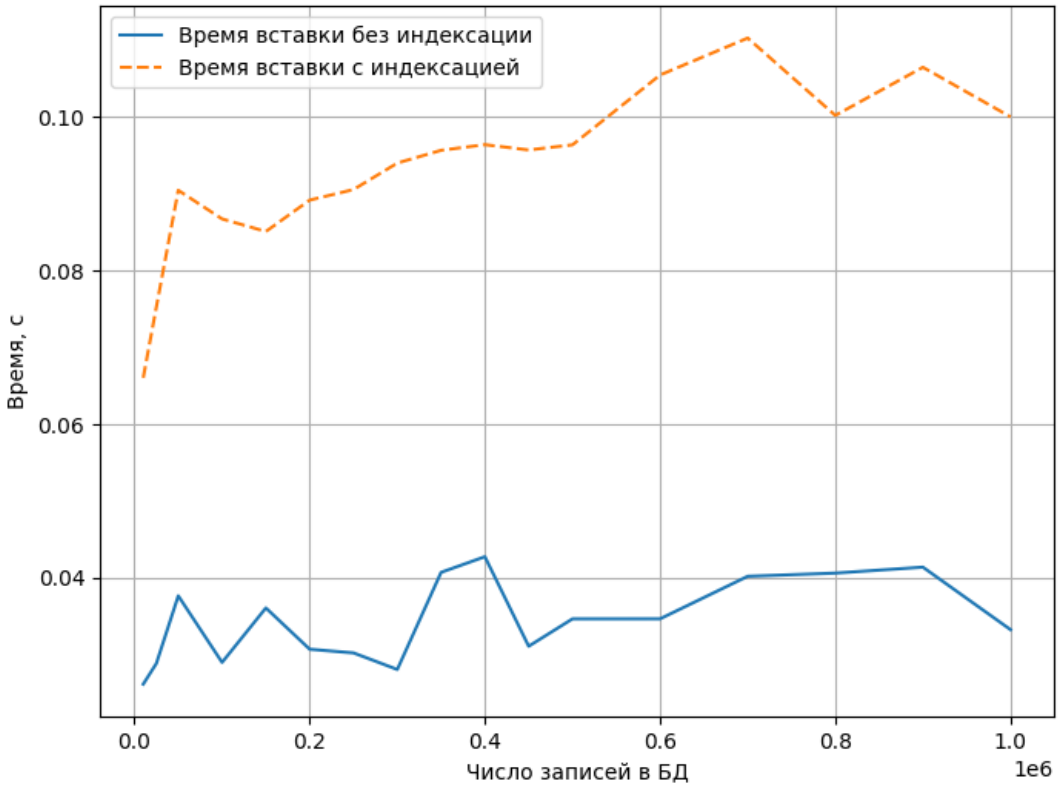

Рисунок 4.7 – Время вставки 10000 записей в таблицу

В данном случае индексация позволила многократно (в 100-200 раз) сократить время выполнения запроса. Однако вместе с тем время индексации таблицы увеличилось до 2.5 с (при 1 000 000 записей) и возросло время вставки в таблицу (в 2 - 3 раза).

4.2 Вывод из раздела

В результате эксперимента обнаружено, что индексация способно многократно (по полученным данным до 200 раз) ускорить выполнение запросов к базе данных, однако имеет несколько недостатков: оно способно оказать воздействие на время выполнения запросов только при использовании уникальных для пользователя данных, увеличивает время массовой вставки в таблицу в 1,5 - 3 раза и построение индекса над таблицей также занимает время.

Заключение

В рамках курсового проекта была создана база данных для хакатонов по направлению «Big Data».

Был проведён анализ существующих решений, предметной области, построена её модель в виде ER-диаграммы, выделены ролевые модели в системе, конкретизированы хранимые данные и выбрана модель базы данных.

Были формализованы сущности системы и спроектирован триггер AFTER на обновление рейтинга после присвоения призовых мест командам, участвовавшим в хакатоне.

