МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 1–40 01 01 Программное обеспечение информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Реализация базы данных для веб-приложения мессенджер с применением технологии мультимедийных типов данных»

Выполнил студент Гайков Дмитрий Викторович
(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна
(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2023

Содержание

Введение 4
1 Постановка задачи 5
1.2 Аналитический обзор аналогов 5
1.2 Разработка функциональных требований, определение вариантов использования 7
1.3 Вывод 9
2 Проектирование базы данных 10
2.1 Обобщенная структура управлением приложения 10
2.2 Диаграммы UML, взаимосвязь всех компонентов. 10
2.3 Описание информационных объектов 11
2.4 Вывод 11
3 Разработка объектов базы данных 12
3.1 Создание необходимых объектов 12
3.2 Создание таблиц 12
3.3 Создание ролей для разграничения 13
3.4 Создание пакетов процедур для базы данных 13
3.4.1 Выборка данных из таблиц 16
3.4.2 Выборка данных по поисковому запросу 17
3.4.3 Заполнение таблиц 100 000 строк 17
3.4.4 Добавление данных в таблицы 18
3.4.5 Удаление данных в таблицы 18
3.4.6 Дополнительные функции 19
3.5 Представления базы данных 19
3.6 Индексы базы данных 20
3.7 Триггеры базы данных 21
3.8 Вывод 21
4 Описание процедур импорта и экспорта 22
5 Тестирование 23
6 Описание технологии и ее применение в базе данных 24
7 Краткое описание приложения для демонстрации 25
7.1 Краткое описание приложения для демонстрации со стороны посетителя 25
7.2 Краткое описание приложения для демонстрации со стороны пользователя 26
7.3 Краткое описание приложения для демонстрации со стороны администратора 26
7.4 Вывод 27
8 Руководство пользователя 28
8.1 Руководство пользователя 28
8.2 Установка приложения 33
8.3 Вывод 33
Заключение 34
Список используемых источников 35
Приложение А 36

Введение

Цель данной работы заключается в создании реляционной базы данных для веб-приложения мессенджер, которое позволит пользователям обмениваться сообщениями, голосовыми сообщениями и сообщениями в виде стикеров.

База данных – это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Postgres SQL, поскольку эта система обладает высокой надежностью и производительностью, что позволяет обеспечить эффективное хранение, обработку и управление музыкальными данными.

Также необходимо разработать приложение для демонстрации функциональности базы данных и взаимодействия с ней. Приложение было реализовано с использованием фреймворка NestJS для платформы Node.js и фреймворка React с TypeScript.

Для гарантированной безопасности пользователей приложения в курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении аудио и картинок.

Основные требования к приложению:

– реализация ролей администратора и пользователя;

– отправка, редактирование и удаление сообщений;

– добавление в друзья, удаление из друзей;

– возможность подписаться на другого или отписать от него;

– взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1 Постановка задачи

1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

Первый аналог – WhatsApp.

WhatsApp обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, обмен файлами и стикерами, а также возможность создания и управления каналами. Этот мессенджер также позволяет пользователям отправлять местоположение и документы.

WhatsApp имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в левой части экрана и списка чатов в правой части. WhatsApp также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн WhatsApp – довольно минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и зеленого, что создает ощущение легкости и чистоты.

Блок сообщения сделан закругленным. Текст для имени пользователя подсвечивается другим цветом. Текст черного цвета на белом фоне. На данное сообщение довольно-таки приятно смотреть.

Интерфейс приложения представлен на рисунке 1.1.

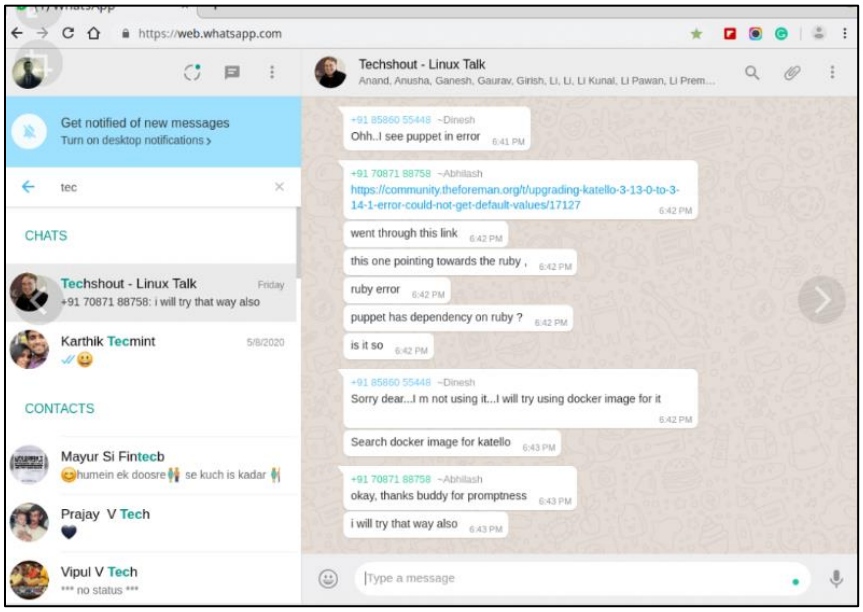

Рисунок 1.1 – Приложение «WhatsApp»

Второй аналог – это приложение Viber.

Viber позволяет отправлять текстовые сообщения, стикеры, фотографии, видео и аудиофайлы. Вы также можете создавать групповые чаты для общения с несколькими людьми одновременно. Viber позволяет совершать бесплатные

голосовые звонки и видеозвонки с другими пользователями Viber по всему миру. Для этого необходимо, чтобы оба пользователя были подключены к интернету. имеет светлый интерфейс с фиолетовыми элементами дизайна.

Главный экран отображает список ваших чатов, а также кнопки для совершения голосовых и видеозвонков. В верхней части экрана расположены кнопки для доступа к контактам, магазину стикеров и настройкам. В целом, интерфейс Viber интуитивно понятен и удобен в использовании.

Дизайн Viber представляет собой современный и лаконичный интерфейс с использованием ярких цветов и плоских иконок. Он имеет минималистичный дизайн, который делает приложение легким и простым в использовании.

Страница пользователя имеет картинку человека, на которого мы зашли. Кнопки в нем подсвечены и выделяются на общем фоне. Сразу видно, куда жать.

Интерфейс приложения представлен на рисунке 1.2.

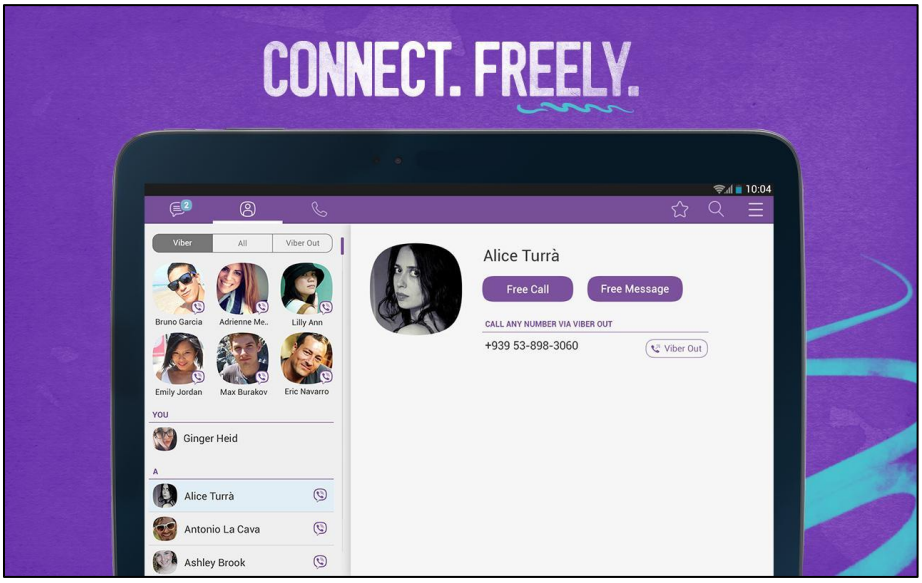

Рисунок 1.2 – Приложение «Viber»

Третий аналог – это Telegram.

Telegram обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, демонстрация экрана, обмен файлами и стикерами, а также возможность создания каналов и ботов. Этот мессенджер также предлагает возможность создания секретных чатов с функцией автоматического удаления сообщений.

Telegram имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в нижней части экрана и списка чатов в верхней части. Telegram также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн Telegram отличается от других мессенджеров своей минималистичностью и простотой в использовании. Он имеет темно-синий цветовой фон, который выделяет контент на экране. В верхней части экрана расположено главное меню, которое позволяет быстро переключаться между разделами мессенджера. В центре экрана находится список чатов и диалогов.

Каждый чат представлен в виде миниатюрного значка с изображением фото профиля пользователя или группы, а также отображается последнее сообщение. При нажатии на любой из чатов, открывается окно переписки, которое также имеет темно-синий фон и белый шрифт. Дизайн приятный и легко пользоваться им.

Интерфейс приложения представлен на рисунке 1.3.

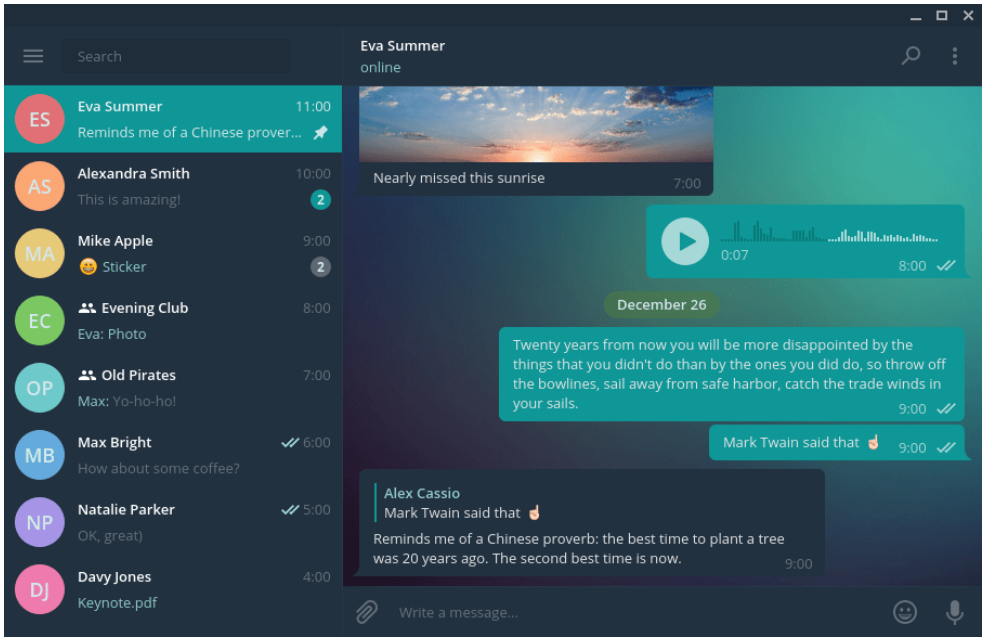

Рисунок 1.3 – Приложение «Telegram»

Анализируя аналоги WhatsApp, Viber и Telegram, можно сделать вывод, что все три приложения предлагают широкий спектр функций, включая обмен сообщениями, голосовые и видеозвонки, групповые чаты и возможность отправки файлов. Каждое из приложений имеет свою уникальную цветовую схему и дизайн интерфейса, но общий тренд – простота и удобство использования. Все они предлагают настраиваемый интерфейс с возможностью выбора тем оформления

1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для мессенджера, как будут хранится сообщения, как будут храниться аудиосообщения, как будут храниться стикеры и фотографии пользователей.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от

своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Роли пользователя – это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

– RL_VISITOR;
– RL_USER;
– RL_ADMIN;
– RL_MAINADMIN.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.4.

