МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных управления криптокошельками с технологией Настройка системы безопасности сервера СУБД»

**Исполнитель**

студент 2 курса 1 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф. А. Кудрицкий

подпись, дата

**Руководитель**

доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Блинова

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Блинова

подпись дата инициалы и фамилия

Минск 2023

**Содержание**

[Введение 4](#_Toc135199360)

[1 Анализ требований к программному средству 5](#_Toc135199361)

[1.1 Аналитический обзор аналогов 5](#_Toc135199362)

[1.1.1 Аналог Trust Wallet 5](#_Toc135199363)

[1.1.2 Аналог Binance 6](#_Toc135199366)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc135199367)

[1.3 Вывод 9](#_Toc135199368)

[2 Разработка архитектуры проекта 10](#_Toc135199369)

[2.1 Обобщенная структура управлением приложения 10](#_Toc135199370)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc135199372)

[3 Разработка модели базы данных 14](#_Toc135199373)

[3.1 Создание необходимых объектов 14](#_Toc135199374)

[3.1.1 Представления базы данных 14](#_Toc135199375)

[3.1.2 Индексы базы данных 15](#_Toc135199376)

[3.1.3 Триггеры базы данных 15](#_Toc135199377)

[3.2 Описание используемой технологии 16](#_Toc135199378)

[3.3 Вывод 19](#_Toc135199379)

[4 Установка, настройка и использование PosgtreSQL 14.5 20](#_Toc135199380)

[4.1 Установка PostgreSQL 20](#_Toc135199381)

[4.2 Создание таблиц 20](#_Toc135199382)

[4.3 Создание ролей для разграничения 21](#_Toc135199383)

[4.4 Создание пакетов процедур для базы данных 23](#_Toc135199384)

[4.4.1 Выборка данных из таблиц 23](#_Toc135199385)

[4.4.2 Заполнение таблиц 100 000 строк 24](#_Toc135199386)

[4.4.3 Добавление данных в таблицы 25](#_Toc135199387)

[4.4.4 Удаление данных в таблицы 26](#_Toc135199388)

[4.4.5 Изменение данных в таблицы 26](#_Toc135199389)

[4.4.6 Дополнительные функции 27](#_Toc135199390)

[4.5 Описание процедур экспорта и импорта 28](#_Toc135199391)

[4.6 Вывод 30](#_Toc135199392)

[5 Тестирование 31](#_Toc135199393)

[5.1 Тестирование производительности базы данных 31](#_Toc135199394)

[5.2 Вывод 32](#_Toc135199395)

[6 Руководство по использованию программного средства 33](#_Toc135199396)

[6.1 Руководство пользователя 33](#_Toc135199397)

[6.2 Установка приложения 36](#_Toc135199398)

[6.3 Вывод 37](#_Toc135199399)

[Заключение 38](#_Toc135199400)

[Список литературных источников 39](#_Toc135199401)

[Приложение А 40](#_Toc135199402)

# Введение

Цель данной работы заключается в создании базы данных сервиса с онлайн играми и смарт-контрактами с применением технологии резервного копирования. База данных будет хранить информацию о зарегистрированных пользователях, текущих играх и их результатах. Также необходимо разработать клиент-серверное приложение для демонстрации функциональности базы данных.

База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом 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 данными клиентов сайта.

В курсовой работе также требуется создать приложение для демонстрации функциональности базы данных и ее взаимодействия с пользователем. Для этого приложение было разработано с использованием языков программирования React js, nest js, socket.io, typescript.

Основные требования к приложению:

* Реализация ролей администратора и пользователя;
* Изменение данных аккаунта пользователя;
* Регистрация и вход пользователя;
* Взаимодействие с базой данных при помощи процедур и функций.

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

1. Анализ требований к программному средству

## Аналитический обзор аналогов

В мире современных игр человечество достигло немалых результатов. Одним из ярких представителей являются игры онлайн на некоторые денежные суммы. Они весьма популярны в связи со своим удобством.

Занимающаяся в этой сфере компания обязана обеспечивать максимальную надёжность и безопасность данных пользователей, а также стабильную работу и привлекательный интерфейс.

### 1.1.1 Аналог Web3games

Сайт адаптивен под разные размеры экранов и разные устройства.

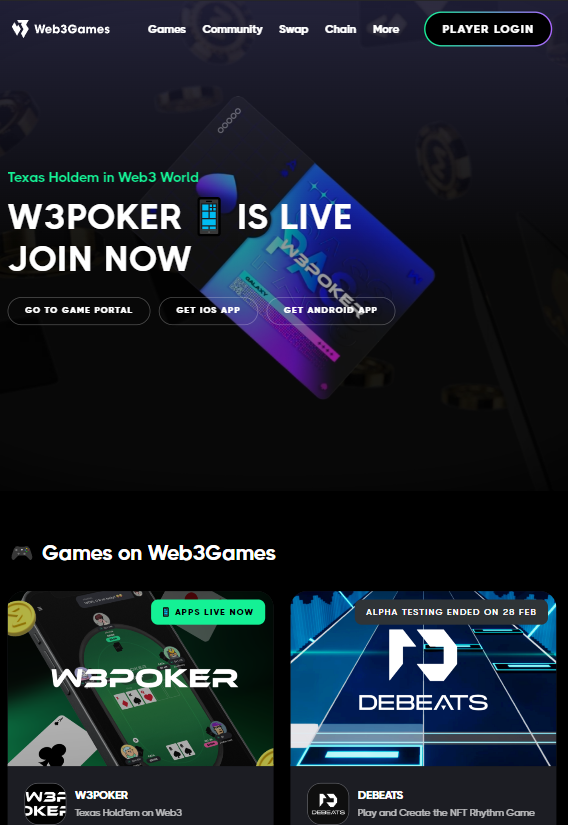


Рис. 1.1 – главная страница web-сайта при просмотре с планшета

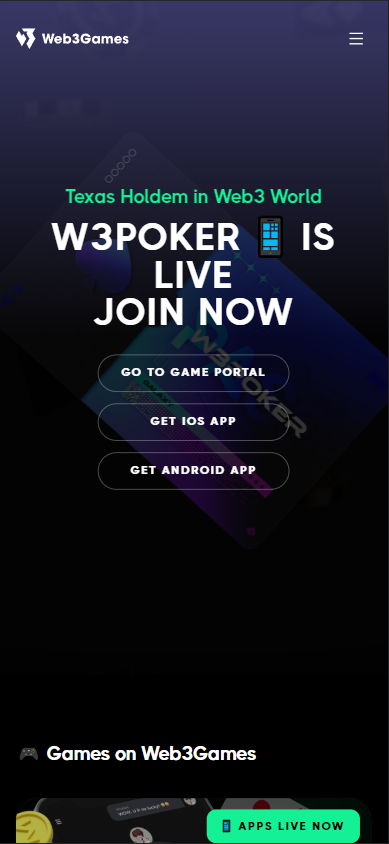


Рис. 1.2 – главная страница web-сайта при просмотре с мобильного устройства

Преимущества сайта https://web3games.com/:

1. Наличие большого количества игр: на сайте представлены игры разных жанров и с разнообразным функционалом.

2. Удобный интерфейс: сайт имеет интуитивно понятный интерфейс, который несомненно упрощает поиск и использование сервиса.

3. На сайте также есть новостной раздел, что позволяет пользователям получать информацию о новых событиях и игрых на площадке.

4. Сервис предоставляет возможность использовать его не только в браузере, но и как приложение на телефонах.

Недостатки сайта https://music.yandex.ru/:

1. Плохая оптимизация: пользуясь сервисом со стационарного компьютера уже можно заметить переодические подтормаживания и задержку в отрисовке UI, при эмуляции использования с телефона, ситуация аналогичная.

2. Сервис предоставляет услуги посредника: пользователю предоставлен функционал поиска и просмотра информации об играх, но для непосредственного их использования, пользователь должен перейти на сторонние сайты, где и реализована нужная ему игра

3. Отсутствие полноценного личного кабинета пользователя: зарегистрированному пользователю предоставляется крайне мало функционала и информации. Пользователь не может посмотреть историю своих игр, свой баланс и тп.

### 1.1.2 Аналог Alchemy

Web-сайт адаптивен под разные размеры экранов, и под разные устройства.

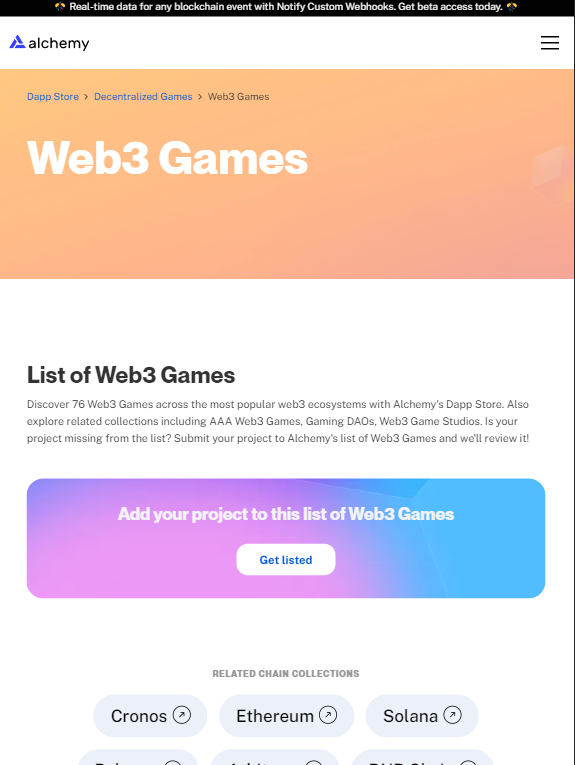


Рис. 1.3 – страница с играми web-сайта при просмотре с планшета

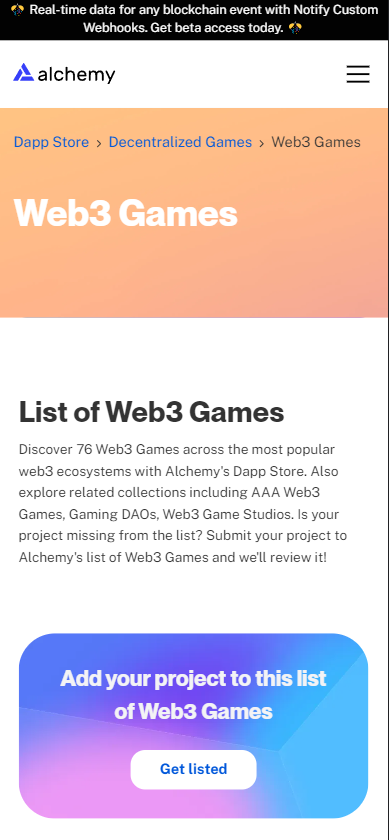


Рис. 1.4 – страница с играми web-сайта при просмотре с мобильного устройства

Преимущества сайта https://www.alchemy.com/best/web3-games:

1. Огромный каталог игр: сервис предоставляет внушительное количество и разнообразие игр на своей платформе.

2. Пользовательский опыт: сайт имеет удобный и интуитивно понятный интерфейс, который позволяет легко ориентироваться по сайту и находить нужную информацию.

3. Широкое разнообразие предлагаемых токенов: на площадке можно найти много игр в основе которых лежат разные токены (chain колекции).

4. Надёжность: сервис существует уже много лет и, не смотря на наличие раздела с играми, компания специализируется в первую очередь на разработке и интеграции web3 инструментов. Это позволяет быть уверенным в честности и стабильность предлагаемого функционала.