Была выбрана СУБД, средства реализации приложения, описано создание триггера, ролей и выделение им прав. Было представлено описание генерации данных для базы и примеры работы ПО. Было проведено нагрузочное тестирование по трём различным пользовательским сценариям, в ходе которого было выяснено, что разработанное приложение плохо справляется с большим количеством пользователей.

Был проведён эксперимент, в ходе которого было сравнение времени выполнения различных select запросов при наличии индексирования и без него. Полученные данные свидетельствуют о том, что индексирование способно до 200 раз ускорить выполнение запросов, однако имеет ограниченную применимость, замедляет массовую вставку в таблицу и требует времени для построение индекса над таблицей.

Все поставленные задачи были выполнены: разработана база данных и интерфейс для взаимодействия с ней с использованием различных ролевых моделей.

Список использованных источников

1. CoderNet – Портал для помощи программистам [Электронный ресурс]. – Режим доступа: https://codernet.ru/articles/drugoe/xakaton_opredelenie_prostyimi_slovami_i_zachem_on_provoditsya/ (Дата обращения: 15.05.2023).

2. Цифровой прорыв 2021 [Электронный ресурс]. – Режим доступа: https://leadersofdigital.ru/, (Дата обращения: 14.06.2023).

3. HYPE Hackathon Solution [Электронный ресурс]. – Режим доступа: https://www.hypeinnovation.com/en/solutions/hackathons (Дата обращения: 14.05.2023).

4. Eventornado [Электронный ресурс]. – Режим доступа: https://eventornado.com/, свободный – (14.05.2023).

5. Hackarthon Manager [Электронный ресурс]. – Режим доступа: https://github.com/codeRIT/hackathon-manager, свободный – (14.05.2023).

6. SkillFactory - Хакатон [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/hackathon/, (Дата обращения: 28.05.2023).

7. Цифровой прорыв - Вопросы и ответы [Электронный ресурс]. – Режим доступа: https://hacks-ai.ru/faq.html, (Дата обращения: 28.05.2023).

8. Карпова И. П. Базы данных. Учебное пособие. – М.: Московский государственный институт электроники и математики (Технический университет), 2009. – 131 с.

9. SQL – Structured Query Language [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?view=sql-server-ver15 (Дата обращения: 14.05.2023).

10. SQL – Structured Query Language [Электронный ресурс]. – Режим доступа: http://bseu.by/it/tohod/lekcii2_4.htm, (Дата обращения: 14.05.2023).

11. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/ (Дата обращения: 14.05.2023).

12. Python – Documentation [Электронный ресурс]. – Режим доступа: https://www.python.org/, (Дата обращения: 18.05.2023).

13. Flask – Documentation (2.3.x) [Электронный ресурс]. – Режим доступа: https://flask.palletsprojects.com/en/2.3.x/, (Дата обращения: 18.05.2023).

14. FreeCodeCamp – Object Relational Mapping [Электронный ресурс]. – Режим доступа: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-t (Дата обращения: 20.05.2023).

15. SQLAlchemy – The Database Toolkit for Python [Электронный ресурс]. – Режим доступа: https://www.sqlalchemy.org/, (Дата обращения: 20.05.2023).

16. Postman API Platform [Электронный ресурс]. – Режим доступа: https://www.postman.com/, (Дата обращения: 14.06.2023).

17. PL/pgSQL – SQL Procedural Language [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/plpgsql.html (Дата обращения: 14.05.2023).

18. IBM – REST API Documentation [Электронный ресурс]. – Режим доступа: https://www.ibm.com/docs/en/inventory-visibility?topic=apis-rest-api-documentation (Дата обращения: 16.05.2023).

19. MDN – HTTP Authorization [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication, (Дата обращения: 20.05.2023).

