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](#_Toc3105)

[1 Анализ требований к программному средству 5](#_Toc19126)

[1.1 Аналитический обзор аналогов 5](#_Toc26855)

[1.1.1 Аналог DnD.su 5](#_Toc19831)

[1.1.2 Аналог DnD.wizards.com 7](#_Toc15242)

[1.1.2 Аналог Paizo 9](#_Toc21115)

[1.2 Разработка функциональных требований, определение вариантов использования 11](#_Toc3292)

[1.3 Вывод 13](#_Toc14346)

[2 Разработка архитектуры проекта 14](#_Toc24512)

[2.1 Обобщенная структура управлением приложения 14](#_Toc1104)

[Для обеспечения комфортного и удобного пользования справочником по вселенной D&D через приложение необходимо предусмотреть функции быстрого и интуитивно понятного использования платформы. Также пользователи должны иметь возможность создавать статьи и сообщения на форуме, а также. 14](#_Toc15671)

[2.2 Диаграмма базы данных, взаимосвязь всех компонентов. 14](#_Toc29099)

[3 Разработка модели базы данных 22](#_Toc32263)

[3.1 Создание необходимых объектов 22](#_Toc17979)

[3.1.1 Представления базы данных 23](#_Toc9845)

[3.1.2 Индексы базы данных 23](#_Toc6452)

[3.1.3 Триггеры базы данных 24](#_Toc3298)

[3.2 Описание используемой технологии 25](#_Toc31618)

[3.3 Вывод 26](#_Toc2645)

[4 Установка, настройка и использование PosgtreSQL 14.5 27](#_Toc26297)

[4.1 Установка PostgreSQL 27](#_Toc11306)

[4.2 Создание таблиц 27](#_Toc11373)

[4.3 Создание ролей для разграничения 28](#_Toc22668)

[4.4 Создание пакетов процедур для базы данных 29](#_Toc12648)

[4.4.1 Выборка данных из таблиц 30](#_Toc9054)

[4.4.2 Заполнение таблиц 100 000 строк 30](#_Toc23373)

[4.4.3 Добавление данных в таблицы 31](#_Toc16314)

[4.4.4 Удаление данных в таблицы 32](#_Toc3251)

[4.4.5 Изменение данных в таблицы 32](#_Toc6714)

[4.4.6 Дополнительные функции 33](#_Toc10941)

[4.6 Вывод 34](#_Toc19101)

[5 Тестирование 35](#_Toc16562)

[5.1 Тестирование производительности базы данных 35](#_Toc16267)

[5.2 Вывод 35](#_Toc14632)

[6 Руководство по использованию программного средства 36](#_Toc23940)

[6.1 Руководство пользователя 36](#_Toc8642)

[6.2 Установка приложения 39](#_Toc21057)

[6.3 Вывод 40](#_Toc21900)

[Заключение 41](#_Toc10247)

[Список литературных источников 42](#_Toc2931)

[Приложение А 43](#_Toc16582)

# Введение

Цель данной работы заключается в создании базы данных для игровой вселенной Dungeons and Dragons с применением технологии полнострочного поиска. База данных будет хранить информацию о зарегистрированных пользователях, статьях игровой энциклопедии, а также сообщениях на форуме. 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 энциклопедии и её пользователей.

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, js, css и php. Для обеспечения безопасности пользователей приложения был использован метод шифрования паролей перед их сохранением в базе данных.

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

Игровые вселенные - воображаемые миры, которые строят игроки. Они получили распространение ещё в прошлом веке и остаются популярными до сих пор. Игровые вселенные представляют собой продукт плодотворной работы как её авторов, так и сообщества, и представляют собой сложную информационную структуру.

В настоящее время различные вселенные содержат множество фанатов по всему миру. Ресурсы, посвящённые им, должны содержать информацию об этой вселенной, а также предоставлять возможность контакта поклонников. Такие ресурсы являются эффективным способом взаимодействия фанатской базы, и поэтому продолжают оставаться востребованными.

### 1.1.1 Аналог DnD.su

Сайт адаптивен под разные размеры экранов и разные устройства.

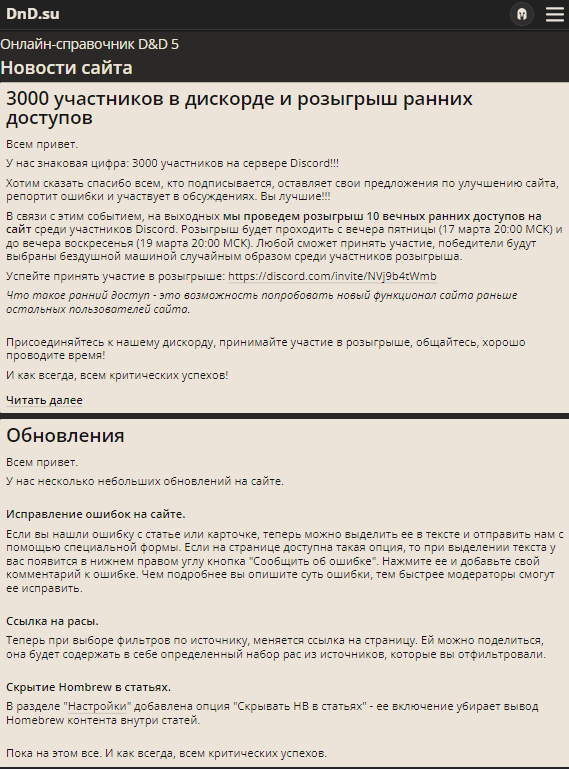


Рис. 1.1 – главная страница web-сайта при просмотре с планшета

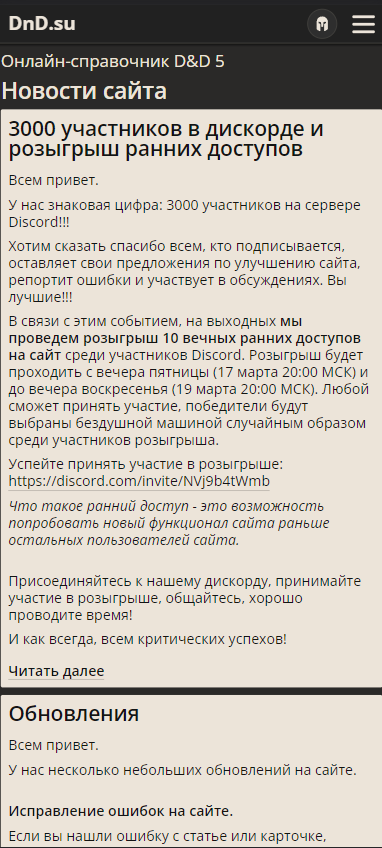


Рис. 1.2 – главная страница web-сайта при просмотре с мобильного устройства

Преимущества сайта https://dnd.su/:

1. Наличие огромной коллекции статей: на сайте представлены разнообразные тематические статьи в большом количестве.