5. Наличие справочной информации: в случае отсутствия у пользователя опыта работы со смарт-контрактами, сервис предлагает подробную справочную информацию о платформе и как ее использовать.

Недостатки сайта https://www.alchemy.com/best/web3-games:

1. Мало информации об игре: пользователь может только прочитать краткую сводку об выбранной игре, подробное описание или демонстрация недоступны незарегистрированному пользователю.

2. Ограниченный функционал личного кабинета: в личном кабинете пользователя мало информации об играх в которые он играл, балансе его кошелька и тп.

3. Ограничения бесплатного тарифа: после регистрации пользователь может играть в часть игр, однако даже в доступных ему есть множество ограничений связанных с бесплатным использованием.

### 1.1.2 Аналог Defiance.games

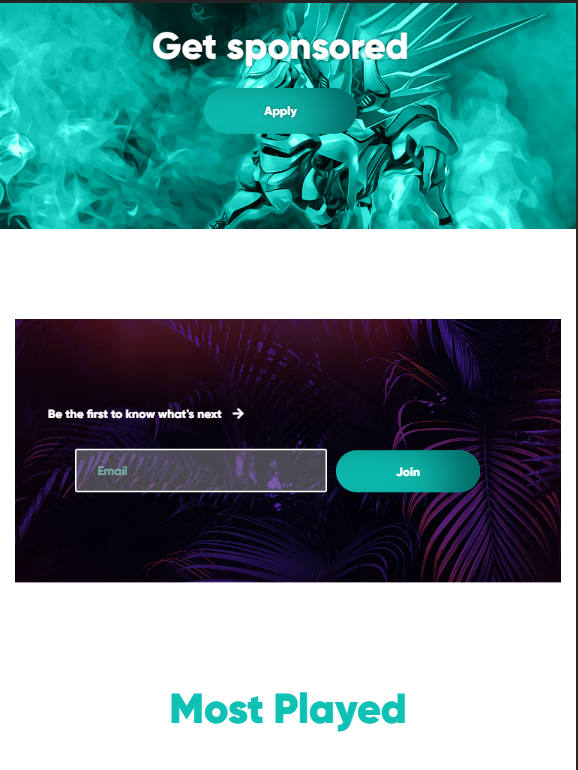


Рис. 1.5 – главная страница web-сайта при просмотре с планшета

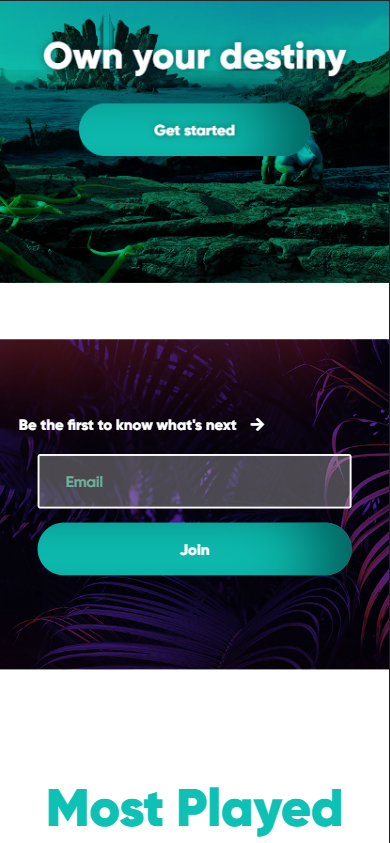


Рис. 1.6 – страница с играми web-сайта при просмотре с мобильного устройства

Преимущества сайта https://defiance.games/:

1. Простота: пользователь может сразу выбрать интересующую его игру и начать играть в нее бесплатно почти без ограничений.

2. Широкий выбор: на сайте представлена большая коллекция игр.

3. Быстрота и плавность: сервис работает быстро и стабильно на любых устройствах даже под нагрузкой.

4. Полноценный личный кабинет: пользователь может добавлять игры в избранное, просматривать историю игр, свой баланс.

Недостатки сайта https://defiance.games/:

1. Устаревший и неудобный интерфейс: UI не соответствует современным стандартам и выглядит невпечатляюще. UX также на плохом уровне, пользователю будет трудно разораться, где и что находится, а многие элементы интерфейса расположены не интуитивно.

2. Нестабильность: некоторые разделы и элементы интерфейса работают странно или не работаю вообще.

3. Стоимость: Нет конкретного разделения на бесплатный и платный контент, в некоторых условно бесплатных играх у пользователя будут списываться средства за игру, хотя четко и ясно нигде это до начала игрового процесса не обозначено.

## 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 игровой вселенной функциональные требования могут включать в себя функции для хранения информации о клиентах, обработки текущих игр, вычисления результатов.

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:

* Гость.
* Пользователь.
* Администратор.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.3.

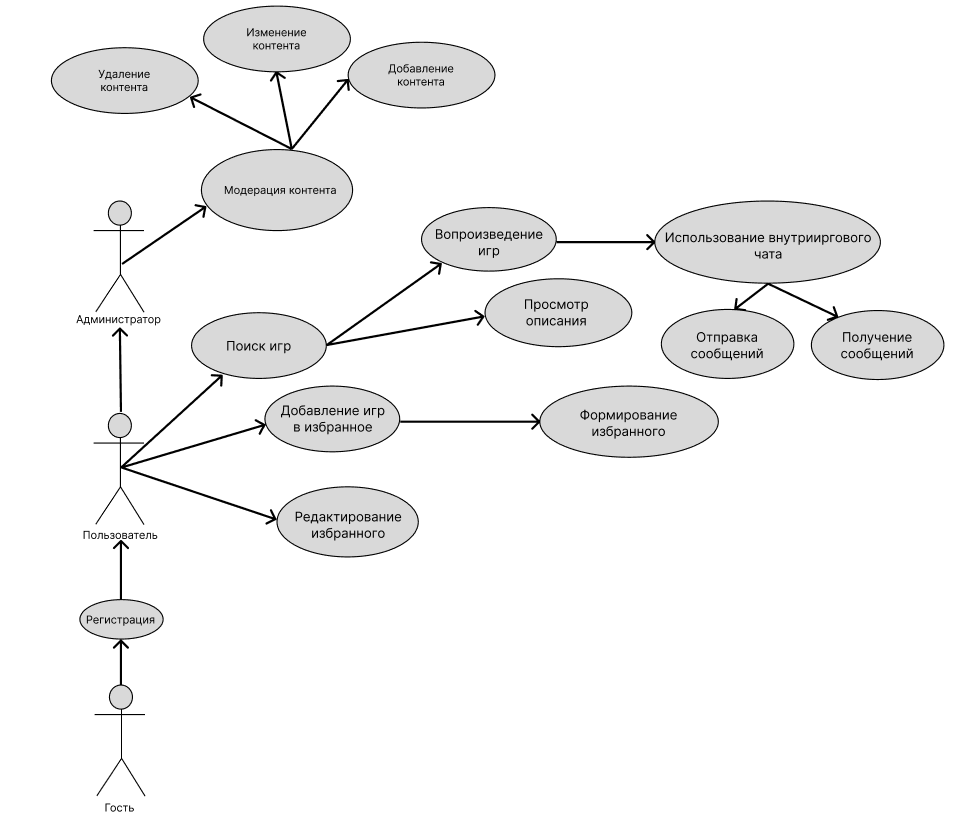


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет предложена регистрация. После регистрации клиент получает роль Пользователь.

Роль Пользователь получает доступ к поиску и воспроизведению игр, чату, а изменению избранного.

Роль Администратор заключается в модерации контента.

## 1.3 Вывод

Был проведен аналитический обзор аналогов справочников по игровым вселенным. 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.

1. Разработка архитектуры проекта

## 2.1 Обобщенная структура управлением приложения

Для обеспечения удобного и эффективного управления данными при использовании базы данных в приложении необходимо разработать интуитивно понятный интерфейс, который позволит пользователям взаимодействовать с базой данных. В этом могут помочь оптимизированные запросы для вставки, обновления и удаления данных, а также механизмы для извлечения и обработки информации из базы данных.

## Для обеспечения комфортного и удобного пользования сервиса с онлайн играми через приложение необходимо предусмотреть функции быстрого и интуитивно понятного использования платформы. Также пользователи должны иметь возможность создавать комнаты для игр и заходить в существующие.

## 2.2 Диаграмма базы данных, 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 будет представлена на рисунке 2.1.

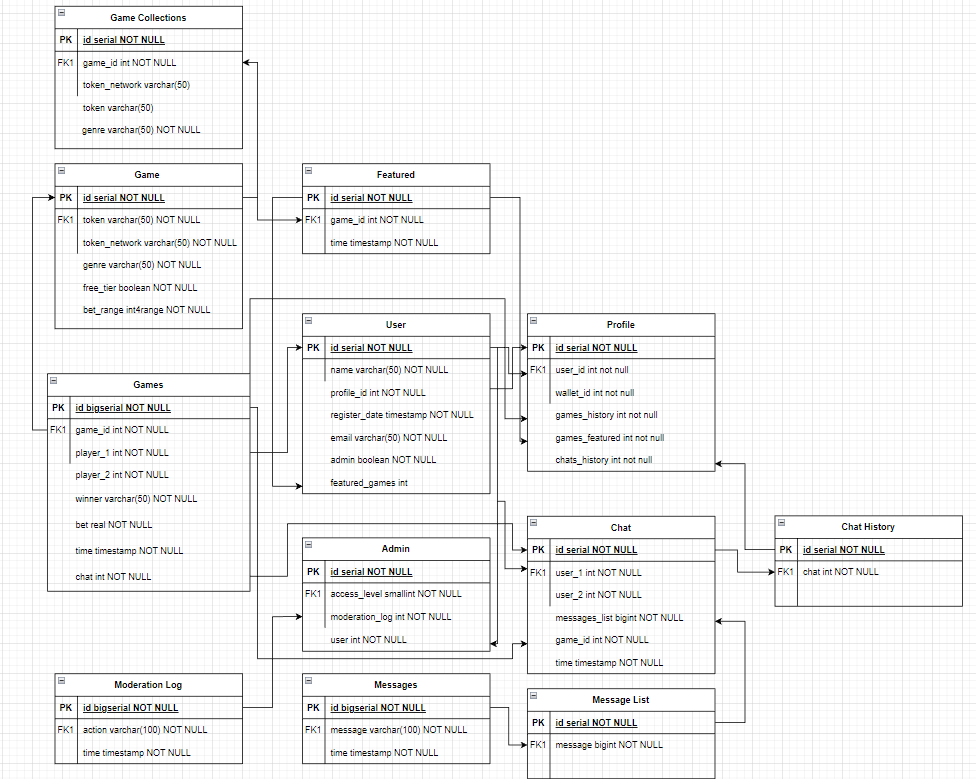


Рисунок 2.1 – Диаграмма базы данных

Таблица "users":

Id: serial (первичный ключ)

Name: varchar(20) (имя пользователя, не пустое значение)

Register\_date: timestamp (дата и время регистрации, не пустое значение)

Email: varchar(50) (адрес электронной почты, уникальное значение, не пустое значение)

Admin: boolean (флаг администратора, значение по умолчанию: false)

Games\_history: bigint (идентификатор истории игр, уникальное значение)

Games\_featured: bigint (идентификатор особо рекомендуемых игр, уникальное значение)

Таблица "featured":

Id: serial (первичный ключ)

User\_id: bigint (идентификатор пользователя, связано с полем "Id" таблицы "users", не пустое значение)

Game\_id: bigint (идентификатор игры, связано с полем "Id" таблицы "games", не пустое значение)

Time: timestamp (время)

Таблица "games":

