**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**МБОУ «Школа № 34» им. Героя РФ П. В. Кривова г. Рязани**

**ПРАКТИКО-ОРИЕНТИРОВАННЫЙ ПРОЕКТ ПО ИНФОРМАТИКЕ**

**Тема работы:**

# «BIVREOST - КОШЕЛЕК КРИПТОВАЛЮТЫ, ИНТЕГРИРОВАННЫЙ В TELEGRAM».

**Автор:** Сермягин Кирилл Андреевич.
Ученик МБОУ «Школа № 34» им. Героя РФ П. В. Кривова, 11 «А» класса.
**Научный руководитель:** Учитель информатики и математики Любакова Мария Васильевна

**г. Рязань, 2024**

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

**Актуальность темы.**

Криптовалюты являются одним из самых актуальных и интересных разработок в финансовой индустрии последних лет. Они предлагают новые возможности для инвестирования и управления активами, и мы видим, что все больше и больше людей начинают интересоваться этой темой.

**Проблема проекта.** Европейский союз 03 июня 2022 г. ввел санкционный пакет против России. Как следует из опубликованного документа, европейские власти отключили от SWIFT Сбербанк, МКБ, РСХБ. Одно из больших преимуществ криптовалюты по сравнению с другими платежными средствами – это то, что любой человек вне зависимости от его национальности может отправить другому человеку те же самые деньги, только в криптовалюте, обойдя любую политическую ситуацию, которая сложилась между странами, где они проживают. Никто не выбирает, где родиться, а запрещать отправлять финансовые активы – это неправильно, потому что люди могут быть родственниками. У любого человека может случиться плохой период в жизни и ему может потребоваться финансовая помощь. Эта проблема послужила мотивацией для создания этого проекта. Мой кошелек будет доступен каждому, вне зависимости от национальности, религиозных исповеданий и т.д.

**Продукт проекта**: WEB (TWA)-приложение (SPA).

Однако, управление криптовалютными активами может быть довольно сложным и вызывать много вопросов, особенно для тех, кто новичок в этой области. В этом контексте, я разработал кошелек криптовалюты, интегрированный в Telegram. Мой кошелек предлагает удобный и простой инструмент для управления вашими криптовалютными активами. Он позволяет вам хранить все ваши криптовалюты в одном месте и доступен для использования прямо из приложения Telegram. Он оснащен такими функциями

как: переводы пользователям кошелька, переводы по адресу и отслеживание портфеля. Вы сможете легко и быстро управлять своими активами и принимать информированные решения об инвестировании. Кроме того, я понимаю, что безопасность ваших активов является приоритетом, поэтому сделал все возможное, чтобы обеспечить высочайший уровень безопасности для ваших финансовых активов. Мой кошелек использует надежные и проверенные алгоритмы, чтобы защитить ваши данные и финансы. В целом, кошелек криптовалюты, интегрированный в Telegram, предлагает вам удобный, безопасный и мощный инструмент для управления вашими криптовалютными активами. Я постоянно работаю над улучшением и расширением функциональности кошелька, чтобы обеспечить вам еще более высокий уровень удобства и эффективности. В целом, я верю, что мой кошелек, станет вашим надежным партнером в мире криптовалют и поможет вам достигать ваших финансовых целей.

**Цель:** Разработать безопасный, децентрализованный кошелек криптовалюты, который будет доступен всем желающим, с помощью него можно будет удобно отправлять и получать токены криптовалюты IOTA (MIOTA, SHIMMER).

**Задачи:**

- воссоздать архитектуру приложения, выбрать технологии, с помощью которых будет реализован проект;
- разработать алгоритмы безопасности хранения и отправления данных к пользователю;
- разработать понятный и дружелюбный интерфейс приложения;
- протестировать кошелек на тестовых и live-net сетях, выпустить проект и собрать фидбек от пользователей.

# ГЛАВА 1. РАЗРАБОТКА АРХИТЕКТУРЫ И ВЫБОР ТЕХНОЛОГИЙ

## § 1. Определение и характеристика элементов приложения

В апреле 2022 года Telegram представил технологию TWA – (англ. Telegram Web Application) WEB-приложение telegram. Эта технология позволила расширить функционал ботов в telegram, она поддерживает бесконечно гибкие интерфейсы JavaScript, способные заменить целевые WEB-приложения. С помощью этой технологии и был реализован проект.

**Telegram бот**. Элемент приложения, с которым впервые встречается пользователь моего продукта. Он требуется для представления моего продукта пользователю и получения кнопки, с помощью которой пользователь сможет открыть WEB-приложение **WEB-приложение (Клиент)**. Клиентская часть веб-приложения (также известная как клиент) является важным компонентом системы, поскольку она отвечает за отображение интерфейса пользователя и управление взаимодействием с ним.

На клиенте выполняется рендеринг HTML, CSS и JavaScript кода, что позволяет отображать интерфейс и реагировать на действия пользователя. Он также может взаимодействовать с сервером, запрашивая и получая данные, которые необходимы для отображения информации на странице.

Использование клиента в веб-приложении позволяет реализовать более гибкую и интерактивную архитектуру, которая может улучшить пользовательский опыт. Также это помогает снизить нагрузку на сервер, так как многие действия выполняются на клиенте.

**WEB-приложение (Сервер)**. Сервер веб-приложения является важным компонентом системы, поскольку он отвечает за хранение, обработку и отдачу данных. Сервер хранит все необходимые данные, такие как базы данных, файлы,

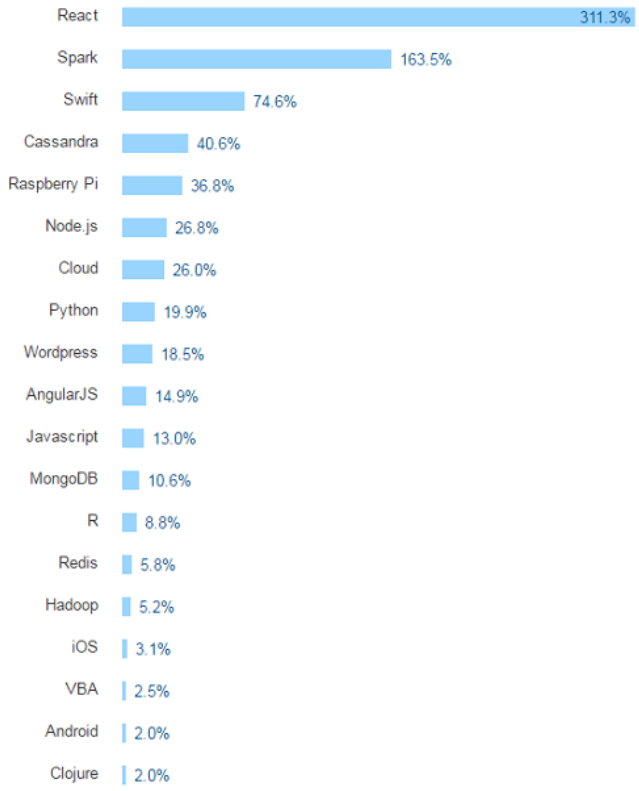
изображения и т.д. Эти данные могут быть запрошены клиентом при необходимости и отданы ему для отображения. Сервер также может выполнять логику бизнес-процессов и обработку данных, такую как валидация, аутентификация и авторизация пользователей, а также сохранение информации в базу данных.

Использование сервера в веб-приложении позволяет реализовать более надежную и безопасную архитектуру, так как все важные данные хранятся и обрабатываются на сервере, а не на клиенте. Также это позволяет реализовать масштабирование приложения, так как сервер может обрабатывать запросы от множества клиентов одновременно. Кроме того, сервер может обеспечивать бекапы и защиту данных, что важно для вашего приложения и его пользователей. В общем, сервер является неотъемлемой частью веб-приложения и обеспечивает масштабируемость, безопасность и функциональность приложения.