2. Быстродействие: сайт работает быстро и плавно на любых устройствах.

3. Новостная информация: сайт предоставляет много новостной и справочной информации.

4. Надежность: сайт работает уже несколько лет, также растет количество пользователей.

Недостатки сайта https://dnd.su/:

1. Внешний вид: сайт выглядит устаревшим, а многие элементы расположены не интуитивно.

2. Отсутствие коммуникации между пользователями: ресурс не предоставляет возможность пользователям общаться, комментировать или обсуждать что-то на площадке.

3. Отсутствие взаимодействия пользователей с ресурсом: пользователи не могут добавлять, изменять статьи или как-то иначе учавствовать в развитии ресурса.

### 1.1.2 Аналог DnD.wizards.com

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

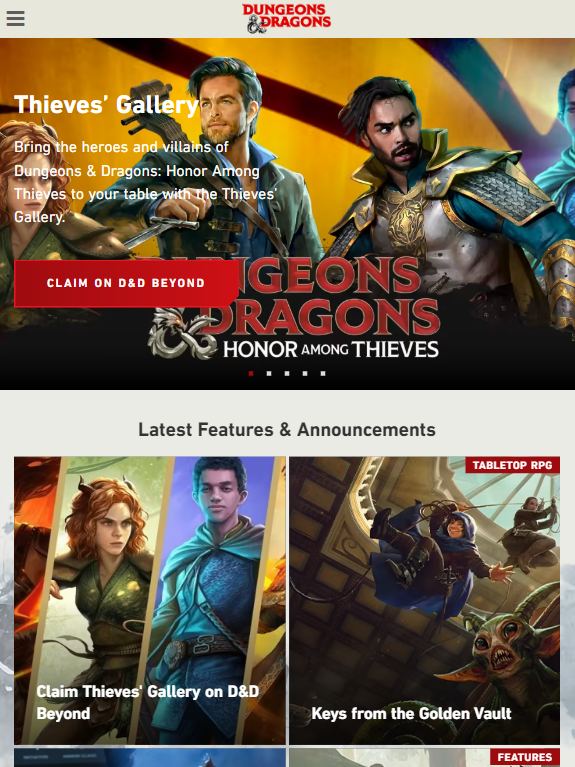


Рис. 1.3 – главная страница web-сайта при просмотре с планшета

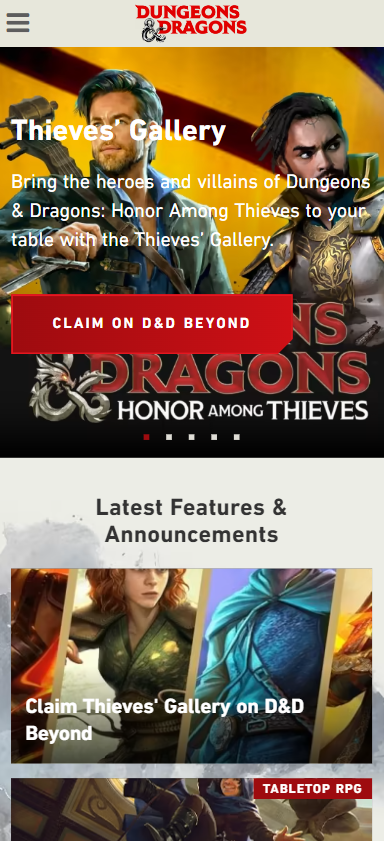


Рис. 1.4 – главная страница web-сайта при просмотре с с мобильного устройства

Преимущества сайта https://dnd.wizards.com/:

1. Удобный интерфейс: сайт имеет удобный и интуитивно понятный интерфейс, который позволяет легко найти нужную статью или новость.

2. Официальный ресурс: сайт создан и поддерживается компанией-разработчком D&D, поэтому на данном сайте можно найти самые новые статьи и товары.

3. Наличие магазина: dnd.wizards.com содержит раздел с магазином, где заинтересованный пользователь может приобрести тематические товары.

4. Плавность работы: сервис работает быстро на различных устройствах и содержит много анимаций.

Недостатки сайта https://dnd.wizards.com/:

1. Отсутствие взаимодействия с сообществом: сервис не предлагает какого-либо вовлечения вользователя в работу сервиса.

2. Малое количество информации: несмотря на наличее новых статей и событий, их общее количество куда меньше чем у других аналогичных ресурсов.

3. Нестабильность: некоторые разделы и элементы интерфейса работают некорректно или не функционируют вообще.

### 1.1.2 Аналог Paizo

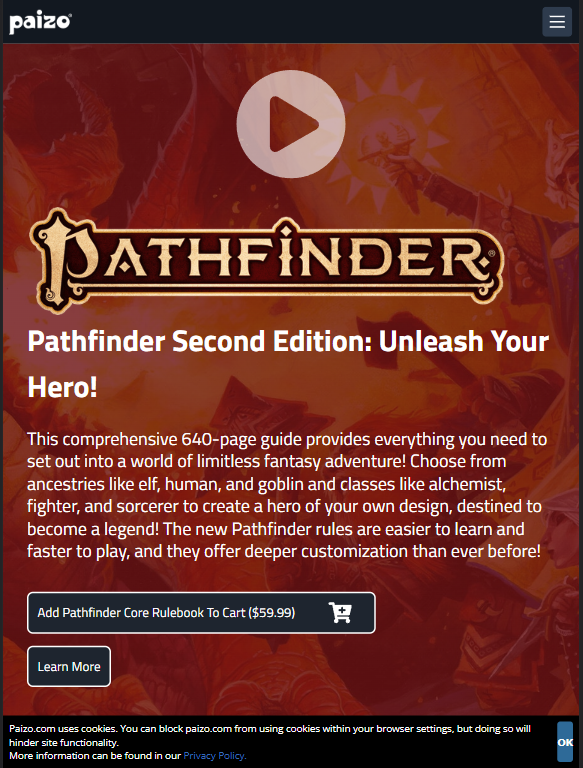


Рис. 1.5 – главная страница web-сайта при просмотре с планшета



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

Преимущества сайта https://paizo.com/pathfinder:

1. Простота использования: сервис имеет простой, понятный интерфейс, что способствует удобному и быстрому поиску информации на сайте.
2. Множество статей: сервис предоставляет множество разнообразных статей и справочной информации.

3. Наличие вспомогательных разделов: paizo.com предоставляет пользователю много разнообразных разделов на сайте, как основных, так и второстепенных, где можно приобрести товары, прочитать новости и тп.

