Реферат

Пояснительная записка состоит из двух глав, списка литературы, состоит из 171 страниц, 1 таблиц, 40 рисунков.

Ключевые слова: блокчейн, Proof Of Authority, python, PostgreSQL, Нода, консенсус, хеш, сервер, клиент-сервер, масштабируемость, децентрализация, валидация.

Объектом исследования ВКР является современная автоматизация проведения трансграничных транзакций посредством разработки блокчейн-систем на основе Proof of Authority (PoA). Такие системы обеспечивают высокую скорость и безопасность при сохранении децентрализованной природы блокчейна. Объектом исследования служит разработка эффективной и безопасной блокчейн-платформы для проведения трансграничных транзакций. В работе уделяется внимание процессу консенсуса, интеграции различных узлов и нод, обеспечению масштабируемости и взаимодействию с пользователями посредством веб-интерфейса.

Цель заключается в создании блокчейн-системы на основе PoA, способной автоматизировать проведение трансграничных транзакций с обеспечением высокой скорости, безопасности и масштабируемости. Объект исследования также стремится внедрить веб-интерфейс, который обеспечит пользователям интуитивное управление транзакциями и доступ к информации о текущих операциях. Актуальность работы обоснована растущим интересом к технологии блокчейн и необходимостью развития систем, способных обеспечить безопасные и эффективные трансграничные платежи.

В условиях активного международного взаимодействия современному бизнесу и частным лицам требуется инструмент, сочетающий высокую скорость, прозрачность и устойчивость. Практическим результатом данной вкр является создание блокчейн-системы на алгоритме Proof Of Authority с системой масштабируемых нод и первичного интерфейса, позволяющей пользователям безопасно и быстро проводить транзакции. Внедрение такой системы поможет ускорить трансграничные платежи и оптимизировать

взаимодействие между различными экономическими субъектами, укрепляя доверие к процессу.

Содержание

Введение

В современной эпохе быстрого прогресса в области цифровых технологий и финансовых инноваций наблюдается нарастающая потребность в разработке эффективных механизмов для выполнения трансграничных финансовых операций с использованием различных цифровых валют. Ответом на эту потребность является создание передовых технологических решений, которые обеспечивают высокую степень безопасности, прозрачности и операционной эффективности.

В рамках данной ВКР представлен механизм прямых трансграничных финансовых операций между цифровыми валютами разных стран, основанный на технологии блокчейн с использованием протокола Proof of Authority (PoA). Этот механизм является инновационным решением, призванным устранить множество трудностей, связанных с международными финансовыми транзакциями.

Перед детальным изучением разработанного механизма следует уделить внимание историческим предпосылкам. С появлением таких цифровых валют, как Bitcoin и Ethereum, возникла необходимость их применения в международных финансовых операциях. Однако традиционные методы сталкиваются с рядом ограничений, включая вопросы безопасности, время подтверждения транзакций и высокие комиссии.

Применение блокчейн-технологии на основе протокола Proof of Authority (PoA) предлагает решение данных проблем, обеспечивая высокий уровень безопасности и операционной эффективности за счёт включения ограниченного числа доверенных участников для подтверждения транзакций.

Целью данной работы является разработка и реализация механизма прямых трансграничных финансовых операций, который обеспечит высокую надёжность, прозрачность и операционную эффективность с использованием блокчейн-технологии на основе протокола Proof of Authority. Исследование включает анализ требований,

проектирование системы, её реализацию и тестирование для гарантии полной функциональности и соответствия заданным критериям.

В конечном итоге, успешное внедрение предложенного механизма способствует углублению международных финансовых отношений, упрощению процессов трансграничных операций и повышению доверия к цифровым валютам как инструменту международных расчётов.

Для успешной реализации системы прямых трансграничных финансовых операций между цифровыми валютами различных стран на базе блокчейн-технологии с применением протокола Proof of Authority (PoA), следует выполнить следующие ключевые задачи:

1. Анализ требований: Идентификация функциональных и нефункциональных требований системы. Это включает обеспечение возможности прямых трансграничных финансовых операций, гарантирование безопасности, минимизация времени подтверждения транзакций, совместимость с разнообразными цифровыми валютами и соответствие нормативным требованиям.

2. Проектирование системы: Разработка комплексной архитектуры системы, выбор подходящих блокчейн-технологий, структурирование сети с особым вниманием к механизмам синхронизации узлов, а также определение интерфейсов для взаимодействия с системой.

3. Реализация синхронизации узлов: Создание механизмов для синхронизации узлов, позволяющих модифицировать код узла и осуществлять асинхронную работу, включая управление собственными ветками блоков.

4. Реализация веб-интерфейса и API: Разработка пользовательского веб-интерфейса для управления аккаунтами, просмотра истории транзакций, а также создание API для интеграции системы с внешними приложениями и сервисами.

5. Генерация адресов транзакций: Разработка системы для генерации уникальных адресов транзакций, что повысит безопасность и гарантирует уникальность каждой операции.

6. Развертывание блокчейн сети: Организация и конфигурация блокчейн сети на основе протокола Proof of Authority, выбор участников сети (доверенных узлов) и настройка параметров сети.

7. Интеграция с внешними системами: Разработка интерфейсов для взаимодействия с внешними системами, такими как цифровые кошельки, криптовалютные биржи и финансовые учреждения.

8. Тестирование и оптимизация: Проведение всестороннего тестирования системы для проверки соответствия требованиям, исправление выявленных ошибок и оптимизация производительности и безопасности.

9. Запуск и поддержка: Внедрение системы в эксплуатацию, обеспечение ее стабильной и непрерывной работы, включая мониторинг производительности, обновление ПО и реагирование на технические проблемы.

Эти задачи, учитывая специфику цифровых валют и международных финансовых операций, обеспечат создание эффективной, безопасной и надежной системы для проведения прямых трансграничных финансовых операций на основе блокчейн-технологии

Раздел 1. Аналитический

						ННГАСУ-09.03.02 – 2024		
Изм.	Кол.уч	Лист	№ док.	Подпись	Дат			
Зав. Каф.		Кислицын Д.И.				Раздел 1 Аналитический	Стадия	Лист	Листов
Руковод.		Кислицын Д.И.					ВКР	11	25
Разраб.		Самедов Н.Ю.					Каф. ИСТ гр. ИС-29		
Консульт.		Кислицын Д.И.							
Н. Контр.		Суханова Н.Т.							

Глава 1. Анализ предметной области

1.1 Блокчейн и его версии

Блокчейн – это распределенный цифровой реестр, который записывает транзакции в цепочке блоков. Каждая транзакция проверяется участниками сети и добавляется в блок, который затем связывается с предыдущими блоками, создавая непрерывную и неизменную цепочку данных. Этот процесс обеспечивает высокую степень безопасности, прозрачности и неизменности записей.

Основные характеристики блокчейна:

– Децентрализация: Управление и валидация транзакций осуществляется множеством участников сети, что исключает необходимость центрального органа.

– Безопасность: Использование криптографических методов для защиты данных и предотвращения их изменения.

– Прозрачность: Все участники сети имеют доступ к полной истории транзакций, что обеспечивает высокий уровень прозрачности.

– Неизменность: после добавления в блок данные становятся неизменяемыми, что предотвращает их фальсификацию.

Поколения блокчейнов:

1. Первое поколение (Blockchain 1.0). Первые блокчейны, разработанные для криптовалют, таких как Bitcoin. Основное внимание уделяется децентрализованной передаче цифровых активов и обеспечению безопасности транзакций. Это поколение решает проблему двойного расходования и обеспечивает надежную передачу ценности.

2. Второе поколение (Blockchain 2.0). Введение смарт-контрактов, что расширяет функциональность блокчейнов. (Ethereum). Смарт-контракты позволяют автоматизировать выполнение договорных условий и расширяют возможности

использования блокчейнов за пределами простых финансовых транзакций, включая децентрализованные приложения (dApps) [26].

3. Третье поколение (Blockchain 3.0). Решение проблем масштабируемости, интероперабельности и улучшение производительности. (Cardano, Polkadot). Это поколение направлено на улучшение скорости транзакций и пропускной способности сети, а также на обеспечение взаимодействия между различными блокчейн-сетями.

4. Четвертое поколение (Blockchain 4.0). Интеграция блокчейна в промышленность и корпоративные приложения, с акцентом на безопасность, масштабируемость и эффективность. (VeChain, Hyperledger). Платформы этого поколения предназначены для промышленного применения, предлагая высокую производительность, надежность и адаптацию под бизнес-потребности. Они способствуют цифровизации различных отраслей и улучшению бизнес-процессов [4].

Моё блокчейн-решение относится к четвертому поколению (Blockchain 4.0) и обладает следующими ключевыми характеристиками:

Технологическая основа и алгоритм PoA. Решение основано на алгоритме консенсуса Proof of Authority (PoA), который обеспечивает высокую скорость и производительность благодаря использованию авторизованных валидаторов. Это позволяет достичь высокой пропускной способности и низких задержек при обработке транзакций.

Децентрализация и безопасность. Несмотря на использование авторизованных валидаторов, решение сохраняет элементы децентрализации, что снижает зависимость от одного центра управления и повышает устойчивость к атакам. Строгий отбор валидаторов и улучшенные механизмы защиты данных обеспечивают высокий уровень безопасности и надежности системы.

Гибкость и интеграция. Блокчейн-решение предназначено для интеграции в корпоративные и промышленные системы, предлагая гибкость и адаптацию под специфические бизнес-потребности. Оно поддерживает быстрое масштабирование и интеграцию с различными цифровыми сервисами и инновационными бизнес-моделями.

Экономическая эффективность. Разработанный блокчейн снижает операционные издержки и делает транзакции более экономичными благодаря низким затратам на поддержку инфраструктуры и отсутствие необходимости выполнения сложных вычислений, как в системах на основе Proof of Work (PoW).

Масштабируемость и инновации. Созданная система решает проблемы масштабируемости, предлагая надежные и эффективные решения для бизнеса и международных транзакций. Это способствует поддержке и развитию блокчейн-экосистемы, а также открывает возможности для создания новых бизнес-моделей и услуг.

1.2 Изучение существующих аналогов блокчейнов

При анализе текущих блокчейн-систем и их механизмов консенсуса можно выявить ряд важных аспектов. Давайте рассмотрим ключевые характеристики некоторых из них:

Bitcoin (Proof of Work):

– Механизм консенсуса: Proof of Work (PoW) требует от майнеров решения вычислительно сложных задач для генерации новых блоков и подтверждения транзакций.

– Недостатки: Высокое энергопотребление, замедление скорости подтверждения транзакций, потенциальная централизация майнинга крупными игроками [1].

Ethereum (Proof of Stake):

– Механизм консенсуса: Ethereum в 2022 году успешно перешел с PoW на Proof of Stake (PoS), где верификация транзакций осуществляется участниками, пропорционально их доле валюты в сети.

– Недостатки: В текущем состоянии Ethereum сталкивается с теми же проблемами, что и Bitcoin [8]. В перспективе PoS может привести к концентрации криптовалюты в руках небольшой группы участников.

Ripple (Протокол консенсуса):

– Механизм консенсуса: Ripple использует уникальный протокол консенсуса, который позволяет достигать согласия по транзакциям без необходимости в вычислительных задачах.

– Недостатки: Система зависит от доверия к центральной организации, управляющей списком узлов, участвующих в процессе консенсуса.

Cardano (Ouroboros Proof of Stake):

– Преимущества: Энергоэффективный протокол PoS, строгий академический подход к разработке и высокий уровень безопасности.

– Недостатки: относительно новая платформа, которая еще не достигла уровня распространения Ethereum или Bitcoin.

Hyperledger Fabric (Permissioned Blockchain):

– Преимущества: Высокий уровень контроля и безопасности, способность к эффективной масштабируемости в корпоративных приложениях.

– Недостатки: Отсутствие децентрализации, что может не соответствовать требованиям проектов, предполагающих широкое участие неконтролируемого числа узлов.

Tezos (On-chain Governance):

– Преимущества: Механизмы управления, позволяющие автоматически обновлять протокол без хард-форков.

– Недостатки: Сложности, связанные с принятием изменений всеми участниками сети, могут замедлить процесс развития платформы.

Теперь рассмотрим, почему механизм консенсуса Proof of Authority (PoA) является предпочтительным для трансграничных финансовых операций:

– Эффективность и скорость: PoA обеспечивает быструю верификацию транзакций за счет использования доверенных узлов, что критически важно для операций в реальном времени.

– Низкая стоимость транзакций: PoA исключает необходимость в значительных вычислительных ресурсах, снижая стоимость транзакций по сравнению с PoW и PoS.

– Высокий уровень безопасности: при адекватной настройке и управлении списком доверенных узлов, PoA может обеспечить высокую защиту от атак.