## § 2. Выбор технологий

**1. Telegram Бот.** Для создания бота я выбрал Python 3.11 и библиотеку pyTelegramBotAPI (4.9.0), потому что на этом языке программирования и с помощью данной библиотеки можно развернуть бота намного быстрее, чем с использованием других языков и библиотек, бот – это примитивная вещь, которая особо не требует детализации, конкретно в моем случае, именно поэтому был выбран Python.

**2. WEB-приложение (Клиент).** Для его создания я решил использовать язык программирования JavaScript (TypeScript), фреймворк – React (18.2.0). Главный подход React фреймворка – создание переиспользуемых компонентов. Благодаря этому соблюдается один из самых главных принципов




программирования – DRY (англ. Don’t repeat yourself) не повторяй сам себя. Компонентный подход ускоряет разработку приложения из-за того, что компоненты можно переиспользовать. Также React имеет большую аудиторию, что позволяет быстро решать появившиеся проблемы. Virtual DOM позволяет эффективно обновлять и отображать компоненты. Производительность достаточно высока, благодаря оптимизированным обновлениям и рендерингу. React имеет большое преимущество по сравнению с другими javascript-фреймворками, именно поэтому был выбран он.

**3. WEB-приложение (Сервер)** К выбору реализации сервера нужно подойти максимально ответственно и выбрать тот язык, который позволит создать высоконагруженную API. Самое главное, для этого языка должно быть написано API взаимодействия с криптовалютой IOTA. Список поддерживаемых языков IOTA, достаточно невелик это – Python, JavaScript (Node.JS), Rust. Rust – язык, схож с C++ / C, он создавался для системного программирования, хорошо подходит для разработки низкоуровневого ПО. По-моему мнению, данный язык не подходит для разработки WEB-приложений, потому что для написания WEB-приложения потребуется язык более высокого уровня, потому что скорость разработки на Rust будет довольно низка. Node.JS по сравнению с Python превышает его в аспекте производительности. Масштабируемость примерна одинакова в Python и Node.JS по простой причине в python много библиотек, которые позволяют использовать асинхронное программирование для обработки больших миграций данных. Также в Python намного лучше обусловлена система работы с файлами, она довольно проста, по сравнению с fs в JavaScript.

Синтаксические особенности и обработка ошибок – это тот критерий, который стал для меня решающим в выборе python – он более дружественен в этом плане, чем Node.JS, а с обработкой ошибок придется сталкиваться на каждом углу, потому что будет выполняться разработка приложения, которое работает с криптовалютой, что однозначно непросто. Для создания API мой взгляд упал на FastAPI, потому что это один из лучших, быстрых, простых фреймворков для создания REST API.

Итоговый список *основных* технологий, применяемых в проекте:

1. Python 3.11
   1.1. pyTelegramBotAPI 4.9.0
   1.2. FastAPI 0.90.0
   1.3. CoinGeckoAPI
2. JavaScript (TypeScript)
   2.1. React 18.2.0
   2.2. react-router-dom 6.8.1
   2.3. lottie-react 2.3.1
3. SASS (SCSS)
4. Базы данных
   4.1. MySQL 8.0.32
   4.2. PostgreSQL 15.1
5. Выгрузка проекта
   5.1. Docker
   5.2. Git
   5.3. Linux (Ubuntu 22.04)

**React-router-dom.** Библиотека для создания маршрутизации в React-приложении, имеет обширный функционал, простую настройку роутов.

**Lottie-react.** Дает возможность использовать lottie-анимации в react-приложении.

**SASS (SCSS).** Для такого обширного приложения, с достаточно большой верстkой я решил выбрать именно SCSS, потому что написание кода будет

быстрым и будет удобная поддержка уже написанного кода, за счет его преимуществ перед обычным CSS.

**MySQL.** Данная база данных используется для хранения общей информации, в Main-API. Тут не была важна скорость выборки данных из таблиц, поэтому для Main-API было решено использовать данный тип.

**PostgreSQL.** Этот тип базы данных я решил выбрать, потому что на сервисах IOTA-API и SHIMMER-API была важна скорость выборки данных из таблиц. Этот тип оптимальнее и производительнее MySQL в 2,41 раза.

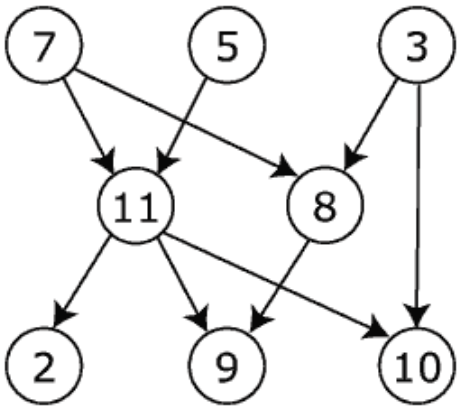
**Docker.** Один из самых удобных ПО для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Он позволит ускорить тестирование и создание всего приложения.

**Git.** Система контроля версий требовалась для написания абсолютно всего проекта.

**Linux.** Эта операционная система потребовалась для деплоя всего проекта на сервер, под мои задачи подходила именно Ubuntu. Поэтому было выбрано решение остановиться именно на ней.

## § 3. Выбор криптовалюты

Мой выбор пал на перспективную и прогрессивную криптовалюту IOTA и ее токен Shimmer. Эта криптовалюта имеет достаточно большие преимущества по сравнению с другими. Процесс формирования сети нельзя назвать блокчейном потому, что в нем нет привычного создания блоков, связанных между собой, как в биткоине. Структура данных представлена в виде ориентированного ациклического графа. Предназначена для эффективной и защищенной передачи



данных и платежей без комиссий между устройствами в интернете вещей. IOTA была разработана благодаря доктору математических наук Сергею Попову, Домиником Шинером и Дэвидом Сэнстебё.

Это оргграф, в котором отсутствуют направленные циклы. Направленный ациклический граф является обобщением дерева (точнее, их объединения – леса). Направленные ациклические графы широко используются в нейронных сетях, компиляторах и машинном обучении.

Плюсы в использовании криптовалюты IOTA:

1. отсутствие централизации со стороны майнеров;
2. микротранзакции (Пользователи могут совершать микропереводы без комиссий);
3. скорость транзакций и масштабируемость (Минимальная функция технологии proof of work (принцип защиты сетевых систем от злоупотребления услугами) выполняется, как и в любой другой технологии распределенного реестра. Но она ничтожно мала. В биткоине чтобы выполнить блок PoW нужна мощность нескольких мощных компьютеров. Для криптовалюты IOTA достаточно самого простого процессора);
4. безопасность (Современная криптография в основе покажет себя устойчивее даже с учетом появления квантовых компьютеров. В биткоине используются старые подходы к криптографии.

Именно исходя из таких огромных преимуществ, которыми обладает IOTA, я и решил выбрать именно ее.

## § 4. Функции приложения

1. Регистрация:
   1.1. Создание пароля;
   1.2. Создание мнемонической фразы.

2. Авторизация.

3. Получение информации о кошельке.

3.1. Баланс:

3.1.1. Общий баланс;

3.1.2. Отдельный баланс по токенам пользователя (IOTA, SHIMMER);

3.1.3. Баланс токенов по актуальному курсу доллара;

3.2. Курс токенов:

3.3. Просмотр актуальных адресов кошельков токенов:

3.3.1. В виде QR-кода:

3.3.2. В обыденном виде;

4. Взаимодействие с зарегистрированными пользователями:

4.1. Вывод первых 15 пользователей;

4.1.1. Аватарка пользователя (Telegram);

4.1.2. Имя пользователя (Telegram);

4.1.3. Уникальный username пользователя (Telegram);

4.2. Поиск по пользователям;

4.3. Избранное;

4.3.1. Добавление;

4.3.2. Удаление;

4.4. Пользователи, добавленные в избранные, выводятся первыми в списке;

4.5. Возможность нажать на пользователя и отправить ему токены;

5. Отправка токенов:

5.1. По адресу;

5.2. По пользователю из списка.

