Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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 (ИУ)

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка базы данных для многопользовательского календаря

Студент группы **ИУ7-66Б** _____ **Солопов Ю. В.**

Руководитель курсовой работы _____ **Романова Т. Н.**

2023 г.

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

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

_____ Рудаков И. В.

«3» марта 2023 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине

Базы данных

Студент группы **ИУ7-66Б**

Солопов Юрий Витальевич

Тема курсовой работы

Разработка базы данных для многопользовательского календаря

График выполнения работы: 25% к 4 нед., 50% к 8 нед., 75% к 13 нед., 100% к 15 нед.

Задание

Провести анализ предметной области. Сформулировать требования к базе данных и приложению. Сформулировать описание пользователей проектируемого приложения. Спроектировать архитектуру базы данных и ограничения целостности. Спроектировать ролевую модель на уровне базы данных. Выбрать средства реализации. Реализовать спроектированную БД и необходимый интерфейс для взаимодействия с ней. Исследовать характеристики разработанного программного обеспечения.

Оформление курсовой работы:

Расчетно-пояснительная записка на 25-40 листах формата А4. Презентация на 12-18 слайдах.

Дата выдачи задания «3» марта 2023 г.

Руководитель курсовой работы _____ **Романова Т. Н.**

Студент _____ **Солопов Ю. В.**

РЕФЕРАТ

Расчётно-пояснительная записка содержит 44 с., 13 рис., 4 табл., 8 ист.

Тема работы: разработка базы данных для многопользовательского календаря.

Ключевые слова: базы данных, реляционная модель, поисковые системы, OLAP, OLTP, PostgreSQL, ElasticSearch.

Исследование: сравнение эффективности фильтрации и сортировки при использовании различных видов баз данных.

Содержание

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

OLAP — online analytical processing — интекративная аналитическая обработка.

OLTP — online transaction processing — транзакционная система.

БД — база данных.

СУБД — система управления базами данных.

ВВЕДЕНИЕ

Планирование является неотъемлемым процессом в современной жизни. Необходимость постоянно держать в голове большое количество встреч, мероприятий, событий и т.д. привела к появлению средств, которые позволяют хранить подобную информацию. Одним из видов таких средств является календарь, хранящий информацию о событиях. Но в одном событии может учавствовать сразу несколько человек, одного из которых можно назвать инициатором этого события, а остальных — гостями. По этой причине появляется необходимость в создании информационной системы, позволяющей нескольким пользователям совместно планировать события.

Целью курсовой работы является разработка базы данных для многопользовательского календаря.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) проанализировать существующие решения;
2) формализовать задачу и определить необходимый функционал;
3) рассмотреть модели баз данных и выбрать наиболее подходящую;
4) проанализировать существующие СУБД и выбрать наиболее оптимальные;
5) спроектировать и разработать базу данных;
6) спроектировать и разработать приложение, взаимодействующее с разработанной базой данных;
7) провести сравнительный анализ эффективности фильтрации и сортировки при использовании различных СУБД.

1 Аналитическая часть

В данном разделе рассмотрены существующие решения, формализована задача и данные, выделены категории пользователей, а также выбрана модель базы данных.

1.1 Существующие решения

В связи с существованием потребности в системах планирования, на рынке уже существуют решения, предоставляющие различный функционал.

Рассмотрим только самые популярные из них, такие как:

- Яндекс.Календарь;
- календарь Outlook;
- календарь Google.

Выделим следующие критерии для сравнения выбранных решений.

1) Возможность приглашать с событиям участников.
2) Возможность гибкой настройки прав доступа для каждого участника.
3) Возможность добавлять теги.
4) Возможность гибкого поиска событий.

Результаты сравнения выбранных решений по заданным критериям представлены в таблице 1.

Таким образом, ни одно из четырех рассмотренных решений не удовлетворяет всем четырем критериям сравнения. Также стоит отметить, что ни в одной из рассмотренных систем нет возможности назначать права доступа к событию индивидуально для каждого приглашенного пользователя.

Таблица 1 – Существующие решения поставленной задачи

Название проекта	**Возм. приглашать участников**	**Возм. настройки прав доступа**	**Возм. добавлять теги**	**Возм. гибкого поиска событий**
Яндекс. Календарь [1]	Да	Только для всех участников сразу	Нет	Нет
Календарь Outlook [2]	Да	Нет	Да	Да
Календарь Google [3]	Да	Только для всех участников сразу -	Нет	Да

1.2 Формализация задачи

В ходе выполнения курсовой работы необходимо разработать базу данных для хранения информации о пользователях, событиях, приглашениях, правах доступа и тегах. Также необходимо спроектировать и разработать приложение, которое будет предоставлять API-интерфейс для взаимодействия с базой данных.

Необходимо предусмотреть возможность создавать события с указанием названия, временной метки, описания, типа, тегов и приглашенных пользователей. Реализовать возможность для приглашенного пользователя просматривать, менять событие или приглашать других участников в зависимости от выданных ему прав доступа к событию. Помимо этого, требуется реализовать возможность фильтрации и сортировки по различным данным события.

1.3 Формализация данных

В разрабатываемой базе данных можно выделить следующие сущности:

1. пользователь — User;
2. событие — Event;
3. приглашение — Invitation;
4. право доступа — Access Right;
5. тег — Tag.

На рисунке 1 представлена ER-диаграмма сущностей в нотации Чена.

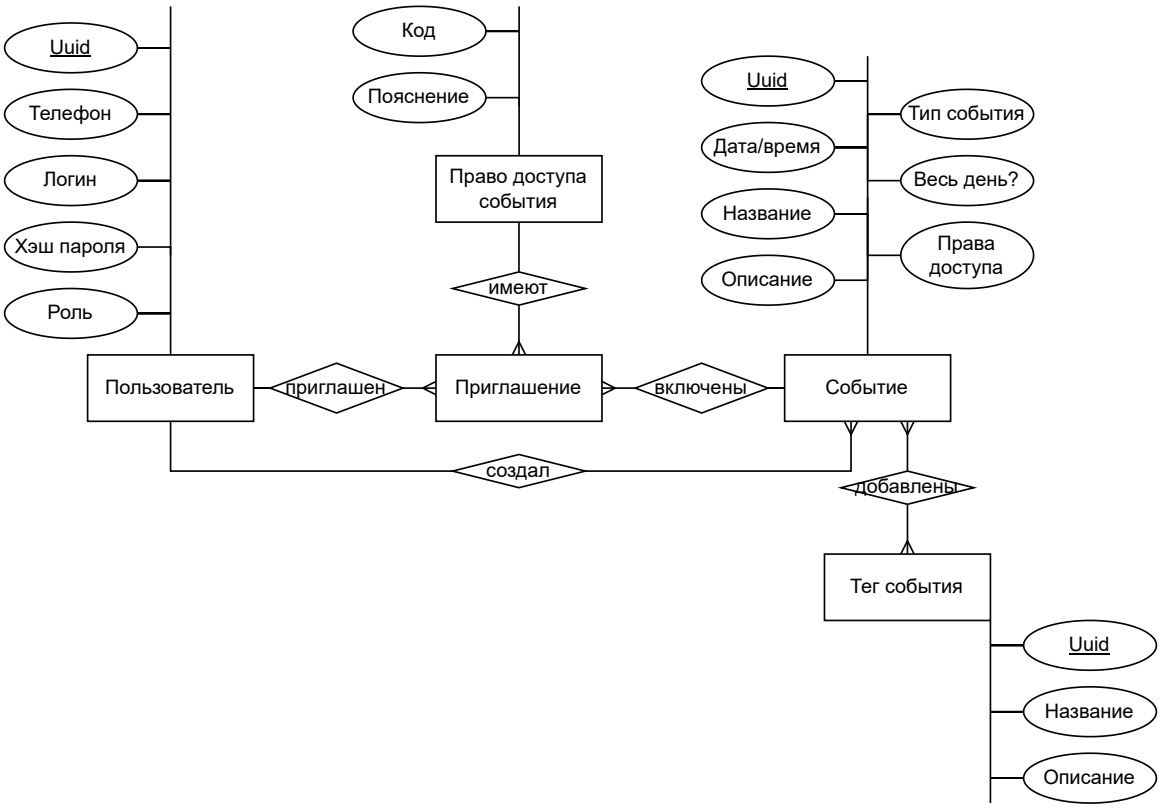

Рисунок 1 – ER-диаграмма в нотации Чена

1.4 Формализация категорий пользователя

Для взаимодействия с приложением было выделено 3 следующие категории пользователей.

1. **Обычный пользователь** после входа в приложение может только созда-

вать, изменять и удалять события и изменять данные своего пользователя.