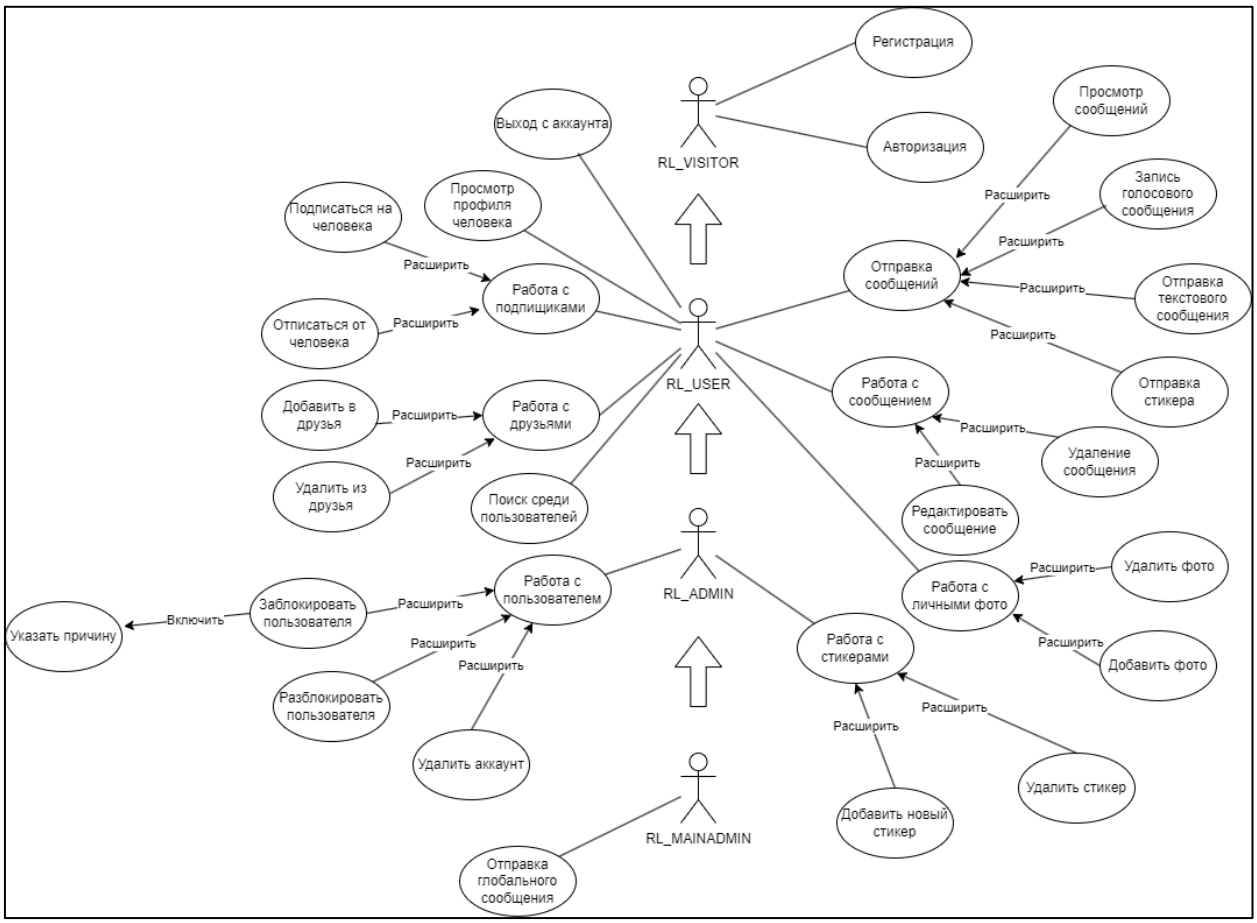

Рисунок 1.4 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является посетителем. Ему будут доступны лишь две функции: авторизация и регистрация. После регистрации пользователь становится пользователем RL_USER.

Роль RL_USER дает уже функцию отправлять разные виды сообщений, просматривать все свои сообщения, подписываться на кого-либо или отписаться от

кого кого-нибудь, добавлять человека в друзья и удалять людей из друзей, добавлять, удалять фото, искать пользователей, редактировать или удалять свои сообщения.

Роль RL_ADMIN может также просматривать чужие аккаунты, но цель найти недопустимую информацию. Если такая есть, он может заблокировать пользователя, указав при этом причину, или же вовсе удалить данный аккаунт. Также администратору доступно удаление или добавление стикеров.

Роль RL_MAINADMIN имеет доступ к тому же функционалу, что и RL_ADMIN, но также ей доступно отправлять глобальное сообщение, которое за раз дойдет всем пользователям приложения.

Приложение обладает четырьмя ролями, на которые разложены все основные функции для корректного использования.

1.3 Вывод

Итого, был проведен аналитический обзор аналогов мессенджеров и сервисов, которые уже существуют на рынке. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Также были определены функциональные требования базы данных, а также роли пользователей и варианты использования системы в зависимости от этих ролей. Была разработана UML-диаграмма, на которой отображены основные функции, которые доступны для каждой из ролей пользователей.

2 Проектирование базы данных

2.1 Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для общения должны входить функции для удобного поиска людей, функции добавления в друзья, различные виды сообщение и другие подобные функции.

2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.1.

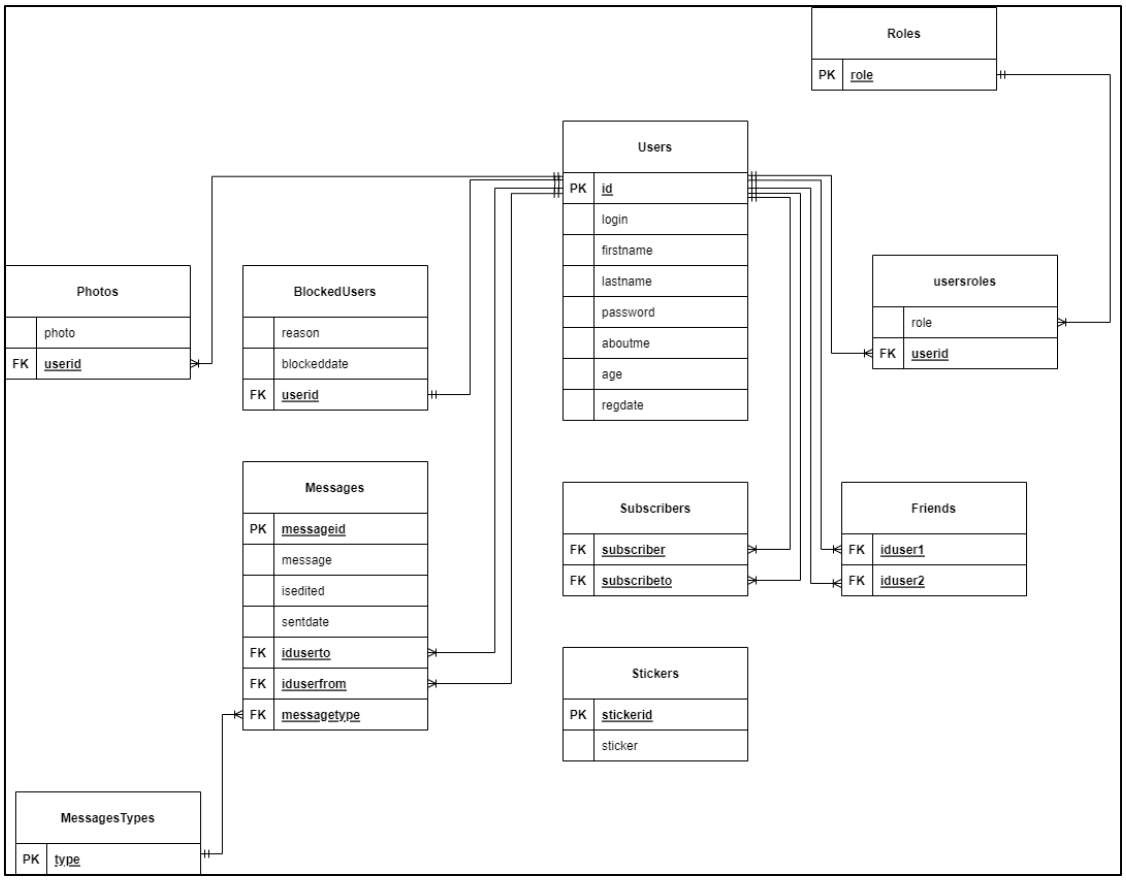

Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-одному", "один-ко-

многим", "многие-ко-многим". Например, таблица Users связана с таблицами Photos, BlockedUsers, UsersRoles, Messages, Subsribers, Friends через внешние ключи.

2.3 Описание информационных объектов

Для реализации базы данных было разработано 10 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Roles, Users, UsersRoles, BlockedUsers, MessagesTypes, Messages, Subscribers, Friends, Stickers и Photos. Ниже будет описание про каждую из них более подробно.

Таблица Roles отвечает за хранение всех ролей доступных в приложении. Поле здесь лишь одно – role varchar(20) primary key.

Таблица Users отвечает за хранение всех пользователей. Здесь следующие поля: id varchar(50) primary key, login varchar(20), firstname varchar(20), lastname varchar(20), password varchar(256), aboutme varchar(300), age integer, regdate timestamp.

Таблица UsersRoles отвечает за хранение ролей доступных для определенных пользователей. Здесь два поля: userid varchar(50) FK(users), role varchar(20) FK(roles).

Таблица BlockedUsers отвечает за хранение заблокированных администратором пользователей. Здесь три поля: userid varchar(50) FK(users), blockeddate timestamp, reason varchar(300).

Таблица MessagesTypes отвечает за хранение всех доступных типов сообщения. Здесь только одно поле: type varchar(20).

Таблица Messages отвечает за хранение сообщений между пользователями. Здесь следующие поля: iduserto varchar(50) FK(users), iduserfrom varchar(50) FK(users), message varchar(3000), messagetype varchar(20) FK(MessagesTypes), isedited boolean, sentdate timestamp, messageid varchar(256) primary key.

Таблица Subscribers отвечает за хранения подписчиков. Здесь два поля: subscriber varchar(50) FK(users), subscribeto varchar(50) FK(users).

Таблица Friends отвечает за хранение друзей. Здесь два поля: iduser1 varchar(50) FK(users), iduser2 varchar(50) FK(users).

Таблица Stickers отвечает за хранение всех доступных в приложении стикеров. Здесь два поля: sticker varchar(100), stickerid varchar(50) primary key.

Таблица Photos отвечает за хранения фотографий пользователей. Здесь два поля: userid varchar(50) FK(users), photo varchar(1000).

2.4 Вывод

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться.

3 Разработка объектов базы данных

3.1 Создание необходимых объектов

Для создания web-приложения мессенджера должны быть реализованы таблицы упомянутые в предыдущем разделе. Для более удобной работы с ними должны быть созданы индексы, функции, процедуры и представления. Скрипты создания таблиц представлены в приложении А.

3.2 Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличное пространство.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 3.1.

```
create tablespace DataTablespace
    location 'D:\DB\PSQL\data\RECVY\DataTB';

create tablespace IndexTablespace
    location 'D:\DB\PSQL\data\RECVY\IndexTB';

create tablespace SystemTablespace
    location 'D:\DB\PSQL\data\RECVY\SystemTB';
```

Листинг 3.1 – Скрипт для создания табличных пространств

Для базы данных создадим десять основных таблиц: Roles, Users, UsersRoles, Photos, BlockedUsers, Stickers, Friends, Subscribers, MessagesTypes, Messages.

В табличном пространстве DataTablespace будут храниться данные из таблицы BlockedUsers, Friends, Subscribers, Users, Messages и Photos. Тут собраны данные связанные напрямую с пользователем. В табличном пространстве IndexTablespace будут хранится все индексы данной БД. В табличном пространстве SystemTablespace будут хранится системные данные такие, как Stickers, Roles, UsersRoles, MessagesTypes.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у каждого пользователя может быть множество фотографий, которые он добавил. Для этого мы добавим в таблицу Photos внешний ключ (FOREIGN KEY) на таблицу Users, который будет указывать на идентификатор пользователя.