6. Список транзакций (История):

6.1. Просмотр последних 15 транзакций;

6.2. Возможности просмотра;

6.2.1. От кого или кому;

6.2.2. Аватарка пользователя;

6.2.3. Время подтверждения транзакции в сети;

6.2.4. Стоимость в долларах.

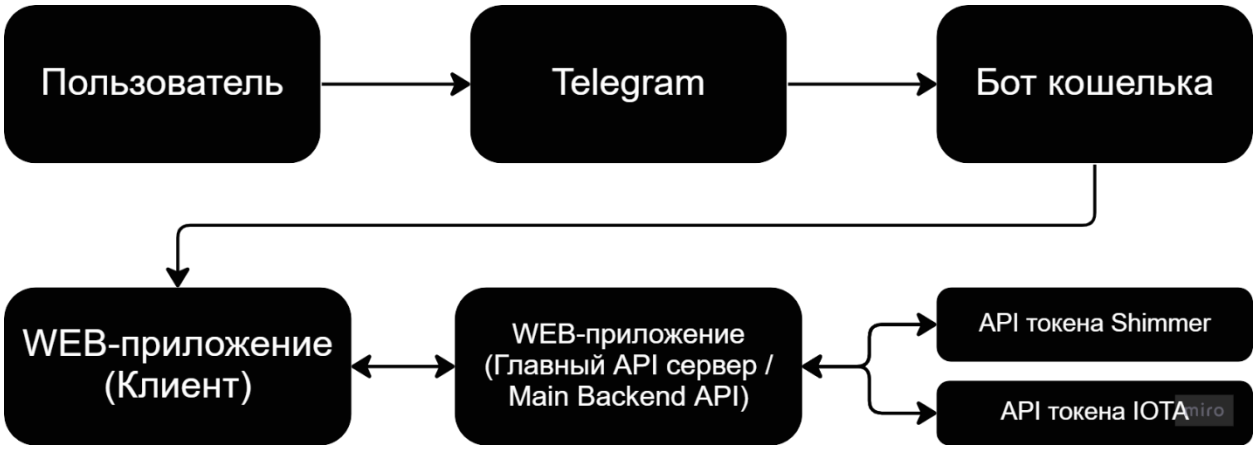
7. Поддержка светлой и темной темы (интеграция вместе с Telegram).

8. При осуществлении транзакции отправлять уведомление тому, кому были отправлены токены.

## § 5. Разработка архитектуры приложения

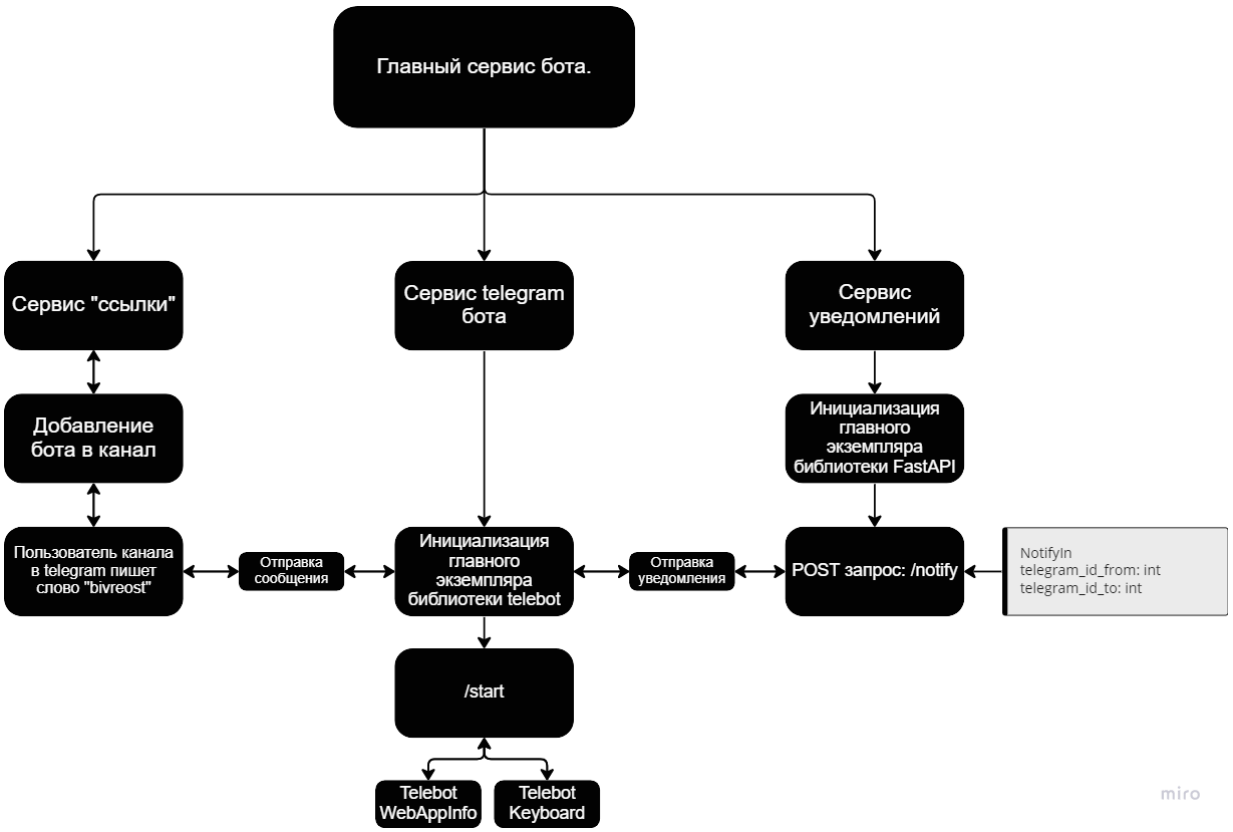
**Общая архитектура.**

Имея описания компонентов архитектуры приложения, можно ее составить. Я имею 4 главных аспекта приложения – клиент, сервер, API криптовалюты, бот. Всю архитектуру приложения я хочу представить в виде схемы.




Из этой общей схемы можно понять, как пользователь получает данные своего продукта, идя от высокоуровневой части проекта к более низким уровням приложения. Далее, мне хотелось бы описать каждую часть архитектуры приложения.

**Бот кошелька.**



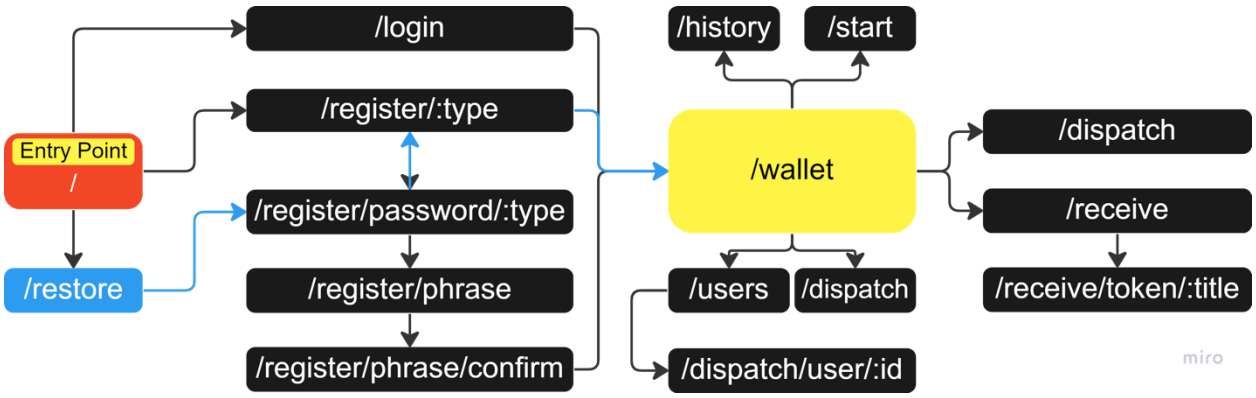

**Сервис уведомлений.** Этот сервис описывает реализацию уведомлений. Когда осуществляется транзакция и она подтверждается API криптовалюты, то автоматически отправляется уведомление пользователю, который получил свои токены.