2. **Премиум пользователь** после входа в приложение может создавать, изменять и удалять события, изменять данные своего пользователя, а также производить поиск событий с фильтрацией, сортировкой и пагинацией.
3. **Администратор** после входа в приложение может создавать, изменять и удалять события и теги, производить поиск событий с фильтрацией, сортировкой и пагинацией, а также изменять и удалять любых пользователей.

Более того, доступ конкретного пользователя к событиям ограничивается указанными для него правами доступа при приглашении. Тем не менее, у администратора есть доступ к любым созданным событиям.

На рисунках 2 – 4 представлены диаграммы вариантов использования приложения для каждой категории пользователей.

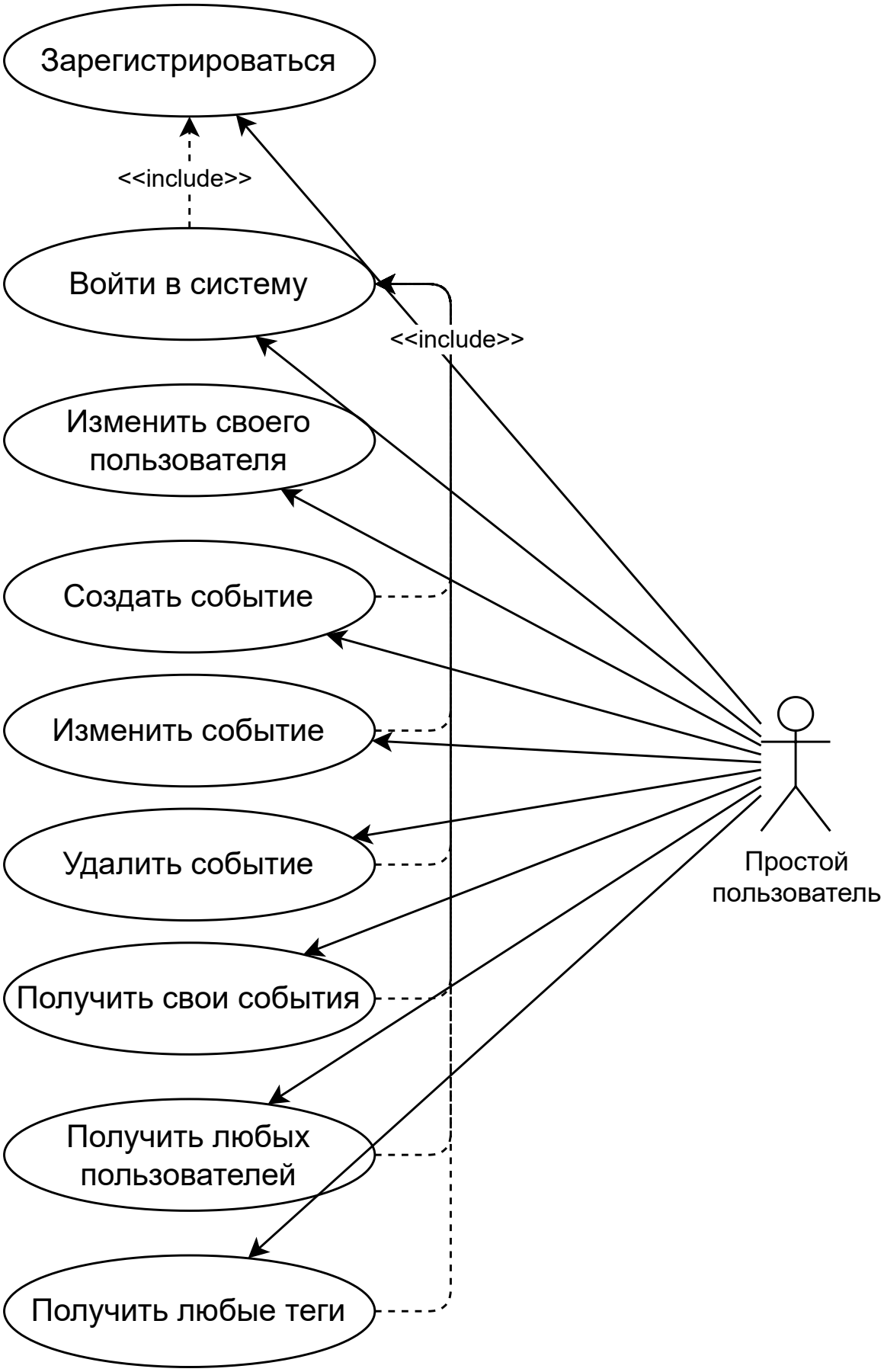

Рисунок 2 – Use-case диаграмма простого пользователя

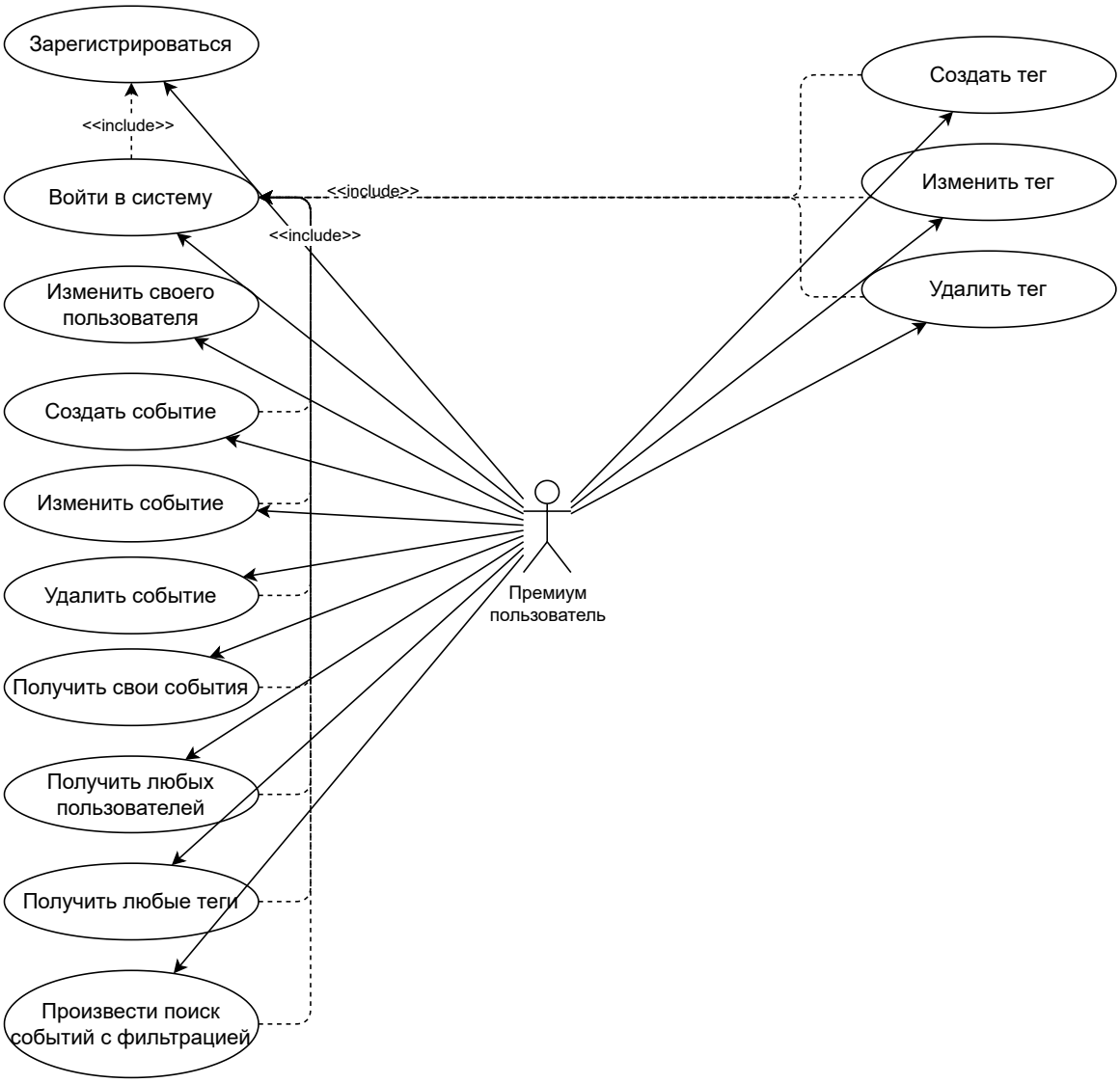

Рисунок 3 – Use-case диаграмма премиум пользователя

Рисунок 4 – Use-case диаграмма администратора

1.5 Классификация СУБД

Система управления базами данных, **СУБД** — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. Главные задачи СУБД – это создание базы данных и манипуляция данными. Такая система позволяет обеспечить надежность хранения данных, их целостность и неизбыточность.