Другой тип связи - это связь "многие ко многим" (many-to-many), которая определяет отношение между множеством записей в одной таблице и множеством записей в другой таблице. Например, есть три таблицы Users, Roles, UsersRoles, у множества пользоватей может быть множество ролей так же, как и у множества ролей может быть множество пользователей. Скрипт создание таблицы User будет представлен на листинге 3.2.

```
create table Users (
    id varchar(50) primary key,
    login varchar(20) unique,
    firstName varchar(20),
    lastName varchar(20),
    password varchar(256),
    aboutMe varchar(300) null,
    age integer check (
        age > 6 and
        age < 150
    ),
    regDate timestamp default CURRENT_TIMESTAMP,
    role varchar(20) default 'user',
    foreign key (role) references roles(role)
) tablespace DataTablespace;
```

Листинг 3.2 – Скрипт создание таблицы User

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы четыре табличных пространства: TS_USER, TS_TRACK, TS_PLAYLIST и TS_LIBRARY, в каждом из которых будут храниться соответствующие таблицы.

3.3 Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

3.4 Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц.
2. Выборка данных по поисковому запросу.
3. Заполнение таблиц 100 000 строк.

4. Добавление данных в таблицы.
5. Удаление данных из таблиц.
6. Изменение данных в таблицах.
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

Будут созданы четыре роли для разграничения доступа к базе данных: RL_VISITOR, RL_USER, RL_ADMIN, RL_MAINADMIN. Описание этих ролей будет идти далее.

Роль RL_VISITOR имеет доступ лишь к функциям, которые позволяют добавить нового пользователя. То есть ей доступна, вставка в таблицы Users, Photos, UsersRoles, выборка из таблиц Users, BlockedUsers, Photos, Roles, UsersRoles, также это роли доступно выполнение двух функций signin и signup, а еще и процедуры setuserrole. Реализация данной роли показана на листинге 3.3.

```
create role RL_VISITOR;

grant connect on database recvy to RL_VISITOR;

grant SELECT on users to RL_VISITOR;
grant INSERT on users to RL_VISITOR;

grant insert on photos to RL_VISITOR;
grant select on photos to RL_VISITOR;

grant select on blockedusers to RL_VISITOR;
grant select on roles to RL_VISITOR;
grant select on usersroles to RL_VISITOR;
grant insert on usersroles to RL_VISITOR;
grant execute on procedure setuserrole to RL_VISITOR;
grant execute on FUNCTION signin to RL_VISITOR;
grant execute on FUNCTION signup to RL_VISITOR;
```

Листинг 3.3 – Реализация роли RL_VISITOR

Роль RL_USER имеет право на выполнение всех операций, связанных с объектами, хранящимися в DataTablespace, что позволяет этой роли пользоваться всем функционалом данной БД. Выданные привилегии роли RL_USER можно увидеть на листинге 3.4.

```
create role RL_USER;
grant RL_VISITOR to RL_USER;
grant insert on friends to RL_USER;
grant delete on friends to RL_USER;
grant select on friends to RL_USER;
grant insert on messages to RL_USER;
grant delete on messages to RL_USER;
grant update on messages to RL_USER;
grant select on messages to RL_USER;
grant select on messagestypes to RL_USER;
grant select on stickers to RL_USER;
grant insert on subscribers to RL_USER;
grant delete on subscribers to RL_USER;
grant select on subscribers to RL_USER;
grant delete on photos to RL_USER;
grant select on users to RL_USER;
grant update on users to RL_USER;
grant select on returned_users to RL_USER;
```

Листинг 3.4 – Привилегии, выданные роли RL_USER

Роль RL_ADMIN имеет все те же права, что и предыдущая роль, но плюс ко всему ей доступна работа с объектами из SystemTablespace. Данная работа включает возможность заблокировать и разблокировать пользователя, удалить аккаунт пользователю, добавить или удалить стикеры и тому подобные. Выданные привилегии роли RL_ADMIN можно увидеть на листинге 3.5.

```
create role RL_ADMIN;

grant RL_USER to RL_ADMIN;

grant execute on function messagetojson to RL_ADMIN;
grant execute on procedure usersfromjson to RL_ADMIN;

grant delete on stickers to RL_ADMIN;
grant insert on stickers to RL_ADMIN;
grant update on stickers to RL_ADMIN;

grant delete on blockedusers to RL_ADMIN;
grant insert on blockedusers to RL_ADMIN;
grant update on blockedusers to RL_ADMIN;
grant update on messagestypes to RL_ADMIN;
grant delete on messagestypes to RL_ADMIN;
grant insert on messagestypes to RL_ADMIN;
grant delete on roles to RL_ADMIN;
grant insert on roles to RL_ADMIN;
grant update on roles to RL_ADMIN;
grant delete on users to RL_ADMIN;
grant delete on usersroles to RL_ADMIN;
grant select on blocked_users to RL_ADMIN;
```

Листинг 3.5 – Привилегии, выданные роли RL_ADMIN

Роль RL_MAINADMIN позволяет делать тоже самое, что и RL_ADMIN, но плюс ко всем ей становится доступным функционал для создания глобального сообщения с помощью функции sendToEvereone.

3.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: getBlockedUsers, searchBlockedUsers, getChats, getFriends, searchFriends, getMessages, getPhotosById, getStickers, getSubscribersOf, searchSubscribers, getUsers, searchUser. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

GetUsers – функции для выборки всех пользователей приложения.

GetBlockedUsers - функция для выборки списка всех заблокированных пользователей. Возвращаемое множество похоже на результат GetUsers, только плюс ко всему еще есть поле причины блокировки.

GetChats – функция позволяющая сгенерировать чаты на основе всех сообщений.

GetFriends - функции для выборки друзей.

GetMessages – для получения всех сообщений между двумя пользователями.

GetPhotos, GetStickers - функции для получения фотографий пользователя и стикеров приложения.

GetSubscribersOf – функции для получения списка подписчиков определенного пользователя.

На листинге 3.6 будет функция GetUsers, которая предназначена для выборки всех пользователей из таблицы Users.

```
create or replace function getUsers(skip integer = null, take
integer = null)
returns setof returned_users as $$
begin
    call checkskipandtake(skip, take);

    return query (
        select
            *
        from
            returned_users
        offset skip
        limit take
        );

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;
```

Листинг 3.6 – Функция GetUsers

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

3.4.2 Выборка данных по поисковому запросу

Для поиска были разработаны разные функции. SearchUser для поиска среди всех пользователей, searchFriends для поиска среди друзей определенного пользователя, searchSubs для поиска среди всех подписчиков определенного пользователя. Данные функции принимают текст и по нему ищут пользователей по фамилии, или имени, или совместно. Пример одной из функций представлен на листинге 3.7.

```
create or replace function searchUser(stext text, skip integer =
null, take integer = null)
returns setof returned_users as $$
    declare
        searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);
    return query
   (
    select
        *
    from
        returned_users as u
    where
        lower(u.firstname || ' ' || u.lastname) ilike searchtext  or
        lower(u.lastname || ' ' || u.firstname) ilike searchtext or
        lower(u.firstname) ilike searchtext or
        lower(u.lastname) ilike searchtext
    offset skip
    limit take);

    exception
        when others then
            raise exception '%', SQLERRM;
end;
$$ language plpgsql;
```

Листинг 3.7 – Функция searchUser

Таким образом, в данном разделе были представлены примеры функций, которые позволяют искать треки.

3.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы Messages была разработана процедура fill, которая вставляет 100000 строк в таблицу. Данная процедура использует цикл while, который с каждой итерацией увеличивает значение переменной i на 1 и делает новую запись в таблицу Messages. Процедура представлена на листинге 3.8.

```
create or replace procedure fill()
as $$
declare
  i int;
  _messageid text;
begin
    i := 1;
    while i <= 100000 loop
        select messageid into _messageid from
sendMessage('IMSv8XlkJwtsBVPKpUY18zQ8KT27BGF6kZdm73XJH890hm8sYw',
'IMSv8XlkJwtsBVPKpUY18zQ8KT27BGF6kZdm73XJH890hm8sYw', 'Hello mir '
|| cast(i as text), 'text');
        i := i + 1;
    end loop;
end;
$$ language plpgsql;
```

Листинг 3.8 – Процедура заполнения таблицы Messages

Данная функция заполняет таблицу в течении несколько секунд. Производительность базы данных будет проверена дальше.

3.4.4 Добавление данных в таблицы

Были разработаны множество функций для добавления данных в таблицы. SignUp для реализации регистрации пользователя. SendMessage функция для добавления нового сообщения. BlockUser для добавления пользователя в таблицу BlockedUsers. AddFriend функция для добавления записи в таблицу Friends. Функция Subscribe для создания новой записи в таблицу Subscribers. AddSticker функция для добавления нового стикера.

Реализации этих функций и процедур будет в приложении А.

3.4.5 Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры: unBlockUser для разблокировки пользователя, deleteFromFriend для удаления из друзей, dellChat для удаления чата, dellMessage для удаления собщения, dellPhoto для удаления фотографии, deleteSticker для удаления стикера, describe для удаления записи из таблицы Subscribers.

На листинге 3.9 будет процедура dellMessage.

```
create or replace procedure dellMessage(id varchar(256))
language plpgsql as $$
begin
    delete from messages where messageid = $1;
end;
$$;
```

Листинг 3.9 – Процедура для удаления сообщения

Все остальные процедуры будут аналогичны, также предназначены для удаления соответствующих данных из основных таблиц базы данных.

3.4.6 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Функция toHash создана дял хеширования пароля. Она принимает пароль и возвращает захешированное значение . Функция представлена на листинге 3.10.

```
create or replace function toHash (password varchar(256))
returns varchar(256)
as $$
begin
    return sha256(password::bytea);
end;
$$ language plpgsql;
```

Листинг 3.10 – Функция toHash

Функция generateException для создания пользовательского исключения. Принимает ключ и значение, а возвращает текст «ключ:значение» как единую строку. Функция представлена на листинге 3.11.

```
create or replace function generateException(key text, error text)
returns text
as $$
begin
    return key || ':' || error;
end;
$$ language plpgsql;
```

Листинг 3.11 – Функция generateException

В целом, эти функции могут быть полезны для повышения безопасности хранения паролей пользователей, а также для получения дополнительной информации о треках в базе данных.

3.5 Представления базы данных

Представление в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы два представления:

– returned_users, которое позволяет вернуть пользователей с включенными массивами фотографий и ролей;

– blocked_users, которые позволяет вернуть заблокированных пользователей, но с включенными массивами фотографий и ролей плюс ко всем с причиной и датой блокировки.

Представление returned_users представлено на листинге 3.12.

```
     CREATE OR REPLACE VIEW returned_users AS
SELECT
    users.id,
    users.firstname,
    users.lastname,
    users.aboutme,
    users.age,
    users.regdate,
    getphotosbyid(users.id) AS photos,
    getrolesof(users.id) AS role
FROM
    users;
```

Листинг 3.12 – Представление returned_users

Данные представления помогаю работать с таблицами более гибко.

3.6 Индексы базы данных

Индекс – объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Для быстрой генерации чатов и для быстрого поиска сообщений были разработаны 4 индека для таблицы Messages. IDX_MESSAGEID создан для более быстро поиска при удаление или изменении сообщения. IDX_IDUSERFROM и IDX_IDUSERTO созданы для более быстрого поиска сообщений относящихся к определенному пользователю. IDX_USERISERTO_IDUSERFROM создан для более быстрой генерации чатов. Сами индексы представлены на листинге 3.13.

```
create unique index idx_messageid on messages(messageid) tablespace
indextablespace;

create index idx_iduserfrom on messages(iduserfrom) tablespace
indextablespace;
create index idx_iduserto on messages(iduserto) tablespace
indextablespace;
create index idx_iduserto_iduserfrom on messages(iduserto,
iduserfrom) tablespace indextablespace;
```

Листинг 3.13 – Индексы базы данных для таблицы Messages

Кроме этих индексов, есть и другие. Они позволяют более быстро искать пользователей, находить фотографии для определенного пользователя и многое другое. Эти индексы представлены в приложении А.

3.7 Триггеры базы данных

Триггер базы данных – это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для обеспечения целостности данных и контроля доступа к данным, а также для автоматической обработки данных при выполнении определенных операций в таблице.