– Гибкость и управляемость: PoA позволяет гибко управлять списком узлов, адаптируя сеть под специфические требования трансграничных операций [3].

Nonce в блокчейне – это уникальное число, используемое в процессе майнинга для нахождения хеша, соответствующего заданным условиям. В блокчейнах на основе Proof of Work (PoW), майнеры изменяют nonce и пересчитывают хеш до тех пор, пока не найдут подходящее значение, что позволяет добавить новый блок в цепочку. Nonce обеспечивает уникальность и безопасность транзакций [6].

Хеш - результат применения хэш-функции к набору данных, который преобразует входные данные произвольной длины в строку фиксированной длины. Этот хеш является уникальным и практически невозможно восстановить оригинальные данные из него. Хеши используются в криптовалюте для различных целей, таких как создание уникальных идентификаторов для блоков и транзакций, проверка целостности данных, а также обеспечение безопасности и подтверждение подлинности информации.

Блокчейн на базе PoA не требует использования nonce, что необходимо в PoW для генерации блоков. Это исключает ненужные вычисления, позволяя снизить энергопотребление и повысить общую производительность системы. Важно отметить, что PoW подвержен риску 51% атаки, в то время как PoA, благодаря своей структуре и механизмам защиты, значительно менее уязвим к внешним угрозам. В случае попытки взлома одной из узлов, происходит форк сети, и узлы автоматически рассинхронизируются, что предотвращает возможные внешние взломы.

Таким образом, Proof of Authority представляет собой оптимальное решение для системы трансграничных транзакций, обеспечивая высокую скорость обработки, низкую стоимость, повышенную безопасность и удобство управления.

Консенсус-алгоритмы играют ключевую роль в обеспечении безопасности и целостности блокчейна, определяя, каким образом новые блоки добавляются в цепочку и какие правила следует соблюдать для подтверждения транзакций.

В данной главе мы рассмотрим несколько основных консенсус-алгоритмов, а именно Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS) и Practical Byzantine Fault Tolerance (PBFT). Каждый из этих алгоритмов имеет свои особенности, преимущества и недостатки, которые необходимо учитывать при выборе подходящего для конкретного блокчейн-проекта.

Цель данной главы - провести сравнительный анализ этих консенсус-алгоритмов, выявить их основные характеристики, преимущества и недостатки, а также оценить их применимость в различных сценариях использования. Этот анализ поможет разработчикам и исследователям принять обоснованное решение при выборе консенсус-алгоритма для своего данной вкр, учитывая специфику задачи, требования к производительности, масштабируемости и безопасности.

1.3.1 Описание консенсус механизма Proof Of Work (PoW)

Proof of Work (PoW) – это один из наиболее известных и широко применяемых консенсус-механизмов в блокчейне, который был впервые предложен в биткойн-протоколе. Этот механизм был разработан для обеспечения безопасности и надежности сети путем подтверждения транзакций и создания новых блоков [2].

Принцип работы PoW основан на концепции "трудозатратности", где участники сети должны продемонстрировать выполнение определенной

вычислительной работы, чтобы добавить новый блок в цепочку. Этот процесс требует значительных вычислительных ресурсов и времени, что делает его затратным с точки зрения энергии и вычислительной мощности.

Ключевые особенности PoW:

1. Майнеры: Участники сети, называемые майнерами, конкурируют между собой за право создания нового блока. Майнеры решают криптографическую задачу, известную как "хеш-паззл", которая требует значительных вычислительных ресурсов для его решения.

2. Доказательство выполненной работы: после того как майнер найдет корректное решение криптографической задачи, он отправляет его на сеть. Другие участники могут легко проверить, что майнер действительно выполнил работу, проверив результат через простой алгоритм.

3. Сложность задачи: Сложность криптографической задачи (также известной как "сложность хеша" или "цель хеша") регулируется сетью с целью поддержания постоянной скорости создания блоков. Чем выше сложность задачи, тем больше вычислительных ресурсов необходимо для ее решения.

4. Скорость блокогенерации: Протокол PoW регулирует скорость создания новых блоков путем изменения сложности задачи. В среднем, блоки генерируются примерно раз в 10 минут в биткойн-сети.

Преимущества PoW:

– Децентрализация: PoW обеспечивает высокую степень децентрализации, поскольку любой участник сети может стать майнером, внося свой вклад в поддержание безопасности сети.

– Надежность: благодаря высоким затратам на вычислительные ресурсы для атаки сети, PoW считается одним из наиболее надежных консенсус-механизмов.

Недостатки PoW:

– Высокое энергопотребление: Процесс майнинга требует больших объемов энергии, что может привести к негативным экологическим последствиям.

– Склонность к централизации: С развитием специализированных устройств для майнинга (ASIC), сети PoW становятся более склонными к централизации в руках крупных игроков, обладающих большими вычислительными ресурсами.

В целом, PoW был первым успешным консенсус-механизмом в блокчейне, который продемонстрировал высокую степень безопасности и надежности. Однако его недостатки, включая высокое энергопотребление и склонность к централизации, побудили исследователей и разработчиков искать альтернативные консенсус-механизмы, такие как Proof of Stake (PoS) и другие [30].

1.3.2 Описание консенсус механизма Proof Of Stake (PoS)

Proof of Stake (PoS) – это альтернативный консенсус-механизм в блокчейне, который используется для подтверждения транзакций и создания новых блоков. В отличие от Proof of Work (PoW), где участники сети (майнеры) конкурируют за право создания блоков, в PoS новые блоки создаются на основе доли (стейка) владения криптовалютой [27].

Принцип работы PoS основан на следующих основных принципах:

1. Стейкинг (Staking): Участники сети, называемые стейкерами, блокируют определенное количество своей криптовалюты в смарт-контракте или специальном кошельке в обмен на право участвовать в процессе создания блоков и получения вознаграждения.

2. Выбор блокопроизводителя (Validator Selection): В PoS блокопроизводители (валидаторы) выбираются случайным образом среди стейкеров на

основе их доли в сети. Чем больше стейк у участника, тем выше вероятность того, что он будет выбран для создания блока.

3. Подписывание блоков (Block Signing): Выбранный валидатор создает новый блок, подписывает его и добавляет в цепочку блоков. Этот процесс требует намного меньше вычислительных ресурсов по сравнению с PoW, так как нет необходимости решать сложные криптографические задачи.

4. Подтверждение блоков (Block Confirmation): Другие участники сети проверяют подписанный блок на его корректность и валидность. Если блок проходит проверку, он добавляется в блокчейн, и валидатор получает вознаграждение за создание блока.

Преимущества PoS:

– Энергоэффективность: PoS потребляет значительно меньше энергии по сравнению с PoW, так как процесс создания блоков не требует вычислительных ресурсов для решения сложных математических задач.

– Децентрализация: PoS также обеспечивает высокую степень децентрализации, поскольку любой участник сети, имеющий достаточный стейк, может стать валидатором и участвовать в создании блоков.

– Низкая вероятность централизации: Поскольку выбор валидаторов осуществляется на основе стейка, а не вычислительной мощности, сети PoS менее подвержены централизации в руках крупных игроков.

Недостатки PoS:

– Проблема "Ничего в обороте" (Nothing at Stake): Некоторые критики утверждают, что PoS подвержен риску "ничего в обороте", когда стейкеры могут одновременно подписывать несколько ветвей блокчейна, что может нарушить его целостность.

– Сосуществование с большими владельцами: PoS также может привести к проблеме сосуществования с крупными владельцами криптовалюты, которые могут иметь большее влияние на сеть из-за своего большого стейка.

В целом, PoS представляет собой эффективный и энергоэффективный консенсус-механизм, который становится все более популярным в блокчейн-пространстве. Он обеспечивает высокую степень децентрализации и безопасности сети при сниженном энергопотреблении по сравнению с PoW.

1.3.3 Описание консенсус механизма Delegated Proof of Stake (DPoS)

Delegated Proof of Stake (DPoS) - это вариация консенсус-алгоритма Proof of Stake (PoS), который предлагает улучшенную масштабируемость и эффективность за счет делегирования прав на создание блоков отдельным участникам сети. В DPoS валидаторы (или делегаты) выбираются голосованием участников сети, что позволяет держателям токенов делегировать свои права на создание блоков другим лицам.

Основные принципы работы Delegated Proof of Stake:

1. Выбор делегатов (Validators Selection): В сети DPoS держатели токенов могут голосовать за кандидатов на роль валидаторов, которые будут создавать блоки и поддерживать работу сети. Число делегатов обычно ограничено, что повышает эффективность сети и улучшает ее масштабируемость.

2. Получение права на создание блоков (Block Production Rights): Делегаты, получившие большее количество голосов от держателей токенов, получают право на создание блоков в сети. Эти делегаты затем могут создавать и подписывать новые блоки в соответствии с протоколом сети.

3. Распределение вознаграждений (Reward Distribution): Делегаты, создавшие блоки, получают вознаграждение в виде токенов сети за свою работу. Эти вознаграждения затем распределяются между делегатами и теми, кто им доверяет и поддерживает их.

4. Проверка подписей (Signature Verification): Другие участники сети проверяют подписанные блоки на их корректность и валидность. Если блок проходит проверку, он добавляется в блокчейн, и делегаты получают вознаграждение за создание блока.

Преимущества Delegated Proof of Stake:

– Масштабируемость: DPoS обеспечивает высокую масштабируемость благодаря ограниченному числу делегатов, что позволяет сети обрабатывать большее количество транзакций в секунду по сравнению с PoW или обычным PoS.

– Эффективность: DPoS потребляет меньше энергии и вычислительных ресурсов, так как процесс создания блоков выполняется ограниченным числом делегатов, выбранных голосованием, в отличие от PoW, где каждый майнер конкурирует за право создания блока.

– Децентрализация и управление: DPoS предоставляет держателям токенов возможность участвовать в управлении сетью, голосуя за делегатов и влияя на решения, принимаемые в сети.

Недостатки Delegated Proof of Stake:

– Централизация ресурсов: DPoS может быть более подвержен централизации, поскольку делегаты, получившие большее количество голосов, могут контролировать значительную часть сети и принимать ключевые решения.

– Потенциальный риск атаки на 51%: В сети DPoS существует риск атаки на 51%, когда один или несколько делегатов получают достаточно большую долю голосов, чтобы контролировать более половины сети.

В целом, DPoS представляет собой эффективный и масштабируемый консенсус-механизм, который обеспечивает высокую производительность и децентрализацию в блокчейн-сетях. Однако необходимо учитывать потенциальные риски централизации и атаки на 51%, а также рассмотреть механизмы смягчения этих рисков при разработке и использовании сети DPoS.

1.3.4 Описание консенсус механизма Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) – это алгоритм консенсуса, разработанный для обеспечения безопасности и надежности в распределенных системах, подверженных вредоносным действиям или сбоям узлов. Он является одним из наиболее известных и широко применяемых алгоритмов в блокчейн-технологиях и распределенных системах.

Вот основные принципы работы алгоритма PBFT:

1. Лидер (Leader): В PBFT узлы в сети поочередно выступают в роли лидера, который инициирует процесс формирования блоков и рассылает их другим узлам. Лидер выбирается из числа узлов с помощью алгоритма выборов.

2. Предложение блоков (Block Proposals): Лидер формирует блок, содержащий транзакции, и рассылает его всем остальным узлам в сети.

3. Подтверждение блоков (Block Confirmation): когда узлы получают предложенный блок, они проводят проверку на его корректность и валидность. Если большинство узлов подтверждает блок, он считается окончательным и добавляется в блокчейн.

4. Консенсусное голосование (Consensus Voting): В PBFT узлы в сети голосуют за подтверждение блока. Если достигается кворум голосов, то блок считается подтвержденным и добавляется в блокчейн.

Преимущества алгоритма PBFT:

– Высокая скорость транзакций: PBFT обеспечивает высокую производительность благодаря параллельной обработке транзакций и относительно небольшому числу узлов, участвующих в консенсусе.

– Отказоустойчивость: Алгоритм PBFT способен обеспечить надежность и отказоустойчивость даже в случае отказа или вредоносных действий части узлов в сети.

– Финальность транзакций: после подтверждения блока большинством узлов он считается финальным и не может быть отменен или изменен.

Недостатки алгоритма PBFT:

– Централизация: В PBFT обычно существует определенное количество узлов, участвующих в процессе консенсуса, что может привести к централизации в системе.

– Требовательность к бесперебойной работе узлов: Для успешной работы алгоритма все узлы должны быть доступны и работоспособны. Отказ или сбой даже небольшого числа узлов может снизить производительность и надежность сети.

В целом, Practical Byzantine Fault Tolerance является эффективным алгоритмом консенсуса, который обеспечивает высокую производительность и отказоустойчивость в распределенных системах. Однако необходимо учитывать его требовательность к централизации и непрерывной работе узлов при проектировании и внедрении в блокчейн-системы.