Id: serial (первичный ключ)

Token: varchar(50) (токен игры, не пустое значение)

Token\_network: varchar(50) (сеть токена игры)

Game\_genre: varchar(50) (жанр игры, не пустое значение)

Free\_tier: boolean (флаг бесплатной версии игры, значение по умолчанию: true)

Min\_bet: decimal (минимальная ставка, не пустое значение)

Max\_bet: decimal (максимальная ставка, не пустое значение)

Таблица "rooms":

Id: serial (первичный ключ)

Created\_at: timestamp (дата и время создания комнаты, не пустое значение)

Created\_by: bigint (идентификатор создателя комнаты, связано с полем "Id" таблицы "users", не пустое значение)

Game\_id: bigint (идентификатор игры, связано с полем "Id" таблицы "games", не пустое значение)

Таблица "room\_users":

Id: serial (первичный ключ)

User\_id: bigint (идентификатор пользователя, связано с полем "Id" таблицы "users", не пустое значение)

Connection\_date: timestamp (дата и время подключения, не пустое значение)

User\_type: varchar(20) (тип пользователя)

Room\_id: bigint (идентификатор комнаты, связано с полем "Id" таблицы "rooms", не пустое значение)

Таблица "game\_process":

Id: serial (первичный ключ)

Bet: decimal (ставка, не пустое значение)

Creation\_time: timestamp (дата и время создания, не пустое значение)

Finish\_time: timestamp (дата и время окончания)

Game\_id: bigint (идентификатор игры, связано с полем "Id" таблицы "games", не пустое значение)

Room\_id: bigint (идентификатор комнаты, связано с полем "Id" таблицы "rooms", не пустое значение)

Таблица "results":

Id: serial (первичный ключ)

User\_id: bigint (идентификатор пользователя, связано с полем "Id" таблицы "users", не пустое значение)

Balance\_results: money (результаты баланса, не пустое значение)

Game\_process\_id: bigint (идентификатор игрового процесса, связано с полем "Id" таблицы "game\_process", не пустое значение)

Таблица "admins":

Id: serial (первичный ключ)

User\_id: bigint (идентификатор пользователя, связано с полем "Id" таблицы "users", не пустое значение)

Access\_level: smallint (уровень доступа, не пустое значение)

Таблица "moderation\_logs":

Id: serial (первичный ключ)

Admin\_id: bigint (идентификатор администратора, связано с полем "Id" таблицы "admins", не пустое значение)

Action: varchar(50) (действие, не пустое значение)

Time: timestamp (время)

Таблица "messages":

Id: serial (первичный ключ)

Sender\_id: bigint (идентификатор отправителя, связано с полем "Id" таблицы "users", не пустое значение)

Receiver\_id: bigint (идентификатор получателя, связано с полем "Id" таблицы "users", не пустое значение)

Game\_id: bigint (идентификатор игры, связано с полем "Id" таблицы "games", не пустое значение)

Content: varchar(100) (содержание сообщения, не пустое значение)

**2.3 Описание информационных объектов**

Таблица "users":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для пользователей.

Name: varchar(20) CHECK(Name != '') NOT NULL - поле, содержащее имя пользователя. Ограничено до 20 символов. Проверяет, чтобы значение поля не было пустым.

Register\_date: timestamp NOT NULL - поле, содержащее дату и время регистрации пользователя. Не может быть пустым.

Email: varchar(50) CHECK(Email != '') UNIQUE NOT NULL - поле, содержащее адрес электронной почты пользователя. Ограничено до 50 символов. Проверяет, чтобы значение поля не было пустым и уникальным в таблице.

Admin: boolean DEFAULT false - поле, указывающее, является ли пользователь администратором. Значение по умолчанию - false.

Games\_history: bigint CONSTRAINT games\_history\_key UNIQUE - поле, содержащее идентификатор истории игр пользователя. Уникальное значение для каждого пользователя.

Games\_featured: bigint CONSTRAINT games\_featured\_key UNIQUE - поле, содержащее идентификатор особо рекомендуемых игр для пользователя. Уникальное значение для каждого пользователя.

Таблица "featured":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

User\_id: bigint UNIQUE NOT NULL CONSTRAINT fk\_user\_featured REFERENCES users (Id) - поле, содержащее идентификатор пользователя, связанное с полем "Id" таблицы "users". Уникальное значение. Ссылается на идентификатор пользователя.

Game\_id: bigint UNIQUE NOT NULL CONSTRAINT fk\_game\_featured REFERENCES games (Id) - поле, содержащее идентификатор игры, связанное с полем "Id" таблицы "games". Уникальное значение. Ссылается на идентификатор игры.

Time: timestamp NOT NULL - поле, содержащее дату и время.

Таблица "games":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для игр.

Token: varchar(50) CHECK(Token != '') NOT NULL - поле, содержащее токен игры. Ограничено до 50 символов. Проверяет, чтобы значение поля не было пустым.

Token\_network: varchar(50) CHECK(Token\_network != '') - поле, содержащее сеть токена игры. Ограничено до 50 символов. Проверяет, чтобы значение поля не было пустым.

Game\_genre: varchar(50) CHECK(Game\_genre != '') NOT NULL - поле, содержащее жанр игры. Ограничено до 50 символов. Проверяет, чтобы значение поля не было пустым.

Free\_tier: boolean DEFAULT true - поле, указывающее, является ли игра бесплатной. Значение по умолчанию - true.

Min\_bet: decimal NOT NULL - поле, содержащее минимальную ставку игры. Не может быть пустым.

Max\_bet: decimal NOT NULL - поле, содержащее максимальную ставку игры. Не может быть пустым.

Таблица "rooms":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для комнат.

Created\_at: timestamp NOT NULL - поле, содержащее дату и время создания комнаты. Не может быть пустым.

Created\_by: bigint NOT NULL CONSTRAINT fk\_rooms\_user REFERENCES users (Id) - поле, содержащее идентификатор пользователя, создавшего комнату. Ссылается на идентификатор пользователя.

Game\_id: bigint NOT NULL CONSTRAINT fk\_rooms\_game REFERENCES games (Id) - поле, содержащее идентификатор игры, связанное с полем "Id" таблицы "games". Ссылается на идентификатор игры.

Таблица "room\_users":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

User\_id: bigint NOT NULL CONSTRAINT fk\_room\_users\_user REFERENCES users (Id) - поле, содержащее идентификатор пользователя, связанное с полем "Id" таблицы "users". Ссылается на идентификатор пользователя.

Connection\_date: timestamp NOT NULL - поле, содержащее дату и время подключения пользователя к комнате. Не может быть пустым.

User\_type: varchar(20) NOT NULL - поле, содержащее тип пользователя в комнате. Не может быть пустым.

Room\_id: bigint NOT NULL CONSTRAINT fk\_rooms\_users\_room REFERENCES rooms (Id) - поле, содержащее идентификатор комнаты, связанное с полем "Id" таблицы "rooms". Ссылается на идентификатор комнаты.

Таблица "game\_process":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

Bet: decimal NOT NULL - поле, содержащее ставку игры. Не может быть пустым.

Creation\_time: timestamp NOT NULL - поле, содержащее дату и время создания игрового процесса. Не может быть пустым.

Finish\_time: timestamp - поле, содержащее дату и время завершения игрового процесса.

Game\_id: bigint NOT NULL CONSTRAINT fk\_game\_process\_game REFERENCES games (Id) - поле, содержащее идентификатор игры, связанное с полем "Id" таблицы "games". Ссылается на идентификатор игры.

Room\_id: bigint NOT NULL CONSTRAINT fk\_game\_process\_room REFERENCES rooms (Id) - поле, содержащее идентификатор комнаты, связанное с полем "Id" таблицы "rooms". Ссылается на идентификатор комнаты.

Таблица "results":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

User\_id: bigint NOT NULL CONSTRAINT fk\_results\_user REFERENCES users (Id) - поле, содержащее идентификатор пользователя, связанное с полем "Id" таблицы "users". Ссылается на идентификатор пользователя.

Balance\_results: money NOT NULL - поле, содержащее результат баланса пользователя. Не может быть пустым.

Game\_process\_id: bigint NOT NULL CONSTRAINT fk\_results\_game\_process REFERENCES game\_process (Id) - поле, содержащее идентификатор игрового процесса, связанное с полем "Id" таблицы "game\_process". Ссылается на идентификатор игрового процесса.

Таблица "admins":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

User\_id: bigint NOT NULL CONSTRAINT fk\_admin\_user REFERENCES users (Id) - поле, содержащее идентификатор пользователя, связанное с полем "Id" таблицы "users". Ссылается на идентификатор пользователя.

Access\_level: smallint NOT NULL - поле, содержащее уровень доступа администратора. Не может быть пустым.

Таблица "moderation\_logs":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

Admin\_id: bigint NOT NULL CONSTRAINT fk\_moderation\_logs\_admin REFERENCES admins (Id) - поле, содержащее идентификатор администратора, связанное с полем "Id" таблицы "admins". Ссылается на идентификатор администратора.

Action: varchar(50) NOT NULL - поле, содержащее действие, выполненное администратором. Не может быть пустым.

Timme: timestamp NOT NULL - поле, содержащее дату и время выполнения действия администратором. Не может быть пустым.

Таблица "messages":

Id: serial (первичный ключ) - поле, автоматически генерирующее уникальные идентификаторы для записей в таблице.

Sender\_id: bigint NOT NULL CONSTRAINT fk\_messages\_sender REFERENCES users (Id) - поле, содержащее идентификатор отправителя сообщения, связанное с полем "Id" таблицы "users". Ссылается на идентификатор пользователя.

Resiever\_id: bigint NOT NULL CONSTRAINT fk\_messages\_resiever REFERENCES users (Id) - поле, содержащее идентификатор получателя сообщения, связанное с полем "Id" таблицы "users". Ссылается на идентификатор пользователя.

Game\_id: bigint NOT NULL CONSTRAINT fk\_messages\_game REFERENCES games (Id) - поле, содержащее идентификатор игры, связанное с полем "Id" таблицы "games". Ссылается на идентификатор игры.

Content: varchar(100) NOT NULL - поле, содержащее содержимое сообщения. Не может быть пустым.

**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.

Эта база данных может быть использована для создания сервиса с онлайн играми, управления им.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для сайта игровой вселенной необходимо создать следующие таблицы:

Таблица "users":

Хранит информацию о пользователях системы.

Поля таблицы включают идентификатор пользователя (Id), имя пользователя (Name), дату регистрации (Register\_date), электронную почту (Email), статус администратора (Admin), идентификатор истории игр (Games\_history) и идентификатор выделенных игр (Games\_featured).

Таблица "featured":

Хранит информацию о выделенных играх, которые отображаются на главной странице.

Поля таблицы включают идентификатор (Id), идентификатор пользователя (User\_id), идентификатор игры (Game\_id) и дату и время (Time).

Таблица "games":

Хранит информацию о играх.

Поля таблицы включают идентификатор игры (Id), токен игры (Token), сеть токена (Token\_network), жанр игры (Game\_genre), флаг бесплатной версии игры (Free\_tier), минимальную ставку (Min\_bet) и максимальную ставку (Max\_bet).

Таблица "rooms":

Хранит информацию о комнатах, созданных для игр.

Поля таблицы включают идентификатор комнаты (Id), дату и время создания (Created\_at), идентификатор создателя комнаты (Created\_by) и идентификатор игры (Game\_id).

Таблица "room\_users":

Связующая таблица, которая отображает связь между пользователями и комнатами.

Поля таблицы включают идентификатор (Id), идентификатор пользователя (User\_id), дату и время подключения (Connection\_date), тип пользователя (User\_type) и идентификатор комнаты (Room\_id).