Первый триггер создан для того, чтобы захешировать пароль перед вставкой в таблицу Users. Данный триггер представлен в листинге 3.14.

```
create or replace function signUpTrigger()
returns trigger
as $$
begin
    new.password := tohash(new.password);
    return new;
end;
$$ language plpgsql;
```

Листинг 3.14 – Скрипт триггера signUpTrigger

Второй триггер setRoleTrigger отвечает за выдачу роли пользователя после операции вставки в таблицу Users.

```
create or replace function setRoleTrigger()
returns trigger
as $$
begin
    call setuserrole(new.id);
    return new;
end;
$$ language plpgsql;
```

Листинг 3.4 – Скрип триггера setRoleTrigger

Два эти триггер созданы для процесса регистрации пользователя.

3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для мессенджера, где мы увидели основные объект данной базы данных.

4 Описание процедур импорта и экспорта

База данных имеет возможность импортировать данные для таблицы Messages в формат JSON и экспортировать данные из JSON в таблицу Users. Это может быть полезно в случае необходимости переноса данных на другой сервер или резервного копирования данных.

Для импорта была разработана функция messageToJson. Она позволяет вернуть сообщения для определенного пользователя или же все сообщения. Также она позволяет ограничить количество сообщений на выходе за счет параметров skip и take. Данная функция представлена в листинге 4.1.

```
create or replace function messageToJson(
    _from varchar(50) = null, _to varchar(50) = null,
    skip integer = null, take integer = null)
    returns json
    language plpgsql as $$
declare json json;
begin
    call checkskipandtake(skip, take);
    create temporary table _messages as
    select *
    from messages
    where iduserfrom = _isnull($1, iduserfrom)
      and iduserto = _isnull($2, iduserto)
    offset skip limit take;
    _json := (select json_agg(to_json(_messages))
              from _messages);
    drop table _messages;
    return _json;
exception
    when others then
        raise exception '%', SQLERRM;
end;
$$;
```

Листинг 4.1 – Функция messageToJson

Для экспорта из формата JSON была разработана функция usersFromJson. Которая принимает JSON-текст, преобразовывает и регистрирует пользователей. Данная функция представлена в листинге 4.2.

```
create or replace procedure usersFromJson(_json json) as $$ begin
    SELECT signup(login, firstname, lastname, age, password)
    FROM json_populate_recordset(null::users, $1);
end; $$ language plpgsql;
```

Листинг 4.2 – Функция usersFromJson

Данные функции могут пригодится в будущем для создания архивов сообщений, для резервного копирования данных из БД и во многом другом.

5 Тестирование

Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица Messages, содержащая больше всего данных. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

```
explain analyse select * from
getchats('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
```

Листинг 5.1 – Запрос к таблице Messages

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 9451.419 мс, а время планирования – 0.118 мс. Результаты запроса будут представлены на рисунке 5.1.

	QUERY PLAN
1	Function Scan on getchats (cost=0.25..10.25 rows=1000 width=168) (actual time=9432.715..9432.716 rows=5 lo
2	Planning Time: 0.118 ms
3	Execution Time: 9451.419 ms

Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса можно создать четыре индекса. Первый позволит более быстро искать сообщение по идентификатору сообщения, другие индексы ускорят работу с поиском по идентификаторам пользователей в полях iduserto, iduserfrom. Результат будет представлен на рисунке 5.2.

	QUERY PLAN
1	Function Scan on getchats (cost=0.25..10.25 rows=1000 width=168) (actual time=23.270..23.271 rows=5 loops=
2	Planning Time: 0.050 ms
3	Execution Time: 23.299 ms

Рисунок 5.2 – Результат выполнения запроса

После создания индексов видно, что чаты стали генерировать намного быстрее. Время планирования заняло 0.050 мс, а время выполнения 23.299 мс, что более чем в 300 раз быстрее.

Результаты тестирования говорят о том, что индексы помогают быстрее сформировать чаты из данных из таблицы Messages.

6 Описание технологии и ее применение в базе данных

В данной базе данных используется мультимедийность для хранения музыкальных файлов и изображений. Например, таблица Photos хранит фотографии пользователей. Поле photo хранит название файла фотографии, чтобы сервер на Node.js понял, какой файл взять и отправить клиенту. По такому принципу работает и таблица Stickers, где поле sticker хранит название файла со стикером.

Когда пользователь на стороне клиента загружает мультимедийные файлы (например, изображения), они отправляются на сервер Node.js для обработки и сохранения в базе данных. Для этого файлы передаются на сервер Node.js, который сначала переименовывает файл, а потом сохраняет его в статическую папку на сервере, а имя файла сохраняет в БД. Пример скрипта для добавления фотографии во время регистрации представлен на листинге 6.1.

```
async signUp(signUpDto : SignUpDto, file : MemoryStoredFile) :
Promise<Token> {
  if(file) {
    const newFileName = await generateString(250, true);
    this.filesService.rename(file, newFileName);
    signUpDto.photo = file.originalName;
  }

  const id : string = await this.usersService.addUser(signUpDto);

  if(file) {
    await this.filesService.saveUserFile(file);
  }

  return await this.jwtService.signAsync({ id });
}
```

Листинг 6.1 – Скрипт регистрации с сохранением фотографии пользователя

Когда же фотографии запрашиваются из БД, сервер получает имя файла из БД, но он дописывает путь относительно статической папки на сервере. Скрипт получения нового пути к файлу представлен на листинге 6.2.

```
getNameForUserImg(name : string) : string {
  return `/static/img/users/${name}`
}
```

Листинг 6.2 – Скрипт преобразования имени файла

Таким образом, была рассмотрена тема хранения мультимедийных данных в базах данных. Мы рассмотрели, как фотографии пользователей хранятся в БД.

7 Краткое описание приложения для демонстрации

7.1 Краткое описание приложения для демонстрации со стороны посетителя

Когда человек зайдет на сайт и до этого он не входил в аккаунт, ему будут доступны лишь две функции: регистрации и авторизация.

Форма регистрации содержит 6 полей: логин, который должен состоять от 6 до 20 символов русского или английского алфавита, а также чисел; пароль, который должен содержать как минимум 8 символов и как максимум 64 символа; имя, которое должно содержать имя пользователя, оно должно состоять только из либо русский, либо английских букв; фамилия, которое по правилам идентично полю имени; возраст, которое должно содержать число от 6 до 150 лет; файловое поле, которое является необязательным и может содержать ссылку на файл-изображения.

Форма авторизации содержит два поля: логин и пароль. Этих данных хватит для прохождения идентификации, аутентификации и авторизации. Они были описаны в прошлом абзаце.

При вводе неверных данных пользователь получит ошибки и уточнения к ним. Отображение ошибки показано на рисунке 7.1.

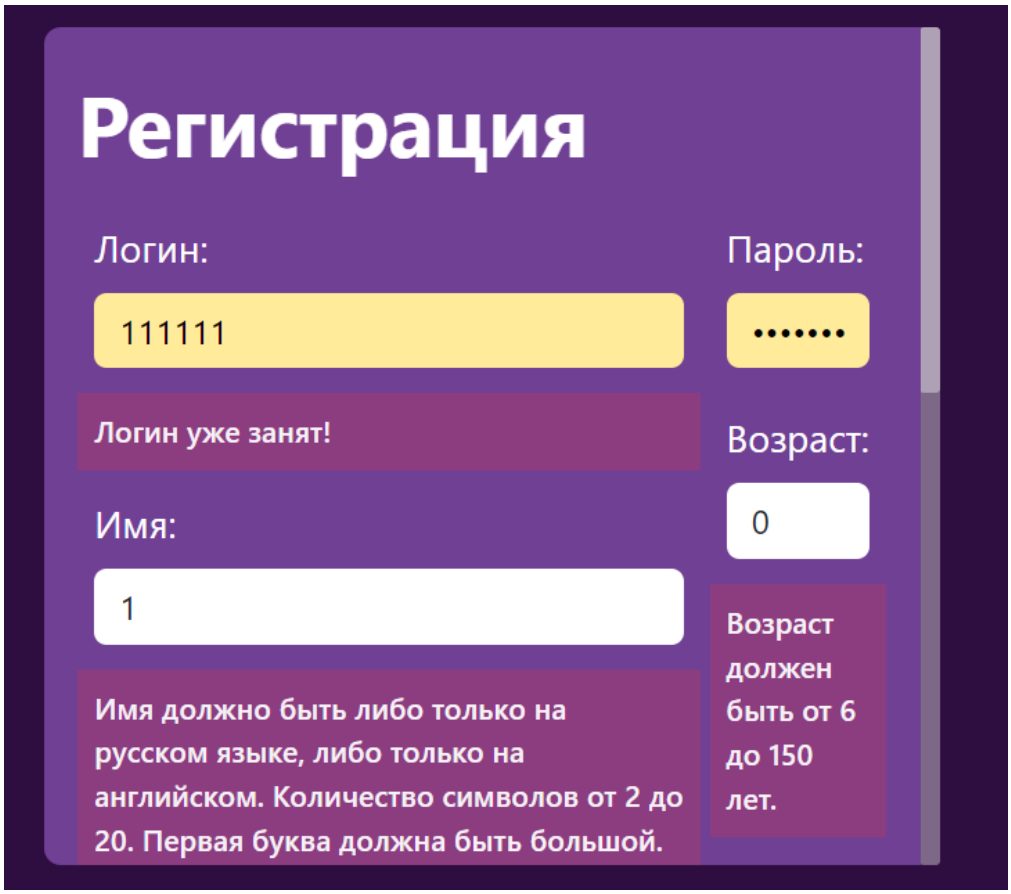

Рисунок 7.1 – Отображения ошибок при регистрации

Тем самым пользователю доступно лишь две функции и, чтобы воспользоваться приложением полностью, он должен создать или войти в существующий аккаунт.

7.2 Краткое описание приложения для демонстрации со стороны пользователя

Пользователю приложения доступно уже намного больше функций. Он может отправлять текстовые, голосовые сообщения и сообщения в виде стикеров. Также ему доступен раздел «Люди», где он может найти себе друзей, подписываться на других людей и начинать общение с ними.

Основная задача мессенджера – возможность общаться. Если пользователь отправил неправильное сообщение, то он может отредактировать его или вовсе удалить его. Также пользователь может удалить весь чат без восстановления.

Также пользователь может придумать себе уникальное описание или же изменить старые данные в меню редакции.

Приложение рассчитано на огромное количество пользователей, и под одним именем может скрывать несколько разных людей, и для более точно поиска нужного человека введена функция добавления нескольких фотографий. Она позволит узнать человека по лицу.

Пользователю доступно намного большем, чем обычному посетителю. Эта роль позволяет насладиться всем функционалом приложения RECVY.

7.3 Краткое описание приложения для демонстрации со стороны администратора

Администратор является обычным пользователем, но с большим функционалом, который позволяет сохранять в приложении порядок.

Главный функции администратора – функции блокировки пользователя и удаления аккаунта. Для того, чтобы заблокировать пользователя, нужно указать причину, которая будет указана пользователю после попытки входа в аккаунт. Пример попытки входа на заблокированный аккаунт показан на рисунке 7.2. Для удаления кого-либо аккаунта нужно просто лишь нажать на кнопку в разделах «Люди» или «Нарушители». Удаление с невозможным восстановлением.

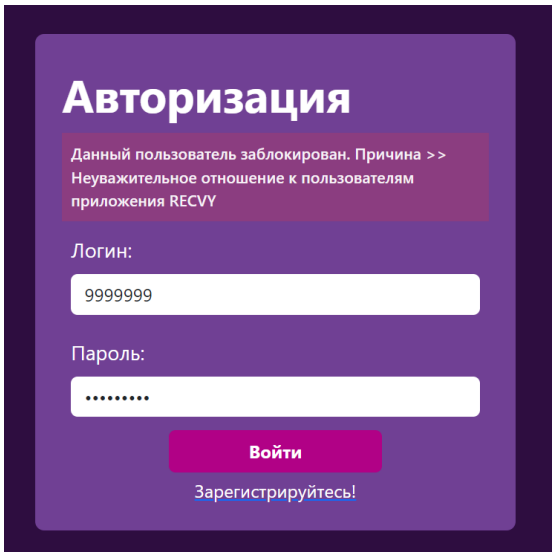

Рисунок 7.2 – Попытка входа на заблокированный аккаунт

Имя администратора на странице чатов подсвечено розовым цветом и при наведении будет надпись, что данный пользователь является администратором, как показано на рисунке 7.3.