Недостатки сайта https://paizo.com/pathfinder:

1. Плохая оптимизация: сервис работает быстро на стационарных компьютерах, но имеет проблемы с производительностью на мобильных устройствах.

2. Нагруженный интерфейс: новому пользователю будет сложно разобраться в работе сервиса.

3. Реклама: при использовании сервиса в глаза бросается наличие огромного количества рекламы, что мешает пользованию сайтом.

## 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 статьях энциклопедии и пользователях, поиска статей, общения на форуме, а также функции для оценки контента.

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.

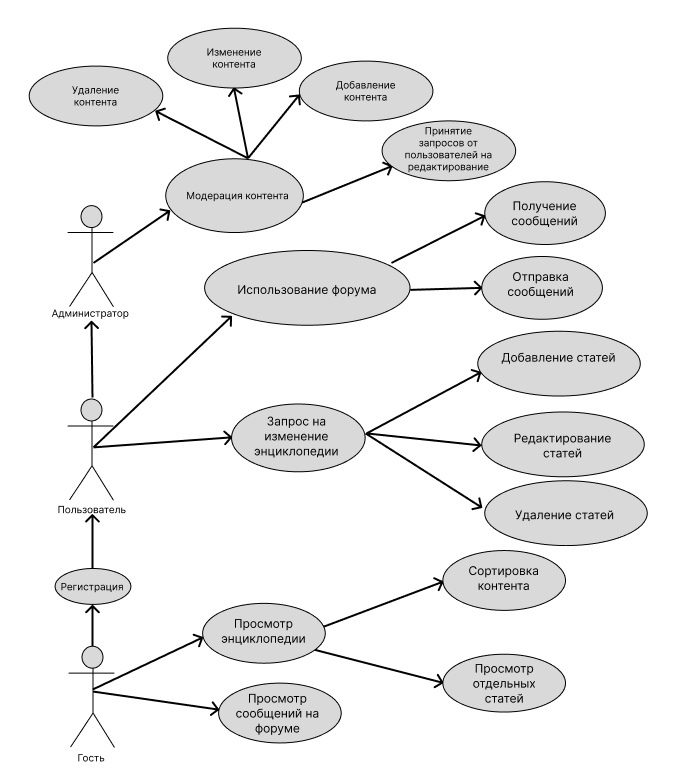


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности изменять её. После регистрации клиент получает роль Пользователь.

Роль Пользователь получает доступ к написанию сообщений на форуме, а также способен запросить возможность изменить контент энциклопедии.

Роль Администратор заключается в непосредственном отслеживании контента и управлении им. Он может вносить правки в статьи, добавлять и удалять их, а также добавлять запросы на подобные действия от пользователей.

## 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 справочником по вселенной D&D через приложение необходимо предусмотреть функции быстрого и интуитивно понятного использования платформы. Также пользователи должны иметь возможность создавать статьи и сообщения на форуме, а также.

## 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.

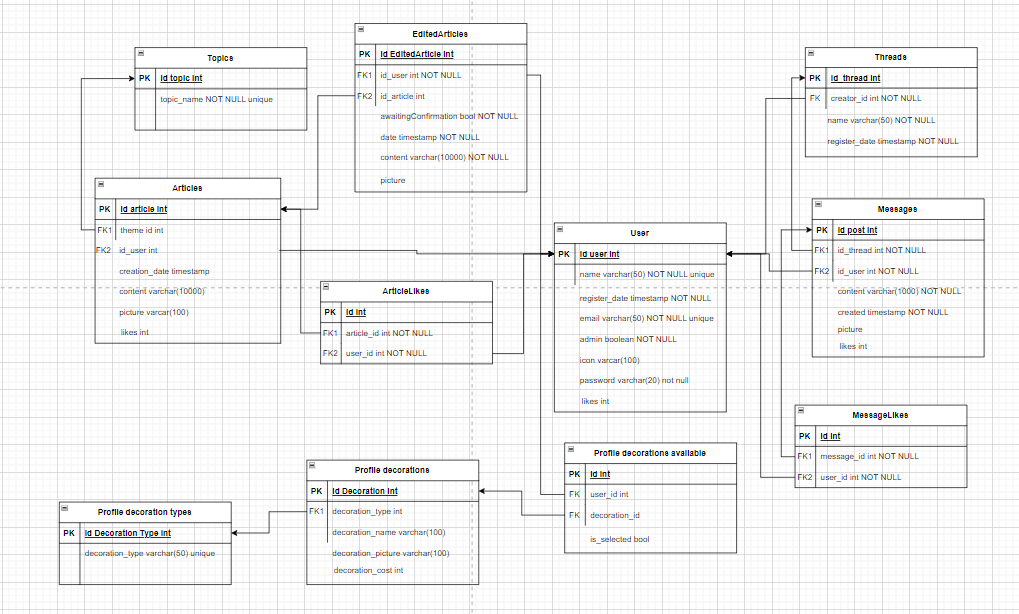


Рисунок 2.1 – Диаграмма базы данных

Таблица "users" содержит поля Id (тип: serial, первичный ключ), Name (тип: varchar(50), не пустое значение), Register\_date (тип: timestamp, значение по умолчанию: текущая дата и время), Email (тип: varchar(50), уникальное значение, не пустое значение), Password (тип: varchar(50), не пустое значение, длина больше 3 символов), Admin (тип: boolean, значение по умолчанию: false), Icon (тип: varchar(100), может быть пустым)

Таблица "p\_decoration\_types" содержит поля Id (тип: serial, первичный ключ), Type (тип: varchar(50), уникальное значение, не пустое значение)

Таблица "profile\_decoration" содержит поля Id (тип: serial, первичный ключ), Type (тип: varchar(50), не пустое значение, связано с полем "Type" таблицы "p\_decoration\_types"), Name (тип: varchar(50), не пустое значение), Picture (тип: varchar(50), не пустое значение), Cost (тип: decimal, не пустое значение)

Таблица "user\_decorations" содержит поля id (тип: serial, первичный ключ), decoration\_id (тип: int, не пустое значение, связано с полем "Id" таблицы "profile\_decoration"), user\_id (тип: int, не пустое значение, связано с полем "Id" таблицы "users"), is\_selected (тип: bool, значение по умолчанию: false)

Таблица "threads" содержит следующие поля: Id: Уникальный идентификатор записи в таблице (тип: serial), Creator\_id: Идентификатор создателя (тип: int), связан с полем "Id" таблицы "users", Creation\_date: Дата и время создания (тип: timestamp), устанавливается на текущую дату и время по умолчанию, Name: Название темы (тип: varchar(50)).