20. MDN – Working with JSON [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON, (Дата обращения: 20.05.2023).

21. Ryzen 5 3500u [Электронный ресурс]. – Режим доступа: https://www.amd.com/en/products/apu/amd-ryzen-5-pro-3500u (дата обращения: 27.05.2023).

22. Ubuntu 20.04 LTS [Электронный ресурс]. – Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 27.06.2023).

23. Apache JMeter – Overview [Электронный ресурс]. – Режим доступа: https://jmeter.apache.org/, (Дата обращения: 28.05.2023).

24. NN Group – Respone times: the 3 important limits [Электронный ресурс]. – Режим доступа: https://www.nngroup.com/articles/response-times-3-important-limits/, (Дата обращения: 28.05.2023).

25. PostgreSQL Documentation – Index Types [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/indexes-types.html, (Дата обращения: 14.06.2023).

Приложение А

Данное приложение содержит сценарии создания и заполнения базы данных.

Листинг 4.1 – Создание базы данных и ее сущностей

```
create table if not exists Roles
(
    id serial primary key,
    description text
);

create table if not exists Request_Types
(
    id serial primary key
);

create table if not exists Statuses
(
  id serial primary key
);

create table if not exists Teams
(
    id serial primary key,
    name text,
    n_members int,
    rating float,
    cap_id int,
    foreign key (cap_id) references Users(id)
);

create table if not exists Users
(
  id serial primary key,
  name text,
  login text,
  password text,
  role serial,
  foreign key (role) references Roles(id),
  unique (login)
);

create table if not exists T_P_List
(
    part_id serial,
```

Листинг 4.2 – Продолжение листинга 4.1

```
    team_id serial,
    foreign key (team_id) references Teams(id),
    foreign key (part_id) references Users(id)
);
create table if not exists Hackathons
(
    id serial primary key,
    name text,
    address text,
    date date,
    theme text,
    duration float,
    id_first serial,
    id_second serial,
    id_third serial,
    foreign key (id_first) references Users(id),
    foreign key (id_second) references Users(id),
    foreign key (id_third) references Users(id)
);

create table if not exists Requests
(
    id serial primary key,
    type serial,
    requester_id serial,
    requester_type serial,
    status serial,
    name text,
    adm_id serial,
    comment text,
    foreign key (type) references Request_Types(id),
    foreign key (requester_id) references Roles(id),
    foreign key (status) references Statuses(id),
    foreign key (adm_id) references Users(id)
);
```

Листинг 4.3 – Заполнение базы данных

```
import psycopg2

open_files = []

def get_insert_user_query(args):
    name = args[0]
    login = args[1]
    password = args[2]
    role = args[3]

    query = f"insert into Users values (default, '{name}', '{login}',
    '{password}', {role})"

    return query

def get_insert_team_query(args):
    name = args[0]
    n_members = args[1]
    cap_id = args[2]
    rating = args[3]

    query = f"insert into Teams values (default, '{name}',
    '{n_members}', '{rating}', {cap_id})"

    return query

def get_insert_t_p_query(args):
    part_id = args[0]
    team_id = args[1]

    query = f"insert into t_p_list values ('{team_id}', '{part_id}')"
    return query

def get_insert_t_h_query(args):
    team_id = args[0]
    hack_id = args[1]

    query = f"insert into t_p_list values ('{team_id}', '{hack_id}')"

    return query

def get_insert_hackathon_query(args):
    name = args[0]
    address = args[1]
    date = args[2]
    theme = args[3]
```