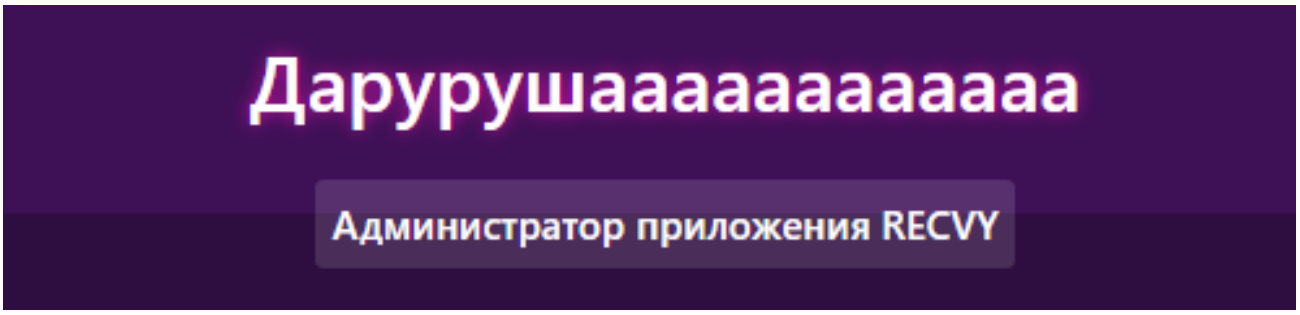

Рисунок 7.3 – Аккаунт администратора

Аккаунт администратора позволяет следить за порядком в приложении и по ситуации наказывать нарушении.

7.4 Вывод

В данном разделе были рассмотрены все основные роли, которые были реализованы в приложении, а также было рассмотрено, какие привилегии они дают. Приложение предоставляет 3 роли, на которые распределен весь функционал, что позволяет улучшить систему безопасности внутри сайта. Система ролей способствует эффективному использованию ресурсов приложения. Например, посетителям доступны только базовые функции, тогда как администраторам предоставлены расширенные возможности управления. Это позволяет каждой категории пользователей получать доступ к тем функциям, которые соответствуют их ролям и потребностям в приложении.

8 Руководство пользователя

8.1 Руководство пользователя

При открытии веб-приложения RECVY вас будет встречать форма авторизации, в которой вы можете ввести данные и зайти под определенным пользователем или же вы можете нажать на гиперссылку «Зарегистрироваться» и перейти на страницу регистрации. Рисунок формы авторизации представлен на рисунке 8.1.

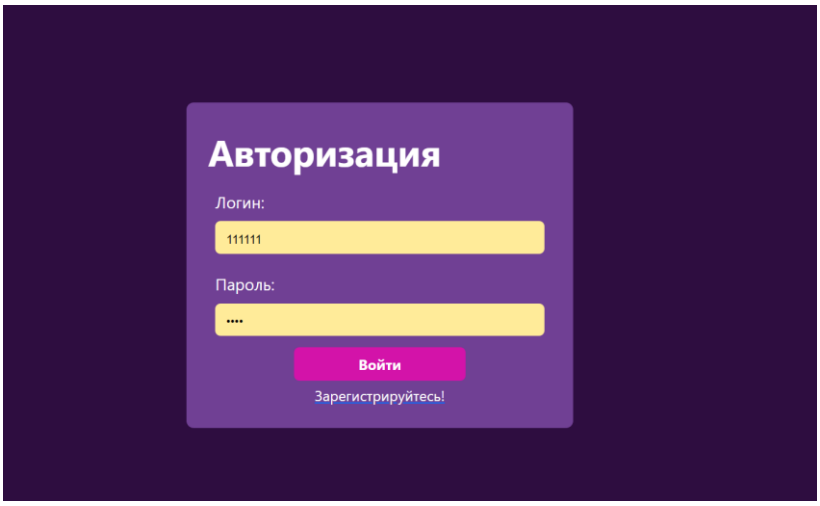

Рисунок 8.1 – Главная страница

После входа под определенным пользователем вам откроется главная страница приложения, где вы сможете увидеть чаты с людьми, с которыми вы переписывались ранее. Также здесь видно, кто из этих людей онлайн. Также на данной странице вам доступно навигационное меню, где вы можете перейти на страницу со всеми пользователями приложения, где будут находится ваши друзья и подписчики. Главная страница представлена на рисунке 8.2.

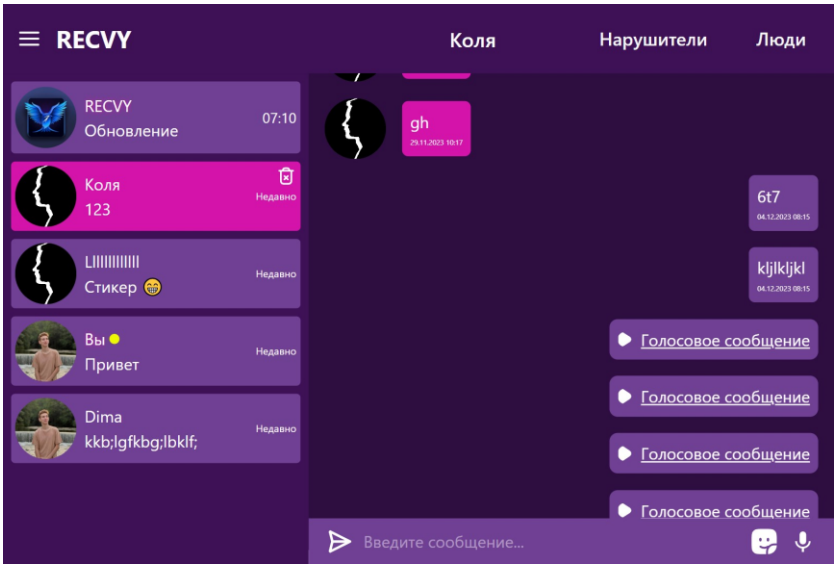

Рисунок 8.2 – Главная страница

Пользователь может создать новый аккаунт. Для этого ему надо перейти на страницу регистрации, где ему будет доступна интуитивно понятная форма. Рисунок формы регистрации представлен на рисунке 8.3.

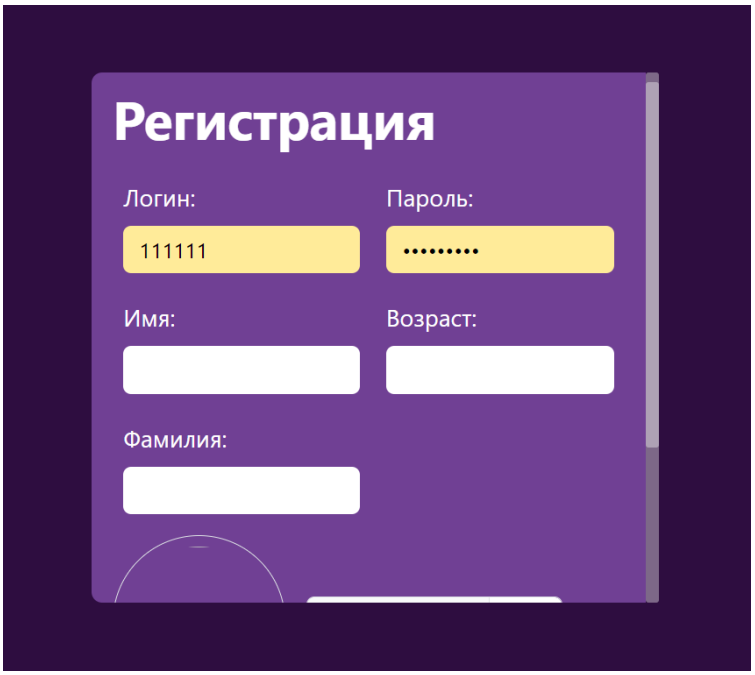

Рисунок 8.3 – Страница регистрации

Пользователь может отправлять сообщения различных видов: текстовые, голосовые и стикеры. Для отправки текстового сообщения он должен ввести текст в текстовое поле, показанное на рисунке 8.4.

Рисунок 8.4 – Блок для отправки сообщения

Для отправки голосового сообщения нужно нажать на кнопку в виде микрофона. После нажатия она загорится розовым цветом, как показано на рисунке 8.5, что означает, что запись голоса начата. Для отправки голосового сообщения, нужно снова нажать на эту кнопку.

Рисунок 8.5 – Режим записи голоса

Для отправки стикера нужно нажать на соответствующую кнопку, после чего откроется меню со стикерами, как показано на рисунке 8.6. После чего нужно нажать на любой стикер для отправки. Чтобы закрыть данное меню, нужно еще раз нажать на кнопку в виде стикера.

Рисунок 8.6 – Меню со стикерами

Пользователю доступны функции редактирования текстового сообщения и удаления любого своего сообщения. Для активации этого режима он должен дважды нажать на свое сообщение, после чего откроется меню, которое показано на рисунке 8.7. Для редактирования вам нужно нажать на сообщение, и вы сможете изменять его содержимое, после чего вы должны нажать на кнопку в виде карандаша. Для удаления вы должны нажать на кнопку в виде корзины.

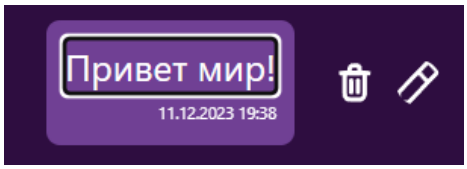

Рисунок 8.7 – Режим мутации сообщения

Если пользователь захочет нажать на кнопку «Бургер меню», то ему будет доступны информация о себе, возможность добавить себе фотографии или удалить их; кнопка для выхода из аккаунта; кнопка для редакции информации о себе. Содержимое бургер-меню показано на рисунке 8.8.

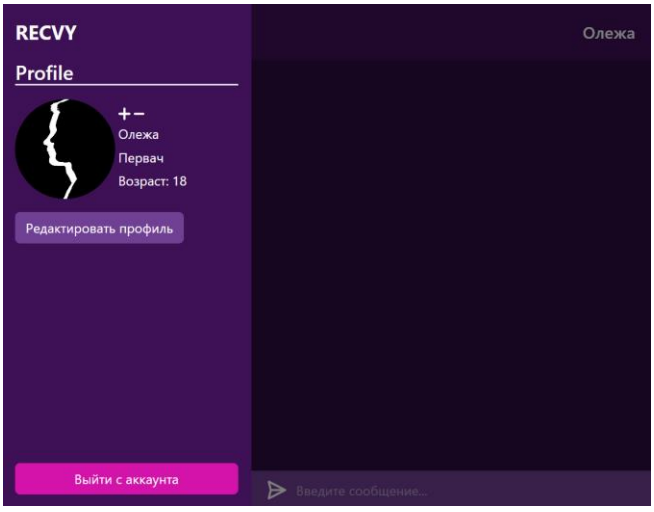

Рисунок 8.8 – Бургер-меню

Если пользователь захочет информацию о себе, он должен нажать на кнопку «Редактировать профиль». После чего ему откроется форма, показанная на рисунке 8.9. Тут он может ввести новые данные и подтвердить их, нажав на кнопку «Редактировать».

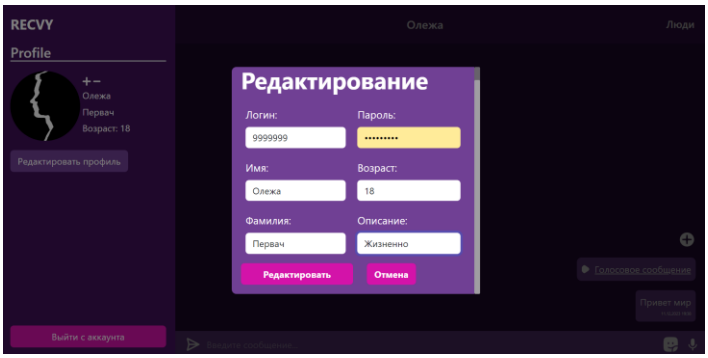

Рисунок 8.9 – Форма редактирования

Также любой пользователь может посмотреть информацию о любом другом, нажав на его имя в заголовочном блоке. Описание содержит фотографии, имя, фамилию, возраст и описание. Описание пользователя показано на рисунке 8.10.