Таблица "messages" содержит следующие поля: Id: Уникальный идентификатор записи в таблице (тип: serial), Thread\_id: Идентификатор треда, к которому принадлежит сообщение (тип: bigint), связан с полем "Id" таблицы "threads", User\_id: Идентификатор пользователя, отправившего сообщение (тип: bigint), связан с полем "Id" таблицы "users", Content: Содержание сообщения (тип: varchar(50)), Picture: Имя файла изображения, прикрепленного к сообщению (тип: varchar(100)), Creation\_date: Дата и время создания сообщения (тип: timestamp).

Таблица "messages\_likes" содержит следующие поля: Id: Уникальный идентификатор записи в таблице (тип: serial), Message\_id: Идентификатор сообщения, которое было лайкнуто (тип: bigint), связан с полем "Id" таблицы "messages", User\_id: Идентификатор пользователя, который поставил лайк (тип: bigint), связан с полем "Id" таблицы "users".

Таблица "topics" содержит поля Id: Уникальный идентификатор записи в таблице (тип: serial), Name: Название темы (тип: varchar(100)).

Таблица "article" содержит поля Id: Уникальный идентификатор записи в таблице (тип: serial), Theme: Тема статьи (тип: varchar(100)), связан с полем "Name" таблицы "topics", User\_id: Идентификатор пользователя, создавшего статью (тип: bigint), связан с полем "Id" таблицы "users", Content: Содержание статьи (тип: varchar(100)), Picture: Имя файла изображения, прикрепленного к статье (тип: varchar(100)), Creation\_date: Дата и время создания статьи (тип: timestamp), is\_confirmed: Флаг подтверждения статьи (тип: boolean).

Таблица "article\_likes" содержит следующие поля: Id: Уникальный идентификатор записи в таблице (тип: serial), Article\_id: Идентификатор статьи, которая была лайкнута (тип: bigint), связан с полем "Id" таблицы "article", User\_id: Идентификатор пользователя, который поставил лайк (тип: bigint), связан с полем "Id" таблицы "users".

Таблица "edited\_articles" содержит следующие поля: Id: Уникальный идентификатор записи в таблице (тип: serial), User\_id: Идентификатор пользователя, отредактировавшего статью (тип: bigint), связан с полем "Id" таблицы "users", Article\_id: Идентификатор отредактированной статьи (тип: bigint), связан с полем "Id" таблицы "article", Creation\_date: Дата и время создания записи об отредактированной статье (тип: timestamp), Picture: Имя файла изображения, связанного с отредактированной статьей (тип: varchar(100)).

**2.3 Описание информационных объектов**

Для реализации базы данных было разработано 5 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Users, Crypto\_Currencies Exchange\_Rates, Wallets, Transactions. Ниже будет описание про каждую из них более подробно.

Таблица "Users" - хранит информацию о пользователях. Содержит следующие поля:

id (serial primary key): Уникальный идентификатор записи в таблице "user\_decorations". Значение этого поля генерируется автоматически и увеличивается с каждой новой записью.

decoration\_id (int not null): Внешний ключ, связанный с полем "Id" таблицы "profile\_decoration". Он указывает на идентификатор декорации, присвоенный данной записи в таблице.

user\_id (int not null): Внешний ключ, связанный с полем "Id" таблицы "users". Он указывает на идентификатор пользователя, связанного с данной записью в таблице.

is\_selected (bool not null default false): Логическое поле, указывающее, является ли данная декорация выбранной пользователем. По умолчанию, значение этого поля равно "false".

Таблица "profile\_decoration" - хранит информацию о доступной кастомизации профиля:

Id (serial primary key): Уникальный идентификатор записи в таблице "profile\_decoration". Значение этого поля генерируется автоматически и увеличивается с каждой новой записью.

Type (varchar(50) not null): Строковое поле, указывающее тип декорации. Это поле связано с полем "type" таблицы "p\_decoration\_types" посредством внешнего ключа. Оно указывает на тип декорации, к которой относится данная запись.

Name (varchar(50) check(Name !='') not null): Строковое поле, содержащее название декорации. Значение этого поля не может быть пустым, поскольку установлено ограничение CHECK, которое проверяет, что значение поля "Name" не равно пустой строке.

Picture (varchar(50) check(Picture !='') not null): Строковое поле, содержащее имя файла изображения декорации. Значение этого поля не может быть пустым, поскольку установлено ограничение CHECK, которое проверяет, что значение поля "Picture" не равно пустой строке.

Cost (decimal not null): Десятичное поле, указывающее стоимость декорации. Значение этого поля не может быть пустым (null), так как установлено ограничение NOT NULL.

Таблица "p\_decoration\_types" - хранит информацию о типах декораций. Содержит следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор типа декорации.

Type: символьная строка (varchar) с максимальной длиной 50 символов - тип декорации.

Таблица "user\_decorations" - хранит информацию о приобретённых пользователем декорациях. Содержит следующие поля:

id: серийный ключ (serial) - уникальный идентификатор декорации пользователя.

decoration\_id: целочисленное значение (int) - идентификатор декорации профиля.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице profile\_decoration (CONSTRAINT fk\_u\_dec\_prof\_dec references profile\_decoration(Id)).

user\_id: целочисленное значение (int) - идентификатор пользователя.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_users\_u\_dec references users(Id)).

is\_selected: логическое значение (boolean) - указывает, выбрана ли декорация пользователем.

Значение по умолчанию: false.

Ограничения: не может быть пустым (not null).

Таблица threads содержит информацию о всех разделах общения. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор темы обсуждения.

Creator\_id: целочисленное значение (int) - идентификатор создателя темы.

Ограничения: не может быть равным 0 (check(Creator\_id != 0)), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_threads\_creator references users(Id)).

Creation\_date: временная метка (timestamp) - дата и время создания темы.

Значение по умолчанию: текущая дата и время (default CURRENT\_TIMESTAMP).

Ограничения: не может быть пустым (not null).

Name: символьная строка (varchar) с максимальной длиной 50 символов - название темы обсуждения.

Ограничения: не может быть пустым (not null), должно отличаться от пустой строки (check(Name != '')).

Таблица messages содержит информацию о всех сообщениях, отправленных пользователями. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор темы обсуждения.

Creator\_id: целочисленное значение (int) - идентификатор создателя темы.

Ограничения: не может быть равным 0 (check(Creator\_id != 0)), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_threads\_creator references users(Id)).

Creation\_date: временная метка (timestamp) - дата и время создания темы.

Значение по умолчанию: текущая дата и время (default CURRENT\_TIMESTAMP).

Ограничения: не может быть пустым (not null).

Name: символьная строка (varchar) с максимальной длиной 50 символов - название темы обсуждения.

Ограничения: не может быть пустым (not null), должно отличаться от пустой строки (check(Name != '')).

Id: серийный ключ (serial) - уникальный идентификатор сообщения.