1.3.5 Описание консенсус механизма Proof Of Authority (PoA)

Proof of Authority (PoA) – это алгоритм консенсуса, который используется для подтверждения блоков в блокчейне. В отличие от более распространенных алгоритмов, таких как Proof of Work (PoW) или Proof of Stake (PoS), где участники сети майнят блоки или ставят на них стейки, PoA опирается на авторитет определенных участников сети, называемых авторитетными нодами [14].

Вот основные принципы работы алгоритма Proof of Authority:

1. Авторитетные ноды (Authority Nodes): В сети PoA существует ограниченное количество авторитетных узлов, которые обладают правом на создание

новых блоков и подтверждение транзакций. Эти ноды часто выбираются на основе их репутации, надежности и доверия.

2. Подтверждение блоков (Block Confirmation): Авторитетные ноды формируют новые блоки, содержащие транзакции, и подписывают их своими приватными ключами. После этого они распространяют блоки по сети.

3. Проверка подписей (Signature Verification): Участники сети, получившие новый блок, проверяют подпись авторитетных нод и валидируют блок. Если подпись корректна и блок проходит проверку, он добавляется в блокчейн.

4. Децентрализация: хотя в PoA существует ограниченное число авторитетных нод, сеть может быть децентрализованной в том смысле, что участники сети могут быть размещены в разных географических точках и не зависят от одного центрального управления.

Преимущества алгоритма Proof of Authority:

– Высокая производительность: поскольку авторитетные ноды обладают правом на создание блоков, процесс консенсуса в PoA может быть быстрым и эффективным.

– Надежность: Авторитетные ноды обычно имеют высокий уровень доверия и репутации, что повышает надежность сети и защищает ее от вредоносных атак.

Недостатки алгоритма Proof of Authority:

– Централизация: поскольку авторитетные ноды выбираются централизованным образом и имеют контроль над созданием блоков, сеть может быть более централизованной по сравнению с другими алгоритмами консенсуса.

– Низкая стойкость к атакам 51%: если злоумышленник получит контроль над большинством авторитетных нод, он сможет контролировать сеть и вносить изменения в блокчейн [24].

В целом, Proof of Authority – это эффективный алгоритм консенсуса, который подходит для частных блокчейн-сетей или сетей, где важны скорость и надежность

транзакций. Однако следует учитывать его недостатки, такие как возможность централизации и низкая стойкость к атакам 51%.

1.4 Сравнение существующих блокчейн решений на PoA

Рассмотрим несколько известных блокчейн-решений, использующих PoA, и сравним их по основным характеристикам.

Ethereum (в тестовой сети Kovan). Ethereum – это одна из самых популярных и широко используемых блокчейн-платформ. В тестовой сети Kovan применяется алгоритм PoA для повышения скорости и снижения стоимости транзакций.

Преимущества:

– Быстрая обработка транзакций.

– Высокая масштабируемость.

– Широкая поддержка сообщества и разработчиков.

Недостатки:

– Не подходит для основного Ethereum-сети (которая использует PoW и PoS).

– Тестовая сеть, не предназначенная для коммерческих приложений.

VeChain. VeChain – блокчейн-платформа, ориентированная на бизнес-приложения и корпоративные решения. Она использует PoA для обеспечения высокой производительности и безопасности.

Преимущества:

– Высокая скорость транзакций.

– Низкие операционные издержки.

– Фокус на корпоративные приложения.

– Интеграция с IoT для отслеживания и управления цепочками поставок.

Недостатки:

– Центральность системы может вызвать вопросы доверия у пользователей.

– Ограниченная децентрализация по сравнению с другими блокчейнами.

POA Network. POA Network – это независимая блокчейн-платформа, специально разработанная для использования алгоритма PoA. Она ориентирована на создание и управление децентрализованными приложениями (dApps).

Преимущества:

– Высокая скорость и пропускная способность сети.

– Простота развертывания dApps.

– Низкие комиссии за транзакции.

Недостатки:

– Меньшая известность и принятие по сравнению с Ethereum.

– Ограниченное количество валидаторов может вызвать вопросы безопасности и устойчивости.

Xooa. Xooa предоставляет блокчейн-платформу с использованием PoA, ориентированную на разработчиков, чтобы упростить создание и развертывание блокчейн-приложений.

Преимущества:

– Простота использования и интеграции.

– Высокая скорость транзакций.

– Поддержка корпоративных приложений.

Недостатки:

– Ограниченная децентрализация.

– Низкая известность по сравнению с крупными блокчейн-платформами.

Каждое из рассмотренных блокчейн-решений на основе PoA имеет свои уникальные преимущества и недостатки. Ethereum (Kovan) и POA Network подходят для разработчиков, ищущих платформу для создания dApps с высокой производительностью

и низкими комиссиями. VeChain и Xooa предлагают решения, ориентированные на корпоративные приложения, с акцентом на высокую скорость транзакций и низкие операционные издержки. Выбор подходящей платформы зависит от конкретных потребностей и требований проекта, а также от баланса между децентрализацией, скоростью и безопасностью. Моё блокчейн-решение на PoA обеспечивает уникальные преимущества, которых нет у существующих решений. Оно предлагает более быструю обработку транзакций и повышенную пропускную способность благодаря оптимизированным алгоритмам консенсуса. Усовершенствованные механизмы синхронизации нод обеспечивают большее удобство пользования.

1.5 Решение вопросов масштабирования и безопасности

1. Масштабирование: В контексте выбора консенсус-механизма для блокчейн-системы критическое значение имеет способность обеспечения масштабируемости. Каждый из рассмотренных механизмов обладает своими преимуществами и ограничениями в этом отношении. Например, Proof of Work (PoW) хорошо масштабируется за счет распределенной природы майнинга, но сталкивается с проблемами энергопотребления и скорости транзакций. С другой стороны, Proof of Stake (PoS) и Delegated Proof of Stake (DPoS) обычно более эффективны с точки зрения энергопотребления и скорости, но могут столкнуться с проблемами централизации и безопасности. Решение вопроса масштабирования может включать в себя комбинацию различных механизмов консенсуса или применение масштабируемых технологий, таких как sidechains или state channels.

2. Безопасность: при выборе консенсус-механизма необходимо обеспечить высокий уровень безопасности блокчейн-сети. Каждый из рассмотренных механизмов обладает своими уникальными механизмами обеспечения безопасности. Например,

Proof of Work (PoW) обеспечивает высокий уровень защиты благодаря сложности вычислений, необходимых для создания новых блоков. Proof of Stake (PoS) также обеспечивает безопасность, используя стейки для голосования, а Delegated Proof of Stake (DPoS) доверяет валидацию блоков выбранным делегатам. Важно провести анализ рисков и принять решение, основанное на компромиссе между масштабируемостью и безопасностью, учитывая потенциальные атаки и уязвимости.

1.6 Сравнительный анализ и выводы

Сравнив различные консенсус-механизмы, можно сделать следующие выводы:

1. Proof of Work (PoW): Этот механизм обеспечивает высокий уровень безопасности и децентрализации, но сталкивается с проблемами масштабирования и энергопотребления.
2. Proof of Stake (PoS): PoS эффективен с точки зрения энергопотребления и скорости транзакций, но может иметь проблемы с централизацией и безопасностью.
3. Delegated Proof of Stake (DPoS): DPoS обеспечивает высокую производительность и масштабируемость, но может столкнуться с проблемами централизации и доверия к делегатам.
4. Practical Byzantine Fault Tolerance (PBFT): PBFT обеспечивает высокую производительность и безопасность, но требует доверия к узлам сети и может быть менее децентрализованным.
5. Proof of Authority (PoA): PoA обеспечивает высокую производительность и безопасность за счет ограниченного числа авторитетных узлов, но сталкивается с проблемой централизации.

В зависимости от конкретных требований к проекту и контекста его применения, выбор консенсус-механизма может варьироваться. Важно учитывать как

технические, так и организационные аспекты при принятии решения. В случае моей работы, я выбрал Proof of Authority (PoA), так как он лучше всего подходит под требования безопасности и производительности, необходимые для успешной работы нашего блокчейн-приложения.

1.7 Анализ потребностей и проблем, которые приложение должно решать

Данное приложение призвано решать ключевые вызовы в сфере международных финансовых операций. Основная проблема, на решение которой направлено приложение, – неэффективность и высокая стоимость традиционных методов международных платежей. Существующие системы часто обременены высокими комиссиями и длительными сроками обработки транзакций, что затрудняет оперативное и экономически выгодное выполнение трансграничных платежей.

Проблема недостаточной прозрачности и безопасности также стоит на повестке дня. В классических финансовых системах отсутствует должный уровень прозрачности операций и защиты данных, что порождает риски и недоверие среди участников финансовых операций.

Кроме того, существует проблема затяжного и обременительного процесса соблюдения регуляторных требований в рамках международных транзакций. Традиционные системы требуют многочисленных проверок и документации для соответствия регуляторным нормам, что увеличивает время обработки транзакций и ведет к дополнительным издержкам.

Таким образом, приложение должно обеспечивать:

– Снижение комиссий и минимизацию задержек в международных платежах;

– Повышение прозрачности и безопасности операций;

– Упрощение процесса соблюдения регуляторных требований.

Отдельное внимание заслуживает обеспечение доверия к узлам блокчейна при отсутствии механизма Proof of Work. Предлагается механизм, позволяющий каждому владельцу ноды модифицировать свой код таким образом, что при изменениях нода автоматически перестанет синхронизироваться с остальной частью сети, исключая риски недобросовестного поведения и повышая надежность системы.

Для успешной интеграции и функционирования блокчейна на территории России необходимо учесть соответствие местному законодательству, особенно в части требований KYC и анти-отмывания денег. Интеграция с уже существующими информационными системами и сервисами окажет существенное влияние на снижение трудностей при внедрении и повышение общей эффективности использования приложения.

Рассмотрение экономической целесообразности применения блокчейна на базе PoA включает анализ рыночной конкуренции, развитие информационных технологий и потенциальные санкции, что может стимулировать широкое принятие данной технологии как альтернативного решения для обхода ограничений и улучшения финансовых операций на международном уровне.

1.8 Анализ потенциальных уязвимостей и защита от них

Анализ потенциальных уязвимостей и защита от них играет ключевую роль в обеспечении безопасности и надежности блокчейна на основе Proof of Authority (PoA) для проведения трансграничных транзакций. В этой главе проводится детальный анализ потенциальных уязвимостей блокчейна и рассматриваются методы и меры для их предотвращения.

Основные уязвимости и меры защиты:

1. Атака 51%. В традиционных блокчейнах, таких как Bitcoin, атака 51% происходит, когда один участник или группа участников контролирует более половины вычислительной мощности сети, что позволяет им манипулировать подтверждением транзакций [13]. Меры защиты в PoA: В блокчейне на основе PoA, где доверие основано на предварительно утвержденном списке валидаторов, риск такой атаки существенно снижен. Вместе с тем, чтобы минимизировать риски, необходимо тщательное управление и регулярная проверка списка валидаторов, а также внедрение механизмов для автоматической рассинхронизации с сомнительными узлами [5].

2. Вредоносные атаки на смарт-контракты. Смарт-контракты могут содержать уязвимости в коде, которые могут быть эксплуатированы для манипулирования выполнением контракта или кражи средств. Меры защиты: Реализация многоуровневого процесса аудита кода смарт-контрактов, включая автоматические инструменты проверки и ручной аудит экспертами, а также механизмы быстрого отката транзакций в случае обнаружения аномалий.

3. DDoS-атаки (Distributed Denial of Service). Злоумышленники могут направить чрезмерное количество трафика на сеть, что приведет к перегрузке и невозможности обработки легитимных запросов. Меры защиты: Использование распределенной архитектуры сети, методов ограничения скорости запросов, а также применение специализированных сетевых решений для защиты от DDoS-атак.

Дополнительные меры защиты:

– Шифрование данных: необходимо использовать современные алгоритмы шифрования для защиты данных, передаваемых в сети, и хранящихся в блокчейне, включая личные ключи и информацию о транзакциях.

– Механизмы аутентификации и авторизации: Внедрение многофакторной аутентификации и строгих политик доступа для обеспечения того, чтобы только авторизованные пользователи могли выполнять операции.

– Аудит и мониторинг: Разработка систем аудита и мониторинга для отслеживания активности в сети и оперативного реагирования на подозрительные действия.

– Регулярные обновления и патчи: Обеспечение своевременного обновления программного обеспечения и аппаратных компонентов для устранения известных уязвимостей.

– Обучение пользователей: Проведение регулярных тренингов для пользователей с целью повышения осведомленности о потенциальных угрозах и методах их предотвращения.

Эффективная реализация этих мер предосторожности способна обеспечить высокий уровень безопасности и надежности блокчейн-системы на основе PoA, устраняя основные уязвимости и предотвращая потенциальные атаки.