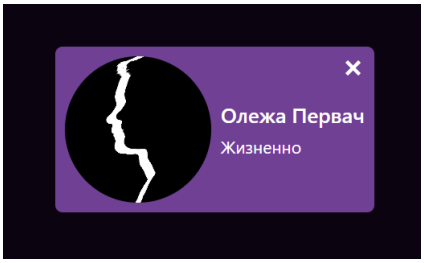

Рисунок 8.10 – Блок с описанием пользователя

Пользователь может перейти на страницу «Люди», где ему доступен удобный поиск пользователь. Тут он можем посмотреть своих друзей и подписчиков. Также здесь он может подписываться на кого-либо, отписываться от кого-либо, добавлять в друзья, удалять из друзей. Также он может перейти в чат с любым пользователем. Кнопка «Еще» позволяет получить еще пользователей, так используется ленивая подгрузка. Страница показана на рисунке 8.11.

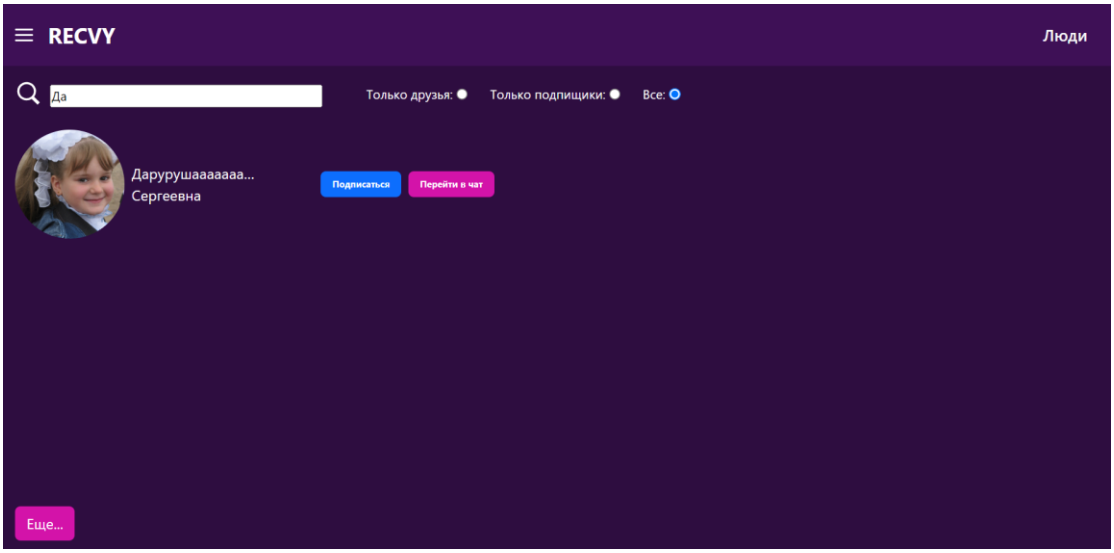

Рисунок 8.11 – Страница «Люди»

Администратору доступны дополнительные кнопки, для блокировки пользователя и удаления аккаунта пользователя. Они показаны на рисунке 8.12.

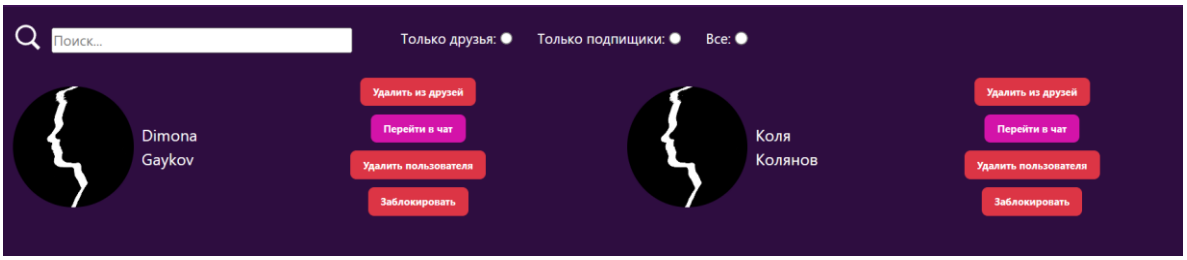

Рисунок 8.12 – Пользователь от лица администратора

Также администратору доступно добавление новых стикеров в их меню. Нужно сначала нажать на кнопку добавления стикеров, после чего выбрать нужные файлы. Данная процедура показана на рисунке 8.13.

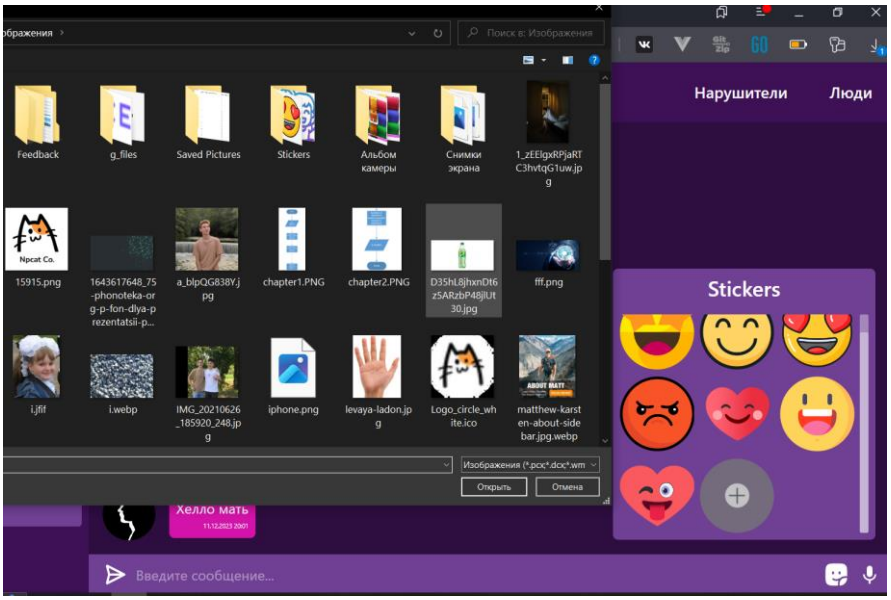

Рисунок 8.13 – Процесс добавления новых стикеров

Администратор имеет доступ к странице «Нарушители», где показаны все заблокированные пользователи и причины блокировки, где можно разблокировать пользователя. Данная страница показана на рисунке 8.14.

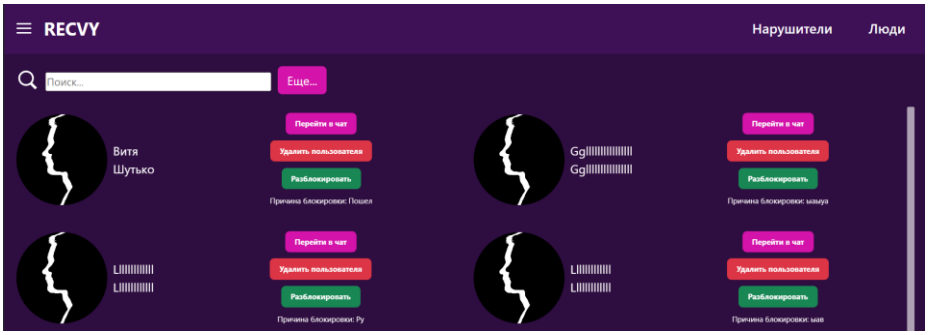

Рисунок 8.14 – Страница «Нарушители»

Мы рассмотрели основной функционал данного приложения. Он должен помочь освоится новому пользователю. Мы увидели, что интерфейс более-менее понятный и отзывчивый. Дизайн приложения довольно-таки привлекательный.

8.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, которая соединяет базу данных и React приложение. Для этого необходимо запустить скрипт, который настроит соединение с базой данных и запустит сервер.
2. Запустить React приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером. Для этого необходимо запустить команду для сборки и запуска React приложения.

После выполнения этих шагов приложение будет полностью готово к работе и пользователь сможет начать использовать его функционал.

8.3 Вывод

В данном разделе были рассмотрены функциональные возможности приложения, а также права доступа для пользователей с различными ролями. Было показано, что наша платформа позволяет пользователям создавать плейлисты, искать треки по названию или исполнителю, а также изменять свой профиль и пароль.

Кроме того, администратор имеет доступ к управлению пользователями, треками и жанрами. Для запуска приложения необходимо запустить серверную часть, которая соединяет базу данных и React-приложение.

Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для музыкальной площадки с использованием технологии применения мультимедийных типов данных в СУБД PostgreSQL.

В процессе выполнения работы были использованы различные объекты, включая таблицы, триггеры и функции, чтобы обеспечить структурированное хранение данных и своевременный доступ к ним. В результате, цель работы была успешно достигнута, и база данных готова к использованию. Были разработаны роли для управления доступом к данным и обеспечения безопасности.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формате JSON, что обеспечило удобство использования и управления данными.

Одной из ключевых особенностей разработанной базы данных является технология хранения мультимедийных данных, что позволяет эффективно управлять медиа-файлами на площадке.

Таким образом, была успешно выполнена задача по разработке базы данных для музыкальной площадки на основе СУБД PostgreSQL. Разработанная база данных позволяет хранить и управлять большим объемом музыкальных данных, обеспечивает безопасный доступ к ним и предоставляет возможность импорта и экспорта данных в различных форматах. Кроме того, технология хранения мультимедийных данных позволяет эффективно управлять медиа-файлами на площадке.

Список используемых источников

1. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.04.2023.
2. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 18.04.2023.
3. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 18.04.2023.
4. Нистюк О. А. курс лекций по дисциплине «Базы данных».

Приложение А

```
create tablespace DataTablespace
    location 'D:\DB\PSQL\data\RECVY\DataTB';

create tablespace IndexTablespace
    location 'D:\DB\PSQL\data\RECVY\IndexTB';

create tablespace SystemTablespace
    location 'D:\DB\PSQL\data\RECVY\SystemTB';
     create table BlockedUsers (
  userId varchar(50),
  blockedDate timestamp default CURRENT_TIMESTAMP,
  reason varchar(300)
) tablespace DataTablespace;

alter table BlockedUsers add foreign key (userid) references
users(id);
alter table BlockedUsers add unique (userId);
     create table friends (
    idUser1 varchar(50),
    idUser2 varchar(50),

    foreign key (idUser1) references users(id),
    foreign key (idUser2) references users(id)
) tablespace DataTablespace;

drop table Friends;
     create table Messages (
  messageId varchar(256) primary key,

  idUserFrom varchar(50),
  idUserTo varchar(50),

  message varchar(3000),
  messageType varchar(20),

  isEdited boolean default false,

  sentDate timestamp default CURRENT_TIMESTAMP,

  foreign key (idUserFrom) references users(id),
  foreign key (idUserTo) references users(id),

  foreign key (messageType) references messagestypes(type)
) tablespace DataTablespace;

drop table Messages;
     create table MessagesTypes (
    type varchar(20) primary key
) tablespace SystemTablespace;

insert into MessagesTypes values ('text');
```

```
insert into MessagesTypes values ('sticker');
insert into MessagesTypes values ('voice');
     create table Roles (
    role varchar(20) primary key
) tablespace SystemTablespace;

insert into Roles values ('user');
insert into Roles values ('admin');
insert into Roles values ('mainadmin');
     create table Photos (
  userId varchar(50),
  photo varchar(1000) unique,

  foreign key (userId) references users(id)
) tablespace DataTablespace;
     create table Stickers (
  stickerId varchar(50) primary key,
  sticker varchar(100)
) tablespace SystemTablespace;
     create table Subscribers (
    subscriber varchar(50),
    subscribeTo varchar(50),

    foreign key (subscriber) references users(id),
    foreign key (subscribeTo) references users(id)
) tablespace DataTablespace;
     create table Users (
    id varchar(50) primary key,
    login varchar(20) unique,

    firstName varchar(20),
    lastName varchar(20),
    password varchar(256),
    aboutMe varchar(300) null,

    age integer check (
        age > 6 and
        age < 150
    ),

    regDate timestamp default CURRENT_TIMESTAMP,
    role varchar(20) default 'user',

    foreign key (role) references roles(role)
) tablespace DataTablespace;

     create table UsersRoles (
  userId varchar(50),
  role varchar(20),

  foreign key (userId) references users(id),
  foreign key (role) references roles(role)
) tablespace systemtablespace;
```