**Сервис Telegram бота.** Главный сервис всей архитектуры бота, он реализует экземпляр библиотеки telebot, который взаимодействует с остальными сервисами данной архитектуры. Также описывает команду: «/start», которая напрямую взаимодействует с пользователем и отправляет ему кнопку с клиентской частью WEB-приложения.

**Сервис «ссылки».** Данный сервис реализует функцию пиара кошелька, при добавлени бота в канал и выдачи ему прав администратора. Это дает возможность отправлять ссылку на наше приложение. Это сделано для того, чтобы, когда человека спросили: «Каким кошельком криптовалюты Вы пользуетесь?». Он просто написал слово «bivreost», и в чат отправилась ссылка на бота.

**WEB-приложение (клиент).**

1. Компоненты: Компоненты являются основными заполнителями интерфейса пользователя. Они могут быть функциональными или классовыми.
2. Данные: в React приложении данные могут храниться на уровне компонентов в виде состояния или в виде свойств (props).
3. Роутинг: react-router-dom используется для настройки маршрутов в приложении, которые связывают URL-адреса с компонентами.
4. Хранилище данных: localstorage используется в качестве хранилища данных.
5. Сетевые запросы: axios используется для выполнения сетевых запросов и получения данных с сервера.
6. Я предпочитаю написание только функциональных компонентов, ибо при написании классовых компонентов читабельность кода ухудшается в разы.




Выше я прикрепил схему, как выглядит архитектура маршрутизации \ роутинга в моем приложении.

**WEB-приложение (сервер).**

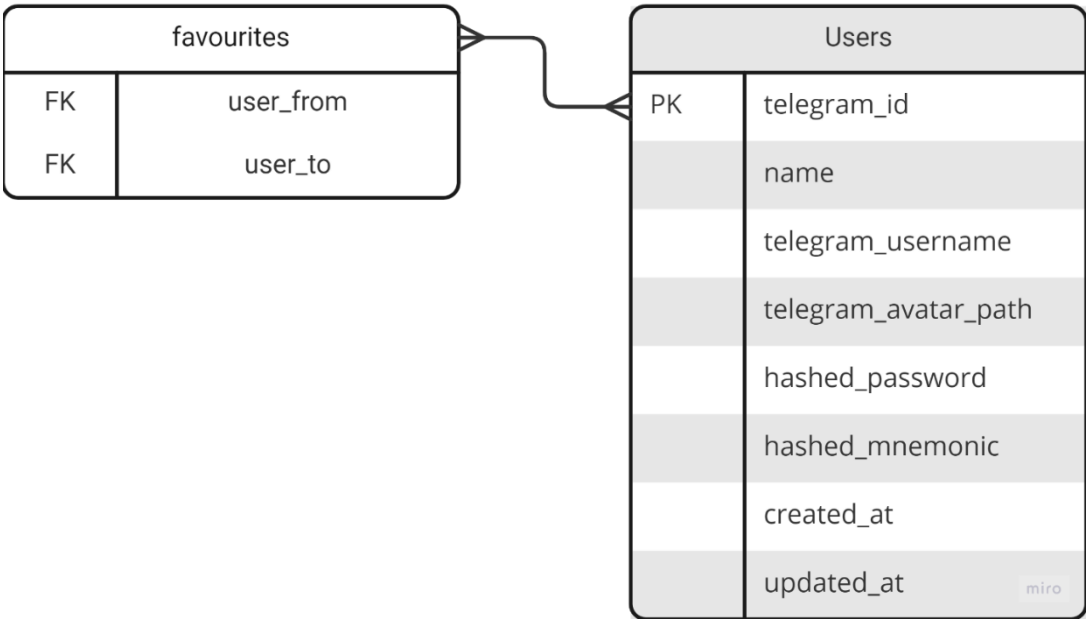
Backend – архитектура состоит из 3 сервисов: Main API (Главный сервис API), Shimmer API (Сервис для работой с токеном Shimmer), Iota API. Каждый сервис выполняет свою обязанность, что соответствует принципу S.O.L.I.D. Конкретно SPR (Single Responsibility Principle) – принцип единой ответственности приложения, это означает, что сущность должна иметь только

одну обязанность и должна быть полностью инкапсулирована. Всю серверную архитектуру я представлю в виде схемы.

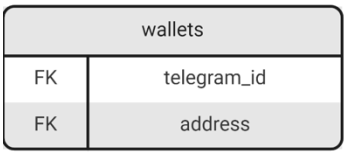

Из этой схемы можно понять, что главный сервис работает с двумя интерфейсами криптовалюты, каждый из которых выполняет, по сути, одинаковые задачи, но интерфейс работы с каждым токеном реализован по-разному. Это и есть реализация паттерна ООП – абстрактной фабрики.

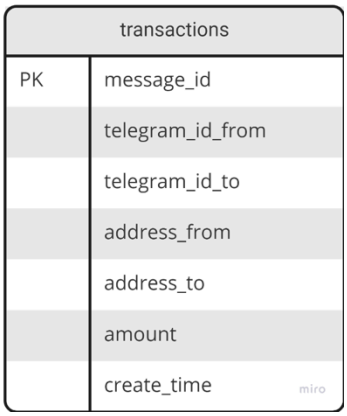
**Базы данных (сервер).**

# Main API




# Shimmer & IOTA API



# ГЛАВА 2. СОЗДАНИЕ АЛГОРИТМОВ БЕЗОПАСНОСТИ И ИХ ИНТЕГРАЦИЯ С TELEGRAM

## § 1. Алгоритм верификации данных (client-server)

Одно из преимуществ моего приложения заключается в интеграции безопасности с Telegram. Итак, чтобы верифицировать отправленные данные с клиента, нужно: отправить строку initData на сервер. Ее возможный вид:

```
"""
query_id=AAFldwgqAgAAAGV3CCoEIsyi&user=%7B%22id%22%3A5000165221%2C%22first_name%22%3A%22abcen7%22%2C%22
last_name%22%3A%22Devnet%22%2C%22username%22%3A%22devnet%22%2C%22language_code%22%3A%22ru%22%7D&
auth_date=1676117926&hash=9fde6b840296be4164625d9aa5516d0fa1b8500d9cc7f942957b65dc47c3aab4
"""
```

Для начала, нужно разбить ее по параметрам, отсортировав их по алфавиту. В том числе будет нужно извлечь параметр hash.

```
('auth_date=1676117926\n'
 'query_id=AAFldwgqAgAAAGV3CCoEIsyi\n'
 'user={"id":5000165221,"first_name":"abcen7","last_name":"Devnet","username":"devnet","language_code":"ru"}',
 '9fde6b840296be4164625d9aa5516d0fa1b8500d9cc7f942957b65dc47c3aab4')
```

auth_date – время входа пользователя в приложение;

query_id – уникальный идентификатор сеанса приложения;

user.id – уникальный идентификатор пользователя;

user.first_name – имя пользователя;

user.last_name – фамилия пользователя;

user.username – уникальное имя пользователя;

user.language_code – код языка в соответствии с ISO 639-1;