1.9 Обзор последних тенденций и инноваций в области блокчейн-технологий

Сегодняшняя эра цифровых технологий непрерывно предоставляет новые возможности для роста и инноваций. Особое внимание заслуживают последние тенденции в блокчейн-технологиях, которые представляют собой катализатор изменений во многих отраслях, начиная от финансов и заканчивая здравоохранением. Цель данной главы - обсудить ключевые инновации и тенденции в блокчейне, особенно в контексте использования механизма Proof of Authority (PoA) для трансграничных транзакций.

Концепция DeFi революционизировала традиционные финансовые услуги, предлагая децентрализованные альтернативы почти каждому финансовому продукту. В рамках DeFi пользователи могут брать и давать взаймы, торговать криптовалютами и получать доход от стейкинга без посредничества традиционных финансовых

институтов. Инновации в DeFi, такие как автоматизированные рыночные механизмы и смарт-контракты, создали основу для новых финансовых моделей [28].

Технология блокчейна постоянно развивается, особенно в области механизмов достижения консенсуса. Важным нововведением является механизм Proof of Authority (PoA), который предоставляет преимущества в виде скорости и энергоэффективности, важных для обработки межграничных транзакций. По сравнению с более ранними системами, такими как Proof of Work (PoW), PoA позволяет немногим выбранным узлам подтверждать транзакции, что делает процесс быстрее и менее ресурсоемким [29].

Интеграция блокчейна с передовыми технологиями, такими как искусственный интеллект (ИИ), интернет вещей (IoT) и облачные вычисления, открывает новые возможности для инноваций. Примеры такой интеграции включают улучшение безопасности данных, управление цепочками поставок и оптимизацию различных бизнес-процессов, повышая тем самым эффективность и прозрачность операций.

Блокчейн находит применение во множестве секторов за пределами финансов, включая здравоохранение, где он помогает в управлении медицинскими данными, и в логистике, где способствует прозрачности и эффективности цепочек поставок. Такие инновации не только улучшают текущие процессы, но и открывают двери для создания новых бизнес-моделей [23].

Одной из ключевых проблем для блокчейна остается его потребление энергии, особенно в сетях, использующих PoW. Однако новые подходы, такие как PoA, значительно снижают энергопотребление, что делает блокчейн более экологически устойчивым. Продолжающиеся исследования и инновации направлены на дальнейшее снижение воздействия блокчейна на окружающую среду [11].

В заключение, последние тенденции и инновации в блокчейне продолжают расширять его возможности и сферы применения. Особое внимание заслуживает PoA, который, благодаря своим уникальным свойствам, открывает новые горизонты для обмена активами. Наблюдая за этими разработками, можно предвидеть значительное влияние блокчейна на будущее цифровой экономики.

Я рассмотрел различные аспекты разработки блокчейна на основе механизма Proof of Authority (PoA) для системы трансграничных транзакций. Проведенный анализ выявил, что такой блокчейн способен эффективно решать ряд ключевых проблем, среди которых - высокие комиссии и задержки в международных платежах, неэффективность традиционных финансовых систем, недостаточная прозрачность и безопасность операций, а также сложности соблюдения регуляторных требований.

Изучение существующих аналогов блокчейнов позволило выявить различные механизмы консенсуса, такие как Proof of Work (PoW), Proof of Stake (PoS) и Consensus Protocol. Однако, по результатам анализа, было установлено, что механизм PoA представляет собой наиболее подходящий вариант для решения задач системы трансграничных транзакций, благодаря своей эффективности, низкой стоимости проведения транзакций, высокому уровню безопасности и гибкости управления.

Кроме того, в рамках анализа были выявлены дополнительные особенности предлагаемой системы, такие как возможность изменения кода узлов и автоматическая рассинхронизация сети в случае попытки взлома, что дополнительно повышает ее безопасность и надежность.

Таким образом, на основании проведенного анализа можно сделать вывод о целесообразности и перспективности использования блокчейна на основе механизма PoA для реализации системы финансовых операций.

Раздел 2. Разработка информационной системы

						НИГАСУ-09.03.02 – 2024		
Изм.	Кол.уч	Лист	№ док.	Подпись	Дат			
Зав. Каф.		Кислицын Д.И.				Раздел 2 Разработка информационной системы	Стадия	Лист	Листов
Руковод.		Кислицын Д.И.					ВКР	36	50
Разраб.		Самедов Н.Ю.					Каф. ИСТ гр. ИС-29		
Консульт.		Кислицын Д.И.							
Н. Контр.		Суханова Н.Т.							

Глава 2. Разработка информационной системы

2.1 Выбор среды разработки

Выбор среды разработки является критически важным этапом в разработке любой технологии, включая блокчейн. Причины выбора Python для данной вкр обусловлены несколькими ключевыми факторами, которые делают его особенно подходящим для разработки сложных систем, таких как блокчейн [7]:

1. Простота и читаемость кода: Python известен своим чистым и легко читаемым синтаксисом, что упрощает разработку, отладку и последующее обслуживание кода. Это особенно важно в проектах, где участвует множество разработчиков и требуется поддержка и обновление кода [15].

2. Богатая экосистема: Python поддерживается обширной библиотекой библиотек и фреймворков, которые могут значительно ускорить разработку и предоставляют инструменты для почти любой задачи – от веб-разработки до работы с базами данных и машинным обучением.

3. Универсальность: Python поддерживается на всех основных платформах, что делает его универсальным выбором для разработки приложений, которые должны работать в различных средах.

4. Поддержка асинхронности и многопоточности: Python обладает отличной поддержкой асинхронного и многопоточного программирования, что критически важно для высокопроизводительных приложений, таких как транзакционные системы на блокчейне.

5. Активное сообщество: Наличие широкого и активного сообщества Python обеспечивает доступ к обширной базе знаний, форумам поддержки и обучающим материалам, что значительно упрощает решение возникающих проблем и ускоряет процесс разработки.

API (англ. Application Programming Interface – программный интерфейс приложения) – это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными [12].

Выбор фреймворка между FastAPI и Flask.

Для разработки веб-интерфейса к блокчейну были рассмотрены два популярных Python фреймворка – FastAPI и Flask. Вот краткое сравнение их основных характеристик [20]:

FastAPI:

– Производительность: благодаря асинхронному программированию FastAPI предлагает высокую производительность.

– Типизация данных и валидация: FastAPI поддерживает автоматическую валидацию данных и предоставляет возможности современной типизации Python, что улучшает безопасность и надежность приложения.

– Документация: FastAPI автоматически генерирует документацию, что упрощает разработку и поддержку API.

– Поддержка стандартов OpenAPI и Swagger: это облегчает интеграцию и использование в различных средах.

Flask:

– Гибкость: Flask предлагает большую гибкость и простоту для разработки простых веб-приложений.

– Меньшая поддержка асинхронности: Flask имеет ограниченную поддержку асинхронного программирования по сравнению с FastAPI. Исходя из анализа, FastAPI является предпочтительным выбором для данной блокчейна в этой вкр из-за его высокой производительности, поддержки асинхронности и улучшенной безопасности, что особенно важно для систем, обрабатывающих транзакции и финансовые данные.

2.1.1 Изучение возможностей фреймворка FastAPI

Фреймворк (с англ. framework – «каркас, структура») – заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код. Он задает структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта.

FastAPI – это современный и эффективный веб-фреймворк на языке Python, который предоставляет разработчикам мощные инструменты для создания веб-приложений. Он позволяет быстро и легко разрабатывать API, обеспечивая высокую производительность за счет использования современных технологий.

Для разработки веб интерфейса к блокчейну FastAPI может быть отличным выбором, так как он обладает рядом полезных возможностей. Например, благодаря асинхронной поддержке можно создать быстродействующие веб-приложение, способное обрабатывать большое количество запросов одновременно. Кроме того, FastAPI предоставляет интуитивный и простой в использовании интерфейс для описания схем данных API с помощью Pydantic, что облегчает разработку и поддержку кода.

Еще одним преимуществом FastAPI является возможность автоматической генерации документации API на основе аннотаций в коде, что значительно упрощает процесс документирования вашего приложения. Кроме того, FastAPI обладает встроенной поддержкой валидации данных, автоматической сериализации и десериализации запросов и многими другими полезными функциями.

Использование FastAPI для блокчейн-приложения позволит не только создать эффективное и надежное веб-приложение, но и изучить современные подходы к разработке API, работе с асинхронным кодом и использованию современных технологий веб-разработки.

2.1.2 Преимущества фреймворка FastAPI

Одним из ключевых достоинств FastAPI является его высокая производительность. Благодаря применению современных технологий асинхронного программирования, данный фреймворк обеспечивает высокую скорость работы и эффективное использование ресурсов, что делает его отличным выбором для создания высоконагруженных веб-приложений.

Кроме того, FastAPI предлагает удобный и интуитивно понятный интерфейс для разработчиков. Он использует аннотации Python для описания эндпоинтов и схем данных API, что делает процесс создания веб-приложений более простым и эффективным. Благодаря автоматической генерации интерактивной документации на основе аннотаций в коде, разработчики могут быстро и легко создавать, и поддерживать документацию без дополнительных усилий.

Еще одним преимуществом FastAPI является встроенная поддержка валидации данных, что обеспечивает безопасность API и помогает избежать ошибок ввода. Благодаря поддержке различных форматов данных, таких как JSON, формы HTML, файлы и другие, FastAPI является универсальным инструментом для разработки разнообразных веб-приложений, обеспечивая разработчикам широкие возможности при создании функциональных и эффективных веб-сервисов.

2.1.3 Недостатки фреймворка FastAPI

FastAPI, несмотря на свои многочисленные достоинства, также имеет некоторые недостатки. В частности, такой фреймворк может показаться избыточным для небольших проектов или приложений, требующих минималистического подхода. Кроме того, изучение особенностей и функционала FastAPI может потребовать времени и

усилий, особенно для начинающих программистов, не знакомых с асинхронным программированием или аннотациями в Python.

Еще одним недостатком FastAPI является отсутствие широкого сообщества и большого количества сторонних библиотек и плагинов, как у более узнаваемых и распространенных фреймворков. Это может затруднить поддержку и развитие проекта в долгосрочной перспективе, особенно если возникнет необходимость в специфической функциональности, которая не предусмотрена по умолчанию в FastAPI.

Кроме того, из-за своей особенности использования асинхронного программирования FastAPI может создавать некоторые сложности в проектировании архитектуры приложения. Это может потребовать дополнительных усилий и времени для овладения соответствующими концепциями и методиками разработки, что также считается недостатком данного фреймворка.

2.2 Проектирование приложения

2.2.1 Определение требований к приложению

Определение требований к приложению блокчейна на основе механизма Proof of Authority (PoA) для проведения трансграничных транзакций включает в себя ряд ключевых аспектов, которые необходимо учесть для успешной разработки и функционирования системы. Эти требования включают в себя:

1. Эффективность трансграничных платежей: Приложение должно обеспечивать быструю и надежную передачу средств между участниками из разных стран с минимальными комиссиями и задержками.

2. Прозрачность и безопасность операций: Система должна обеспечивать высокий уровень прозрачности и безопасности всех транзакций, чтобы участники могли доверять целостности и результатам проведенных операций.

3. Соответствие регуляторным требованиям: Приложение должно быть способно соблюдать регуляторные нормы и требования, действующие в различных странах, чтобы предотвратить возможные конфликты и проблемы с законодательством.

4. Гибкость и масштабируемость: Система должна быть гибкой и легко масштабируемой, чтобы адаптироваться к изменяющимся потребностям рынка и обеспечивать возможность обработки большого объема транзакций.

5. Управление доступом и безопасность: Приложение должно обеспечивать строгий контроль доступа к системе и конфиденциальность информации, защищая данные пользователей от несанкционированного доступа и взлома.

6. Легкость использования и интеграции: Система должна быть удобной в использовании как для конечных пользователей, так и для предприятий, а также легко интегрироваться с существующими финансовыми и технологическими системами.

7. Автоматизация процессов: Приложение должно обеспечивать автоматизацию ключевых процессов, связанных с проведением трансграничных транзакций, чтобы уменьшить ручную работу и повысить эффективность операций.

Учитывая эти требования, разработка и реализация приложения блокчейна на основе механизма PoA должны быть направлены на обеспечение высокого качества обслуживания и удовлетворение потребностей всех заинтересованных сторон.

2.2.2 Анализ потенциальных уязвимостей

Создание такого блокчейна требует отслеживания и устранения уязвимостей.

Примеры:

1. Уязвимости в программном обеспечении (ПО): Уязвимости в смарт-контрактах: Недавние инциденты показали, что смарт-контракты могут содержать ошибки, которые могут привести к утрате средств или даже к полной уязвимости системы блокчейна. Примером является уязвимость "Reentrancy", когда атакующая сторона многократно вызывает внешние контракты перед завершением выполнения основного контракта. Проверка смарт-контрактов на уязвимости и использование проверенных шаблонов контрактов может помочь предотвратить такие атаки.