Thread\_id: целочисленное значение (bigint) - идентификатор темы, к которой относится сообщение.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице threads (CONSTRAINT fk\_message\_thread references threads(Id)).

User\_id: целочисленное значение (bigint) - идентификатор пользователя, отправившего сообщение.

Ограничения: не может быть пустым (not null), не может быть равным 0 (check(User\_id != 0)), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_message\_user references users(Id)).

Content: символьная строка (varchar) с максимальной длиной 50 символов - содержание сообщения.

Ограничения: не может быть пустым (not null), должно отличаться от пустой строки (check(Content != '')).

Picture: символьная строка (varchar) с максимальной длиной 100 символов - путь к изображению, прикрепленному к сообщению.

Ограничения: не может быть пустым (not null), должно отличаться от пустой строки (check(Picture != '')).

Creation\_date: временная метка (timestamp) - дата и время создания сообщения.

Ограничения: не может быть пустым (not null).

Значение по умолчанию: текущая дата и время (default CURRENT\_TIMESTAMP).

Таблица messages\_likes содержит информацию о всех транзакциях, которые были совершены между кошельками пользователей. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор для таблицы "messages\_likes".

Message\_id: целочисленное значение (bigint) - идентификатор сообщения, которому присвоен лайк.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице messages (CONSTRAINT fk\_messages\_likes\_messge references messages(Id)).

User\_id: целочисленное значение (bigint) - идентификатор пользователя, который поставил лайк.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_messages\_likes\_user references users(Id)).

Таблица topics содержит информацию о всех транзакциях, которые были совершены между кошельками пользователей. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор темы.

Name: символьная строка (varchar) с максимальной длиной 100 символов - название темы.

Ограничения: должно быть уникальным (unique), не может быть пустым (not null), должно отличаться от пустой строки (check(Name != '')).

Таблица articles содержит информацию о всех транзакциях, которые были совершены между кошельками пользователей. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор статьи.

Theme: символьная строка (varchar) с максимальной длиной 100 символов - тема статьи.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице topics (CONSTRAINT fk\_article\_theme references topics(Name)).

User\_id: целочисленное значение (bigint) - идентификатор пользователя, создавшего статью.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_article\_user references users(Id)).

Content: символьная строка (varchar) с максимальной длиной 100 символов - содержание статьи.

Ограничения: не может быть пустым (not null), должно отличаться от пустой строки (check(Content != '')).

Picture: символьная строка (varchar) с максимальной длиной 100 символов - путь к изображению, прикрепленному к статье.

Ограничения: может быть пустым, но если указан, должен отличаться от пустой строки (check(Picture != '')).

Creation\_date: временная метка (timestamp) - дата и время создания статьи.

Ограничения: не может быть пустым (not null).

Значение по умолчанию: текущая дата и время (default CURRENT\_TIMESTAMP).

is\_confirmed: логическое значение (boolean) - указывает, подтверждена ли статья администратором.

Значение по умолчанию: false.

Ограничения: не может быть пустым (not null).

Таблица article\_likes содержит информацию о всех транзакциях, которые были совершены между кошельками пользователей. Она включает следующие поля:

Id: серийный ключ (serial) - уникальный идентификатор для таблицы "article\_likes".

Article\_id: целочисленное значение (bigint) - идентификатор статьи, которой присвоен лайк.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице article (CONSTRAINT fk\_article\_likes\_article references article(Id)).

User\_id: целочисленное значение (bigint) - идентификатор пользователя, который поставил лайк.

Ограничения: не может быть пустым (not null), должно соответствовать существующему значению в таблице users (CONSTRAINT fk\_article\_likes\_user references users(Id)).

**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.

В данной базе данных представлены таблицы, которые позволяют хранить информацию о пользователях, криптовалютах, курсах обмена, кошельках и транзакциях. Таблица Users содержит информацию о пользователях, такую как их имя пользователя, адрес электронной почты и пароль. Таблица Crypto\_Currencies содержит информацию о криптовалютах, включая название, описание и символ.

Таблица Exchange\_Rates содержит информацию о курсах обмена, включая идентификатор криптовалюты, курс и время. Таблица Wallets содержит информацию о кошельках, включая идентификатор пользователя, адрес кошелька, баланс и идентификатор криптовалюты. Таблица Transactions содержит информацию о транзакциях, включая идентификатор отправителя, идентификатор получателя, сумму и идентификатор криптовалюты.

Эта база данных может быть использована для создания сервиса обмена криптовалютами, который позволит пользователям создавать кошельки, просматривать курсы обмена и проводить транзакции между своими кошельками и кошельками других пользователей. Также эта база данных может использоваться для анализа криптовалютных транзакций, выявления взаимосвязей между ними и определения тенденций на рынке.

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), пароль (Password), статус администратора (Admin) и путь к иконке профиля (Icon).

Таблица "p\_decoration\_types":

Содержит список доступных типов декораций профилей пользователей.

Одно поле таблицы - идентификатор типа декорации (Id) и название типа (Type).

Таблица "profile\_decoration":

Хранит информацию о доступных декорациях для профилей пользователей.

Поля таблицы включают идентификатор декорации (Id), тип декорации (Type), название декорации (Name), путь к изображению декорации (Picture) и стоимость декорации (Cost).

Таблица "user\_decorations":

Связующая таблица, которая отображает связь между пользователями и выбранными ими декорациями для профилей.

Поля таблицы включают идентификатор (id), идентификатор декорации профиля (decoration\_id), идентификатор пользователя (user\_id) и флаг, указывающий, выбрана ли декорация (is\_selected).

Таблица "threads":

Содержит информацию о созданных темах обсуждений.

Поля таблицы включают идентификатор темы (Id), идентификатор создателя темы (Creator\_id), дату и время создания темы (Creation\_date) и название темы (Name).

Таблица "messages":

Хранит сообщения, связанные с определенной темой обсуждения.

Поля таблицы включают идентификатор сообщения (Id), идентификатор темы (Thread\_id), идентификатор пользователя (User\_id), содержание сообщения (Content), путь к изображению сообщения (Picture) и дату и время создания сообщения (Creation\_date).

Таблица "messages\_likes":

Связующая таблица, которая отображает связь между пользователями и сообщениями, которым они поставили лайк.

Поля таблицы включают идентификатор (Id), идентификатор сообщения (Message\_id) и идентификатор пользователя (User\_id).

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных.