Основными функциями СУБД считаются:

1. управление данными во внешней памяти;
2. управление данными в оперативной памяти с использованием дискового кэша;
3. журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
4. поддержка языков БД.

Классифицировать СУБД можно следующим образом:

1.5.1 По модели хранения

По модели хранения данных СУБД модно разделить на 3 категории:

1. **Дореляционные**
    - **Инвертированные списки** (файлы). БД на основе инвертированных списков представляет собой совокупность файлов, содержащих записи (таблиц). Для записей в файле определен некоторый порядок, диктуемый физической организацией данных. Для каждого файла может быть определено произвольное число других упорядочений на основании значений некоторых полей записей (инвертированных списков). Обычно для этого используются индексы. В такой модели данных отсутствуют ограничения целостности как таковые. Все

ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Одно из немногих ограничений, которое все-таки может присутствовать — это ограничение, задаваемое уникальным индексом.

- **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.
- **Сетевые** — могут быть представлены в виде графа; логика выборки зависит от физической организации данных.

2. **Реляционные** [7]

   В реляционных моделях данные организованы в набор двумерных взаимосвязанных таблиц. Каждая из которых представляет собой набор столбцов и строк, где столбец представляет атрибуты сущности, а строки представляют записи. Использование таблиц для хранения данных обеспечило простой и эффективный способ хранения структурированной информации, доступа к ней, а также легкую сортировку.

3. **Постреляционные** [8]

   Постреляционные базы данных реализуют схему с использованием нереляционной модели данных. Такие БД поддерживают логическую модель данных, ориентированную на обработку транзакций. Постреляционные базы данных включают хранилища данных с ключевыми значениями, сетевые и графовые базы данных, а также базы данных, ориентированные на документы (например, поисковые системы).

   Недостатком такой модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.

1.5.2 По способу доступа к данным

1. **Файл-серверные** — при работе с базой, данные отправляются приложению, которое с ней работает, вне зависимости от того, сколько их нужно. Все операции — на стороне клиента. Файловый сервер периодически обновляется тем же клиентом.
2. **Клиент-серверные** — вся работа производится на сервере, по сети передаются результаты запросов, гораздо меньше информации. Обеспечивается безопасность данных, потому что все происходит на стороне сервера.
3. **Встраиваемые** — библиотека, которая позволяет унифицированным образом хранить большие объемы данных на локальной машине. Доступ к данным может происходить через SQL либо через особые функции СУБД. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют установки сервера, поэтому востребованы в локальном ПО, которое имеет дело с большими объемами данных.
4. **Сервисно-ориентированные** — БД является хранилищем сообщений, промежуточных состояний, метаинформации об очередях сообщений и сервисах;
5. Прочие — пространственная, временная и пространственно-временная.

1.5.3 Выбор модели базы данных

В качестве основной для реализации данного курсового проекта была выбрана реляционная модель данных по следующим причинам:

- задача предполагает постоянное добавление и изменение данных;
- задача предполагает быструю отзывчивость на запросы пользователя;
- задача предполагает надежное хранение данных.

Помимо этого, постановка задачи предполагает возможность осуществления быстрого поиска событий (с использованием фильтрации, сортировки и

пагинации). Такой функционал предоставляют специализированные поисковые базы данных, относящиеся к классу постреляционных. Поэтому было принято решение использовать ElasticSearch для осуществления фильтрации, сортировки и пагинации событий в проектируемом приложении.

2 Конструкторская часть

В данном разделе описаны сущности системы, приведена диаграмма БД, описана проектируемая функция базы данных и приведена формализация бизнес-правил разрабатываемого приложения.

2.1 Описание сущностей

На основе выделенных ранее сущностей спроектированны таблицы реляционной базы данных.

2.1.1 Таблица users

Эта таблица содержит информацию о пользователях системы и включает поля:

1. uuid — идентификатор, тип — uuid, является первичным ключем;
2. created_at — временная метка создания, тип — timestamp;
3. updated_at — временная метка обновления, тип — timestamp;
4. deleted_at — временная метка удаления, тип — timestamp. Будет null, если запись не является удаленной;
5. phone — телефон, тип text, уникальное;
6. login — логин, тип text, уникальное;
7. password_hash — хэш пароля, тип text;
8. role — роль, тип text;

2.1.2 Таблица tags

Эта таблица содержит информацию о тегах и включает поля:

1. uuid — идентификатор, тип — uuid, является первичным ключем;
2. created_at — временная метка создания, тип — timestamp;

3. updated_at — временная метка обновления, тип — timestamp;
4. deleted_at — временная метка удаления, тип — timestamp. Будет null, если запись не является удаленной;
5. name — название, тип text, уникальное;
6. description — описание, тип text;

2.1.3 Таблица events

Эта таблица содержит информацию о событиях и включает поля:

1. uuid — идентификатор, тип — uuid, является первичным ключем;
2. created_at — временная метка создания, тип — timestamp;
3. updated_at — временная метка обновления, тип — timestamp;
4. deleted_at — временная метка удаления, тип — timestamp. Будет null, если запись не является удаленной;
5. timestamp — временная метка события, тип — timestamp;
6. name — название, тип text;
7. description — описание, тип text;
8. type — тип события, тип text;
9. is_whole_day — признак, что событие занимает целый день, тип bool;
10. creator_uuid — ссылка на создателя события, тип uuid;

2.1.4 Таблица events_tags

Эта таблица-связка событий и тегов. Включает поля:

1. event_uuid — ссылка на событие, тип uuid;
2. tag_uuid — ссылка на тег, тип uuid;

2.1.5 Таблица access_rights

Эта таблица содержит информацию о правах доступа и включает поля:

1. code — название, тип text, является первичным ключем;
2. description — описание, тип text;

2.1.6 Таблица invitations

Эта таблица содержит информацию о приглашениях и включает поля:

1. uuid — идентификатор, тип — uuid, является первичным ключем;
2. access_right_code — ссылка на код права доступа, тип text;
3. event_uuid — ссылка на событие, тип uuid;
4. user_uuid — ссылка на приглашенного пользователя, тип uuid;

2.1.7 Схема базы данных

На рисунке 5 представлена диаграмма базы данных.