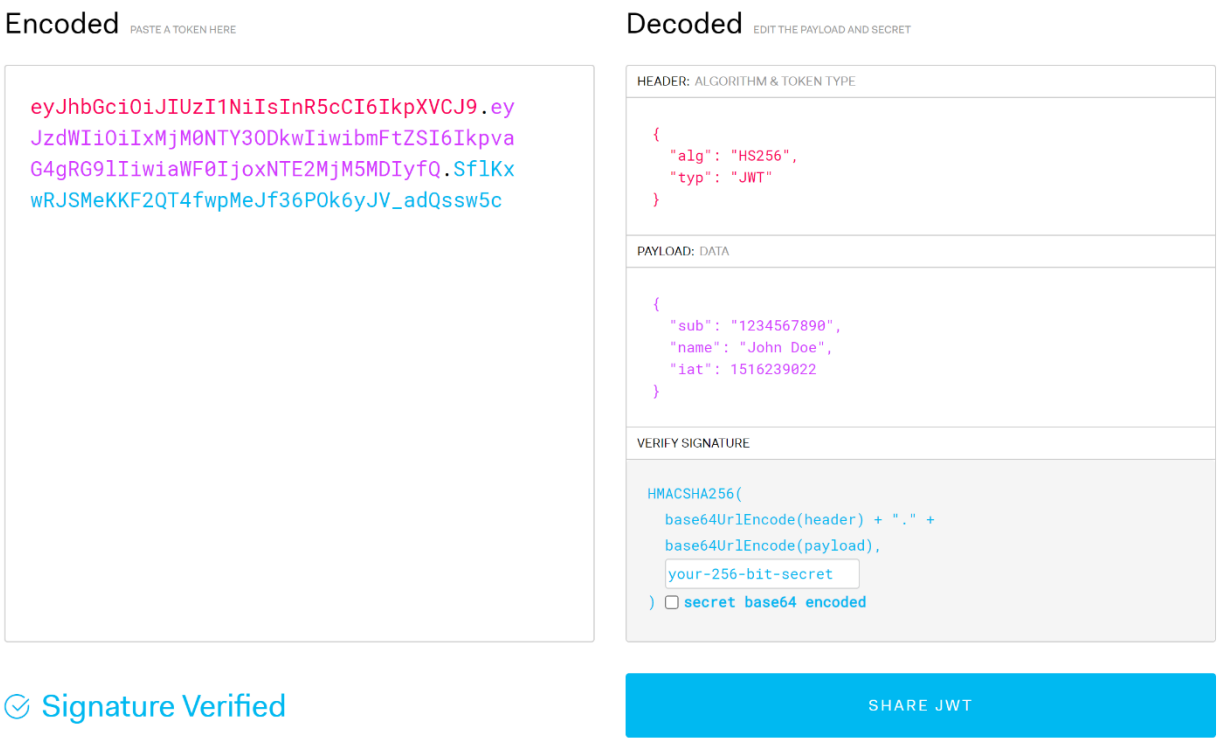
hash – все вышеперечисленное, зашифрованные данные телеграмом.

После этого нужно получить data_check_string. Ее получение простое, нужно соединить все параметры символом перехода на другую строку «\n»,

кроме hash. Далее получаем параметр secret_key. Его значение – это HMAC-SHA-256 сигнатуры токена Telegram бота и константы строки WebAppData, использованное, как ключ. Затем, чтобы понять, валидные ли данные, требуется сравнить шестнадцатеричное представление HMAC-SHA-256 (data_check_string, secret_key) и полученный параметр hash. Если получится равенство, то данные не были изменены и подделаны. Таким образом, будет создана защита от CSRF атак (англ. cross-site request forgery – «межсайтовая подделка запроса».

## § 2. Реализация JWT

JWT – это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Используется для передачи данных для аутентификации в клиент-серверных приложениях.




Здесь представлен дебаггер с официального сайта https://jwt.io

В моей реализации в PAYLOAD я кладу только telegram_id пользователя. Алгоритм шифрования остается тот же: HS256. Чтобы полностью не имплементировать алгоритм создания access token с нуля, уже есть готовый pip пакет jwt (1.3.1) в python. С помощью этого пакета и создания двух классов, полностью реализуется алгоритм JWT.

## § 3. Взаимодействие алгоритмов с API

В каждом запросе API, за исключением:

1. /static/avatars/{telegram_id;
2. /access_token;
3. /users

используется алгоритм верификации данных и jwt алгоритмов. Благодаря, такой архитектуре я имею максимально – защищенное приложение от разных атак.

В конечном итоге, для того, чтобы злоумышленнику получить доступ к кошельку пользователя, нужно:

1. получить доступ к учетной записи клиента;
2. подобрать пароль пользователя;

2.1. или же знать мнемоническую фразу.

Взлом учетной записи пользователя Telegram становится практически невозможным, потому что для этого нужно получить доступ к телефону пользователя, т.к. авторизация полностью на этом завязана.

# ГЛАВА 3. РЕАЛИЗАЦИЯ UI/UX ДИЗАЙНА

## § 1. User-friendly интерфейс

User-friendly – обозначение интерфейса, дружественного пользователю в теории юзабилити. Этим словосочетанием обозначают среду, продуманную с учетом удобства пользователя.

Чтобы сделать интерфейс user-friendly (дружелюбным для пользователя), следует учитывать следующие рекомендации:

1. простота использования: интерфейс должен быть простым в использовании и понятным, чтобы пользователь мог быстро и легко находить то, что ему нужно;
2. ориентированность на пользователя: интерфейс должен быть направлен на потребности и предпочтения пользователей, так чтобы они могли работать с удовольствием;
3. читаемый текст: текст в интерфейсе должен быть читаемым и понятным, для чего необходимо использовать простые слова и ясные инструкции;
4. удобная навигация: пользователь должен легко найти то, что ему нужно, используя удобную навигацию и меню;
5. привлекательный дизайн: интерфейс должен выглядеть привлекательно и профессионально, чтобы пользователь мог с удовлетворением работать с приложением и не отталкиваться от его внешнего вида. Это может включать в себя использование цветов, шрифтов и изображений, которые соответствуют целевой аудитории и цели приложения.

Мой воссозданный интерфейс соответствует основным принципам user-friendly дизайна.

1. **Он интуитивно понятен**. Это первое, о чем нужно позаботиться, ведь если пользователь не поймет, как пользоваться сайтом, то через некоторое время, не добившись нужного результата, покинет ресурс. Возможно, на долгое время или даже навсегда.

2. **Он минималистичен**. Сложный интерфейс тяжело понять. Например, вместо трех фраз всегда лучше использовать одну, если это, конечно, возможно. Дополнительные элементы и категории на главной странице тоже сильно отвлекают внимание. Однако здесь нужно уметь различать «лишние» элементы и по-настоящему необходимые.

3. **Он быстро загружается**. Медленная работа сайта отталкивает очень многих посетителей, и к тому же вызывает неприязнь к ресурсу. Зачастую, это часто имеет прямую связь с минималистичностью – чем меньше на страницах «тяжелых» сложных элементов, тем быстрее он загружается.

4. **Он привлекателен**. Посетители всегда обращают внимание на визуальную составляющую, пусть даже и подсознательно. Им приятнее иметь дело с интерфейсом, который радует глаз. Однако нужно понимать, что оценка красоты дизайна – это очень субъективно. Поэтому нужно ориентироваться на целевую аудиторию и ее потребности.