Таблица "game\_process":

Хранит информацию о текущих игровых процессах в комнатах.

Поля таблицы включают идентификатор (Id), размер ставки (Bet), дату и время создания (Creation\_time), дату и время завершения (Finish\_time), идентификатор игры (Game\_id) и идентификатор комнаты (Room\_id).

Таблица "results":

Хранит информацию о результатах игр пользователей.

Поля таблицы включают идентификатор (Id), идентификатор пользователя (User\_id), баланс результатов (Balance\_results) и идентификатор игрового процесса (Game\_process\_id).

Таблица "admins":

Хранит информацию об администраторах системы.

Поля таблицы включают идентификатор (Id), идентификатор пользователя (User\_id) и уровень доступа (Access\_level).

Таблица "messages":

Хранит информацию о сообщениях между пользователями.

Поля таблицы включают идентификатор (Id), идентификатор отправителя (Sender\_id), идентификатор получателя (Resiever\_id), идентификатор игры (Game\_id) и содержание сообщения (Content).

Для более удобной работы с базой данных можно создать несколько функций и процедур. Например, можно создать функцию для поиска пользователя по его email. Также можно создать процедуру для добавления пользователей в базу данных.

### 3.1.1 Представления базы данных

Представление (view) 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:

* Confirmed\_articles, которое объединяет данные таблиц Users и articles, а также получает количество лайков на статью;
* Full\_messages, которое содержит информацию о сообщении и пользователе, а также лайках к сообщению

Представление Full\_messages было создано для того, чтобы получить полную информацию о пользователях, сообщениях и лайках, объединив данные из трёх таблиц. Оно будет на листинге 3.1. Остальные представления будут аналогичны, только будут работать с другими таблицами.

|  |
| --- |
| CREATE or REPLACE VIEW full\_Messages AS  SELECT m.id, t.name as thread, m.content, m.picture, u.id senderId, u.Name as sender, m.creation\_date, s.message\_likes  FROM messages m  INNER JOIN threads t ON t.id = m.thread\_id  INNER JOIN users u ON u.id = m.user\_id  LEFT JOIN (SELECT l.Message\_id, COUNT(l.Message\_id) as message\_likes  FROM messages\_likes l  GROUP BY l.Message\_id) s  ON s.Message\_id = m.id  ORDER BY m.creation\_date DESC; |

Листинг 3.1 – Представление Full\_messages

Представление Confirmed\_articles было создано для того, чтобы получить информацию о добавленных статьях.

### 3.1.2 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 быстрого полнострочного поиска был создан индекс

|  |
| --- |
| CREATE INDEX textsearch\_idx ON article USING GIN (textsearchable\_index\_col); |

Листинг 3.2 – Индексы базы данных для полнострочного поиска

В целом, использование индексов позволяет существенно ускорить операции поиска, сортировки и фильтрации данных в базе данных, особенно в случае большого объема данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах, поэтому необходимо сбалансировать количество и тип индексов для оптимальной производительности базы данных.

### 3.1.3 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.

AFTER Триггер, созданный в таблице Articles, добавляет вектор на основании трёх других с изменённым весом, которые получают только изменённые данные. Скрипт триггера будет представлен на листинге 3.3.

|  |
| --- |
| CREATE OR REPLACE FUNCTION update\_textsearchable\_index\_col()  RETURNS TRIGGER AS $$  BEGIN  NEW.textsearchable\_index\_col :=  setweight(to\_tsvector('english', coalesce(NEW.name,'')), 'A') ||  setweight(to\_tsvector('english', coalesce('D&D FAQ','')), 'B') ||  setweight(to\_tsvector('english', coalesce(NEW.content,'')), 'D');  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE OR REPLACE TRIGGER trigger\_update\_textsearchable\_index\_col  BEFORE INSERT OR UPDATE ON article  FOR EACH ROW  EXECUTE FUNCTION update\_textsearchable\_index\_col(); |

Листинг 3.3 – Скрипт триггера check\_password

Таким образом, триггер выполняет валидацию данных перед их вставкой или обновлением в таблице Users, обеспечивая целостность данных.

## 3.2 Описание используемой технологии

Полнотекстовый поиск (или просто поиск текста) — это возможность находить документы на естественном языке, соответствующие запросу, и, возможно, дополнительно сортировать их по релевантности для этого запроса. Наиболее распространённая задача — найти все документы, содержащие слова запроса, и выдать их отсортированными по степени соответствия запросу. Понятия запроса и соответствия довольно расплывчаты и зависят от конкретного приложения. В самом простом случае запросом считается набор слов, а соответствие определяется частотой слов в документе.

Операторы текстового поиска существуют в СУБД уже многие годы. В Postgres Pro для текстовых типов данных есть операторы ~, ~\*, LIKE и ILIKE, но им не хватает очень важных вещей, которые требуются сегодня от информационных систем:

Нет поддержки лингвистического функционала, даже для английского языка. Возможности регулярных выражений ограничены — они не рассчитаны на работу со словоформами, например, подходят и подходить. С ними вы можете пропустить документы, которые содержат подходят, но, вероятно, и они представляют интерес при поиске по ключевому слову подходить. Конечно, можно попытаться перечислить в регулярном выражении все варианты слова, но это будет очень трудоёмко и чревато ошибками (некоторые слова могут иметь десятки словоформ).

Они не позволяют упорядочивать результаты поиска (по релевантности), а без этого поиск неэффективен, когда находятся сотни подходящих документов.

Они обычно выполняются медленно из-за отсутствия индексов, так как при каждом поиске приходится просматривать все документы. Полнострочный же поиск устраняет эти недостатки. Вот пример поиска:

|  |
| --- |
| SELECT \*, ts\_rank\_cd(textsearchable\_index\_col, to\_tsquery('aa & q')) as rank  FROM article  WHERE textsearchable\_index\_col @@ to\_tsquery('aa & q')  ORDER BY rank DESC |

Листинг 3.5 – Cкрипт полнострочного поиска

Полнострочный поиск разбивает на лексемы документ на основе словаря, в данном случае применяется английская версия, ниже пример разбиения:

|  |
| --- |
| 'aa':2A 'b':8,26 'bmftp':15 'c':20,24 'cak':19 'codfnjoikd':10 'd':5B,6B 'diudcbbuk':17 'egpwkmq':27 'f':21 'faq':7B 'fm':28 'ii':13 'khwyey':30 'kqbwdrequranla':3A 'lamjdakd':12 'nea':16 'ocntw':14 'onguq':29 'q':1A 'qbg':31 'rirft':23 'wvxqgesstpcpsfrj':11 'x':18 'xr':25 'ye':4A 'ypwsh':9 |

Листинг 3.6 – Вывод разбиения строки

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для крипто-валютного кошелька. Tакже была описана использованная технология полнострочного поиска

4 Установка, настройка и использование PosgtreSQL 14.5

## 4.1 Установка PostgreSQL

PostgreSQL - это мощная реляционная база данных с открытым исходным кодом, которая предлагает множество функций, таких как многопоточность, транзакционность, контроль целостности данных, масштабируемость и многое другое. PostgreSQL широко используется в коммерческих и научных проектах, а также веб-приложениях и мобильных приложениях.

После установки PostgreSQL на сервер, была произведена конфигурация сервера для оптимальной работы с базой данных. Затем была создана база данных с названием kursach\_db, которая будет использоваться в дальнейшем для хранения данных крипто-валютного кошелька.

## 4.2 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 4.1.

|  |
| --- |
| CREATE TABLESPACE tb\_articles '/tablespaces';  CREATE TABLESPACE tb\_messages'/tablespaces';  CREATE TABLESPACE tb\_common'/tablespaces'; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Таблицы articles, article\_likes будут храниться в табличном пространстве tb\_articles, таблицы messages, message\_likes- в табличном пространстве tb\_messages, остальные в tb\_common.

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 кошельков разных валют.

В представленной схеме базы данных есть связь многие ко многим между таблицами "users" и "profile\_decorations", так как каждый пользователь может содержать различные декорации. Скрипт создание таблицы User будет представлен на листинге 4.2.

|  |
| --- |
| create table users  (  Id serial primary key,  Name varchar(50) check(Name !='') not null,  Register\_date timestamp default CURRENT\_TIMESTAMP not null,  Email varchar(50) unique check(Email !='') not null,  Password varchar(50) not null check(length(password) > 3),  Admin boolean not null default false,  Icon varchar(100) check(Icon !='')  ); |

Листинг 4.2 – Cкрипт создание таблицы User

## 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.

Будут созданы три роли для разграничения доступа к базе данных: reglarUser и admin. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли reglarUser являются основными, в то время как роль admin будет иметь более широкий набор привилегий.

Роль admin имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли admin можно увидеть на листинге 4.3.

|  |
| --- |
| GRANT ALL PRIVILEGES ON DATABASE "kursach\_db" TO admin;  GRANT CONNECT ON DATABASE "kursach\_db" TO admin;  GRANT ALL ON TABLESPACE tb\_user TO admin;  GRANT ALL ON TABLESPACE tb\_cryptocurr TO admin; |

Листинг 4.3 – Привилегии, выданные роли admin

Роль reglarUser может зарегистрироваться, входить входить в аккаунт,проводить транзакции, а также пополнять баланс. Кроме того, reglarUser имеет возможность изменять свой профиль и пароль. Роль также может оценивать треки и удалять их из своих плейлистов или библиотеки. Выданные привилегии роли reglarUser можно увидеть на листинге 4.4.

|  |
| --- |
| CREATE ROLE regularUser;  GRANT USAGE ON SCHEMA regularAccess TO regularUser;  GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA regularAccess TO regularUser;  GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA regularAccess TO regularUser;  GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO regularUser;  GRANT USAGE, SELECT ON SEQUENCE users\_id\_seq TO regularUser;  -- И даём её пользователю  CREATE USER serverUser WITH PASSWORD 'qwerty123456';  GRANT regularUser TO serverUser; |

Листинг 4.4 – Привилегии, выданные роли reglarUser

Таким образом, были созданы 2 роли для ограничения доступа к базе данных: reglarUser и admin. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль admin имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль reglarUser имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию.

## 4.4 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. Заполнение таблиц 100 000 строк;
3. Добавление данных в таблицы;
4. Удаление данных из таблиц;
5. Изменение данных в таблицах;
6. 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.

### 4.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: getUserIdByEmail, userExists. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

getUserIdByEmail, userExists - функции для выборки списка пользователей, транзакций и их кошельков.

На листинге 4.5 будет функция get\_users, которая предназначена для выборки всех пользователей из таблицы Users.

|  |
| --- |
| create or replace function getUserIdByEmail(search\_email varchar(50))  returns int  language plpgsql  as  $$  begin  return (SELECT id  FROM users  WHERE email = search\_email);  end;  $$; |

Листинг 4.5 – Функция getUserIdByEmail

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

### 4.4.2 Заполнение таблиц 100 000 строк

Для заполнения таблицы users была разработана процедура insert\_users которая вставляет 100000 строк в таблицу. Процедура представлена на листинге 4.6.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE fill\_articles()  LANGUAGE plpgsql  AS $$  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO ARTICLE (Theme\_id, user\_id, creation\_date, content, picture, name)  values(1, 11, CURRENT\_TIMESTAMP, generate\_random\_string(50, 500), 'picture', generate\_random\_string(5, 20));  END LOOP;  END;  $$; |

Листинг 4.6 – Процедура заполнения таблицы users

Процедура insert\_users была создана для заполнения таблицы users 100000 строками. В теле функции используется цикл FOR, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу users. Процедура не возвращает значение и не принимает аргументов. Для выполнения функции необходимо выполнить SELECT insert\_users ().