2. Уязвимости операционной системы (ОС): Атаки на уровне операционной системы: Вредоносные программы, такие как вредоносные скрипты, руткиты и т. д., могут использоваться для компрометации узлов блокчейна и веб-приложений. Важно регулярно обновлять операционные системы, использовать антивирусное программное обеспечение и следить за активностью на серверах для предотвращения таких атак.

3. Уязвимости веб-приложений: Атаки на сторону клиента (Client-Side Attacks): XSS (межсайтовый скриптинг), CSRF (межсайтовая подделка запросов), а также атаки на сторону клиента могут быть использованы для компрометации веб-приложений, включая те, которые взаимодействуют с блокчейном. Для предотвращения таких атак важно использовать санкционированные методы аутентификации и авторизации, а также правильно настраивать заголовки безопасности HTTP.

4. Методы сертификации: SSL/TLS сертификаты: Веб-приложения, особенно те, которые взаимодействуют с блокчейном и содержат чувствительные данные, должны использовать SSL/TLS сертификаты для обеспечения шифрования трафика и проверки подлинности. Важно правильно настроить серверы, чтобы избежать возможных уязвимостей, связанных с неправильной конфигурацией SSL/TLS.

5. Валидаторы и ноды: DDoS атаки на узлы: Ноды блокчейна и валидаторы могут стать целью DDoS-атак, что приведет к отказу в обслуживании и потере доступности для других участников сети. Для предотвращения таких атак можно использовать средства защиты от DDoS, такие как фильтрация трафика и распределенные системы защиты.

6. Избежание ошибок 0 дня: Регулярное обновление: Один из наиболее важных методов предотвращения ошибок 0 дня – это регулярное обновление программного обеспечения, включая операционные системы, приложения и зависимости. Постоянное отслеживание уязвимостей и применение патчей является необходимым.

2.2.3 Разработка структуры приложения и его компонентов

Основные компоненты приложения и их взаимосвязи:

1. Chain. Этот компонент содержит базу данных блоков и транзакций. Здесь осуществляется генерация мнемоник, создание и проверка подписей, а также получение и создание транзакций и блоков. Взаимосвязь с другими компонентами: Chain взаимодействует с компонентом Crypto для шифрования мнемонических фраз и выполнения математических операций, необходимых для создания и проверки подписей. Также Chain обеспечивает данные, которые могут быть использованы Node для проверки и добавления новых блоков в цепочку.

2. Crypto. Этот модуль отвечает за создание транзакций, составление транзакций и компоновку байтов, а также за шифрование мнемонических фраз и выполнение математических операций. Взаимосвязь с другими компонентами: Crypto используется Chain для создания и проверки подписей транзакций, а также для шифрования мнемонических фраз. Кроме того, он может взаимодействовать с Node для создания новых транзакций и блоков.

3. Node. Этот модуль представляет собой ноду блокчейна, которая выпускает и валидирует блоки. У ноды есть свой API, который используется другими компонентами приложения. Взаимосвязь с другими компонентами: Node использует данные из Chain для проверки целостности блоков и транзакций, а также может

использовать Crypto для создания новых блоков. Его API доступен компоненту Web для взаимодействия с пользователем.

4. Web (Веб-интерфейс). Этот модуль предоставляет графический интерфейс пользователям через веб-сайт. Включает в себя HTML и JS скрипты для взаимодействия с пользователем. Взаимосвязь с другими компонентами: Web взаимодействует с Node через его API для получения данных о блоках и транзакциях, а также для отправки новых транзакций.

Каждый из этих компонентов играет ключевую роль в функционировании приложения, а их взаимосвязь обеспечивает согласованную работу системы в целом (Рисунок 2.1). Chain служит основным хранилищем данных блокчейна, предоставляя информацию о блоках и транзакциях для других компонентов. Crypto обеспечивает безопасность и целостность данных благодаря криптографическим методам, а также осуществляет создание и проверку транзакций. Node является ядром сети, отвечая за создание новых блоков и проверку их на валидность, а также предоставляет API для взаимодействия с другими компонентами. Web обеспечивает пользовательский интерфейс, позволяя пользователям взаимодействовать с блокчейном через веб-интерфейс.

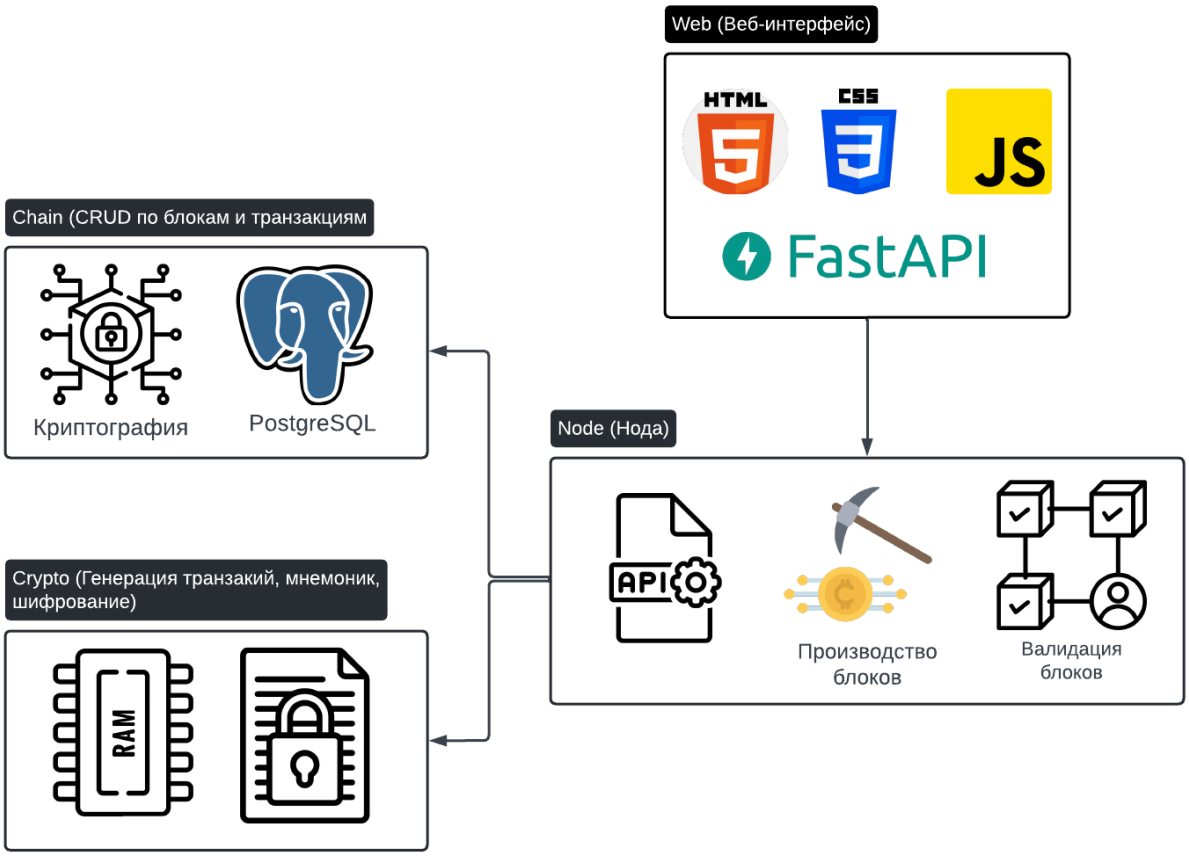

Рисунок 2.1 – Архитектура модулей

Таким образом, компоненты взаимодействуют между собой (Рисунок 2.1), обмениваясь данными и обеспечивая работоспособность и эффективность всей системы блокчейна на механизме PoA для проведения трансграничных транзакций.

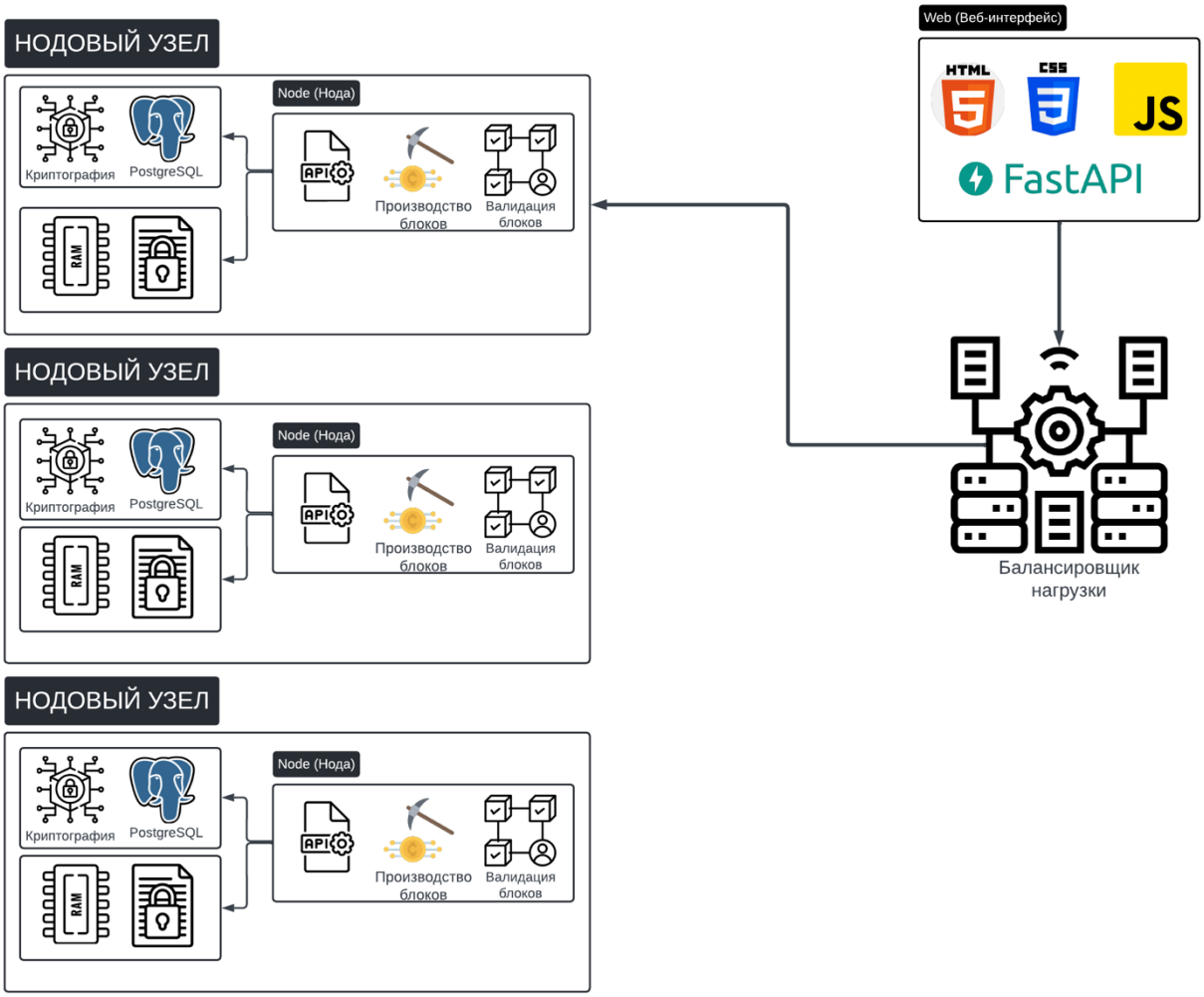

Рисунок 2.1 - Архитектура модулей с балансировщиком

2.2.4 Разработка интерфейса пользователя

В данном разделе рассмотрена разработка интерфейса пользователя для веб версии блокчейна на PoA.

Для разработки интерфейса пользователя блокчейн-приложения с учетом предложенных компонентов следует уделить особое внимание каждому из них.

1. Окно аутентификации (Рисунок 2.2):

– Форма ввода мнемоники для аутентификации пользователя.

– Возможность генерации мнемоники для нового пользователя.