5. **Он сразу демонстрирует все важные опции**. Желательно сразу показывать пользователю все возможности. Использовать выпадающие списки и меню стоит только там, где без этого нельзя обойтись. Если часть возможных действий будет скрыта, пользователь может не догадаться, как их совершить. А если эти действия целевые (например, купить товар, оставить номер телефона), то стоит уделить им особое внимание.

6. **Он умеет общаться с пользователем**. Система должна наглядно демонстрировать пользователю, что его действия обрабатываются. Например, если он отправляет сообщение, то этот процесс должен сопровождаться выводом на экран фразы «сообщение отправляется», а окончание этого процесса – «сообщение отправлено».

7. **Он предсказуем**. Предсказуемость сайта не означает скуку и отсутствие интереса. Речь идет о том, что пользователь, увидев какой-либо элемент сайта должен понимать, как он (элемент) будет «вести себя» при взаимодействии. Если объект выглядит как меню, он должен вести себя как меню, если похож на кнопку, то соответственно, должен куда-либо вести или производить какое-то действие.

8. **Каждая кнопка имеет собственное назначение**. Это может быть переход, раскрытие меню, команда открыть в новом окне, отправить письмо и так далее. Активные и неактивные элементы должны отличаться друг от друга, для этого их можно выделять стилем. Пользователь должен понимать, на какие кнопки он может нажать и перейти на другую страницу, а какие присутствуют просто для декора или дополнительной информации.

Мое приложение представляет удобный для пользователей дизайн user-friendly (доброжелателен для пользователя), где пользователи с первых секунд могут найти нужный им раздел/функцию, не разбираясь в приложении.

Это помогает получить приложению максимальную конверсию от пользователей, а также положительные эмоции при использовании данного приложения.

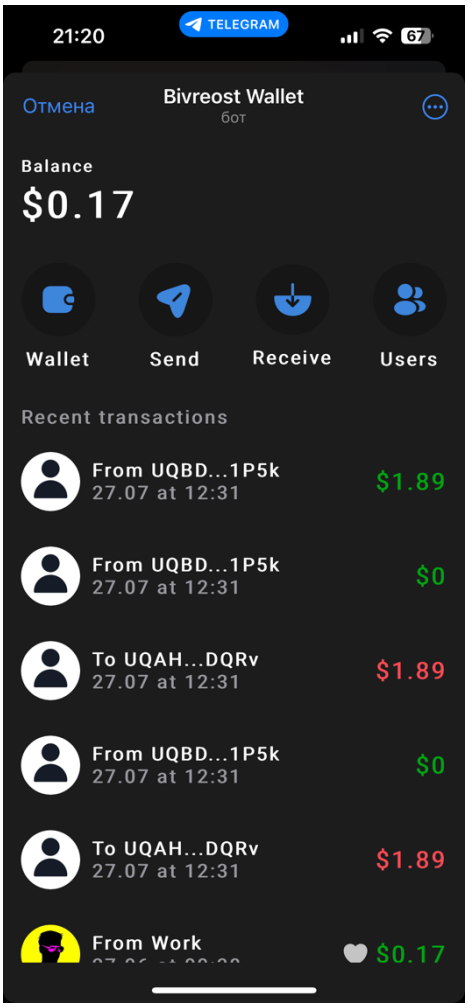
Приложение выполнено в синей гамме, что способствует расслаблению, снижению стресса.

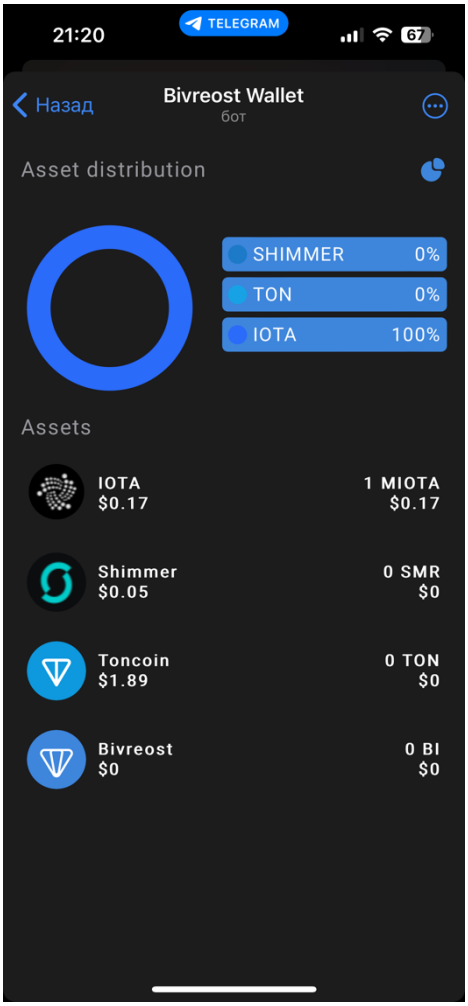
Для удобства пользователя в приложении есть QR код, при сканировании которого копируется адрес кошелька.

Есть поддержка черной и белой темы. Можно заметить, что шрифт, интерфейс никак не меняются, изменяется только цвета фона и шрифта.

В приложении присутствует визуализация картинок, дополняя, расширяя простор приложения.

Во всех разделах кошелька сразу виден баланс. Пользователю не придётся искать, какая сумма на его счете.


21:20
TELEGRAM
Отмена
Bivreost Wallet
бот
Balance
$0.17
Wallet
Send
Receive
Users
Recent transactions
From UQBD...1P5k
27.07 at 12:31
$1.89
From UQBD...1P5k
27.07 at 12:31
$0
To UQAH...DQRv
27.07 at 12:31
$1.89
From UQBD...1P5k
27.07 at 12:31
$0
To UQAH...DQRv
27.07 at 12:31
$1.89
From Work
$0.17


21:20
TELEGRAM
Назад
Bivreost Wallet
бот
Asset distribution
SHIMMER 0%
TON 0%
IOTA 100%
Assets
IOTA
$0.17
1 MIOTA
$0.17
Shimmer
$0.05
0 SMR
$0
Toncoin
$1.89
0 TON
$0
Bivreost
$0
0 BI
$0

## § 2. User-Experience интерфейс

UX (User Experience) дизайн – это наука и искусство создания продуктов, которые оптимизированы для комфортного и удовлетворяющего использования. Он включает в себя исследование, проектирование, разработку и оценку продуктов и услуг, таких как веб-сайты, мобильные приложения, программное обеспечение и электронные устройства, для обеспечения лучшего опыта пользователя. Цель UX дизайна – создать продукты, которые легко использовать, понятные и приятные, повышая эффективность, удовлетворенность и лояльность клиентов.

Создание дизайна – это многоступенчатый процесс, включающий в себя планирование, исследование, концептуализацию и реализацию. Он начинается с определения целей и задач проекта, а затем проводится исследование аудитории, конкурентов и тенденций в области дизайна. На основе этой информации формируется концептуальный дизайн, который может включать в себя схемы, макеты и прототипы.

В процессе создания дизайна я учитывал эстетические и функциональные требования, а также пытался удовлетворить потребности конечных пользователей. Это включало в себя использование эффективных цветовых схем, определение правильного шрифта и выбор удобных интерактивных элементов.

В конечном итоге, созданный дизайн был протестирован и оптимизирован, чтобы удостовериться, что он удовлетворяет всем требованиям и функционирует правильно. Это включало в себя проведение юзабилити-тестирования с пользователями, чтобы узнать, как они воспринимают и взаимодействуют с дизайном.

В целом, создание дизайна требует времени, творчества. Целью является создание продукта, который не только выглядит прекрасно, но и эффективно функционирует и удовлетворяет потребностям пользователей.