### 4.4.3 Добавление данных в таблицы

Были разработаны следующие процедуры и функции для добавления новых строк в основные таблицы базы данных: register\_user, create\_wallet, insert\_transaction, log\_user. Ниже будут описание каждой функции или процедуры.

Были разработаны следующие функции и процедуры для работы с базой данных крипто-валютного кошелька.

addNewUser - процедура, которая добавляет нового пользователя в таблицу users. Принимает значения для полей email, user\_password.

getUserByEmail - функция, которая проверяет правильность ввода пароля для пользователя по указанному email. Принимает значения email и password.

На листинге 4.7 будет функция log\_user, которая логирует пользователя в системе.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE regularAccess.addNewUser (  Name varchar(20),  Email varchar(50),  Password varchar(50),  UserId inout int default 1,  Admin boolean default false  )  LANGUAGE plpgsql  SECURITY DEFINER  AS $$  BEGIN  BEGIN  INSERT INTO public.users (Name, Email, Password, Admin)  VALUES (Name, Email, Password, Admin) RETURNING Id INTO UserId;  EXCEPTION WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  END;  $$; |

Листинг 4.7 – Функция заполнения таблицы log\_user

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.4 Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры: delete\_user;

На листинге 4.8 будет процедура delete\_user

|  |
| --- |
| create or replace procedure regularAccess.deleteUser  (UserId int)  LANGUAGE plpgSQL  SECURITY DEFINER  AS $$  begin  delete from users  where Id = UserId;  EXCEPTION WHEN OTHERS THEN  ROLLBACK;  RAISE;  end;  $$; |

Листинг 4.8 – Процедура для удаления user

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

### 4.4.5 Изменение данных в таблицы

Для изменение данных в базе данных были созданы следующие процедуры: update\_user\_email, update\_user\_password, update\_wallet\_address. Ниже будут описание каждой функции или процедуры.

update\_user\_email - обновляет поле email в таблице Users.

update\_user\_password - обновляет поле пароль в таблице Users.

update\_wallet\_address - обновляет обновляет адрес кошелька пользователя в таблице wallets.

На листинге 4.9 будет процедура UPDATE\_USER\_EMAIL.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE regularAccess.updateUser  (  updateUserId int,  updateName varchar(20) default null,  updateEmail varchar(50) default null,  updatePassword varchar(100) default null,  updateAdmin boolean default null  )  SECURITY DEFINER  LANGUAGE plpgsql  AS $$  DECLARE cursorId int; cursorName varchar(20); cursorEmail varchar(50); cursorPassword varchar(50); cursorAdmin boolean;  DECLARE updateCursor CURSOR FOR SELECT Id, name, email, password, admin FROM users FOR UPDATE;  BEGIN  OPEN updateCursor;  LOOP  FETCH updateCursor INTO cursorId, cursorName, cursorEmail, cursorPassword, cursorAdmin;  IF cursorId = updateUserId THEN  UPDATE users SET name = COALESCE(updateName, cursorName), email = COALESCE(updateEmail, cursorEmail),  password = COALESCE(updatePassword, cursorPassword), admin = COALESCE(updateAdmin, cursorAdmin)  WHERE CURRENT OF updateCursor;  return;  END IF;  EXIT WHEN NOT FOUND;  END LOOP;  END;  $$; |

Листинг 4.9 – Процедура для обновления пользователя

Все остальные процедуры будут аналогичны, также предназначены для изменения соответствующих данных в основных таблицах базы данных.

### 4.4.6 Дополнительные функции

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

Процедура deleteUserMessage позволяет удалить определённое сообщение пользователя. Процедура представлена на листинге 4.10.

|  |
| --- |
| create or replace procedure regularAccess.deleteUserMessage  (  s\_id int default 0,  r\_id int default 0  )  LANGUAGE plpgSQL  AS $$  begin  if s\_id != 0  delete from messages  where sender\_id = s\_id;  elseif  commit;  end;  $$; |

Листинг 4.10 – deleteUserMessage

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования PostgreSQL. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и пакетов процедур для выполнения различных операций с данными.

Были также представлены процедуры экспорта и импорта данных в формате Xml, а также были проведены тесты производительности базы данных. В результате тестирования было установлено, что база данных PostgreSQL обладает высокой производительностью и способна быстро обрабатывать запросы на получение данных.

Итак, можно сделать вывод, что PostgreSQL является мощной и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка PostgreSQL, а также оптимизация запросов, позволят обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

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 wallets, содержащая больше всего данных. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| EXPLAIN ANALYZE SELECT email FROM articles order by id; |

Листинг 5.1 – Запрос к таблице users

Благодаря установки колонки таблицы, выделенной под tsvector и добавлению индекса, удалось уменьшить время запроса до 1 ms

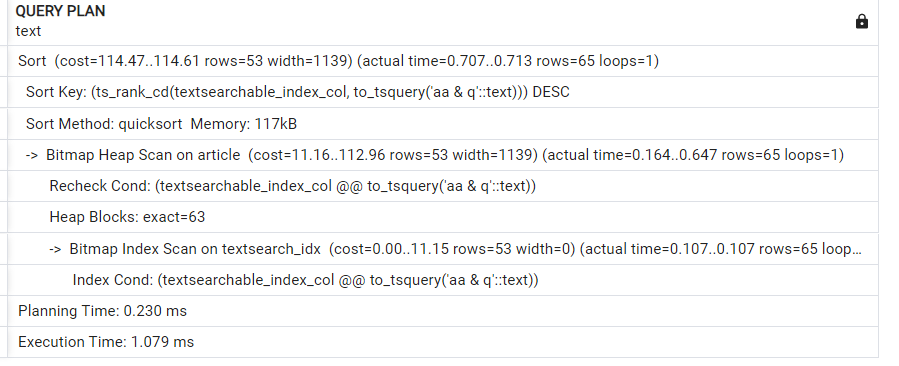


Рисунок 5.2 – Результат выполнения запроса

## 5.2 Вывод

В данном разделе было рассмотрено важное понятие тестирования производительности базы данных. Для проведения тестирования была выбрана таблица articles с большим количеством данных. Проведенный тест показал, что создание индекса на поле textsearchable\_index\_col значительно улучшило производительность запроса к таблице articles, сократив время выполнения запроса и уменьшив стоимость выполнения запроса. Выводом является то, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных, что особенно важно при работе с большими объемами данных.

6 Руководство по использованию программного средства

## Руководство пользователя

Когда пользователь впервые заходит на наш веб-ресурс, его встречает главная страница, где возможно будущий клиент нашего крипто-кошелька может ознакомится с преимуществами работы именно с нашим кошельком, почитать о проекте и узнать много чего нового. Пример главной страницы приведен на рисунке 6.1.

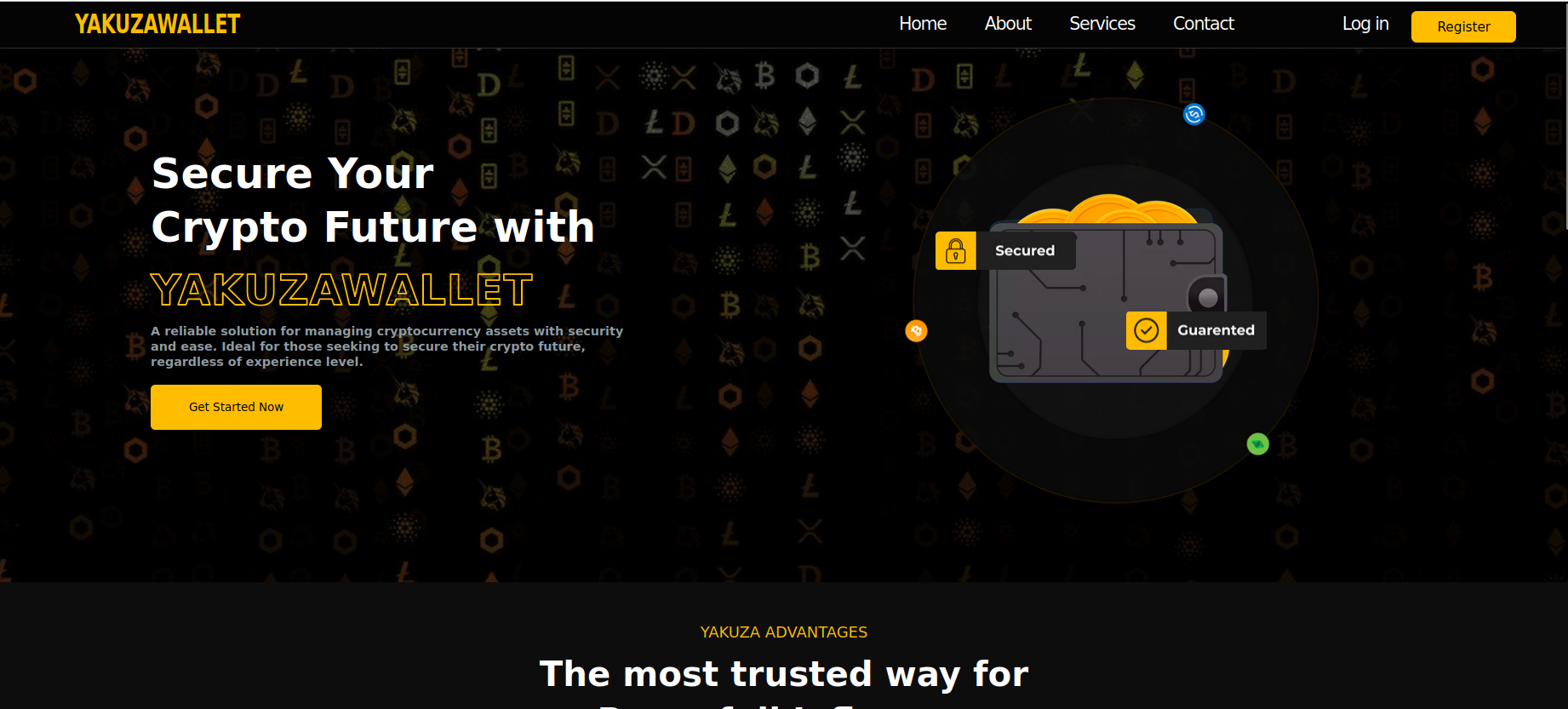


Рисунок 6.1 – Главная страница YAKUZAWALLET

После того как пользователь ознакомился с большей частью информации предствленной на сайте, он может спустится ниже и увидеть множество полезных ссылок, которыми он может воспользоваться, если у него возникнут проблемы. Рисунок 6.2 показывает, как выглядят эти полезные ссылки.