Листинг 1 – Скрипты создания таблиц

```
create role RL_ADMIN;

grant RL_USER to RL_ADMIN;

grant execute on function messagetojson to RL_ADMIN;
grant execute on procedure usersfromjson to RL_ADMIN;

grant delete on stickers to RL_ADMIN;
grant insert on stickers to RL_ADMIN;
grant update on stickers to RL_ADMIN;
grant delete on blockedusers to RL_ADMIN;
grant insert on blockedusers to RL_ADMIN;
grant update on blockedusers to RL_ADMIN;
grant update on messagestypes to RL_ADMIN;
grant delete on messagestypes to RL_ADMIN;
grant insert on messagestypes to RL_ADMIN;
grant delete on roles to RL_ADMIN;
grant insert on roles to RL_ADMIN;
grant update on roles to RL_ADMIN;
grant delete on users to RL_ADMIN;
grant delete on usersroles to RL_ADMIN;
grant select on blocked_users to RL_ADMIN;
     create role RL_MAINADMIN;

grant RL_ADMIN to RL_MAINADMIN;
grant execute on function sendtoeveryone(varchar) to RL_MAINADMIN;
     create role RL_USER;

grant RL_VISITOR to RL_USER;

grant insert on friends to RL_USER;
grant delete on friends to RL_USER;
grant select on friends to RL_USER;
grant insert on messages to RL_USER;
grant delete on messages to RL_USER;
grant update on messages to RL_USER;
grant select on messages to RL_USER;
grant select on messagestypes to RL_USER;
grant select on stickers to RL_USER;
grant insert on subscribers to RL_USER;
grant delete on subscribers to RL_USER;
grant select on subscribers to RL_USER;
grant delete on photos to RL_USER;
grant select on users to RL_USER;
grant update on users to RL_USER;

grant select on returned_users to RL_USER;
     create role RL_VISITOR;
```

```
grant connect on database recvy to RL_VISITOR;

grant SELECT on users to RL_VISITOR;
grant INSERT on users to RL_VISITOR;
grant insert on photos to RL_VISITOR;
grant select on photos to RL_VISITOR;
grant select on blockedusers to RL_VISITOR;
grant select on roles to RL_VISITOR;
grant select on usersroles to RL_VISITOR;
grant insert on usersroles to RL_VISITOR;
grant execute on procedure setuserrole to RL_VISITOR;

grant execute on FUNCTION signin to RL_VISITOR;
grant execute on FUNCTION signup to RL_VISITOR;
     create user VISITOR
    PASSWORD '123123123';

create user APP_USER
    PASSWORD '123123123';

create user APP_ADMIN
    PASSWORD  '123123123';

create user APP_MAINADMIN
    PASSWORD '123123123';

grant
    rl_visitor
to visitor;

grant
    rl_user
to app_user;

grant
    rl_admin
to app_admin;

grant
    rl_mainadmin
to app_mainadmin;
```

Листинг 2 – Скрипты создание ролей и пользователей

```
create or replace function _isnull(value varchar, ifnullvalue
varchar)
returns varchar as $$
BEGIN
    if value is null then
        return ifnullvalue;
    end if;
```

```
    return value;
end;
$$ language plpgsql;
     create or replace function getChat(iduser1 varchar(50), iduser2
varchar(50))
returns table (
        iduserto varchar(50),
        photo varchar(1000),
        firstname varchar(50),
        lastmessage varchar(3000),
        messagetype varchar(30),
        sentdate timestamp
    )
as $$
begin
    return query
        select
            *
        from
            getchats(iduser1) as chats
        where
            chats.iduserto = iduser2;
end;
$$ language plpgsql;

select * from
getchats('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
select * from
getchat('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22',
'81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
     create or replace function getStickers(skip int = null, take
int = null)
returns setof stickers
as $$
begin
    call checkskipandtake(skip, take);

    return query
    select
        *
    from
        stickers
    offset skip
    limit take;

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;

create or replace function getStickerById(id varchar)
returns setof stickers
```

```
as $$
begin
    return query
    select
        *
    from
        stickers
    where
        stickerid = id;
end;
$$ language plpgsql;

select * from
getStickerById('692dT94ZWPa6t87hKq10Te4928VKszAQcOzPY9YZVU93i0H2SD')
;
     create or replace function hisSubscriberOf(his varchar(50), of
varchar(50))
returns boolean
as $$
    declare
        count integer;
begin
    count := (
        select
            count(*)
        from
            subscribers
        where
            subscriber = his and
            subscribeto = of
    );

    return count != 0;
end;
$$ language plpgsql;

select
hisSubscriberOf('YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI'
, 'YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI');
     create or replace function hasUserWithId(_id varchar(50))
returns boolean
as $$
declare
    count integer;
begin
    count := (
    select
        count(*)
    from
        users
    where
        id = _id
    );
```

```
    return count != 0;
end;
$$ language plpgsql;

select
hasUserWithId('lTr4V5hgLwEs1xXPjEEi7MH19iCyu7t2ZpoYCtht27BN1PGX4');
     create or replace function getLastMessage(_from varchar(50),
_to varchar(50))
returns setof messages
as $$
begin
    return query
    select
        *
    from
        messages
    where
        (iduserfrom = _from and iduserto = _to) or
        (iduserfrom = _to and iduserto = _from)
    order by
        sentdate desc
    limit 1;
end;
$$ language plpgsql;

select * from
getLastMessage('YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI',
'YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI');
     create or replace function getMessageById(id varchar(255))
returns setof messages as $$
begin
    return query select * from messages where messageid = $1;
end;
$$ language plpgsql;
     create or replace function generateId(countSymbols integer)
returns text
as $$
declare
    id text = '';
    i integer;
    symbol char;
begin
    for i in 1..countSymbols loop
        -- Генерируем случайный символ буквы, цифры или большой
буквы
        case floor(random() * 3)::integer
            when 0 then
                -- Генерируем случайную букву в нижнем регистре (a-
z)
                symbol = chr(floor(random() * 26 + 97)::integer);
            when 1 then
                -- Генерируем случайную букву в верхнем регистре (A-
```

```
Z)
                symbol = chr(floor(random() * 26 + 65)::integer);
            when 2 then
                -- Генерируем случайную цифру (0-9)
                symbol = chr(floor(random() * 10 + 48)::integer);
        end case;

        id = id || symbol::text;
    end loop;

    return id;
end;
$$ language plpgsql;
```

Листинг 3 – Скрипты создания дополнительных функций

```
create or replace function sendMessage(_from varchar(50), _to
varchar(50), msg varchar(3000), type varchar(20))
returns typemessage as $$
declare
    _id varchar(256) := generateid(256);
    _msg typemessage;
    _isChecked boolean = checksendmessagedto($1, $2, $3, $4);
begin
    insert into
        messages(messageid, iduserfrom, iduserto, message,
messagetype)
    values
        (_id, $1, $2, $3, $4)
    returning
        messageid, iduserfrom, iduserto, message, messagetype,
isedited, sentdate into _msg;

    return _msg;

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;

select * from getusers();
select * from messages;

select * from
sendMessage('YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI',
'YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI', 'jkkk',
'text');
     create or replace function getMessages(_from varchar(50), _to
varchar(50), skip integer = null, take integer = null)
returns setof messages as $$
begin
    call checkskipandtake(skip, take);

    return query
        select * from messages
        where
            (iduserfrom = _from and iduserto = _to) or
            (iduserfrom = _to and iduserto = _from)
        order by sentdate desc
        limit take
        offset skip;

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;
```

```
select iduserfrom, iduserto, message from messages order by sentdate
desc limit 10 offset 0;

select * from messages where iduserfrom =
'MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y' or iduserto =
'MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y';
select * from
getMessages('MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y',
'qQhI80pjjWM6M1bxAa3LJs5aI1f6JJjIro39lzjEsAdEutyxlK');
     create or replace procedure editMessage(id varchar(256),
newvalue varchar(3000))
language plpgsql as $$
begin
    if not exists(select * from messages where messageid = id and
messagetype = 'text') then
        raise exception '%', generateeditmsgerror();
    end if;

    if not checkmessage(newvalue) then
        raise exception '%', generatemsgerror();
    end if;

    update
        messages
    set
        message = newvalue,
        isedited = true
    where
        messageid = id;
end;
$$;

     create or replace procedure dellMessage(id varchar(256))
language plpgsql as $$
begin
    delete from messages
    where
        messageid = $1;
end;
$$;
     create or replace procedure addPhoto(id varchar(50), photo
varchar(1000))
language plpgsql as $$
begin
    if not exists (select * from users where users.id = $1) then
        raise exception '%', generatenotfoundusererror($1);
    end if;
    if checkimage(photo) then
        insert into photos(userid, photo) values (id, photo);
    else
        raise exception '%', generateimageerror();
    end if;
```

```
end
$$;
     create or replace procedure dellPhoto(_userid varchar(50),
_photo varchar(1000))
language plpgsql as $$
begin
    delete from
                photos
            where
                userid = $1 and
                photo = $2;
end;
$$;
     create or replace procedure addSticker(_stickerpath
varchar(1000))
language plpgsql as $$
declare
    _id varchar(50) = generateid(50);
begin
    if not checkimage($1) then
        raise exception '%', generateimageerror();
    end if;

    if exists(select * from stickers where stickers.sticker = $1)
then
        raise exception '%', generatestickeralreadyexisterror();
    end if;

    loop
        if _id not in (select stickerid from stickers) then
            exit;
        end if;

        _id = generateid(50);
    end loop;

    insert into stickers values (_id, $1);
end
$$;
create or replace procedure deleteSticker (idSticker varchar(50))
language plpgsql as $$
begin
    delete from stickers where stickerid = idSticker;
    delete from messages where message = idSticker and messagetype =
'sticker';
end;
$$;
```

Листинг 4 – Скрипты создание функций CRUD

```
create or replace view blocked_users as
select
        returned_users.*,
        blockedusers.reason,
        blockedusers.blockeddate
    from
        blockedusers
    join
        returned_users
            on returned_users.id = blockedusers.userid;
     CREATE OR REPLACE VIEW returned_users AS
SELECT
    users.id,
    users.firstname,
    users.lastname,
    users.aboutme,
    users.age,
    users.regdate,
    getphotosbyid(users.id) AS photos,
    getrolesof(users.id) AS role
FROM
    users;
```

Листинг 5 – Скрипты создание представлений

```
create or replace trigger before_insert_trigger
before insert on users
for each row
execute function signuptrigger();

create or replace trigger after_insert_trigger
after insert on users
for each row
execute function setRoleTrigger();

create or replace function signUpTrigger()
returns trigger
as $$
begin
    new.password := tohash(new.password);
    return new;
end;
$$ language plpgsql;

create or replace function setRoleTrigger()
returns trigger
as $$
begin
    call setuserrole(new.id);
    return new;
end;
$$ language plpgsql;
```