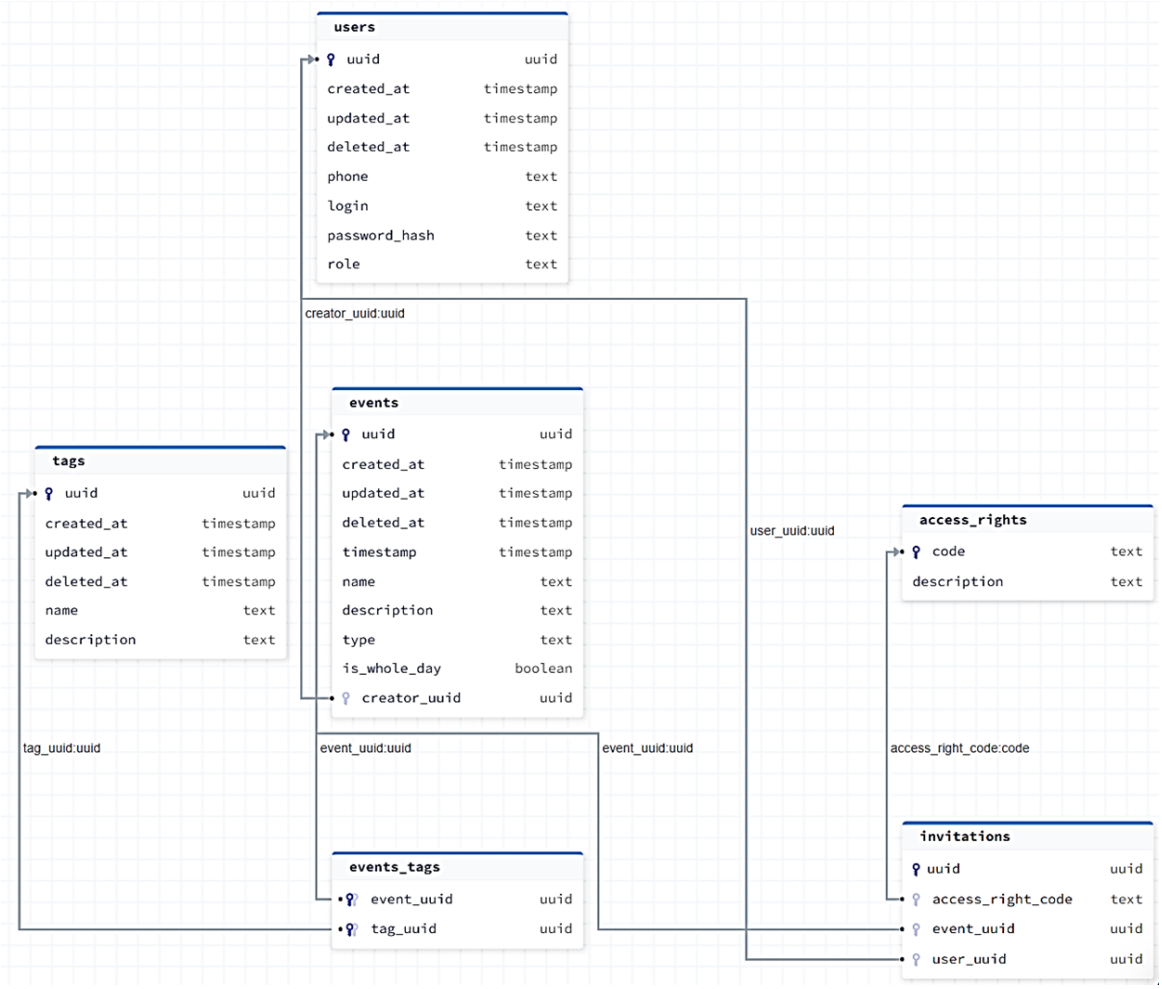

Рисунок 5 – Схема базы данных Postgres

2.1.8 Схема индекса ElasticSearch

На листинге 1 представлено описание схемы индекса events в ElasticSearch.

Листинг 1 – Схема индекса events

```
"mappings": {
    "properties": {
        "@indexed_at": {
            "type": "date"
        },
        "Creator": {
            "properties": {
                "Login": {
                    "type": "text",
                    "fields": {
```

```
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "PasswordHash": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "Phone": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "Role": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "Uuid": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
```

```
                }
            }
        }
    },
    "CreatorUuid": {
        "type": "text",
        "fields": {
            "keyword": {
                "type": "keyword",
                "ignore_above": 256
            }
        }
    },
    "Description": {
        "type": "text",
        "fields": {
            "keyword": {
                "type": "keyword",
                "ignore_above": 256
            }
        }
    },
    "Invitations": {
        "properties": {
            "AccessRightCode": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "UserUuid": {
                "type": "text",
                "fields": {
                    "keyword": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
```

```
                    }
                },
                "Uuid": {
                    "type": "text",
                    "fields": {
                        "keyword": {
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                }
            }
        },
        "IsWholeDay": {
            "type": "boolean"
        },
        "Name": {
            "type": "text",
            "fields": {
                "keyword": {
                    "type": "keyword",
                    "ignore_above": 256
                }
            }
        },
        "Tags": {
            "properties": {
                "Description": {
                    "type": "text",
                    "fields": {
                        "keyword": {
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                },
                "Name": {
                    "type": "text",
                    "fields": {
                        "keyword": {
```

```
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                },
                "Uuid": {
                    "type": "text",
                    "fields": {
                        "keyword": {
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                }
            }
        },
        "Timestamp": {
            "type": "date"
        },
        "Type": {
            "type": "text",
            "fields": {
                "keyword": {
                    "type": "keyword",
                    "ignore_above": 256
                }
            }
        },
        "Uuid": {
            "type": "text",
            "fields": {
                "keyword": {
                    "type": "keyword",
                    "ignore_above": 256
                }
            }
        }
    }
},

```

2.2 Описание проектируемой процедуры

Для корректной работы ролевой системы перед каждым запросом требуется устанавливать необходимую роль текущему пользователю и заносить его *uuid* в переменную текущего сеанса **app.user_uuid**.

Для выполнения описанных выше действий была спроектирована хранимая процедура. Схема ее алгоритма представлена на рисунке 6.

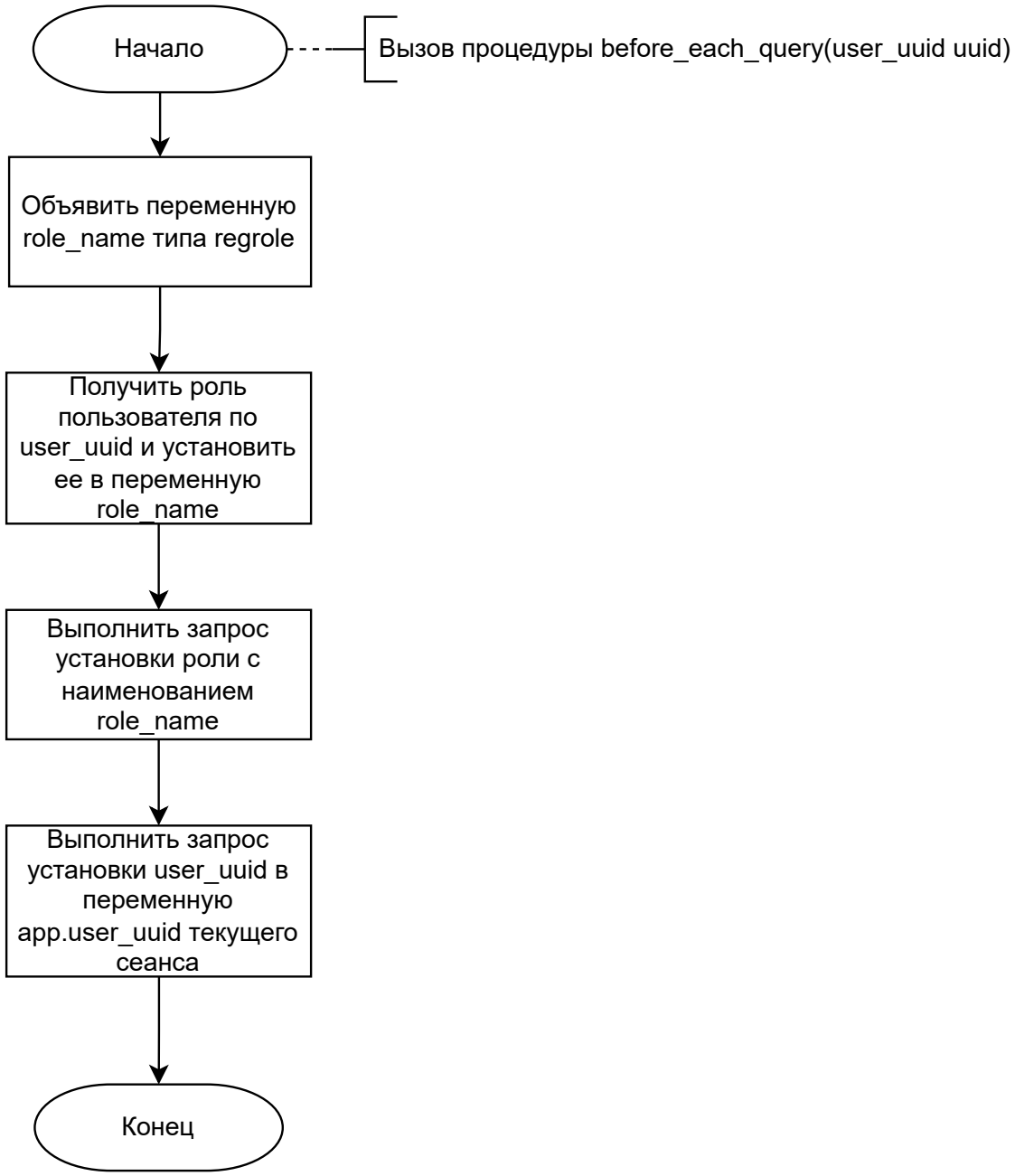

Рисунок 6 – Функция для установки роли и uuid-а текущего пользователя

2.3 Проектирование приложения

В рамках поставленной задачи курсового проекта необходимо разработать веб-приложение с программным интерфейсой (API), обрабатывающим запросы согласно спецификации GraphQL [9]. API должен позволять пользователям выполнять в системе действия, представленные на диаграмме вариантов использования (рисунки 2 – 4).

Незарегистрированный пользователь должен иметь возможность создать новый аккаунт или войти в уже существующий аккаунт. Формализация бизнес-процессов авторизации и регистрации в нотации Business Process Model and Notation (BPMN) представлена на рисунке 7.