# ГЛАВА 4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ В TEST-NET И MAIN-NET СЕТЯХ И ПОЛУЧЕНИЕ ФИДБЕКА ОТ ПОЛЬЗОВАТЕЛЕЙ

## § 1. Выбор конфигурации сервера

Подбор подходящей конфигурации сервера для веб-приложения важен для обеспечения его производительности и доступности. Чтобы подобрать подходящую конфигурацию, необходимо учитывать следующие факторы:

1. Требования к ресурсам: нужно определить, сколько памяти, процессорных ядер и дискового пространства необходимо для работы вашего веб-приложения.
2. Нагрузка на сервер: оцените, сколько ожидается посетителей и как часто они будут использовать ваше веб-приложение.
3. Совместимость с технологиями: нужно убедиться, что выбранная конфигурация сервера совместима с используемыми технологиями в веб-приложении, такими как ОС, язык программирования, веб-фреймворк и база данных.
4. Расширяемость: нужно предусмотреть возможность расширения ресурсов сервера в будущем, если это необходимо.
5. Бюджет: учесть свой бюджет и выбрать конфигурацию, которая удовлетворяет требованиям и входит в рамки бюджета.

Важно помнить, что нет одной идеальной конфигурации сервера, которая была бы подходящей для всех веб-приложений. Конфигурация должна быть подобрана индивидуально для каждого приложения, учитывая его уникальные требования и ограничения. От абстрактных рассуждений перейду к выбору:

**Требования к ресурсам**

**Память.** Собирая docker-compose на своем ПК, я понял сколько потребуется ОЗУ.

1. main-api ~= 1.08 GB
2. iota-api ~= 1.47 GB
3. shimmer-api ~= 4.25 GB
4. client-web ~= 1.04 GB
5. telegram-bot ~= 403 MB
6. postgresql ~= 378.7 MB
7. mysql ~= 517.46 MB
8. volumes ~= 315.1 MB

Итого я имею: 9,45426 GB ОЗУ должно быть выделено только под docker, плюс еще нужно учесть систему и выбрать минимальный тариф, то есть на сервере должно быть **16 GB ОЗУ.**

**Процессор**. Мое приложение также требовательно к процессу, в особенности высокой нагрузке со стороны количества клиентов. Т. к. сказать точно невозможно, оценив нагрузку во всех состояниях приложения, я взял среднее. **8 x 3.00 ГГц.**

**Дисковое пространство.** Проведя анализ веса 1 кошелька и прикинув примерный охват пользователей, можно понять:

- вес 1 средненагруженного кошелька = 1,92 МБ (WMLW – *weight medium-loaded wallet*);

- примерный охват = 500 пользователей (AC – *approximate coverage*);

- вес приложения (вместе с установленными зависимостями) ~= 1168 MB (WoW – *weight of wallet*);

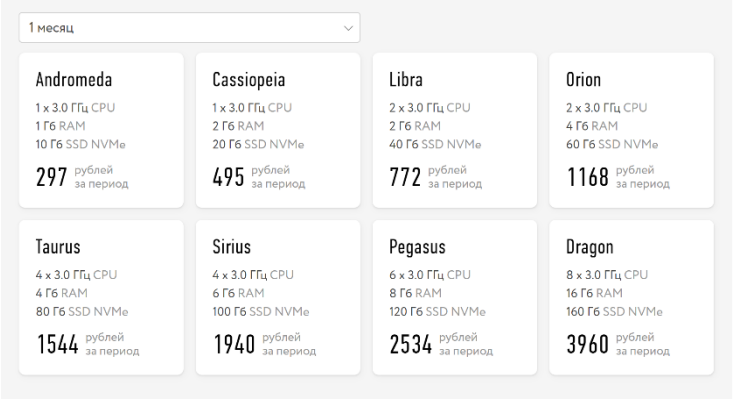
- занятое место = (AC * WMLW) + (WoW) = 1,92 * 500 + 1168 = **2128 MB ⇔ 2,09 GB.**

**Операционная система.** Для операционной системы отлично подходит Linux, конкретно Ubuntu 22.04. Это идеальный выбор под python веб – приложение + npm. Apt великолепно подтягивает все требуемые зависимости, из – за чего мой выбор пал на ubuntu.

**Расширяемость.** Расширяемость приложения в моем случае была не важна, т.к. проводилось только тестирование в тестовой сети, а не постоянный релиз приложения, вследствие чего не требовалось бы и расширять сервер.

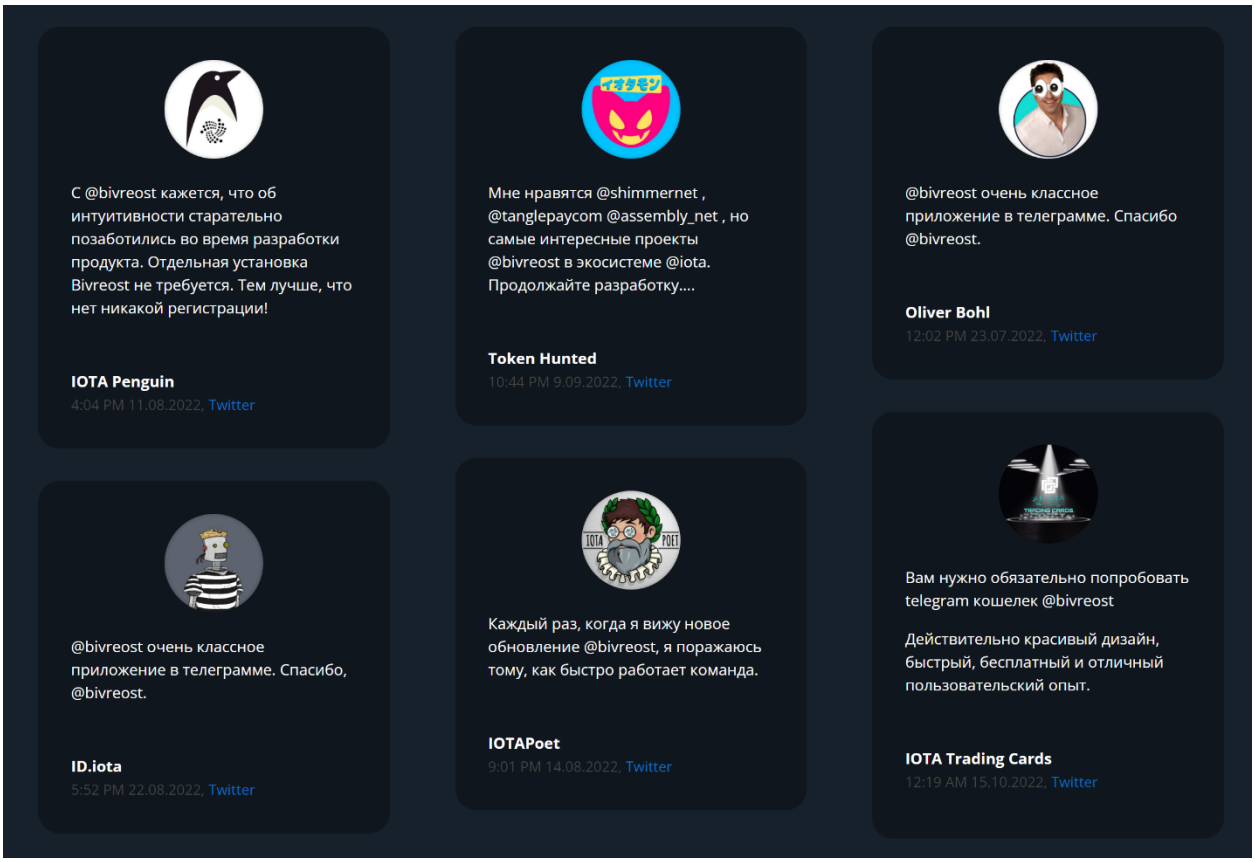
**Бюджет.** В ходе разработки проекта также были распределены силы и на поиск единомышленника – инвестора. Такой человек нашелся, он полностью согласен на все обоснованные траты во благо приложения. Выбор конфигурации я производил из предоставленных VPS серверов https://masterhost.ru, я давно работаю с этой компанией, она ни разу не подводила меня, поэтому я решил остановиться именно на ней. Она предлагает:

Исходя из вышеперечисленных требований, я остановился на тарифе «Dragon» за 3960₽.