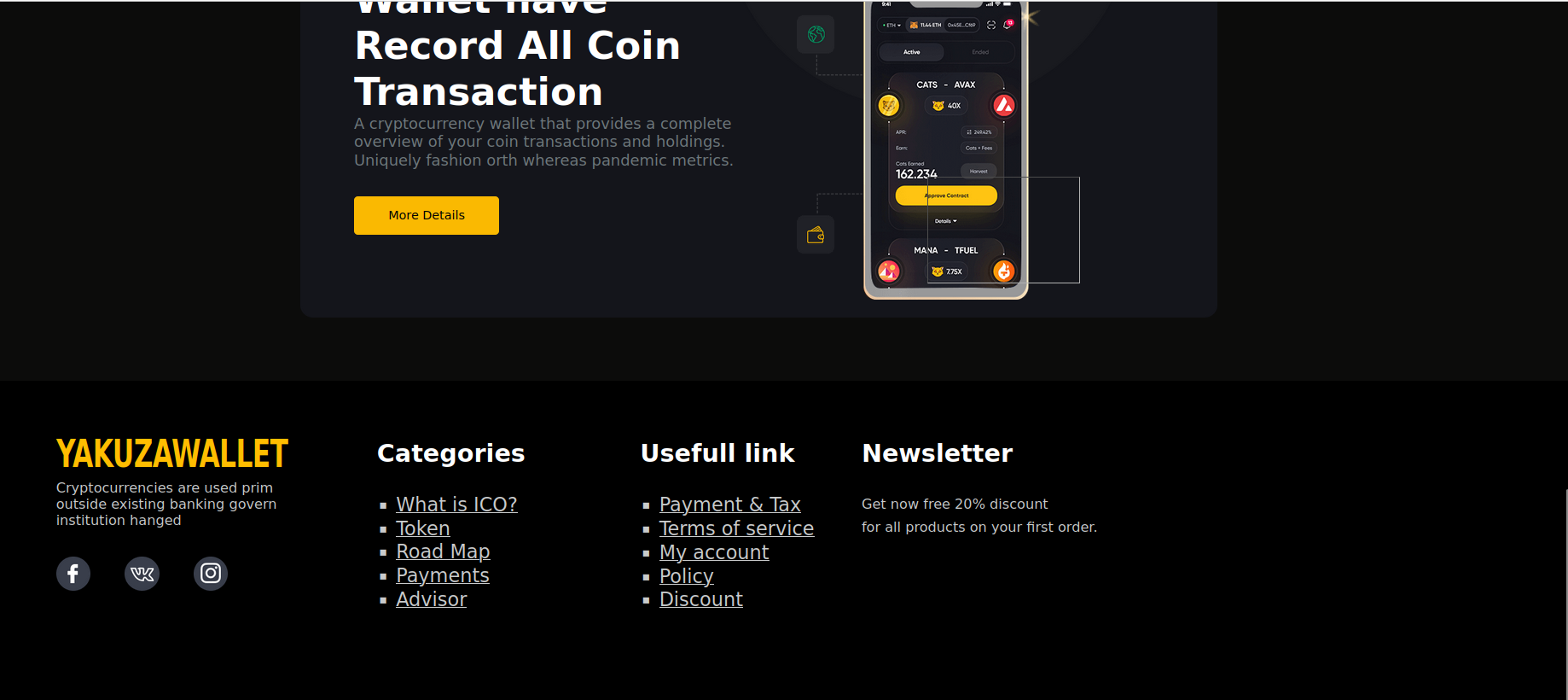


Рисунок 6.2 – Полезные ссылки

После того как пользователь решил попробовать наш крипто-кошелек, он попадает на страницу регистрации, где он может пройти ее. На рисунке 6.3 показана форма регистрации.

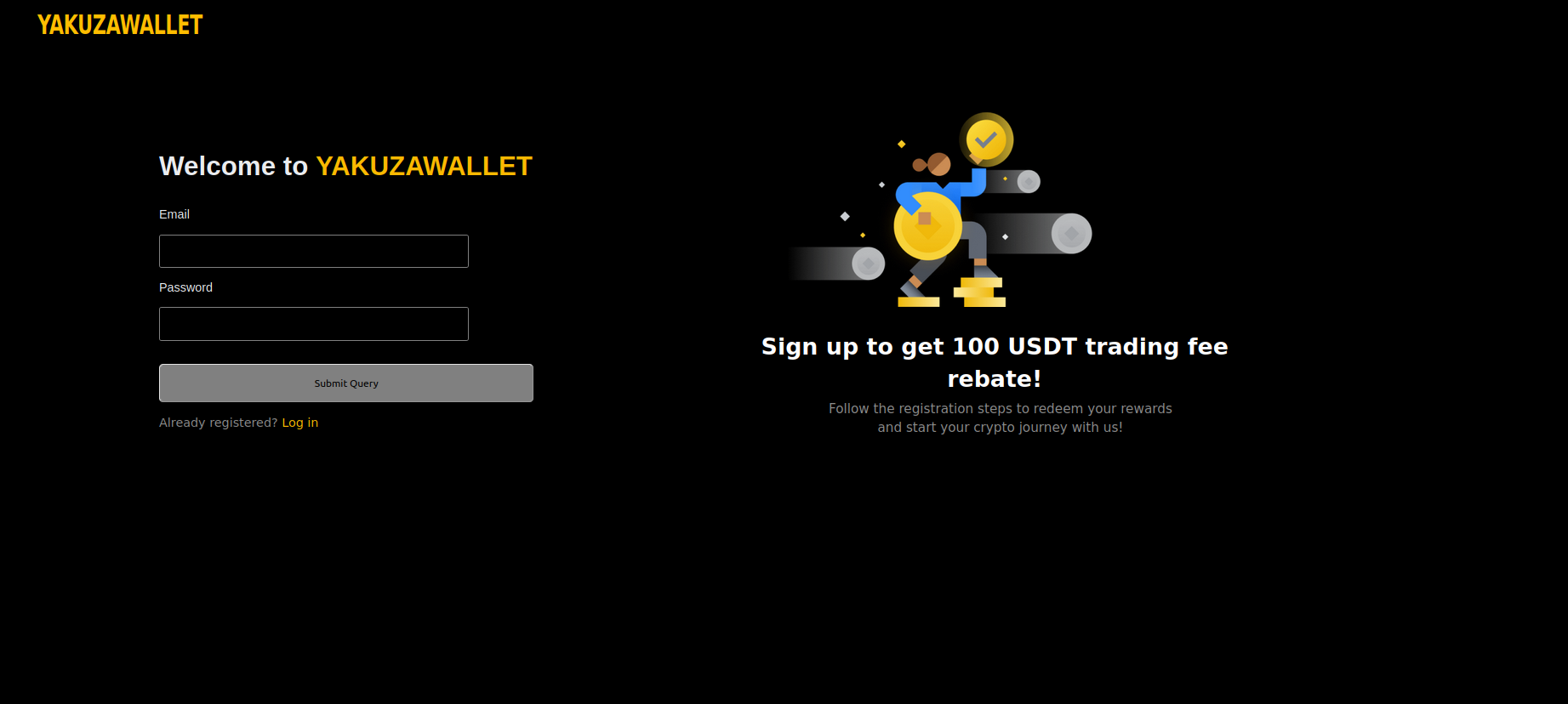


Рисунок 6.3 – Форма регистрации на сайте

Если пользователь уже есть в системе, то он перейдет по ссылке и попадет на страницу входа в кабинет крипто-кошелька. Который представлен на рисунке 6.4.

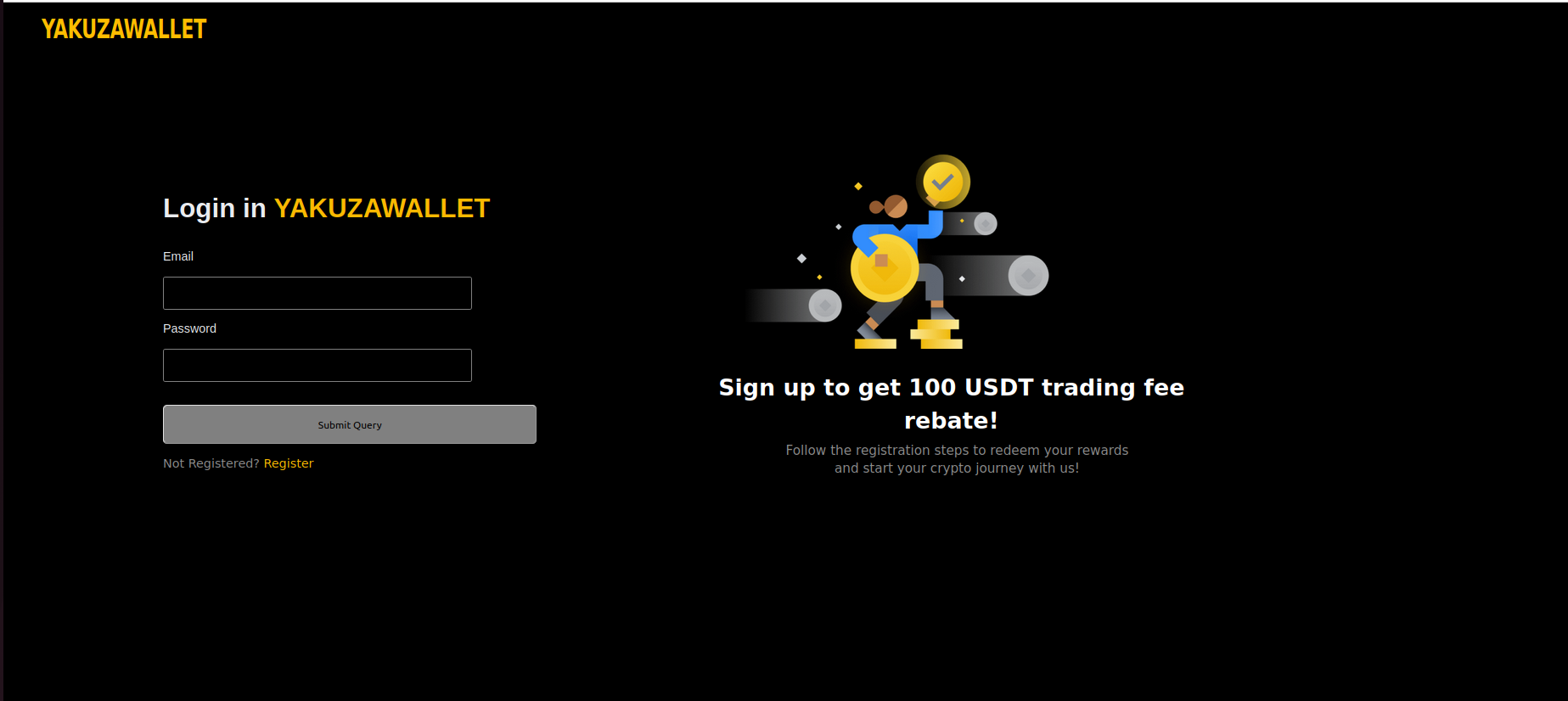


Рисунок 6.4 – Форма входа в кабинет

После того как пользователь прошел форму логина или прошел стадию регистрации он попадает на страницу своего портфеля, где он видит свои активы, на сайте представлены 3 крипто-валюты, такие как: BTC, USDc, ATOM. Рисунок 6.5 показывает страницу с портфелем.