Листинг 4.4 – Заполнение базы данных

```
1
2     duration = args[4]
3     org_id = args[5]
4     first_id, second_id, third_id = args[6], args[7], args[8]
5
6     query = f"insert into hackathons values (default, '{name}',
7     '{address}', '{date}', '{theme}', '{duration}',
8     '{org_id}', '{first_id}', '{second_id}','{third_id}')"
9
10    return query
11
12 connection = psycopg2.connect(user='postgres',
13                                password='postgres',
14                                port=228,
15                                host='localhost')
16
17 cur = connection.cursor()
18
19 filenames = ['admins.csv', 'organizers.csv', 'captains.csv',
20 'participants.csv', 'teams.csv', 't_p_list.csv',
21              'hackathons.csv', "h_t_list.csv"]
22 files = []
23
24 for f in filenames:
25     files.append(open('generated data/' + f))
26
27 for i in range(0, len(files)):
28     for line in files[i]:
29         args = line.split(';')
30         if i < 4:
31             query = get_insert_user_query(args)
32         elif i == 4:
33             query = get_insert_team_query(args)
34         elif i == 5:
35             query = get_insert_t_p_query(args)
36         elif i == 6:
37             query = get_insert_hackathon_query(args)
38         elif i == 7:
39             query = get_insert_t_h_query(args)
40
41         cur.execute(query)
42         connection.commit()
43 for file in files:
44     file.close()
45 cur.close()
46 connection.close()
```

Приложение Б

Данное приложение содержит список разработанных для взаимодействия пользователя с базой данных HTTP-запросов.

- POST («/login») – авторизация в приложении. При этом логин и пароль передаются в заголовке к запросу согласно методу BasicAuth[19];
- POST («/logout») – выход из учетной записи;
- GET («/hackathons») – получить данные о всех хакатонах;
- GET («/hackathon/<id> ») – получить данные о хакатоне с идентификатором id;
- GET («/hackathons/<id>/teams») – получить данные о командах, участвующих в хакатоне;
- GET («/hackathons/my_hackathons») – получить данные о хакатонах текущего пользователя;
- POST («/hackathons») – создать хакатон. Доступно только администратору;
- DELETE («/hackathons/<id> ») – удалить хакатон. Доступно только администратору и создателю хакатона;
- PUT («/hackathons/<id> ») – редактировать данные о хакатоне. Доступно только администратору и создателю хакатона;
- GET («/teams») – получить данные о существующих командах;
- GET («/teams/<id> ») – получить данные о команде с указанным идентификатором;
- GET («/teams/my_team») – получить данные о команде, в которой состоит текущий пользователь;
- GET («/teams/places_available») – получить список команд со свободными местами;

– GET («/teams/<id>/participants») – получить список участников команды;

– POST («/teams») – создать команду. Доступно только администратору;

– PUT («/teams/<id>/leave») – покинуть команду;

– PUT («/teams/<team_id>/set_captain/») – назначить нового капитана команды. Новым капитаном может быть только член команды. Доступно администратору и капитану команды;

– DELETE («/teams/<team_id>/<member_id> ») – удалить из команды идентификатором team_id участника с идентификатором member_id. Доступно администратору и капитану команды;

– GET («/participants») – получить данные обо всех участниках;

– GET («/participants/<id> ») – получить данные об участнике по его идентификатору;

– GET («/organizers») – получить данные обо всех организаторах;

– GET («/organizers/<id> ») – получить данные об организаторе по его идентификатору;

– GET («/requests») – получить данные обо всех заявках. Доступно только администратору.

– GET («/requests/open») – получить данные об открытых заявках. Доступно только администратору.

– GET («/requests/my_requests) – получить данные о заявках, которые подал текущий пользователь и которые назначены на него. Капитаны получают заявки о вступлении в команду, организаторы – об участии команды в хакатоне, администраторы – те заявки, которые они приняли;

– POST («/requests») – подать заявку;

– PUT («/requests/accept/<id> ») – взять заявку в работу. Доступно только администратору;

– PUT («/requests/approve/<id> ») – одобрить заявку. Капитаны могут одобрять заявки о вступлении в команду, организаторы – об участии в хакатоне, администраторы – могут совершать данное действие со всеми заявками;

– PUT («/requests/decline/<id> ») – отклонить заявку.

Приложение В

Данное приложение содержит презентацию к курсовой работе.