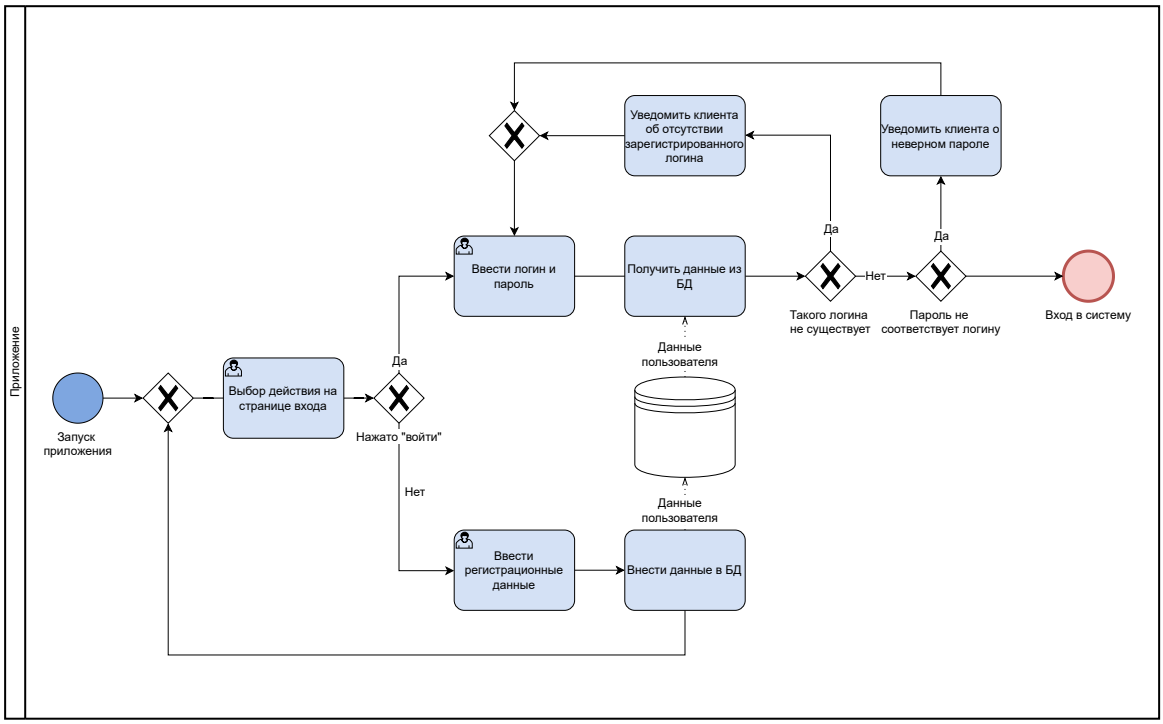

Рисунок 7 – Бизнес-процесс авторизации и регистрации пользователей

Формализация бизнес-процесса создания новых событий с возможностью приглашения к ним участников представлена на рисунке 8.

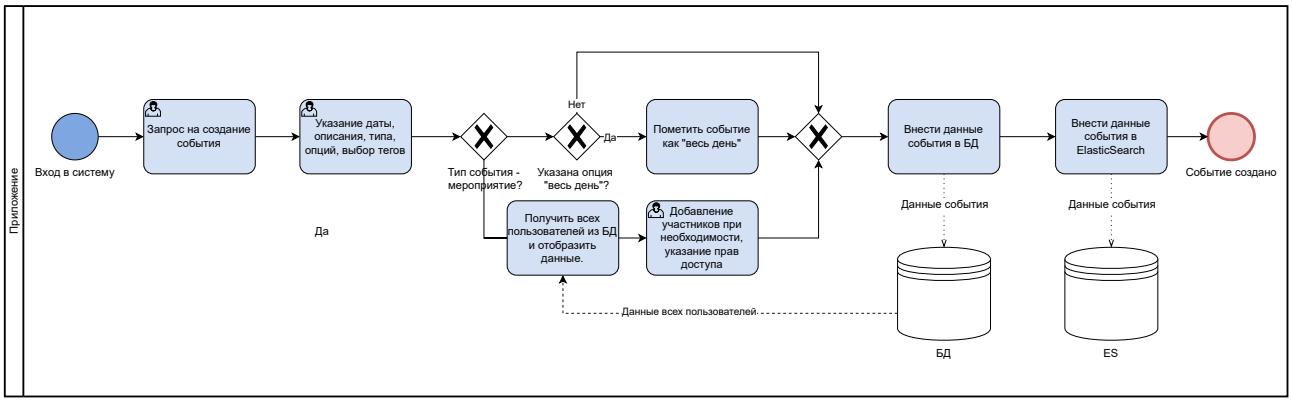

Рисунок 8 – Бизнес-процесс авторизации и регистрации пользователей

3 Технологическая часть

В данном разделе проведен выбор инструментов разработки (языка программирования, библиотек и фреймворков). Также приведено обоснование выбора Postgres в качестве СУБД. Помимо этого, представлены детали реализации приложения и продемонстрирован интерфейс для взаимодействия с приложением.

3.1 Выбор СУБД

В аналитическом разделе при выборе типа модели баз данных была выбрана реляционная модель, следовательно выбор СУБД будет производиться для реляционной модели баз данных. Рассмотрим только самые популярные из них: MySQL, Oracle, PostgreSQL [10], Microsoft SQL Server и SQLite.

Выделим следующие критерии для сравнения выбранных СУБД:

1) открытый исходный код;
2) возможность создания пользователей на уровне БД;
3) опыт работы с СУБД (не является основным требованием, но желательным).

Результаты сравнения выбранных СУБД по заданным критериям представлены в таблице 2.

Таблица 2 – Сравнение выбранных СУБД

Критерий	**MySQL**	**Oracle**	**PostgreSQL**	**Microsoft SQL Server**	**SQLite**
1	+	-	+	-	+
2	+	+	+	+	-
3	-	-	+	-	+

Помимо указанных выше особенностей, стоит отметить, что PostgreSQL активно поддерживается сообществом. На момент 20.05.2023 последнее изме-

нение в исходный код было внесено менее 24 часов назад [11].

По результатам сравнения в качестве СУБД для реляционной базы данных был выбран PostgreSQL.

3.2 Выбор поисковой базы данных

Постановка задачи предполагает возможность осуществления быстрого поиска событий (с использованием фильтрации, сортировки и пагинации). Такой функционал предоставляет поисковая база данных ElasticSearch [12]. Выбор именно этой поисковой БД обоснован возможностью настраивать поиск событий с использованием различных видов фильтрации (включая нечеткое сопоставление), а также поддержкой сортировки событий согласно набранному ими при фильтрации баллу (*_score*).

3.3 Выбор средств реализации

В качестве используемого языка программирования был выбран Golang. Для реализации приложения на этом языке использовались следующие библиотеки:

- .ent [13] — ORM для взаимодействия с базой данных;
- pq [14] — драйвер для работы с PostgreSQL;
- gqlgen [15] — библиотека для реализации GraphQL API.
- fx [16] — система для управления зависимостями.

В качестве среды разработки использовалась среда GoLand.

3.4 Реализация приложения

В данном подразделе приведены листинги кода реализуемого приложения.

3.4.1 Описание таблиц базы данных

На листингах 2 - 3 приведен код описания таблиц *events* и *users* соответственно.

Листинг 2 – Описание таблицы событий

```
// Fields of the Event.
func (Event) Fields() []ent.Field {
    return []ent.Field{
        field.Time("timestamp"),
        field.String("name"),
        field.String("description").Optional().Nillable(),
        field.String("type"),
        field.Bool("is_whole_day"),
        field.String("creator_uuid").SchemaType(map[string]string{
            dialect.Postgres: "uuid",
        }),
    }
}

// Edges of the Event.
func (Event) Edges() []ent.Edge {
    return []ent.Edge{
        edge.To("tags", Tag.Type).StorageKey(
        edge.Table("events_tags"), edge.Columns("event_uuid", "tag_uuid"),
        ),
        edge.To("invitations", Invitation.Type),
        edge.From("creator", User.Type).
        Ref("created_events").
        Field("creator_uuid").
        Unique().
        Required(),
    }
}
```

Листинг 3 – Описание таблицы пользователей

```
// Fields of the User.
func (User) Fields() []ent.Field {
    return []ent.Field{
        field.String("phone").Unique(),
        field.String("login").Unique(),
        field.String("password_hash"),
        field.String("role").GoType(roles.Type("")),
    }
}

// Edges of the User.
func (User) Edges() []ent.Edge {
    return []ent.Edge{
        edge.To("invitations", Invitation.Type),
        edge.To("created_events", Event.Type),
    }
}

```

3.4.2 Создание процедуры

В предыдущем разделе была спроектирована хранимая процедура *before_each_query*. Реализация этой процедуры с использованием языка plpgsql приведена на листинге 4.

Листинг 4 – Реализация процедуры before_each_query

```
CREATE OR REPLACE PROCEDURE before_each_query(IN user_uuid uuid)
    LANGUAGE plpgsql
AS
$$
DECLARE
    role_name regrole;
BEGIN
    role_name = (SELECT role FROM users WHERE users.uuid =
    user_uuid)::regrole;
    execute format('SET role %I', role_name);
```

```
10      execute format('SET app.user_uuid = %L', user_uuid);
11  END;
12  $$;
13
```

3.4.3 Создание ролей и выделение им прав

В аналитическом разделе была разработана ролевая модель, включающая следующие роли:

- simple_user — обычный пользователь;
- premium_user — привелегированный пользователь;
- admin — администратор системы.