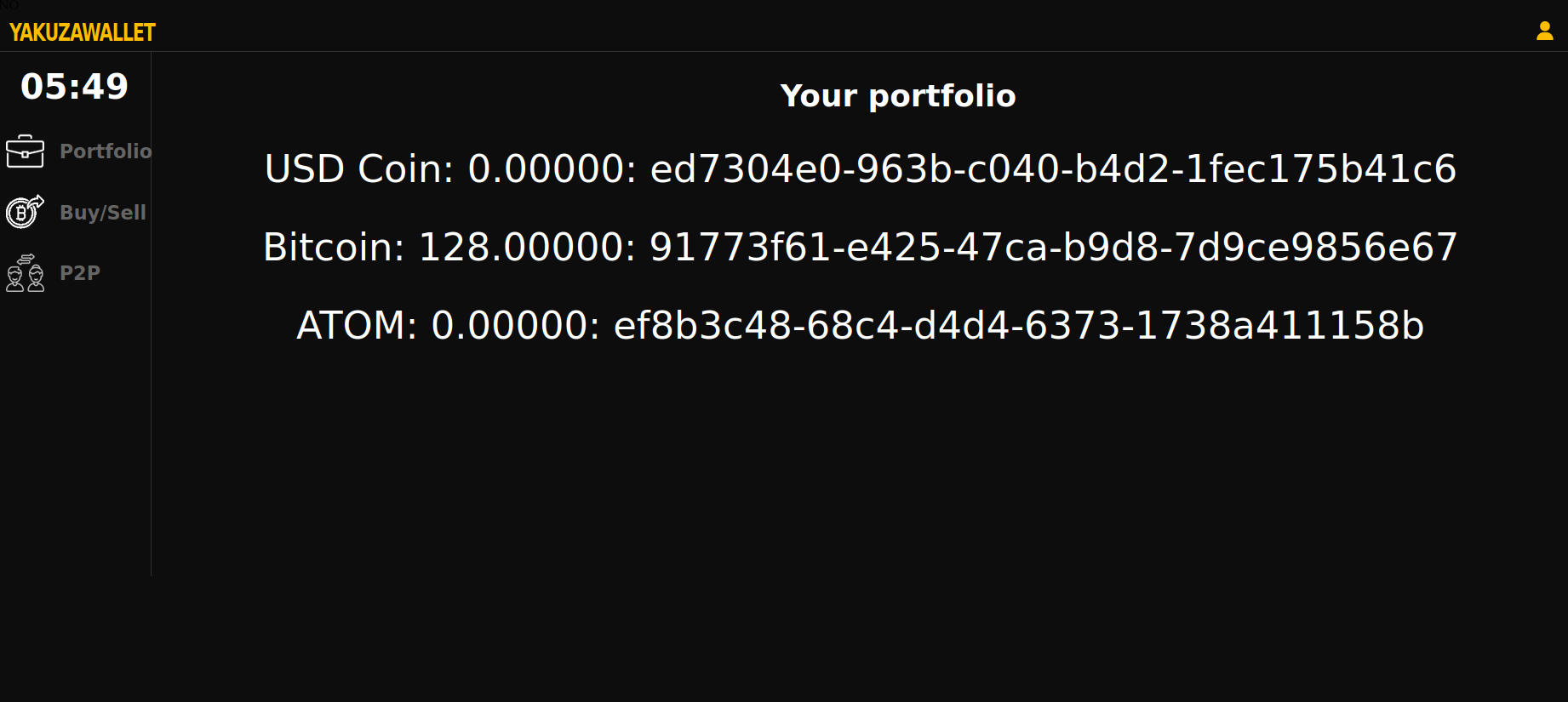


Рисунок 6.5 – Страница портфеля.

После того как пользователь увидит, что у него по нулям на балансе, он захочет пополнить баланс, для этого ему нужно перейти на страницу Buy/Sell, где он сможет пополнить интересующую его крипто-валюту. Пример данной страницу представлен на рисунке 6.6.

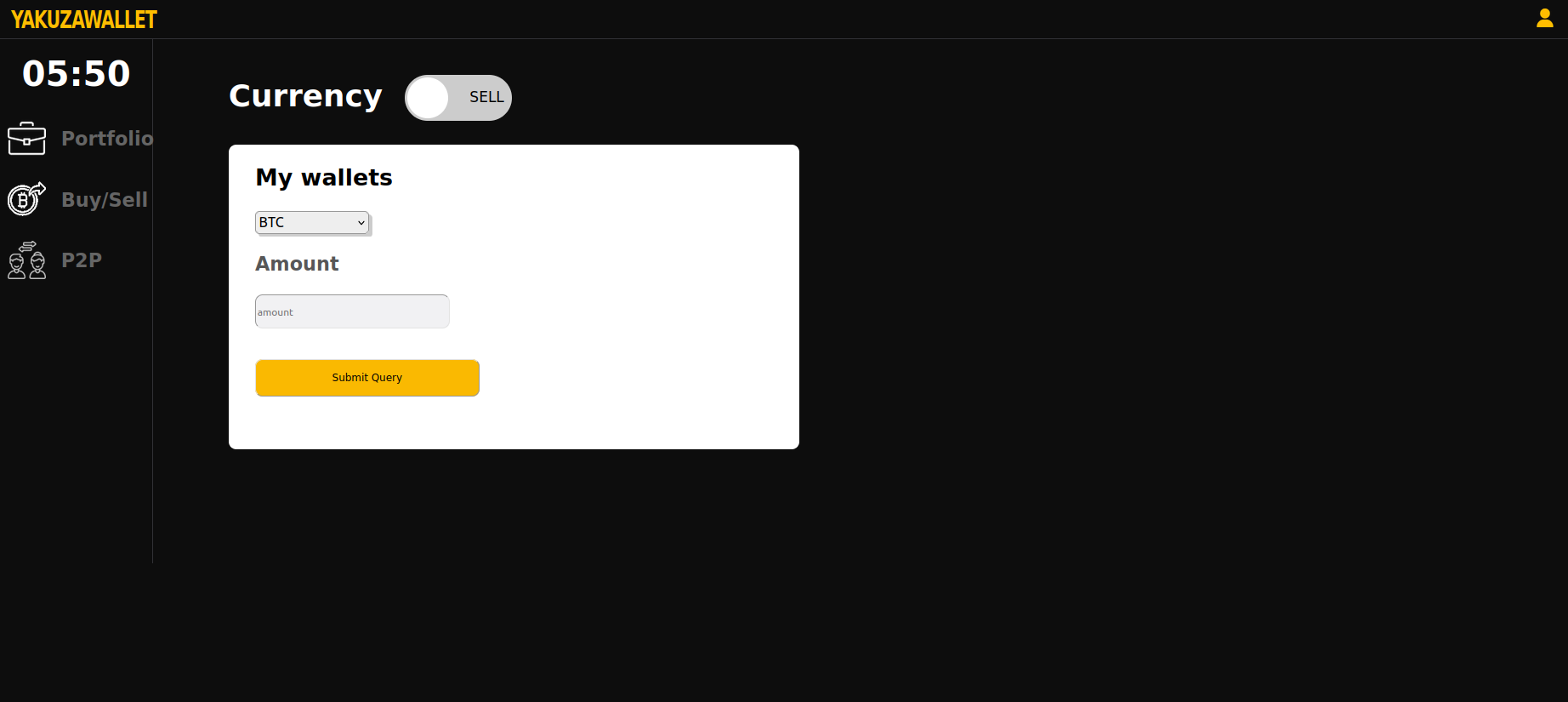


Рисунок 6.6. – Страница покупки/продажи криптовалюты

Вскоре пользователю понадобится переводить свои активы с кошелька на кошелек, проводить какие-то транзакции, в этом ему поможет страница с P2P, что означает, что тут он может перевести определенное количество крипты с одного кошелька на другой. Рисунок 6.7 показывает его интерфейс.

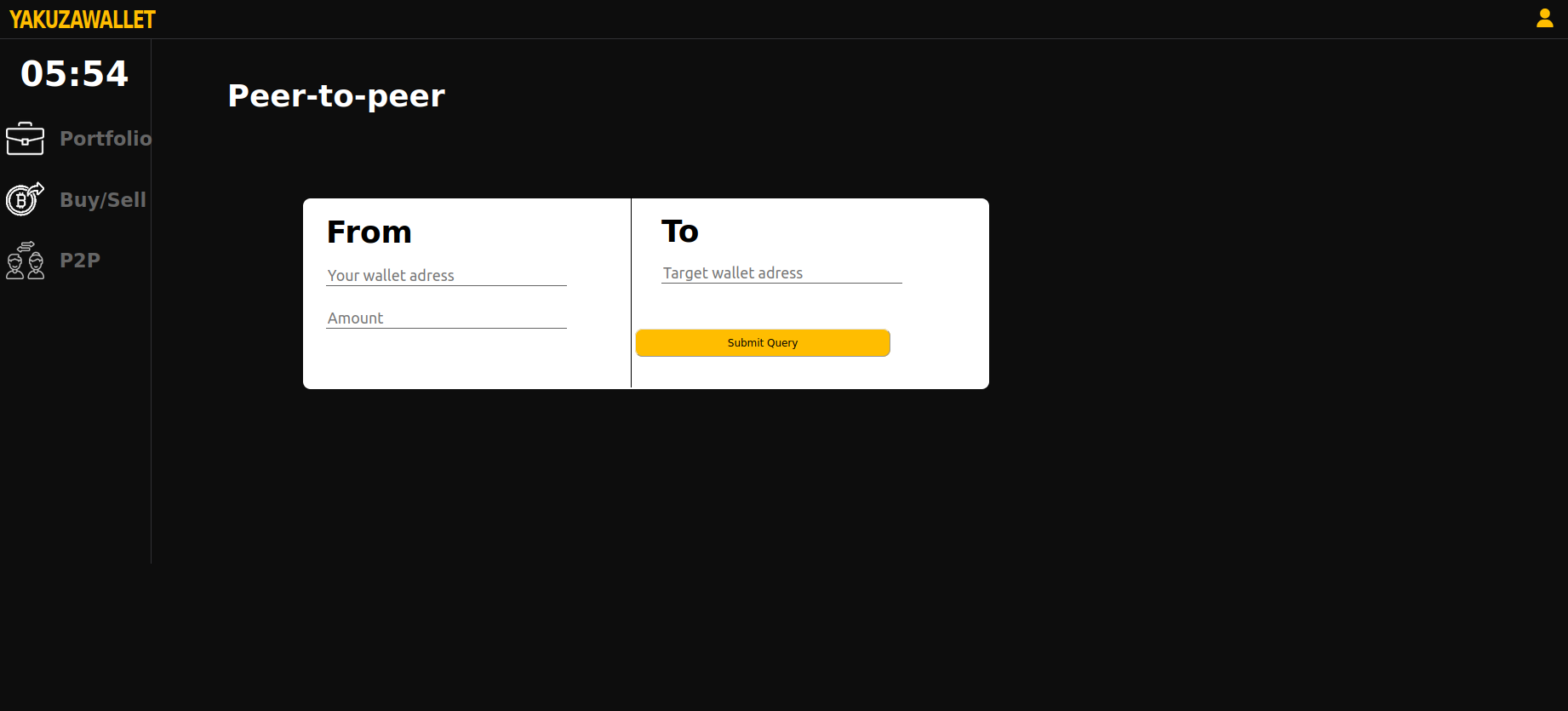


Рисунок 6.7 – Интерфейс P2P

Также пользователь имеет возможность поменять настройки своего аккаунта, такие как email, пароль или установить кастомный адрес крипто-кошелька. Сделать это он может в окне с настройками, который представлен на рисунке 6.8.

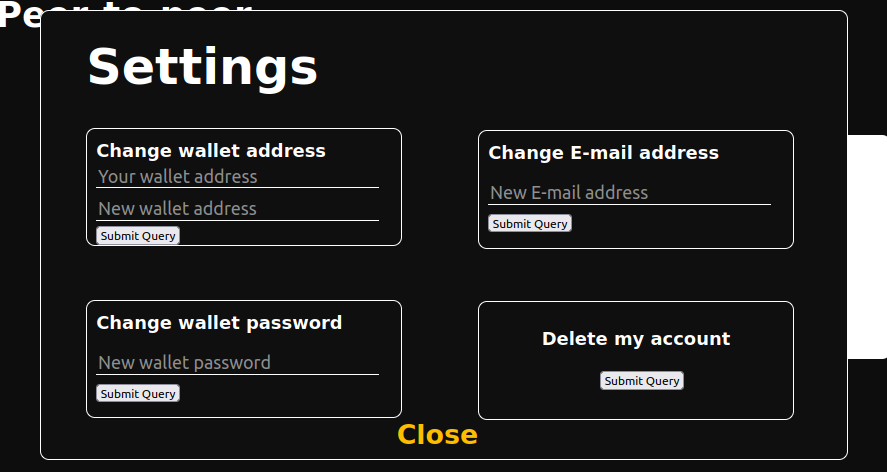


Рисунок 6.8 – Интерфейс настроек

## 6.2 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.

## 6.3 Вывод

В данном разделе были рассмотрены функциональные возможности приложения, а также права доступа для пользователей с различными ролями. Было показано, что может делать пользователь в нашей системе.