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

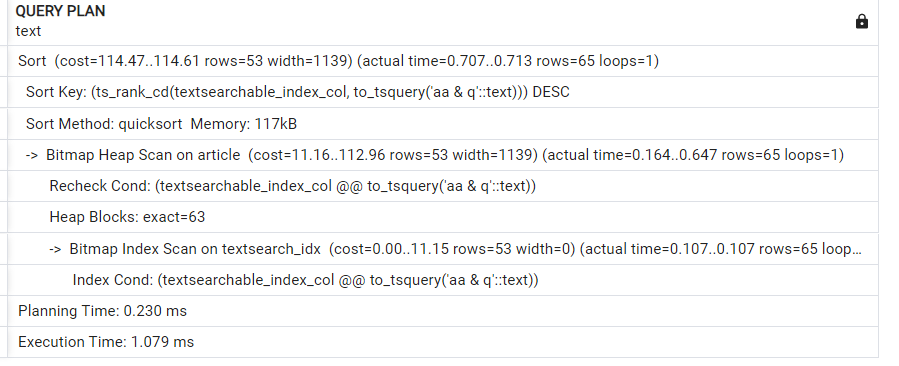


Рисунок 5.2 – Результат выполнения запроса

## 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.

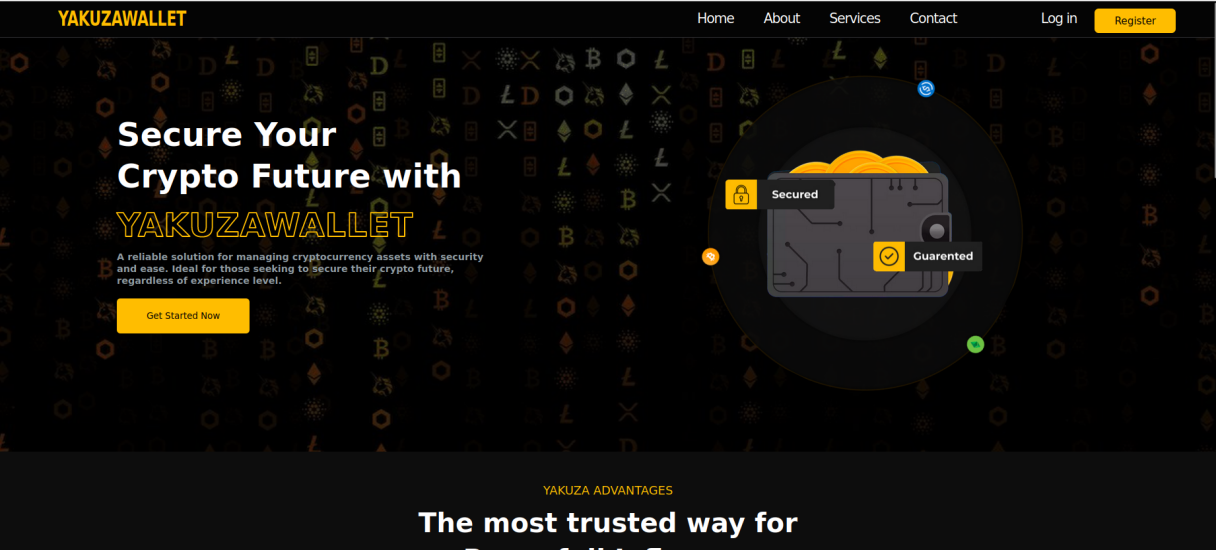


Рисунок 6.1 – Главная страница YAKUZAWALLET

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.

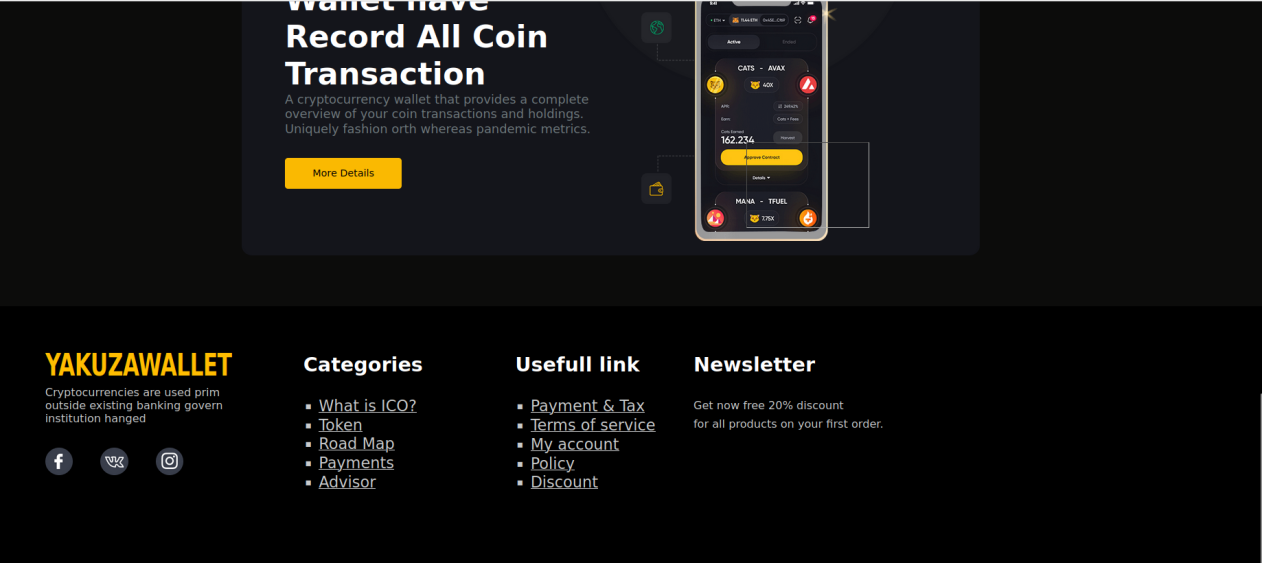


Рисунок 6.2 – Полезные ссылки

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.