В соответствии с разработанной диаграммой вариантов использования (рисунки 2 – 4) пользователя были выделены права. Помимо этого, были созданы политики для ограничения доступа на уровне строк (RLS) к таблице пользователей. Создание ролей, выделение им прав и создание политик представлено на листинге 5.

Листинг 5 – Описание ролей

```
1  create role admin;
2  create role simple_user;
3  create role premium_user;
4
5  grant all on all tables in schema public to admin;
6
7  grant select, update (login, phone, password_hash) on users to simple_user;
8  grant select on access_rights to simple_user;
9  grant all on events to simple_user;
10 grant all on invitations to simple_user;
11 grant all on events_tags to simple_user;
12 grant select on tags to simple_user;
13
14 grant select, update (login, phone, password_hash) on users to premium_user;
15 grant select on access_rights to premium_user;
```

```
16 grant all on events to premium_user;
17 grant all on invitations to premium_user;
18 grant all on events_tags to premium_user;
19 grant all on tags to premium_user;
20
21 ALTER TABLE users
22     ENABLE ROW LEVEL SECURITY;
23
24 DROP VIEW IF EXISTS session;
25 CREATE VIEW session AS
26 SELECT current_setting('app.user_uuid')::uuid AS user_uuid;
27
28 grant all on session to simple_user, premium_user, admin;
29
30 DROP POLICY IF EXISTS users_select ON users;
31 CREATE POLICY users_select
32     ON users
33     FOR SELECT
34     USING (true);
35
36 DROP POLICY IF EXISTS users_update ON users;
37 CREATE POLICY users_update
38     ON users
39     FOR UPDATE
40     TO simple_user, premium_user
41     USING (uuid = (select user_uuid from session));
42
43 DROP POLICY IF EXISTS users_update_admin ON users;
44 CREATE POLICY users_update_admin
45     ON users
46     FOR UPDATE
47     TO admin
48     USING (true);
49
50 DROP POLICY IF EXISTS users_delete_admin ON users;
51 CREATE POLICY users_delete_admin
52     ON users
53     FOR DELETE
54     TO admin
55     USING (true);
```

3.4.4 Интерфейс приложения

Для взаимодействия с приложением был разработан API, обрабатывающий запросы согласно спецификации GraphQL. Для пользователей, событий и тегов были реализованы запросы для получения (всех или по идентификатору), создания, обновления и удаления этих сущностей. Для прав доступа к событиям были реализованы запросы только для их получения. Также был реализован запрос для принудительной синхронизации состояния ElasticSearch с текущим состоянием Postgres. Помимо этого, были реализованы запросы для генерации тестовых данных пользователей, тегов и событий с приглашениями.

Для удобной и быстрой отправки запросов приложение предоставляет специализированный пользовательский интерфейс Apollo Sandbox [17]. На рисунках 9 – 11 приведены несколько примеров запросов к приложению и ответы приложения на эти запросы с использованием этого интерфейса.