# Заключение

В современных организациях база данных играет важную роль в надежном хранении и управлении информацией. В этой работе была выполнена разработка базы данных для игровой вселенной Dungeons&Dragons, в которой была акцентирована внимание на полнострочном поиске.

Были использованы различные объекты, такие как таблицы, триггеры и функции, для обеспечения структурированного хранения данных и оперативного доступа к ним. Проведено тестирование базы данных на большом объеме данных, и полученные результаты оказались положительными.

Одной из ключевых особенностей разработанной базы данных является технология полнострочного поиска, она позволяет быстро и эффективно находить данные на большой выборке, что увеличивает эффективность использования энциклопедии.

# Список литературных источников

1. Книга “PostgreSql изнутри” автор Е.Рогов, год издания 2023.

2.dnd.su [Электронный ресурс] / Режим доступа: https://.dnd.su – Дата доступа: 20.04.2023.

3.dnd.wizards.com[Электронный ресурс] / Режим доступа:

https:// dnd.wizards.com– Дата доступа: 20.04.2023.

5. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 20.04.2023.

6. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 21.04.2023.

# Приложение А

|  |
| --- |
| --------------------- TABLESPACE ---------------------------  create table users  (  Id serial primary key,  Name varchar(50) check(Name !='') not null,  Register\_date timestamp default CURRENT\_TIMESTAMP not null,  Email varchar(50) unique check(Email !='') not null,  Password varchar(50) not null check(length(password) > 3)  Admin boolean not null default false,  Icon varchar(100) check(Icon !=''),  Icon\_decoration\_id bigint CONSTRAINT fk\_user\_icon\_decoration references Profile\_decoration(Id),  Background\_decoration\_id bigint CONSTRAINT fk\_user\_background\_decoration references Profile\_decoration(Id)  );  create table profile\_decoration  (  Id serial primary key,  Type int not null CONSTRAINT fk\_p\_decoration\_types\_profile\_decoration references p\_decoration\_types(Id),  Name varchar(50) check(Name !='') not null,  Picture varchar(50) check(Picture !='') not null,  Cost decimal not null  );  create table p\_decoration\_types  (  Id serial primary key,  Type varchar(50) unique check(Type !='') not null  );  create table threads  (  Id serial primary key,  Creator\_id int check(Creator\_id !=0) not null CONSTRAINT fk\_threads\_creator references users(Id),  Creation\_date timestamp default CURRENT\_TIMESTAMP not null,  Name varchar(50) check(Name !='') not null  );  create table messages  (  Id serial primary key,  Thread\_id bigint not null CONSTRAINT fk\_message\_thread references threads(Id),  User\_id bigint not null check(User\_id !=0) CONSTRAINT fk\_message\_user references users(Id),  Content varchar(50) check(Content !='') not null,  Picture varchar(100) check(Picture !='') not null,  Creation\_date timestamp not null default CURRENT\_TIMESTAMP  );  create table messages\_likes  (  Id serial primary key,  Message\_id bigint not null CONSTRAINT fk\_messages\_likes\_messge references messages(Id),  User\_id bigint not null CONSTRAINT fk\_messages\_likes\_user references users(Id)  );  create table topics  (  Id serial primary key,  Name varchar(100) check(Name !='') unique not null  );  create table article  (  Id serial primary key,  Theme\_id bigint not null CONSTRAINT fk\_article\_theme references topics(Id),  User\_id bigint not null CONSTRAINT fk\_article\_user references users(Id),  Content varchar(10000) check(Content !='') not null,  Picture varchar(100) check(Picture !=''),  Creation\_date timestamp default CURRENT\_TIMESTAMP not null  );  ALTER TABLE article ADD COLUMN name varchar(100) check(name != '') not null;  ALTER TABLE article ADD COLUMN textsearchable\_index\_col tsvector;  UPDATE article SET textsearchable\_index\_col =  setweight(to\_tsvector('english', coalesce(name,'')), 'A') || setweight(to\_tsvector('english', coalesce('D&D FAQ','')), 'B') || setweight(to\_tsvector('english', coalesce(content,'')), 'D');  select \* from article  CREATE INDEX textsearch\_idx ON article USING GIN (textsearchable\_index\_col);  CREATE OR REPLACE FUNCTION update\_textsearchable\_index\_col()  RETURNS TRIGGER AS $$  BEGIN  NEW.textsearchable\_index\_col :=  setweight(to\_tsvector('english', coalesce(NEW.name,'')), 'A') ||  setweight(to\_tsvector('english', coalesce('D&D FAQ','')), 'B') ||  setweight(to\_tsvector('english', coalesce(NEW.content,'')), 'D');  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE OR REPLACE TRIGGER trigger\_update\_textsearchable\_index\_col  BEFORE INSERT OR UPDATE ON article  FOR EACH ROW  EXECUTE FUNCTION update\_textsearchable\_index\_col();  create table article\_likes  (  Id serial primary key,  Article\_id bigint not null CONSTRAINT fk\_article\_likes\_article references article(Id),  User\_id bigint not null CONSTRAINT fk\_article\_likes\_user references users(Id)  );  create table edited\_articles  (  Id serial primary key,  User\_id bigint not null CONSTRAINT fk\_edited\_articles\_user references users(Id),  Article\_id bigint not null CONSTRAINT fk\_edited\_articles\_article references article(Id),  Awaiting\_confirmation boolean not null default true,  Creation\_date timestamp not null,  Picture varchar(100) check(Picture !='') not null  ); |

Листинг 1 – Скрипты создания таблиц

|  |
| --- |
| CREATE SCHEMA regularAccess;  CREATE OR REPLACE FUNCTION regularAccess.my\_procedure1()  RETURNS void AS $$  BEGIN  RAISE NOTICE 'This is my procedure 1';  END;  $$ LANGUAGE plpgsql;  ALTER PROCEDURE addnewuser SET SCHEMA regularAccess;  ALTER PROCEDURE deleteuser SET SCHEMA regularAccess;  ALTER PROCEDURE updateuser SET SCHEMA regularAccess;  ALTER FUNCTION userexists SET SCHEMA regularAccess;  ALTER FUNCTION getuseridbyemail SET SCHEMA regularAccess;  call regularAccess.addNewUser('Fedosdekudrille', 'fedosdekudrille@gmail.com', '1234');  CREATE ROLE regularUser;  GRANT USAGE ON SCHEMA regularAccess TO regularUser;  GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA regularAccess TO regularUser;  GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA regularAccess TO regularUser;  GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO regularUser;  GRANT USAGE, SELECT ON SEQUENCE users\_id\_seq TO regularUser;  CREATE USER serverUser WITH PASSWORD 'qwerty123456';  GRANT regularUser TO serverUser; |

Листинг 2 – Скрипты создание ролей и пользователей

|  |
| --- |
| create or replace function userExists(seekEmail varchar(50), seekPassword varchar(50))  returns boolean  language plpgsql  as  $$  DECLARE cursorEmail varchar(50); cursorPassword varchar(50);  DECLARE checkCursor CURSOR FOR SELECT email, password FROM users FOR READ ONLY;  begin  OPEN checkCursor;  LOOP  FETCH checkCursor INTO cursorEmail, cursorPassword;  IF cursorEmail = seekEmail AND cursorPassword = seekPassword THEN  CLOSE checkCursor;  return true;  END IF;  EXIT WHEN NOT FOUND;  END LOOP;  CLOSE checkCursor;  return false;  end;  $$;  create or replace function getUserIdByEmail(search\_email varchar(50))  returns int  language plpgsql  as  $$  begin  return (SELECT id  FROM users  WHERE email = search\_email);  end;  $$;  CREATE OR REPLACE FUNCTION generate\_random\_string(min\_length int, max\_length int, bool isQuery)  RETURNS TEXT AS  $$  DECLARE  string\_length INT;  random\_string TEXT = '';  i INT;  random\_int INT;  random\_char CHAR;  BEGIN  string\_length := floor(random() \* (max\_length - min\_length)) + min\_length; -- Generate random length between 50 and 500    FOR i IN 1..string\_length LOOP  random\_int := floor(random() \* (90 - 65)) + 65;  random\_char := chr(random\_int); -- Generate random character between ASCII 32 and 126  IF i = 1 OR random() < 0.8 THEN  random\_string := random\_string || random\_char;  ELSE  random\_string := random\_string || ' ' || random\_char;  END IF;  END LOOP;    RETURN random\_string;  END;  $$  LANGUAGE plpgsql;  CREATE OR REPLACE FUNCTION string\_to\_query\_format(str text)  returns text  AS  $$  BEGIN  RETURN regexp\_replace(trim(str), ' {1,}', ' & ', 'g');  END;  $$  LANGUAGE plpgsql; |

Листинг 3 – Скрипты создание функций

|  |
| --- |
| /\*--------------------------———————------------------------------  ----------------------------| VIEW |-----------------------------  ----------------------------———————----------------------------\*/  -------------------- user\_wallet\_info --------------------  CREATE VIEW user\_wallet\_info AS  SELECT Users.username, Users.password, Wallets.address, Wallets.balance  FROM Users  JOIN Wallets ON Users.user\_id = Wallets.user\_id;  SELECT \* FROM user\_wallet\_info;  -------------------- wallet\_currency\_balance --------------------  CREATE VIEW wallet\_currency\_balance AS  SELECT Wallets.address, Crypto\_Currencies.currency\_name, Wallets.balance  FROM Wallets  JOIN Crypto\_Currencies ON Wallets.currency\_id = Crypto\_Currencies.currency\_id;  SELECT \* FROM wallet\_currency\_balance; |

Листинг 5 – Скрипты создание представлений

|  |
| --- |
| CREATE OR REPLACE FUNCTION update\_textsearchable\_index\_col()  RETURNS TRIGGER AS $$  BEGIN  NEW.textsearchable\_index\_col :=  setweight(to\_tsvector('english', coalesce(NEW.name,'')), 'A') ||  setweight(to\_tsvector('english', coalesce('D&D FAQ','')), 'B') ||  setweight(to\_tsvector('english', coalesce(NEW.content,'')), 'D');  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE OR REPLACE TRIGGER trigger\_update\_textsearchable\_index\_col  BEFORE INSERT OR UPDATE ON article  FOR EACH ROW  EXECUTE FUNCTION update\_textsearchable\_index\_col(); |

Листинг 6 – Скрипт создания триггера

|  |
| --- |
| CREATE INDEX textsearch\_idx ON article USING GIN (textsearchable\_index\_col); |

Листинг 7– Скрипт индекса

|  |
| --- |
| CREATE OR REPLACE PROCEDURE fill\_articles()  LANGUAGE plpgsql  AS $$  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO ARTICLE (Theme\_id, user\_id, creation\_date, content, picture, name)  values(1, 11, CURRENT\_TIMESTAMP, generate\_random\_string(50, 500), 'picture', generate\_random\_string(5, 20));  END LOOP;  END;  $$; |

Листинг 8– Скрипты заполнения таблиц