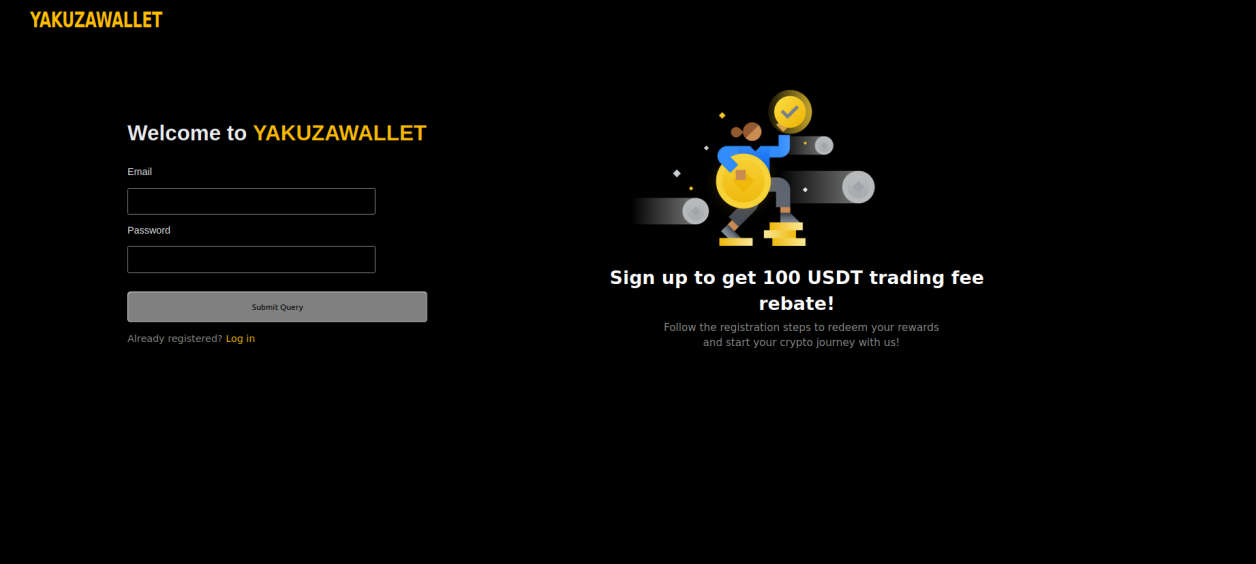


Рисунок 6.3 – Форма регистрации на сайте

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.

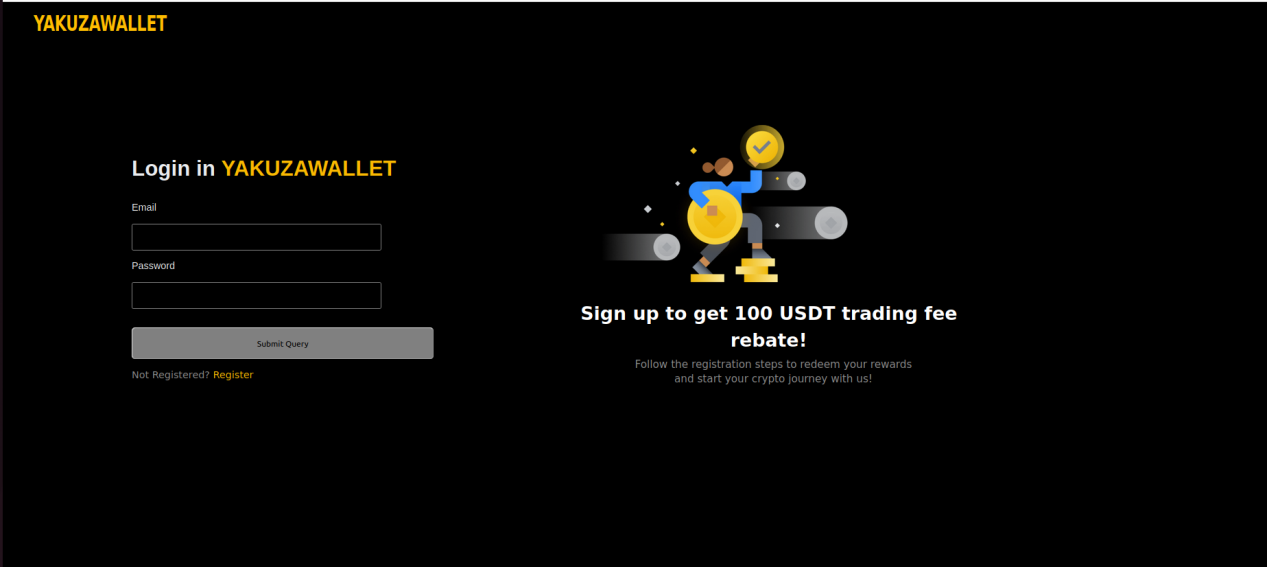


Рисунок 6.4 – Форма входа в кабинет

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.

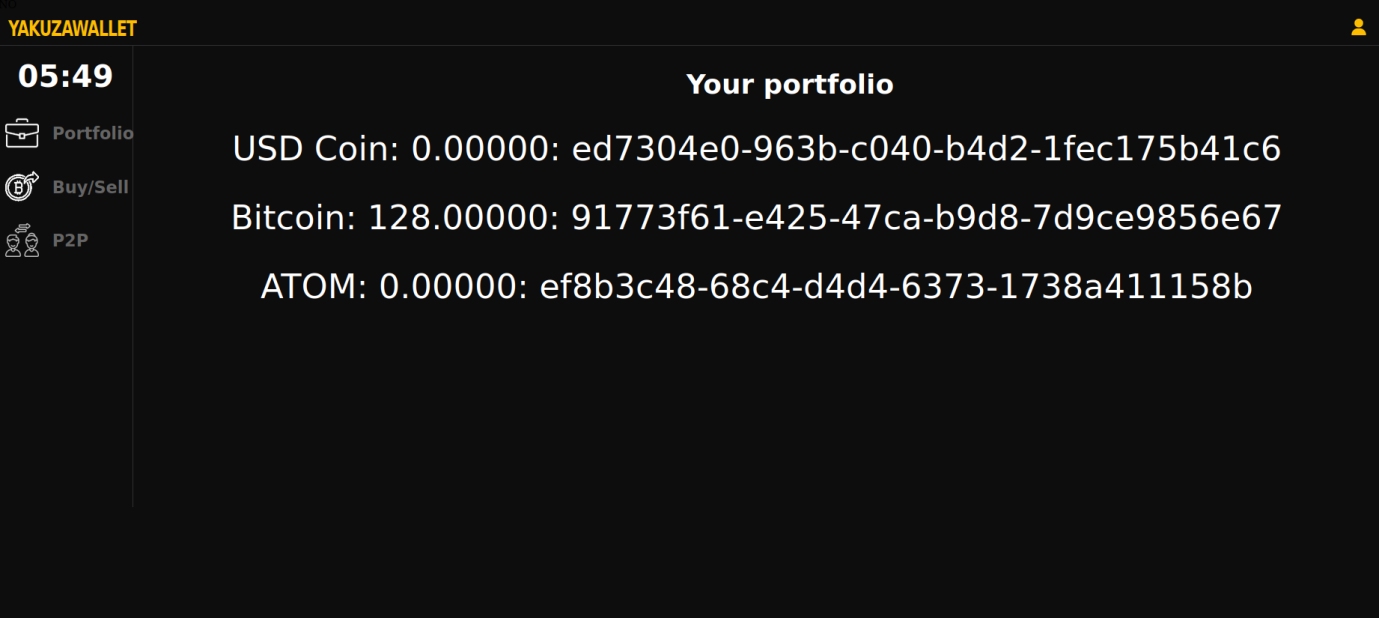


Рисунок 6.5 – Страница портфеля.