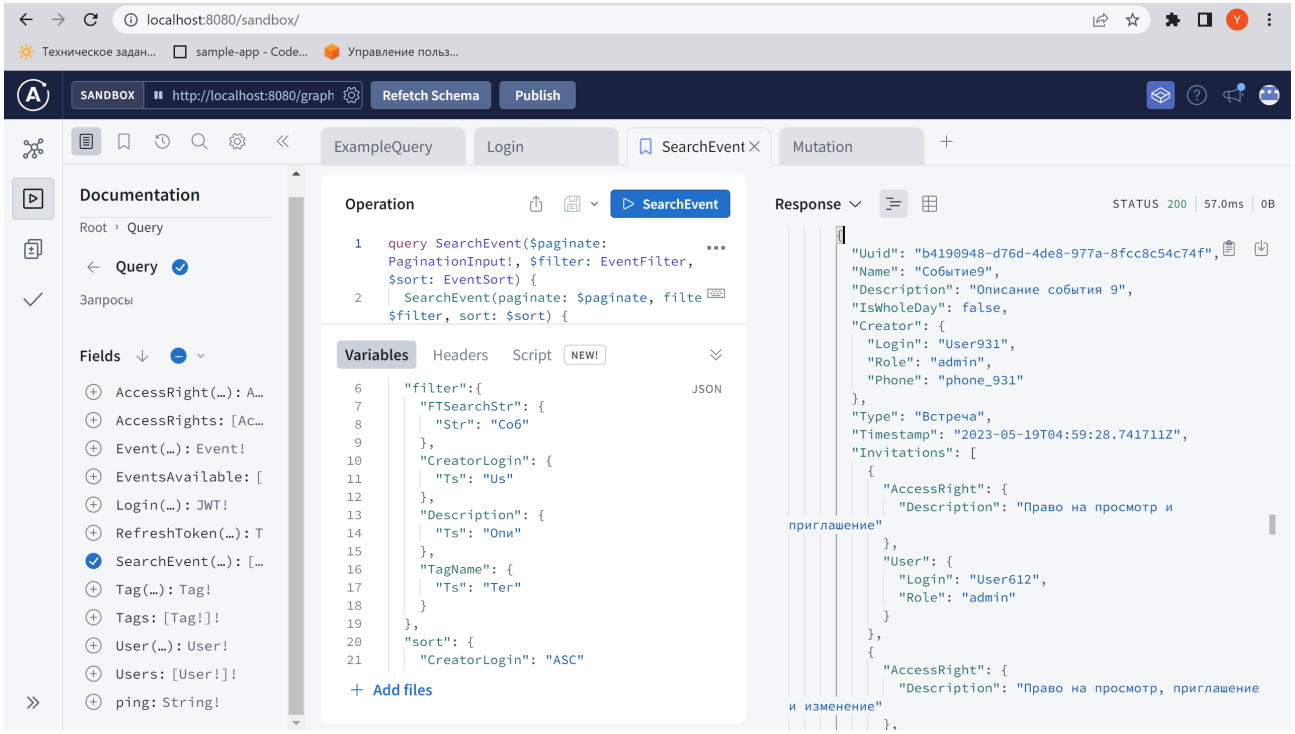

Рисунок 9 – Поиск событий с фильтрами и сортировкой

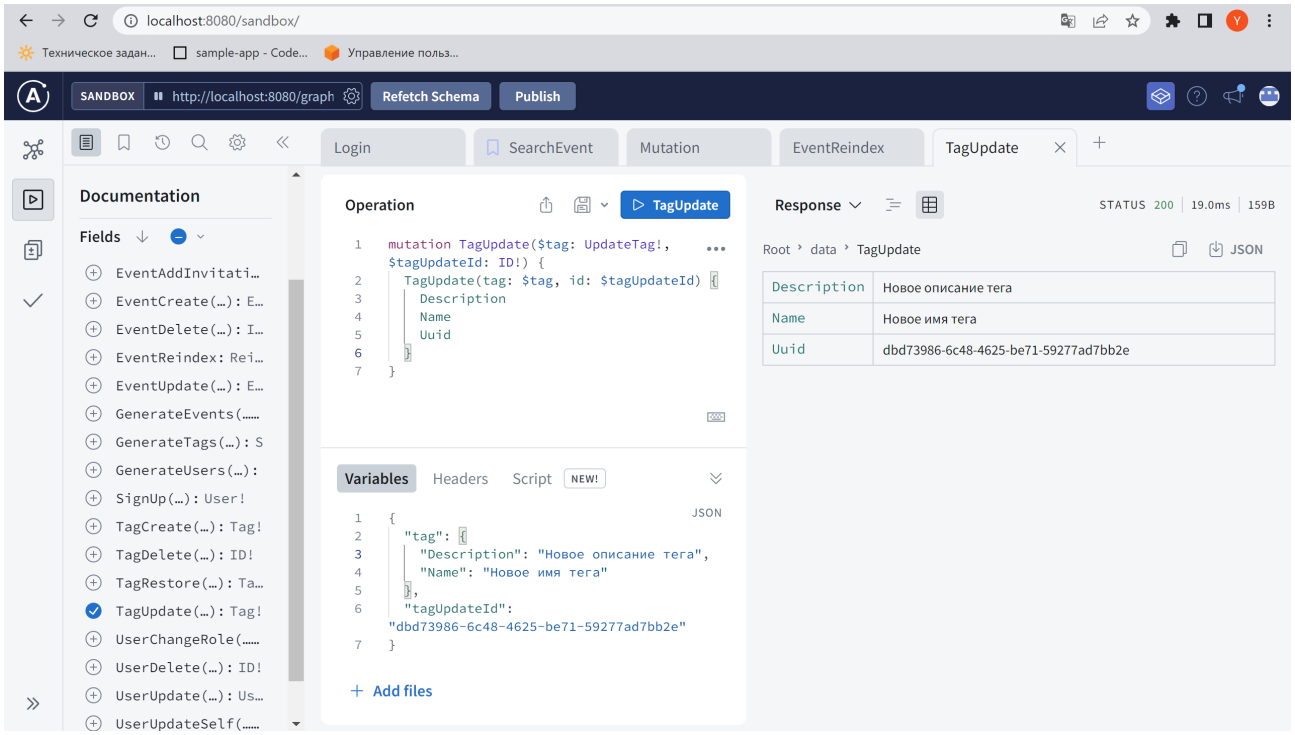

Рисунок 10 – Обновление тега

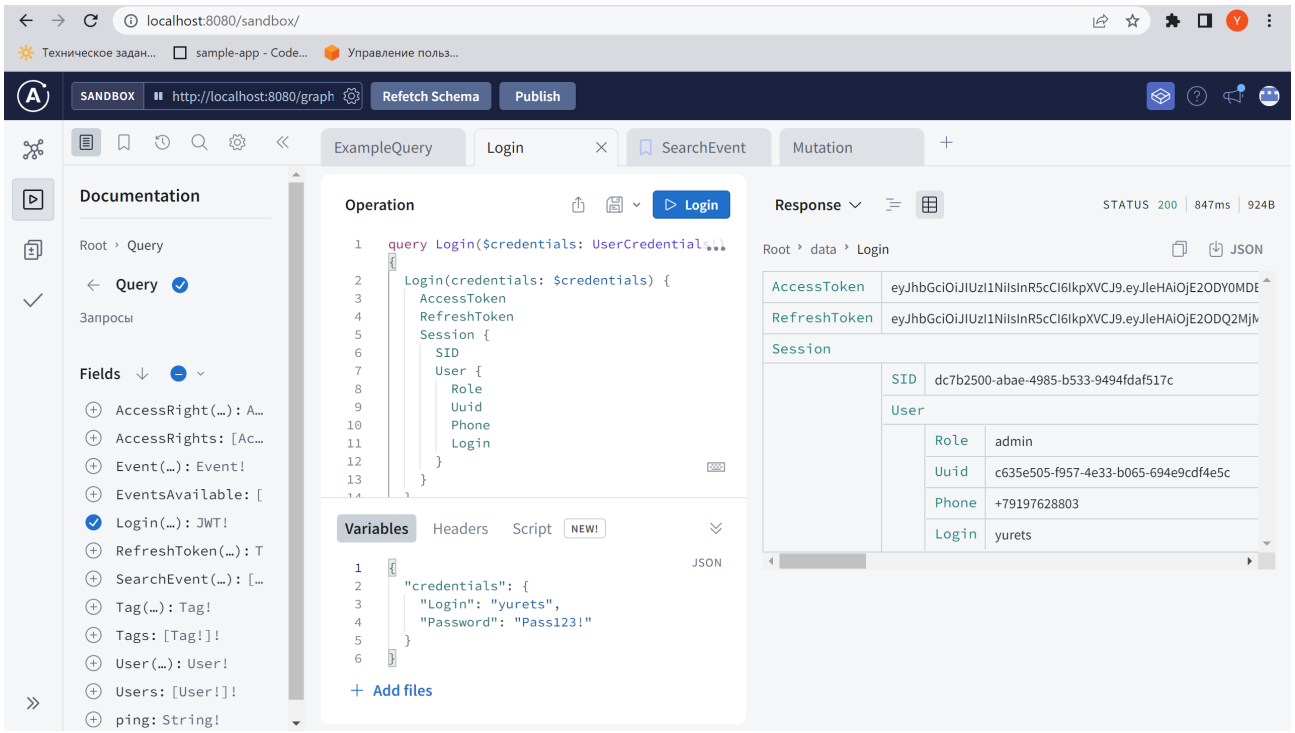

Рисунок 11 – Вход в систему (получение токена)

4 Экспериментальная часть

В данном разделе проведён сравнительный анализ реализаций сервиса поиска событий с использованием PostgreSQL и ElasticSearch

4.1 Технические характеристики

Замеры времени выполнялись на личном ноутбуке. Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

— операционная система Windows 10 Домашняя;
— память 16 Гбайт;
— процессор 3.20 ГГц 4-ядерный процессор Intel Core i5 11-го поколения;
— процессор имеет 4 физических и 8 логических ядер.

Во время замеров ноутбук был включен в сеть электропитания.

4.2 Время выполнения алгоритмов

Для замера времени работы алгоритмов использовалась функция time.Now() из стандартной библиотеки на Golang.

Для проведения замеров времени поиска с фильтрацией и сортировкой использовался запрос с параметрами, приведенными на листинге 6, а параметры запроса для замеров времени поиска без фильтрации и сортировки приведен на листинге 7. В обоих приведенных запросах X – задаваемое количество запрашиваемых записей.

Листинг 6 – Параметры запроса с указанием фильтрациии и сортировки

```
{
    "paginate": {
        "Page": 1,
        "PageSize": X
```

```
    },
    "filter":{
        "FTSearchStr": {
            "Str": "My"
        },
        "CreatorLogin": {
            "Ts": "Us"
        },
        "Description": {
            "Ts": "Desc"
        },
        "TagName": {
            "Ts": "Teg"
        }
    },
    "sort": {
        "CreatorLogin": "ASC"
    }
}

```

Листинг 7 – Параметры запроса без указания фильтрациии и сортировки

```
{
    "paginate": {
        "Page": 1,
        "PageSize": X
    },
}

```

Результаты замеров времени работы реализаций сервиса поиска с использованием PostgreSQL и ElasticSearch приведены в таблице 4. На рисунках 12 и 13 приведена графическая интерпретация замеров времени. В первом столбце таблиц указывается количество запрашиваемых записей, а во втором и третьем — время поиска с фильтрацией и без фильтрации соответственно. Замеры производились по 500 раз для каждого количества запрашиваемых записей. На мо-

мент осуществления замеров в таблице Postgres events было 100000 записей. Столько же документов были и в индексе ElasticSearch events.

Таблица 3 – Результаты замеров времени поиска с использованием PostgreSQL

Кол-во записей	С фильтрами, млс	Без фильтров, млс
10	13.833706	11.289715
20	14.250382	11.753059
50	14.960792	11.462349
100	16.769814	12.062963
200	20.102638	14.037806
300	21.950034	16.361748
400	28.771323	23.641660

Таблица 4 – Результаты замеров времени поиска с использованием ElasticSearch

Кол-во записей	С фильтрами, млс	Без фильтров, млс
10	67.975325	51.430142
20	69.194931	53.183822
50	72.861184	54.079701
100	77.778041	57.135915
200	86.193629	64.008997
300	93.549356	68.152554
400	98.196287	71.863563

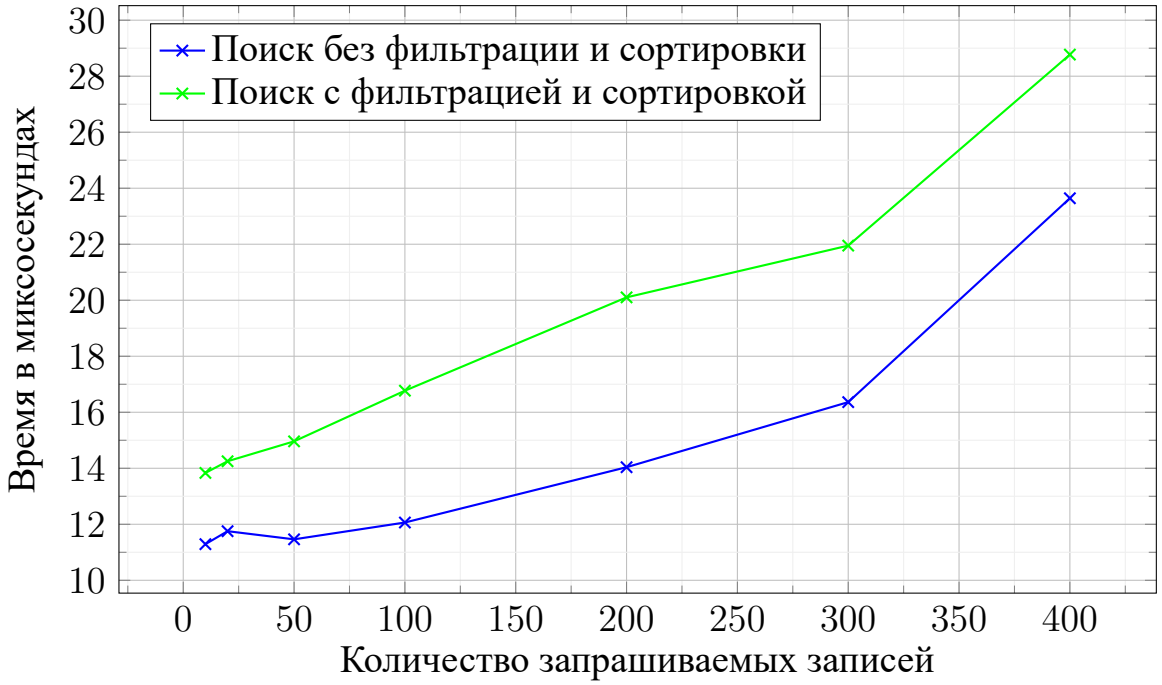

Рисунок 12 – Зависимость времени выполнения поиска в Postgres в зависимости от запрашиваемого количества записей