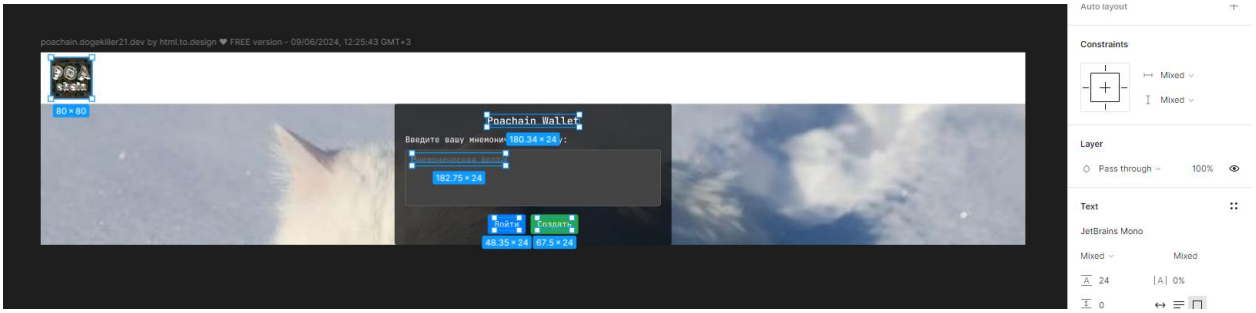

Рисунок 2.2 - Макет окна аутентификации

2. Кошелек (Рисунок 2.3):

– Отображение баланса пользователя и его адрес в удобном для восприятия виде.

– Вкладки для отправки криптовалюты и создания подписей, где пользователь сможет вводить необходимые данные для совершения операции.

– Вкладки для просмотра транзакций, где отображаются все отправленные и полученные транзакции пользователя с возможностью сортировки и фильтрации.

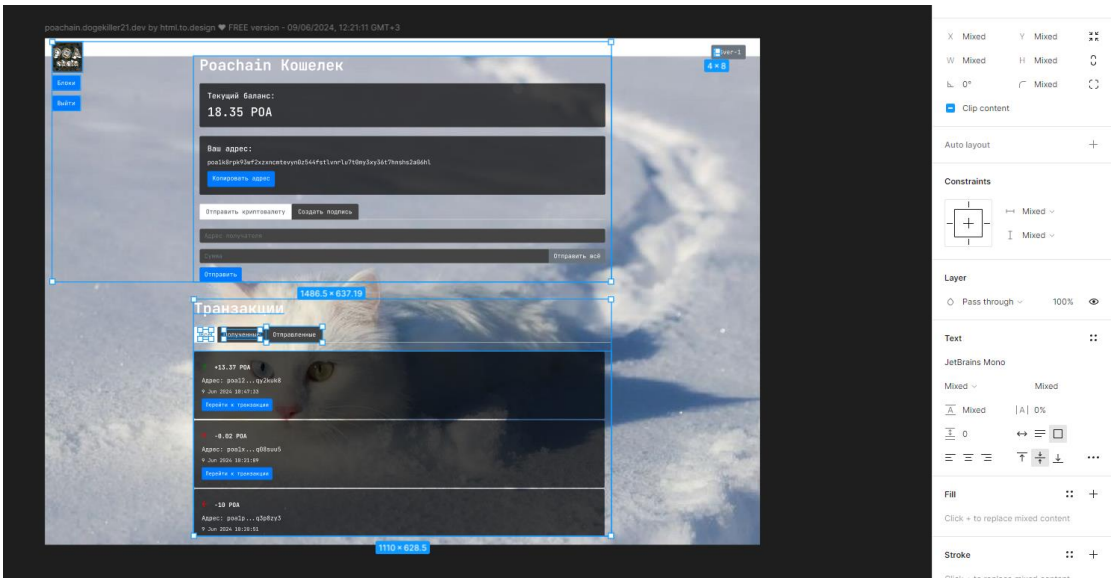

Рисунок 2.3 - Макет страницы кошелька

3. Окно просмотра транзакций (Рисунок 2.4):

– Отображение детальной информации о каждой транзакции, включая ее характеристики и данные блока, в котором она была включена.

– Возможность просмотра связанных транзакций для лучшего понимания контекста операции.

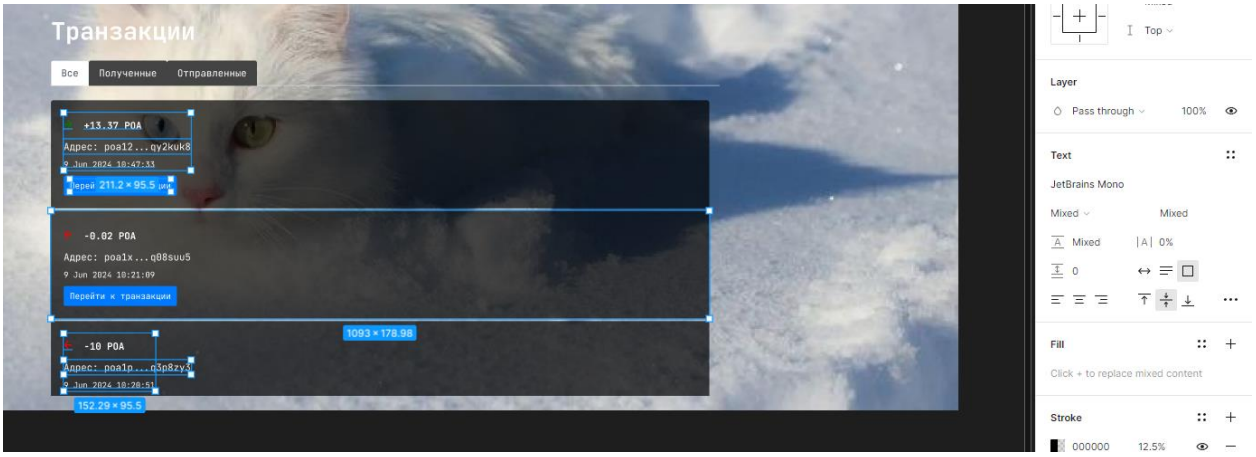

Рисунок 2.4 - Макет интерфейса просмотра транзакций

4. Окно просмотра блока (Рисунок 2.5):

– Отображение всех данных блока, включая список транзакций, включенных в него.

– Возможность просмотра подробной информации о каждой транзакции в блоке.

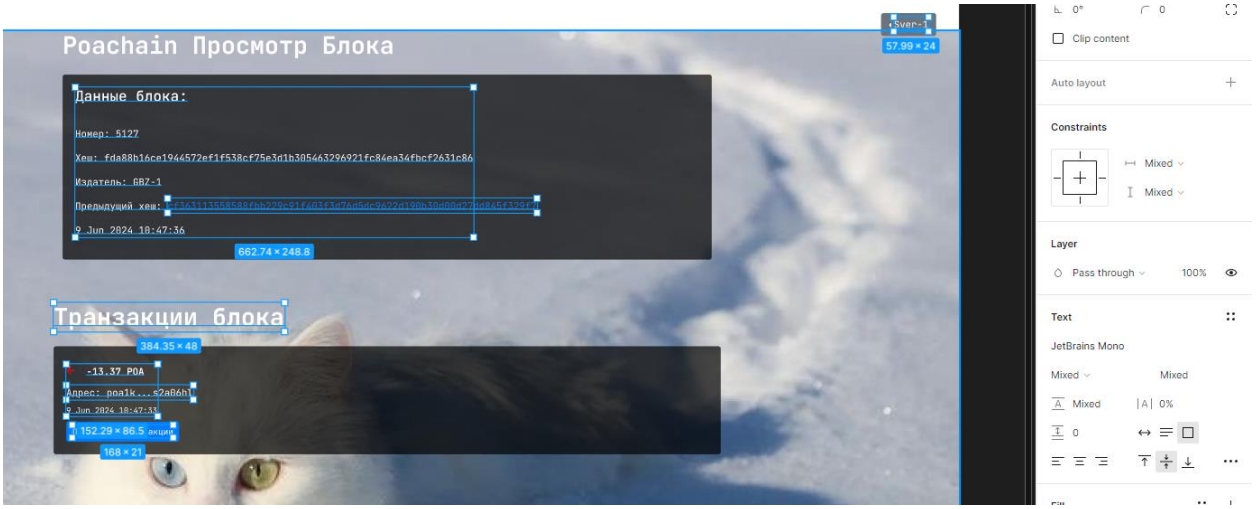

Рисунок 2.5 - Макет просмотрщика блоков

5. Окно обзорщика блоков (Рисунок 2.6):

– Визуализированный процесс генерации блоков в реальном времени.

– Поиск конкретного блока по его номеру или транзакции по ее хешу для быстрого доступа к нужной информации.

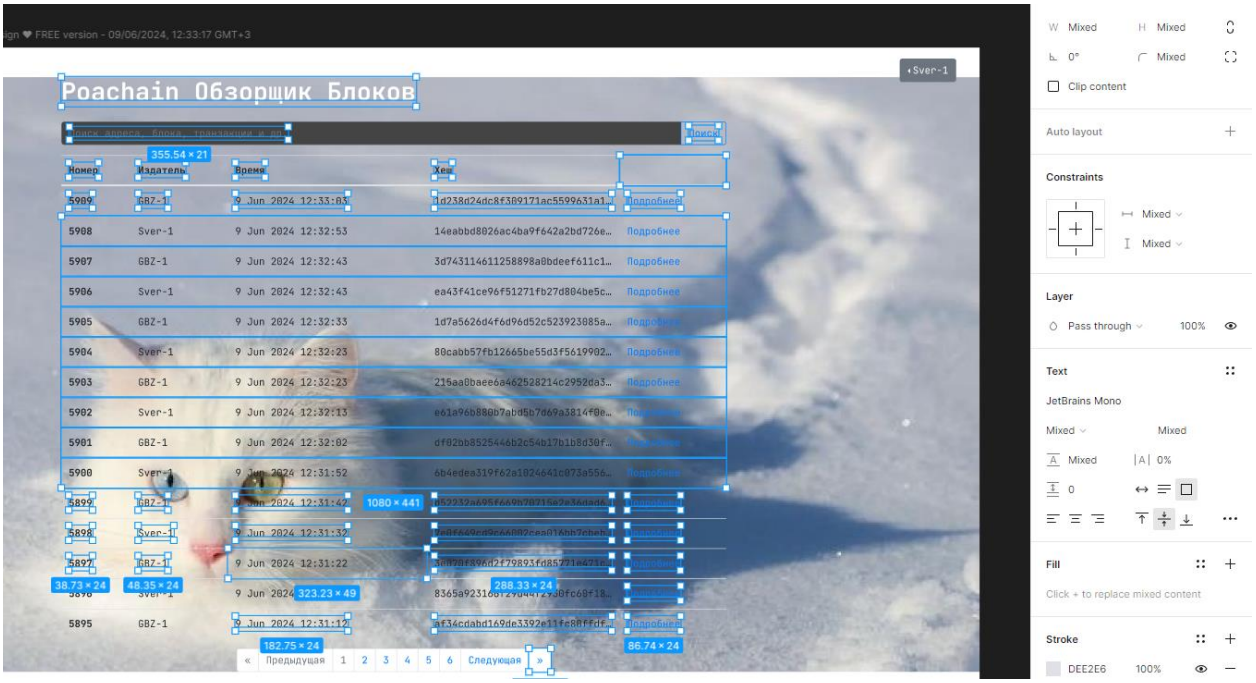

Рисунок 2.6 - Макет обзорщика блоков

Помимо этого, обеспечена плавная навигация между различными окнами и возможность быстрого доступа к основной функциональности приложения. Уделено внимание дизайну интерфейса, он сделан его интуитивно понятным и привлекательным для пользователя.

2.2.5 Описание архитектуры приложения

Архитектура приложения для системы (Рисунок 2.7) трансграничных транзакций в виде блокчейна на механизме Proof of Authority (PoA) включает в себя следующие компоненты:

1. Клиентский веб-интерфейс:

– Веб-приложение, которое предоставляет пользовательский интерфейс для взаимодействия с блокчейном.

– Реализует функциональность аутентификации пользователей, просмотра баланса и истории транзакций, отправки транзакций и просмотра информации о блоках.

– Использует HTML, CSS и JavaScript для создания пользовательского интерфейса и взаимодействия с API нод.

2. Субд Postgresql:

СУБД (Система управления базами данных) – это инструментальная оболочка пользователя, а ввиду того, что такая среда ориентирована на немедленное удовлетворение запросов пользователя – это всегда система интерпретатор.

– Система управления базами данных, используемая для хранения информации о блоках, транзакциях и других данных блокчейна.

– Хранит информацию о пользовательских аккаунтах, балансах, истории транзакций и другие сведения, необходимые для функционирования приложения.

3. Ноды:

– Узлы блокчейна, отвечающие за создание, проверку и хранение блоков.

– Каждая нода в сети обрабатывает транзакции, создает новые блоки и участвует в процессе достижения консенсуса.

– Ноды взаимодействуют между собой для распространения блоков и поддержания целостности сети.