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.

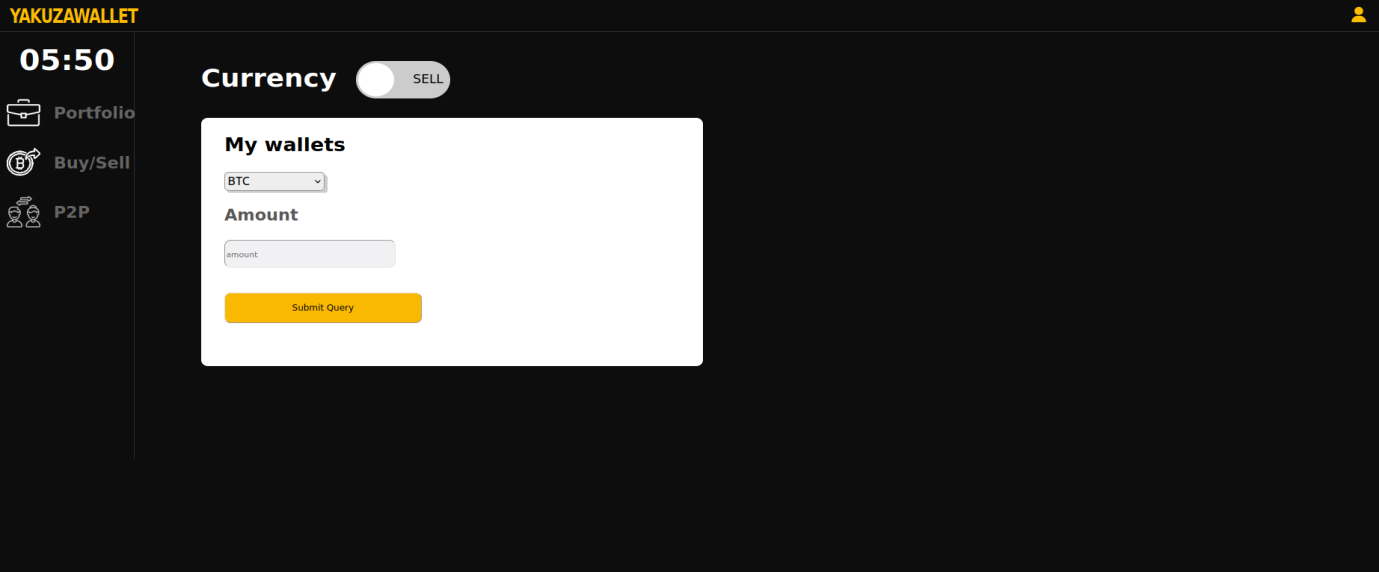


Рисунок 6.6. – Страница покупки/продажи криптовалюты

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.

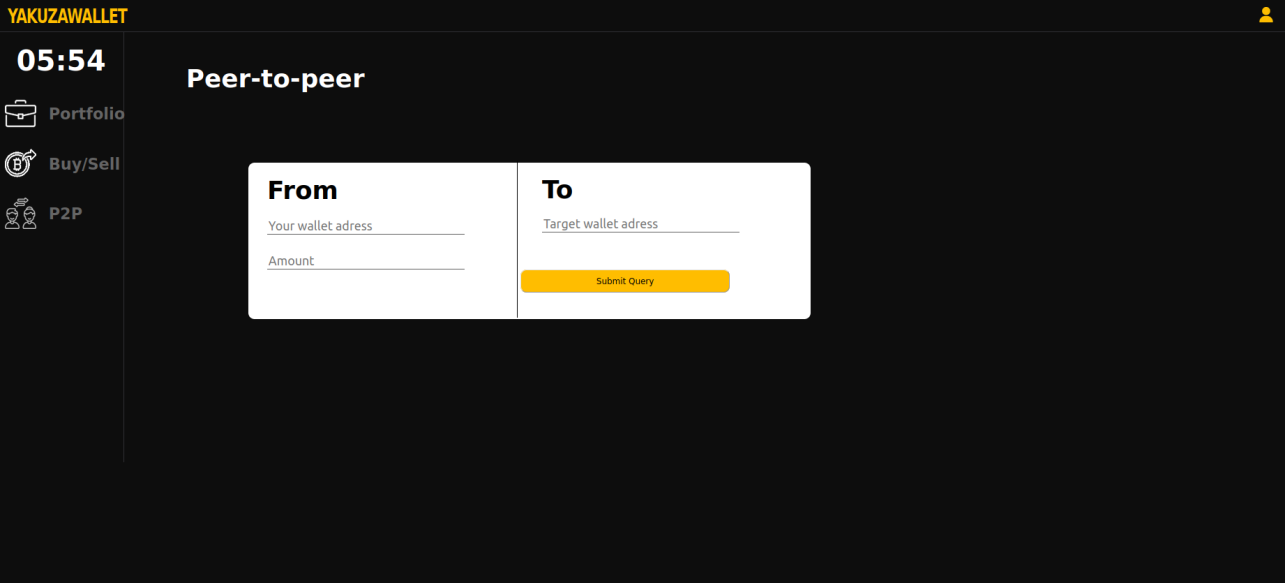


Рисунок 6.7 – Интерфейс P2P

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.

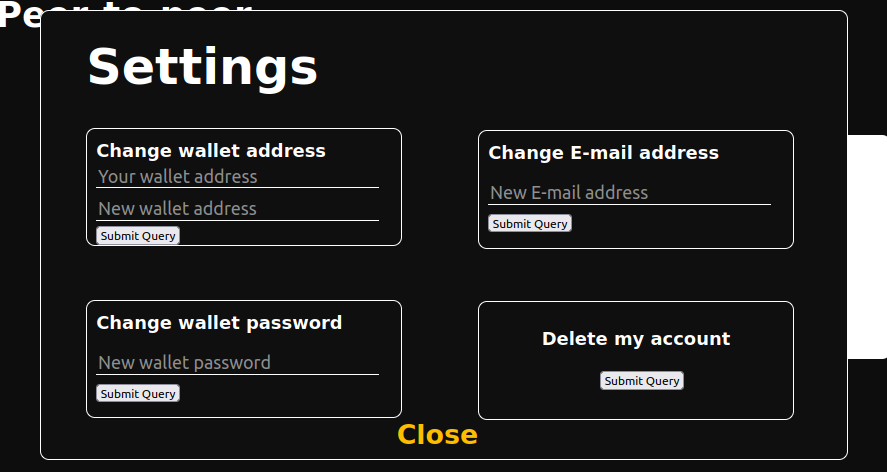


Рисунок 6.8 – Интерфейс настроек

## 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