Листинг 6 – Скрипты создания триггеров

```
     create or replace function messageToJson(
    _from varchar(50) = null,
    _to varchar(50) = null,
    skip integer = null,
    take integer = null)
    returns json
    language plpgsql as
$$
declare
    _json json;
begin
    call checkskipandtake(skip, take);

    create temporary table _messages as
    select *
    from messages
    where iduserfrom = _isnull($1, iduserfrom)
      and iduserto = _isnull($2, iduserto)
    offset skip limit take;

    _json := (select json_agg(to_json(_messages))
              from _messages);

    drop table _messages;

    return _json;
exception
    when others then
        raise exception '%', SQLERRM;
end;
$$;

select *
from users
limit 1;

create or replace procedure usersFromJson(_json json)
as
$$
begin
    SELECT signup(login, firstname, lastname, age, password)
    FROM
        json_populate_recordset(null::users, $1);
end;
$$ language plpgsql;


call usersFromJson('[
  {
    "id": "1",
    "login": "888888999",
    "firstname": "Oleg",
    "lastname": "Kravchenko",
```

```
    "age": 19,
    "password": "12345612331"
  },
  {
    "id": "2",
    "login": "john_doe",
    "firstname": "John",
    "lastname": "Doe",
    "age": 28,
    "password": "password123"
  },
  {
    "id": "3",
    "login": "alice_smith",
    "firstname": "Alice",
    "lastname": "Smith",
    "age": 35,
    "password": "pass987"
  },
  {
    "id": "4",
    "login": "bob_jackson",
    "firstname": "Bob",
    "lastname": "Jackson",
    "age": 22,
    "password": "securepass"
  }
]');

do
$$
    declare
        cur cursor for select *
                       from users
                       where login = '888888999'
                          or login = 'john_doe'
                          or login = 'alice_smith'
                          or login = 'bob_jackson';
    begin
        for rec in cur
            loop
                call deleteuser(rec.id);
            end loop;
    end;
$$ language plpgsql;
```

Листинг 7 – Скрипты для импорта и экспорта

```
create or replace function getBlockedUsers(skip integer = null, take
integer = null)
returns setof blocked_users
as $$
begin
    call checkskipandtake(skip, take);

    return query
        select
            *
        from
            blocked_users
        order by
            blocked_users.blockeddate desc
        offset skip
        limit take;

    exception
        when others then
            raise exception '%', sqlerrm;
end
$$ language plpgsql;

select * from getBlockedUsers(-1);
     create or replace function getBlockReason(_userid varchar(50))
returns text as $$
begin
    return (select reason from blockedusers where userid = _userid);
end;
$$ language plpgsql;
     create or replace function searchBlockedUsers(stext text, skip
integer = null, take integer = null)
    returns setof returned_users as
$$
declare
    searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
        select *
        from returned_users as u
        where u.id in (select bu.userid from blockedusers as bu)
          and (lower(u.firstname || ' ' || u.lastname) ilike
searchtext
            or lower(u.lastname || ' ' || u.firstname) ilike
searchtext
            or lower(u.firstname) ilike searchtext
            or lower(u.lastname) ilike searchtext)
        offset skip limit take;

exception
    when others then
```

```
        raise exception '%', SQLERRM;
end;
$$ language plpgsql;

select *
from searchBlockedUsers('Дар', 0, 8);
     create or replace function getChats(_userid varchar, skip
integer = null, take integer = null)
returns table
    (
        iduserto varchar(50),
        photo varchar(1000),
        firstname varchar(50),
        lastmessage varchar(3000),
        messagetype varchar(30),
        sentdate timestamp
    )
as $$
begin
    call checkskipandtake(skip, take);

    if not exists(select * from users where id = _userid) then
        raise exception '%', generatenotfoundusererror();
    end if;

    return query
    select
        distinct(isThisOrOther($1, m.iduserto,
m.iduserfrom)::varchar) as iduserto,
        (select p.photo from photos as p where p.userid =
isThisOrOther($1, m.iduserto, m.iduserfrom) limit 1) as photo,
        u.firstname as firstname,
        lm.message as lastmessage,
        lm.messagetype as messagetype,
        lm.sentdate as senttime
    from
        messages as m
    join
        users as u on u.id = isThisOrOther($1, m.iduserto,
m.iduserfrom)
    join
        getlastmessage(m.iduserto, m.iduserfrom) as lm on
            lm.iduserto = m.iduserto or
            lm.iduserto = m.iduserfrom
    where
        m.iduserfrom = $1 or
        m.iduserto = $1
    order by
        lm.sentdate desc
    offset skip
    limit take;
```

```
    exception
        when others then
            raise exception '%', SQLERRM;
end;
$$ language plpgsql;

create or replace function isThisOrOther(value text, first text, sec
text)
returns text
as $$
begin
    if value = first then
        return sec;
    else
        return first;
    end if;
end;
$$ language plpgsql;


explain analyse select * from getChats('0');


select login from users where id =
'63dC9kfuuIBkwZ50AL1jRy191JSTA531s9CqJwSmqfiG5A2ICt';
select * from
getChats('63dC9kfuuIBkwZ50AL1jRy191JSTA531s9CqJwSmqfiG5A2ICt', 0,
10);

call dellphoto('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22',
'test.png');
     create or replace function getChat(iduser1 varchar(50), iduser2
varchar(50))
returns table (
        iduserto varchar(50),
        photo varchar(1000),
        firstname varchar(50),
        lastmessage varchar(3000),
        messagetype varchar(30),
        sentdate timestamp
    )
as $$
begin
    return query
        select
           *
        from
            getchats(iduser1) as chats
        where
            chats.iduserto = iduser2;
end;
$$ language plpgsql;
```

```
select * from
getchats('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
select * from
getchat('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22',
'81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
     create or replace function getFriendsOf(userid varchar(50),
skip integer = null, take integer = null)
returns setof returned_users as $$
begin
    call checkskipandtake(take, skip);

    return query (
        select
        returned_users.*
    from
        returned_users
    where
        returned_users.id in (
            select
                friends.iduser1
            from
                friends
            where
                friends.iduser2 = userid
            union
            select
                friends.iduser2
            from
                friends
            where
                friends.iduser1 = userid
        )
        offset skip
        limit take);

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;

select * from
getFriendsOf('81L086RIcklaeZJUciC6zJDUS1q565x1N28V799gazpCTmcD22');
     create or replace function getLastMessage(_from varchar(50),
_to varchar(50))
returns setof messages
as $$
begin
    return query
    select
        *
    from
        messages
```

```
    where
        (iduserfrom = _from and iduserto = _to) or
        (iduserfrom = _to and iduserto = _from)
    order by
        sentdate desc
    limit 1;
end;
$$ language plpgsql;

select * from
getLastMessage('YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI',
'YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI');
     create or replace function getMessages(_from varchar(50), _to
varchar(50), skip integer = null, take integer = null)
returns setof messages as $$
begin
    call checkskipandtake(skip, take);

    return query
        select * from messages
        where
            (iduserfrom = _from and iduserto = _to) or
            (iduserfrom = _to and iduserto = _from)
        order by sentdate desc
        limit take
        offset skip;

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;

select iduserfrom, iduserto, message from messages order by sentdate
desc limit 10 offset 0;

select * from messages where iduserfrom =
'MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y' or iduserto =
'MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y';
select * from
getMessages('MfhE4T8XrV5urZXlXp9gM7f4940emJb8NbnE43UjFz5ZPUdQ6y',
'qQhI80pjjWM6M1bxAa3LJs5aI1f6JJjIro39lzjEsAdEutyxlK');
     create or replace function getPhotosById(id varchar(50))
returns varchar(1000)[]
as $$
begin
    return array(
        select
            photo
        from
            photos
        where
            photos.userId = id);
```

```
end;
$$ language plpgsql;
     create or replace function getStickers(skip int = null, take
int = null)
returns setof stickers
as $$
begin
    call checkskipandtake(skip, take);

    return query
    select
        *
    from
        stickers
    offset skip
    limit take;

    exception
        when others then
            raise exception '%', sqlerrm;
end;
$$ language plpgsql;

create or replace function getStickerById(id varchar)
returns setof stickers
as $$
begin
    return query
    select
        *
    from
        stickers
    where
        stickerid = id;
end;
$$ language plpgsql;

select * from
getStickerById('692dT94ZWPa6t87hKq10Te4928VKszAQcOzPY9YZVU93i0H2SD')
;
     create or replace function searchSubs(stext text, userid
varchar(50), skip integer = null, take integer = null)
    returns setof returned_users as
$$
declare
    searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
        select *
        from returned_users as u
        where issubscribingexist(u.id,userid) and
```

```
            (lower(u.firstname || ' ' || u.lastname) ilike
searchtext
            or lower(u.lastname || ' ' || u.firstname) ilike
searchtext
            or lower(u.firstname) ilike searchtext
            or lower(u.lastname) ilike searchtext)
        offset skip limit take;

exception
    when others then
        raise exception '%', SQLERRM;
end;
$$ language plpgsql;


     create or replace function searchUser(stext text, skip integer
= null, take integer = null)
returns setof returned_users as $$
    declare
        searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
    select
        *
    from
        returned_users as u
    where
        lower(u.firstname || ' ' || u.lastname) ilike searchtext  or
        lower(u.lastname || ' ' || u.firstname) ilike searchtext or
        lower(u.firstname) ilike searchtext or
        lower(u.lastname) ilike searchtext
    offset skip
    limit take;

    exception
        when others then
            raise exception '%', SQLERRM;
end;
$$ language plpgsql;

select * from searchUser('i');
```

Листинг 8 – Скрипты для получения данных

```
create or replace function searchBlockedUsers(stext text, skip
integer = null, take integer = null)
    returns setof returned_users as
$$
declare
    searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
        select *
        from returned_users as u
        where u.id in (select bu.userid from blockedusers as bu)
          and (lower(u.firstname || ' ' || u.lastname) ilike
searchtext
            or lower(u.lastname || ' ' || u.firstname) ilike
searchtext
            or lower(u.firstname) ilike searchtext
            or lower(u.lastname) ilike searchtext)
        offset skip limit take;

exception
    when others then
        raise exception '%', SQLERRM;
end;
$$ language plpgsql;

select *
from searchBlockedUsers('Дар', 0, 8);
     create or replace function searchFriends(stext text, friendsOf
varchar(50), skip integer = null, take integer = null)
    returns setof returned_users as
$$
declare
    searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
        select *
        from returned_users as u
        where isfriendsexist(u.id,friendsOf) and
            (lower(u.firstname || ' ' || u.lastname) ilike
searchtext
            or lower(u.lastname || ' ' || u.firstname) ilike
searchtext
            or lower(u.firstname) ilike searchtext
            or lower(u.lastname) ilike searchtext)
        offset skip limit take;

exception
    when others then
        raise exception '%', SQLERRM;
```

```
end;
$$ language plpgsql;

select * from friends;

select *
from searchFriends('py',
'YLZXq6IP91I7A16Qc4wH5dweWJrwht4ZJFS7SsC7B4Pf8zP7fI');
     create or replace function searchSubs(stext text, userid
varchar(50), skip integer = null, take integer = null)
    returns setof returned_users as
$$
declare
    searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
        select *
        from returned_users as u
        where issubscribingexist(u.id,userid) and
            (lower(u.firstname || ' ' || u.lastname) ilike
searchtext
            or lower(u.lastname || ' ' || u.firstname) ilike
searchtext
            or lower(u.firstname) ilike searchtext
            or lower(u.lastname) ilike searchtext)
        offset skip limit take;

exception
    when others then
        raise exception '%', SQLERRM;
end;
$$ language plpgsql;


     create or replace function searchUser(stext text, skip integer
= null, take integer = null)
returns setof returned_users as $$
    declare
        searchtext text := '%' || lower(stext) || '%';
begin
    call checkskipandtake(skip, take);

    return query
    select
        *
    from
        returned_users as u
    where
        lower(u.firstname || ' ' || u.lastname) ilike searchtext  or
        lower(u.lastname || ' ' || u.firstname) ilike searchtext or
        lower(u.firstname) ilike searchtext or
```

```
        lower(u.lastname) ilike searchtext
    offset skip
    limit take;

    exception
        when others then
            raise exception '%', SQLERRM;
end;
$$ language plpgsql;
```

Листинг 9– Скрипты для поиска данных

```
call fill();

create or replace procedure fill()
as $$
declare
  i int;
  _messageid text;
begin
    i := 1;
    while i <= 100000 loop
        select messageid into _messageid from sendMessage('0', '0',
'Hello mir ' || cast(i as text), 'text');
        i := i + 1;
    end loop;
end;
$$ language plpgsql;
```

Листинг 10– Скрипты заполнение таблиц