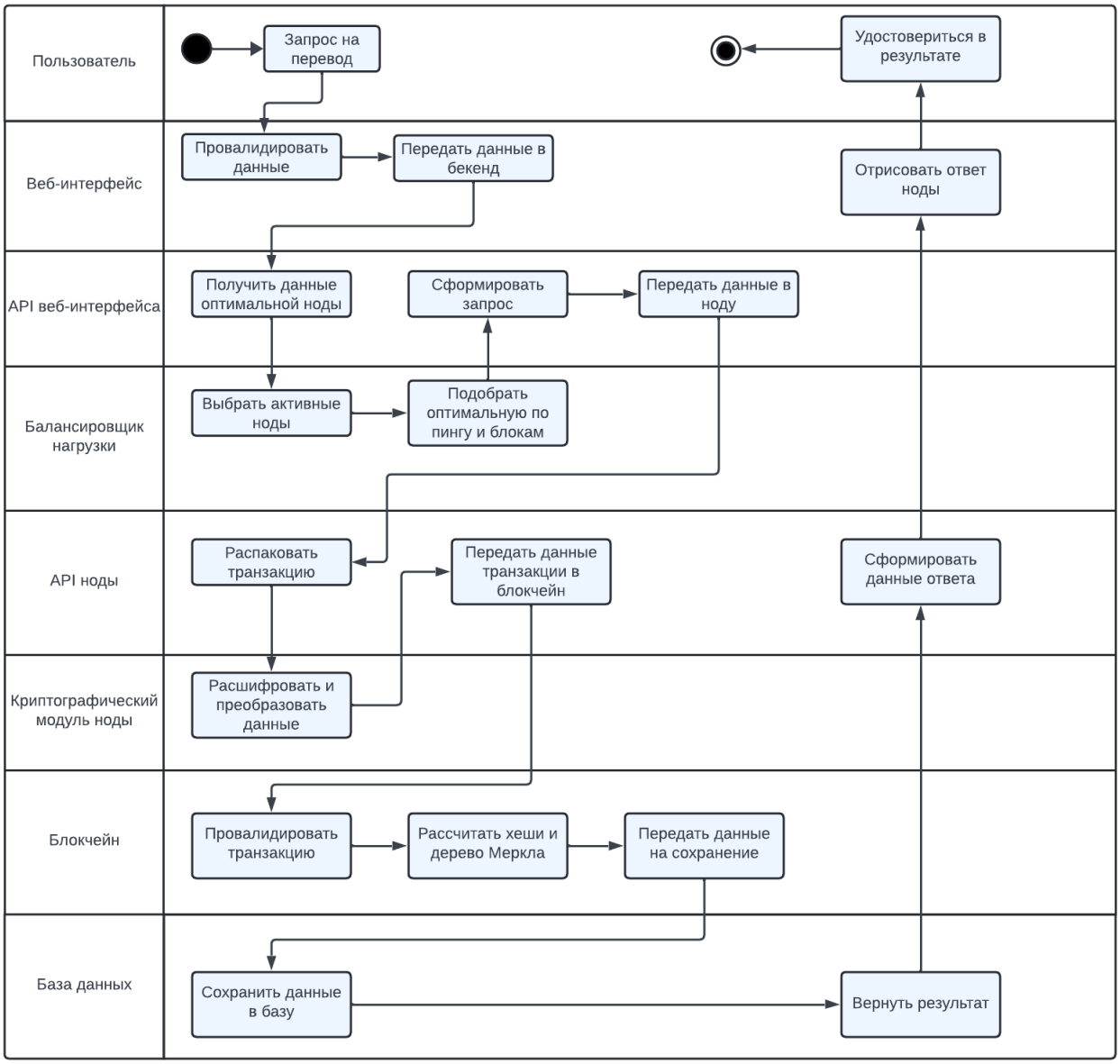

Рисунок 2.7 – Путь запроса по архитектуре

Архитектура приложения предполагает, что клиентский веб-интерфейс взаимодействует с нодами блокчейна через их API для получения информации о блоках, транзакциях и выполнения операций с аккаунтами. Ноды в свою очередь взаимодействуют друг с другом для распространения блоков и поддержания актуального состояния блокчейна. Данные о блоках, транзакциях и аккаунтах хранятся в базе данных Postgresql для обеспечения удобного доступа и быстрой обработки информации.

2.2.6 Выводы по проектированию приложения

Анализируя вышеописанные модули и архитектуру приложения, можно сделать следующие выводы.

Клиентский веб-интерфейс играет ключевую роль в обеспечении пользовательского опыта. Он предоставляет удобный доступ к функционалу блокчейна, включая просмотр информации о транзакциях и блоках, а также выполнение операций с аккаунтами.

Использование системы управления базами данных Postgresql обеспечивает надежное хранение данных блокчейна и обеспечивает быстрый доступ к ним. Это важно для обеспечения эффективной работы приложения и обеспечения целостности данных.

Ноды блокчейна играют ключевую роль в обеспечении безопасности и целостности сети. Механизм PoA обеспечивает высокую эффективность и надежность работы нод, что является важным аспектом для системы трансграничных транзакций.

В целом, архитектура приложения объединяет эти компоненты в единую систему, обеспечивая высокий уровень безопасности, прозрачности и эффективности для пользователей.

2.3 Разработка приложения

2.3.1 Создание хранилища для блокчейна и его описание

В данной главе проведём анализ различных видов хранилищ данных и их применимости для блокчейна на механизме Proof of Authority (PoA). В этой главе сравним следующие виды хранилищ:

1. Реляционные базы данных:

Примеры: PostgreSQL, MySQL, Oracle.

Плюсы: обеспечивают структурированное хранение данных с возможностью использования сложных запросов. Обеспечивают надежность и целостность данных.

Минусы: могут быть менее масштабируемыми по сравнению с NoSQL базами данных. Требуют более тщательного проектирования схемы данных.

2. NoSQL базы данных

Примеры: MongoDB, Cassandra, Redis.

Плюсы: обеспечивают гибкое хранение данных без жесткой схемы. Обладают высокой масштабируемостью и способностью обрабатывать большие объемы данных.

Минусы: могут быть менее эффективными при выполнении сложных запросов, требуют дополнительной работы по обеспечению согласованности данных.

3. Файловые системы:

Примеры: Amazon S3, Google Cloud Storage, IPFS.

Плюсы: Простота использования, низкая стоимость хранения данных. Поддерживают различные типы данных, включая файлы и документы.

Минусы: могут быть менее эффективными при выполнении операций поиска и обработки данных. Могут не обеспечивать такой же уровень надежности и безопасности, как реляционные базы данных.

PostgreSQL является мощной реляционной базой данных, которая обладает рядом характеристик, делающих ее подходящим выбором для использования в блокчейне на механизме Proof of Authority (PoA):

1. Структурированное хранение данных: PostgreSQL позволяет организовать данные в виде таблиц с жесткой схемой, что особенно полезно для блокчейна, где необходимо хранить информацию о блоках, транзакциях, аккаунтах и других аспектах сети.

2. Транзакционная безопасность: PostgreSQL обеспечивает ACID-свойства транзакций (Atomicity, Consistency, Isolation, Durability), что гарантирует целостность

данных и надежность операций в блокчейне. Это особенно важно для обеспечения согласованности данных в децентрализованной среде.

3. Поддержка сложных запросов: PostgreSQL предоставляет широкие возможности для выполнения сложных запросов и аналитики данных. Это позволяет эффективно работать с большим объемом информации, характерным для блокчейна.

4. Масштабируемость: PostgreSQL обладает возможностью масштабироваться как вертикально (путем увеличения ресурсов сервера), так и горизонтально (путем использования репликации и шардинга). Это позволяет блокчейну на PoA обрабатывать большие объемы данных и поддерживать высокую производительность.

5. Большое сообщество и поддержка: PostgreSQL является одной из самых популярных открытых реляционных баз данных, что обеспечивает широкую поддержку, обновления и интеграцию с другими инструментами и технологиями.

6. Открытый исходный код: PostgreSQL распространяется под лицензией открытого исходного кода, что делает его доступным для использования без затрат на лицензирование и обеспечивает прозрачность разработки и независимость от поставщика.

Все эти факторы делают PostgreSQL привлекательным выбором для хранения данных в блокчейне на PoA, обеспечивая надежность, производительность и гибкость в работе с данными.

Перейдем к описанию структуры базы данных (Рисунок 2.8). База данных будет состоять из нескольких таблиц, каждая из которых будет хранить определенный тип данных. Рассмотрим каждую таблицу подробнее.

1. Таблица «block»

Эта таблица содержит в себе цепочку блоков. Состоит из следующих полей:

– id: Первичный уникальный ключ блока.

– block_number: Числовой уникальный номер блока начиная с genesis блока

– block_hash: Хеш блока, созданный на основе его атрибутов (номера блока, дерева меркла, предыдущего блока, времени итд.) (текстовый формат)

– previous_hash: Хеш предыдущего блока (текстовый формат)

– merkle_root: Дерево меркла блока, построенное на основе последовательного хеширования транзакций блока, может быть null если транзакций в блоке нет (текстовый формат)

– authority_id: Опознавательный ключ ноды валидатора которая выпустила данный блок (текстовый формат)

– timestamp: Дата выпуска блока в микросекундах [17]

Дерево Меркла - это структура данных, используемая в криптографии и блокчейне для эффективной проверки целостности и подтверждения содержимого больших объемов данных. Оно строится путем последовательного хэширования пары узлов данных (обычно транзакций) до тех пор, пока не будет получено единое значение, называемое корневым хешем. Этот корневой хеш затем используется для проверки целостности всего дерева: если корневой хеш совпадает, значит, все данные в дереве неизменны и целостны. Деревья Меркла позволяют быстро и эффективно проверять целостность больших объемов данных, что делает их важным элементом в технологиях блокчейна и криптографии [10].

2. Таблица «transaction»

– id: Первичный уникальный ключ транзакции.

– sender_address: Адрес отправителя транзакции (текстовый формат)

– recipient_address: Адрес получателя транзакции (текстовый формат)

– amount: Числовая сумма отправленных активов. Хранится в базе в целочисленном виде, отображается с точностью 2 знака только визуально.

– timestamp: Дата совершения транзакции в микросекундах [18]

– transaction_hash: Хеш транзакции составленный на основе ее данных (отправителя, получателя, суммы, времени) [21]

– block_id: Ключ связанного блока в базе данных

– block_number: Номер блока. После подтверждения транзакции она попадает в блок, а после подтверждения блока в это поле вписывается его номер.

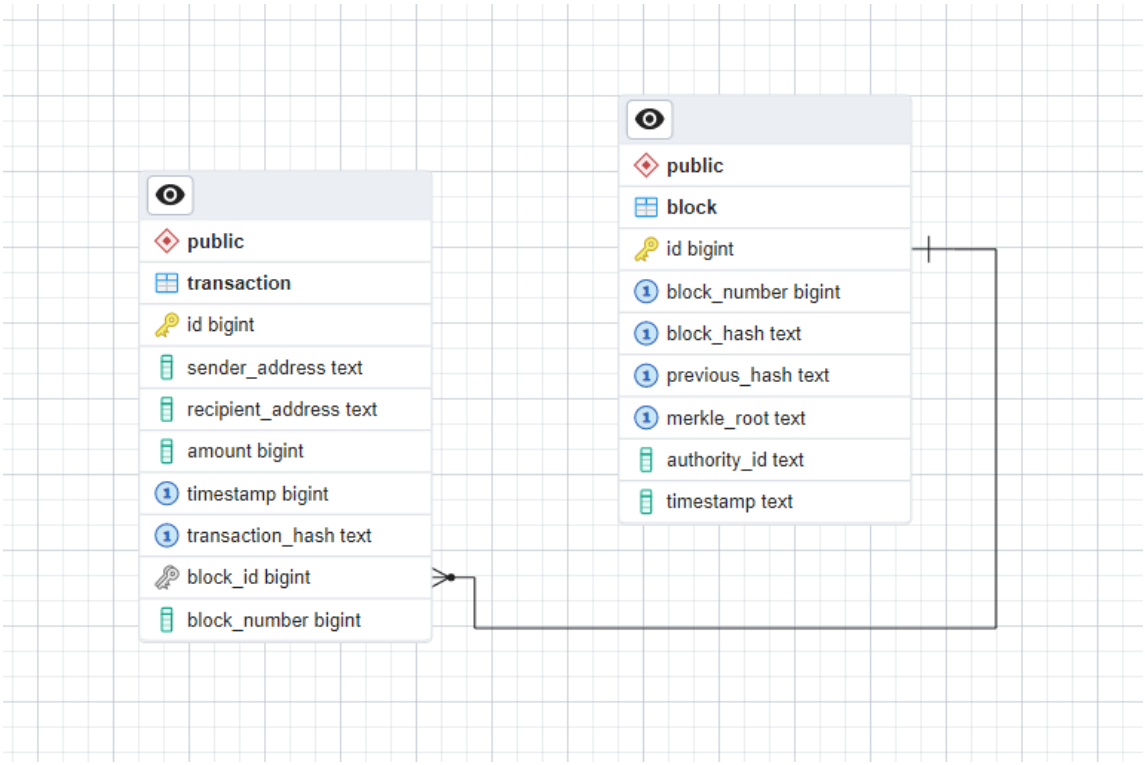

Рисунок 2.8 - Entity Relationship диаграмма базы данных

После проектирования структуры базы данных для блокчейна необходимо приступить к созданию таблиц. Для этого используется язык SQL, который позволяет определить имена таблиц, их атрибуты и типы данных. Каждая таблица также должна иметь первичные ключи, индексы и ограничения в соответствии с требованиями. Этот процесс осуществляется с помощью SQL-инструкций CREATE TABLE.