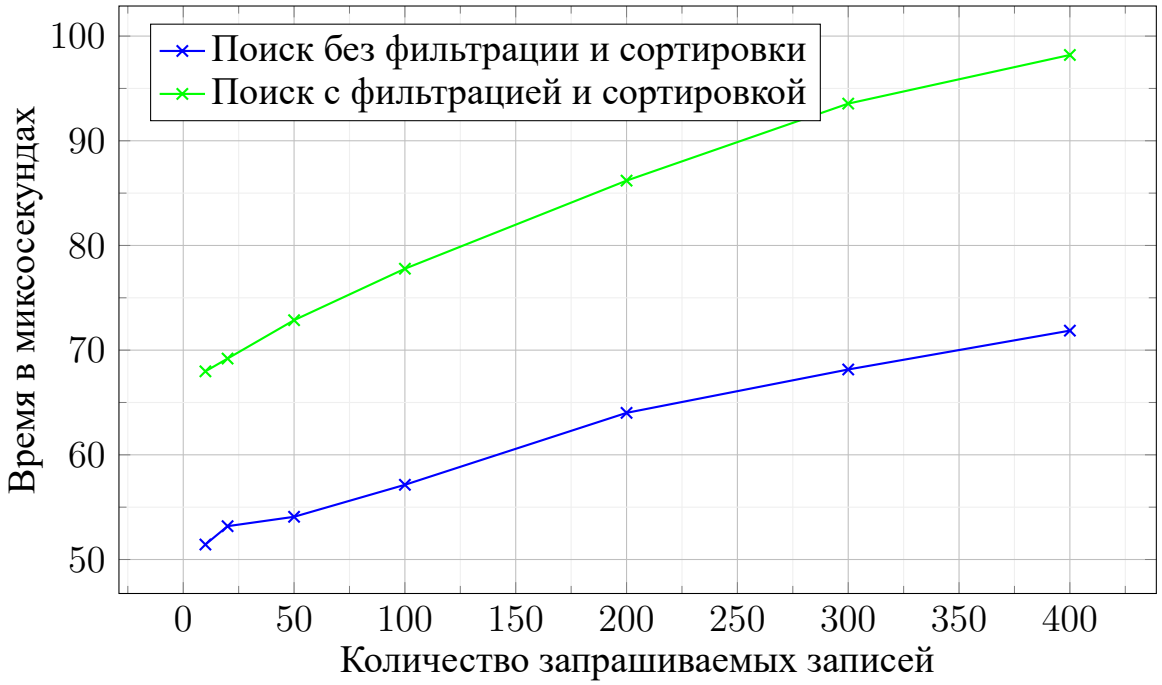

Рисунок 13 – Зависимость времени выполнения поиска в ElasticSearch в зависимости от запрашиваемого количества записей

Из результатов видно, что Postgres с фильтрами дольше примерно на 140%. Это обусловлено необходимостью применения фильтров к различным полям и необходимостью дополнительных JOIN-ов, т.к. некоторые фильтры используют данные из связанных таблиц.

Elastic с фильтрами дольше примерно на 130%. Это обусловлено необходимостью применения фильтров к различным полям и необходимостью сортировки результатов.

Можно заметить, что в среднем Elastic работает медленнее Postgres в 4 раза. Это может быть связанно с тем, что ElasticSearch использует протокол «HyperText Transfer Protocol» (HTTP) [18], в то время как Postgres взаимодействует с приложением по протоколу «Transmission Control Protocol» (TCP), который передает данные быстрее.

Вывод

Несмотря на необходимость выборки данных из нескольких таблиц, поиск событий в Postgres оказался более эффективным, чем поиск событий в ElasticSearch примерно в 4 раза.

Таким образом, для реализации сервиса поиска событий оптимальнее использовать Postgres, чем ElasticSearch.

ЗАКЛЮЧЕНИЕ

Поставленная цель была достигнута — разработана база данных для многопользовательского календаря и приложение, взаимодействующее с разработанной базой данных.

Для достижения поставленной цели были решены следующие задачи:

— проанализированы существующие решения;
— формализована задачу и определен необходимый функционал;
— рассмотрены модели баз данных и выбрана наиболее подходящая;
— проанализированы существующие СУБД и выбраны наиболее оптимальные;
— спроектирована и разработана базу данных;
— спроектировано и разработано приложение, взаимодействующее с разработанной базой данных;
— проведен сравнительный анализ эффективности фильтрации и сортировки при использовании различных СУБД.

Программный продукт представляет из себя приложение с API, разработанным согласно спецификации GraphQL. Приложение позволяет создавать события с указанием названия, временной метки, описания, типа, тегов и приглашенных пользователей. Также присутствует возможность для приглашенного пользователя просматривать, менять событие или приглашать других участников в зависимости от выданных ему прав доступа к событию. Помимо этого, реализован сервис поиска, позволяющий находить события по заданным фильтрам и предусматривающий возможность из сортировки.

Из результатов проведенного сравнительного анализа следует, что для реализации сервиса поиска событий оптимальнее использовать Postgres, чем ElasticSearch.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Яндекс.Календарь [Электронный ресурс]. — Режим доступа: https://calendar.yandex.ru/, свободный (дата обращения: 25.03.2023).

[2] Календарь Outlook [Электронный ресурс]. — Режим доступа: https://outlook.live.com/calendar/0/, свободный (дата обращения: 25.03.2023).

[3] Календарь Google [Электронный ресурс]. — Режим доступа: https://calendar.google.com/, свободный (дата обращения: 25.03.2023).

[4] OLAP [Электронный ресурс]. — Режим доступа: https://www.ibm.com/topics/olap, свободный (дата обращения: 27.03.2023).

[5] OLTP [Электронный ресурс]. — Режим доступа: https://www.ibm.com/topics/oltp, свободный (дата обращения: 27.03.2023).

[6] Роб, П., Коронелл, К. Базы данных: концепции, технологии, применение. - БХВ-Петербург: Вильямс, 2004. - 15 с.

[7] What is a Relational Database (RDBMS)? [Электронный ресурс]. — Режим доступа: https://www.oracle.com/database/what-is-a-relational-database/, свободный (дата обращения: 27.03.2023).

[8] Нереляционные данные и базы данных NoSQL [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/ru-ru/azure/architecture/data-guide/big-data/non-relational-data, свободный (дата обращения: 27.03.2023).

[9] GraphQL Specification [Электронный ресурс]. — Режим доступа: https://spec.graphql.org/October2021/, свободный (дата обращения: 17.04.2023).

[10] PostgreSQL [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/, свободный (дата обращения: 17.04.2023).

[11] Исходный код PostgreSQL [Электронный ресурс]. — Режим доступа: https://github.com/postgres/postgres, свободный (дата обращения: 20.05.2023).

[12] ElasticSearch [Электронный ресурс]. — Режим доступа: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html, свободный (дата обращения: 20.05.2023).

[13] Библиотека .ent [Электронный ресурс]. — Режим доступа: https://entgo.io/, свободный (дата обращения: 20.05.2023).

[14] Библиотека pq [Электронный ресурс]. — Режим доступа: https://pkg.go.dev/github.com/lib/pq, свободный (дата обращения: 20.05.2023).

[15] Библиотека gqlgen [Электронный ресурс]. — Режим доступа: https://gqlgen.com/, свободный (дата обращения: 20.05.2023).

[16] Библиотека fx [Электронный ресурс]. — Режим доступа: https://uber-go.github.io/fx/, свободный (дата обращения: 20.05.2023).

[17] GraphQL Sandbox [Электронный ресурс]. — Режим доступа: https://www.apollographql.com/docs/graphos/explorer/sandbox/, свободный (дата обращения: 20.05.2023).

[18] Протокол HTTP [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview, свободный (дата обращения: 20.05.2023).

ПРИЛОЖЕНИЕ А