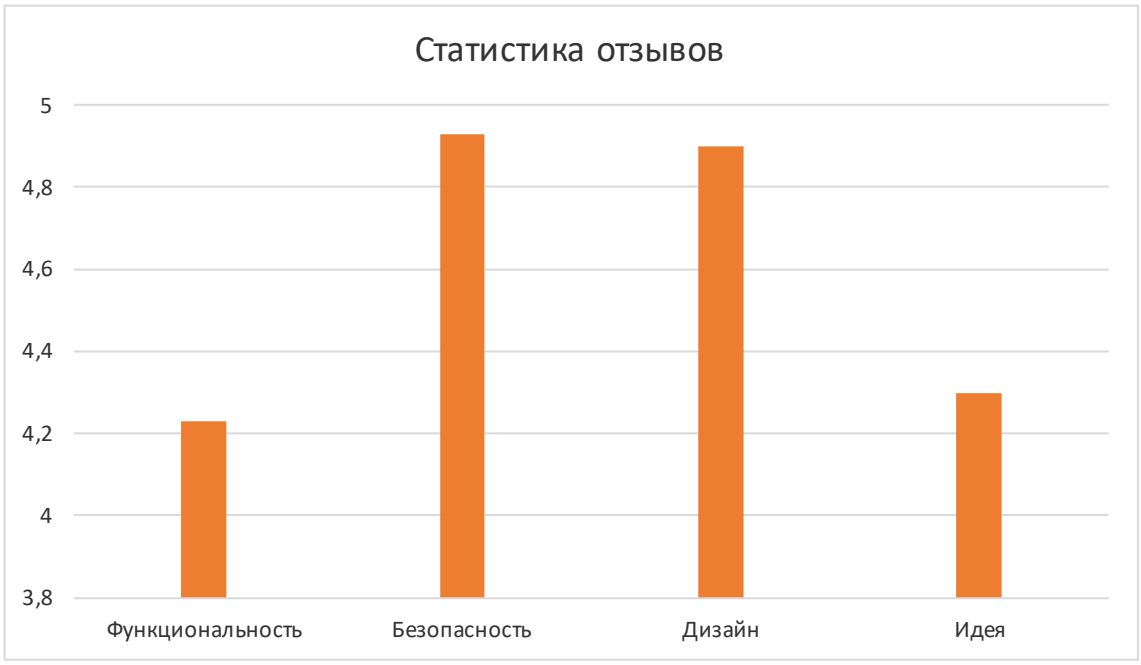



## § 2. Анализ фидбека пользователей

Приложение в работе показало себя уверенно, отрабатывало без багов и падений. Большинству пользователей моего приложения все понравилось, вот некоторые отзывы:




Это только малейшая часть всех отзывов, написанных о моем приложении. Больше всего меня порадовало, что пользователям действительно понравился дизайн приложения.




Я сделал опрос среди 493 участников моего кошелька и попросил их оценить кошелек по 5-бальной шкале и 4 параметрам: функциональность,

безопасность, дизайн, идея. По результатам опроса: функциональность – 4,23; безопасность – 4,93; дизайн – 4,9; идея – 4,3. Средний балл данного приложения: **4,59 / 5.** Понимания, что данный проект имеет потребность в использовании, и клиенты данный продукт закрывает потребности пользователей, я принялся к следующим шагам развития данного проекта.

## § 3. Миграция платформы с TEST-NET на MAIN-NET сети

Хотелось бы начать с отличия TEST-NET от MAIN-NET сетей. TEST-NET сеть – это сущность, которая используется для тестирования токена, в т.ч. для реализации продуктов под криптовалюту. В этой сети пользователи пробуют работу данной криптовалюты. Самое главное, что в ней все токены имитируются и являются синтетическими.

MAIN-NET сеть – это сущность, которая используется уже для повседнего использования. В ней токены уже приравниваются к реальной стоимости, торгуются на криптобиржах и стоят реальных денег.

Самое главное, что стоило учитывать при миграции с TEST-NET на MAIN-NET сеть, что требуется сохранить абсолютно все данные пользователей, за исключением их кошельков, т. о. мы сможем пересоздать только кошельки, сохранив всю информацию об их аккаунтах (статических данных). Такую задачу решает достаточно простой python-скрипт. Его задача обойти все колонки в базе данных, очистить некоторые ячейки, убрать ненужную информацию и переформатировать ее.

## § 4. Анализ данных

Выгрузив все данные с сервера, можно просуммировать мою работу. Данные предоставлены с 1 апреля 2023 года по 7 января 2024 года. Также стоит определиться с курсом токенов MIOTA & SHIMMER, включая курс доллара. Так, как за этот период они менялись, я же возьму средние значения с сайта (https://coinmarketcap.com)

Курс доллара на 07.01.2024 по данным https://cbr.ru: 1$ = 90,85₽

MIOTA: 0,25$ ~= 23₽

SHIMMER: 0,04$ ~= 3.30₽

1. Объем транзакций между кошельками (внутренними и внешними)
    a. MIOTA: 63454 MIOTA ~= 15864$ ~= 1 441 245 ₽
    b. SHIMMER: 35204 SHIMMER ~= 1409$ ~= 128 010₽
2. Количество зарегистрированных пользователей 5631 человек.
3. Ежедневно кошельком пользуются порядка 50-100 человек

## ЗАКЛЮЧЕНИЕ

В ходе разработки и создания всего проекта, я считаю, что я выполнил цель проекта, а также решил все поставленные задачи. Разработанное веб-приложение представляет собой значительный шаг в улучшении удобства использования токенов. Интеграция с популярным мессенджером Telegram позволяет пользователям легко и быстро выполнять операции с криптовалютой, используя единый интерфейс.

Особенностью этого проекта является удобный интерфейс, который стремится упростить жизнь пользователям. Я приложил значительные усилия

для улучшения UX дизайна и интуитивного интерфейса. Тестирование с реальными пользователями позволило получить данные о том, насколько успешно была достигнута эта цель.

В заключение, следует отметить, что интеграция криптовалютного кошелька в платформу обмена сообщениями Telegram может произвести революцию в том, как пользователи управляют своими цифровыми активами. Сочетая удобство и доступность Telegram с безопасностью и функциональностью специального криптовалютного кошелька, пользователи смогут легко и уверенно управлять своими финансами. Чтобы веб-приложение было удобным и безопасным, было проведено тщательное тестирование и исследование пользователей, чтобы выявить любые потенциальные проблемы и оптимизировать взаимодействие с потенциальным клиентом. В конечном счете, успех этого проекта будет зависеть от того, насколько хорошо он отвечает потребностям и ожиданиям своих целевых пользователей, поэтому будет важно продолжать отслеживать отзывы пользователей и вносить улучшения по мере необходимости.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ


1. IOTA – технология - https://goo.su/P0Xj.
2. React (https://ru.reactjs.org/).
3. User-Friendly - https://goo.su/hs5iAFW.
4. Сайт зависимостей npm: https://npmjs.com.
5. Сайт зависимостей python: https://pypi.org.

# **ПРИЛОЖЕНИЕ № 1**

Видео с работоспособностью всего проекта - https://youtu.be/MhDcKLhWh9E