После создания таблиц необходимо заполнить их данными. Для этого используется SQL-инструкция INSERT, которая позволяет добавить записи в таблицы. Например, можно добавить информацию о блоках и транзакциях, которые являются основными элементами блокчейна.

В результате выполнения всех этих шагов будет создана база данных, которая соответствует требованиям блокчейна. База данных будет содержать необходимые таблицы с соответствующей структурой, а также заполнена актуальными данными. Создание базы данных является важным этапом разработки, поскольку она является

основой для работы приложения и обеспечивает эффективное хранение и управление данными.

Таким образом, создание структуры базы данных позволяет эффективно организовать хранение информации о блоках и транзакциях, что является основой для функционирования блокчейна.

2.3.2 Реализации логики работы блокчейна

В данном разделе представлен процесс разработки программного кода, который обеспечивает стабильное и эффективное функционирование блокчейна, нод, криптографических модулей и веб-интерфейса. Основные технические детали и ключевые аспекты работы кода детально рассмотрены с целью выявления основных принципов его функционирования.

Описание каждого модуля происходит в логическом порядке, начиная с архитектурного обзора и перехода к более детальному рассмотрению каждого компонента. Особое внимание уделяется технологии синхронизации нод и их консенсуса, а также возможности автоматического мягкого форка, что представляет собой уникальную особенность разрабатываемого решения.

Форк в блокчейне – это разветвление или разделение цепи блоков на две или более отдельные цепи из-за несогласия в сети по правилам протокола. Это может произойти из-за различий в мнениях участников сети относительно обновлений протокола, конфликтов в консенсусе или атак на сеть. Каждая ветвь может продолжать свое существование, и участники сети должны выбрать, какую ветвь поддерживать.

Помимо этого, в работе происходит анализ плюсов и уникальных особенностей применяемой технологии, а также рассматриваются потенциальные пути для ее

дальнейшего развития и усовершенствования. Все это призвано создать полное и глубокое понимание работы и возможностей разработанного программного продукта.

2.3.3 Архитектура

Для эффективной установки всех необходимых пакетов проекта следует создать файл "requirements.txt", перечислив в нем все библиотеки и их версии, а затем воспользоваться инструментом pip для их установки. Этот подход значительно упрощает процесс управления зависимостями, позволяя легко обновлять библиотеки и гарантировать совместимость среды разработки.

Создание базовой структуры проекта важно для сохранения порядка и удобства работы. Подобная организация каталогов по модулям и функциональности помогает разработчикам легко найти нужные файлы и облегчает сопровождение кода. Каждый модуль может содержать свои собственные файлы, отвечающие за конкретные аспекты проекта, что способствует его более эффективной разработке и поддержке.

Итоговая архитектура проекта в файловой системе (Рисунок 2.9):

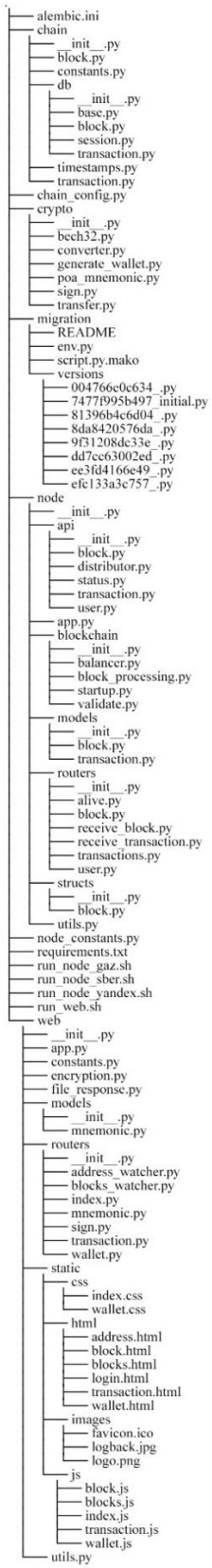

Рисунок 2.9 – Итоговая архитектура проекта в файловой системе

2.3.4 Модуль блокчейна

Модуль блокчейна отвечает за хранение, создание, обновление и удаление блоков и транзакций. А также рассчет хешей на основе внутренних данных блоков и транзакций.

Блоки в блокчейне являются основными элементами данных, которые хранят информацию о транзакциях, контрактах или других событиях. Они связаны между собой в цепочку, обеспечивая непрерывность и безопасность хранения данных. Каждый блок (Рисунок 2.) содержит ссылку на предыдущий блок и хеш, который обеспечивает целостность и невозможность изменения прошлых записей. Блоки позволяют всем участникам системы видеть историю изменений и подтверждать их подлинность, обеспечивая надежность и безопасность работы блокчейна (Рисунок 2.10).

```
27 usages   kesha1225
class Block(Base):
    __tablename__ = "block"

    id = Column(Integer, primary_key=True)
    block_number = Column(Integer, unique=True, nullable=False)

    block_hash = Column(String, unique=True, nullable=False)
    previous_hash = Column(String, unique=True, nullable=False)
    merkle_root = Column(String, unique=True, nullable=True)
    authority_id = Column(String, nullable=False)

    timestamp = Column(BigInteger, nullable=False)

    transactions = relationship( argument: "Transaction", back_populates="block")
```

Рисунок 2.11 – Структура блока в базе данных

```
6 usages  kesha1225
def calculate_block_hash(block: BlockModel) → str:
    block_data = (
        f"{block.block_number}{block.previous_hash}{block.authority_id}"
        f"{block.merkle_root}{block.timestamp}"
    )
    return hashlib.sha256(block_data.encode()).hexdigest()
```

Рисунок 2.10 – Функция расчета хеша блока

Nonce (number used once) – это случайное число, используемое в процессе майнинга блоков в блокчейне Proof of Work (PoW). Он добавляется к блоку вместе с другими данными, чтобы изменить хеш блока и удовлетворить сложность доказательства работы (PoW). В блокчейне на PoW, майнеры должны перебирать различные значения nonce до тех пор, пока не найдут подходящий хеш блока.

В блокчейне на Proof of Authority (PoA) nonce не требуется, поскольку в этом типе консенсуса блоки создаются определенными авторизованными узлами, а не майнерами, конкурирующими за решение математической задачи. Необходимость в nonce отпадает, поскольку в PoA нет необходимости доказывать выполнение работы для создания блоков. Это позволяет упростить процесс формирования блоков и увеличить производительность сети, так как нет необходимости тратить ресурсы на вычисление подходящего nonce.

Отсутствие nonce в блокчейне на PoA можно считать важным улучшением, поскольку это уменьшает энергозатраты, связанные с процессом майнинга, и повышает эффективность сети блокчейна. Благодаря этому блокчейн на PoA становится более экологически чистым и масштабируемым, что важно для его долгосрочной устойчивости и развития.

Отсутствие лимита на количество транзакций (Рисунок 2.11) в блокчейне является важным преимуществом и отличием от других блокчейнов, таких как биткоин и другие медленные криптовалюты. Вот несколько причин, почему это так:

Высокая пропускная способность: Блокчейн без ограничения на количество транзакций в блоке может обрабатывать больше транзакций за тот же период времени, что увеличивает общую пропускную способность сети. Это позволяет ускорить время подтверждения транзакций и обеспечить более быстрый оборот средств (Рисунок 2.12).

Отсутствие комиссий: поскольку блоки могут вмещать больше транзакций, конкуренция за включение транзакций в блок становится менее острой. Это позволяет убрать комиссии за транзакции для пользователей, что делает использование блокчейна более доступным и экономически выгодным.

Улучшенная масштабируемость: Отсутствие ограничений на количество транзакций в блоке способствует улучшению масштабируемости блокчейна. Блокчейны с высокой пропускной способностью могут эффективно обслуживать большое количество пользователей и транзакций без существенного снижения производительности.

Улучшенный пользовательский опыт: более быстрые и дешевые транзакции делают использование блокчейна более удобным для пользователей, улучшая общий пользовательский опыт и привлекательность платформы.

Таким образом, отсутствие лимита на количество транзакций в блоке способствует повышению эффективности и производительности блокчейна, что делает его более конкурентоспособным и привлекательным для пользователей.

```
36 usages   kesha1225
class Transaction(Base):
    __tablename__ = "transactions"

    id = Column(Integer, primary_key=True)
    sender_address = Column(String, nullable=False, index=True)
    recipient_address = Column(String, nullable=False, index=True)
    amount = Column(BIGINT, nullable=False)
    timestamp = Column(BigInteger, nullable=False)
    transaction_hash = Column(String, unique=True, nullable=False)
    block_id = Column(Integer, ForeignKey("block.id"))
    block_number = Column(Integer)
    block = relationship( argument: "Block", back_populates="transactions")
```

Рисунок 2.11 – Структура транзакции в базе данных

```
2 usages  kesha1225
def calculate_block_merkle_root(
    transactions: list[Transaction | TransactionModel],
) → str | None:
    merkle_tree = [tx.transaction_hash for tx in transactions]
    while len(merkle_tree) > 1:
        merkle_tree = [
            hashlib.sha256(
                merkle_tree[i].encode() + merkle_tree[i + 1].encode()
            ).hexdigest()
            for i in range(0, len(merkle_tree), 2)
        ]

    return merkle_tree[0] if merkle_tree else None


4 usages  kesha1225
def calculate_transaction_hash(transaction: Transaction | TransactionModel) → str:
    transaction_str = (
        f"{transaction.sender_address}{transaction.recipient_address}"
        f"{transaction.amount}{transaction.timestamp}"
    )
    return hashlib.sha256(transaction_str.encode()).hexdigest()
```

Рисунок 2.12 – Функции вычисления дерева меркла и хеша транзакции

Добавление случайного числа в транзакции блокчейна используется для предотвращения возникновения одинаковых хешей в случае, если две или более транзакции имеют одинаковые параметры. Это делает атаки на блокчейн сложнее, так как даже небольшие изменения в транзакции приведут к различным хешам блоков.

Однако, в этом случае я использовал более точное время до микросекунд вместо случайного числа, чтобы обеспечить уникальность хеша транзакции [16]. Это значит, что каждая транзакция будет иметь уникальное время создания, которое до микросекунд точно отражает момент создания транзакции. Поскольку время до

микросекунд является уникальным для каждой транзакции, нет необходимости в использовании случайного числа для предотвращения коллизий хешей.

Использование точного времени до микросекунд вместо случайного числа позволяет избежать хранения дополнительных данных в транзакциях и сделать их более легкими и эффективными. Такой подход также обеспечивает безопасность и уникальность каждой транзакции без необходимости генерации случайных чисел.

В базе данных для хранения блоков и транзакций использование индексов на полях sender_address и recipient_address является полезным с точки зрения оптимизации производительности и ускорения выполнения запросов. Индексы на эти поля позволяют значительно снизить время выполнения запросов, связанных с поиском или фильтрацией транзакций по отправителю или получателю.

Использование индексов на полях sender_address и recipient_address (Рисунок 2.13) позволяет базе данных эффективно находить все транзакции, отправленные или полученные конкретным адресом. Это особенно полезно при выполнении запросов на поиск истории транзакций для определенного адреса или при анализе транзакций между определенными адресами.

Благодаря индексам на указанных полях база данных может быстро определить, какие записи соответствуют критериям запроса, что уменьшает время ответа и улучшает общую производительность системы. Такое оптимизированное хранение и доступ к данным важно для эффективной работы блокчейна, особенно при обработке большого объема транзакций.

```sql
create table transactions
(
    id                serial
        primary key,
    sender_address    varchar not null,
    recipient_address varchar not null,
    amount            bigint  not null,
    timestamp         bigint  not null,
    transaction_hash  varchar not null
        unique,
    block_id          integer
        references block
);

alter table transactions
    owner to postgres;

create index ix_transactions_recipient_address
    on transactions (recipient_address);

create index ix_transactions_sender_address
    on transactions (sender_address);
```

Рисунок 2.13 - Описание модели транзакции на DDL вместе с индексами на полях sender_address и recipient_address

Bech32 – это формат адресов, используемый в блокчейне, который представляет собой надежный и эффективный способ кодирования данных. Он основан на алгоритме Безье, который обеспечивает высокую степень надежности и устойчивости к ошибкам при передаче данных [19].

Одним из ключевых преимуществ Bech32 является его способность предотвращать ошибки при копировании и вводе адресов. Это достигается за счет использования символов, которые легко различимы человеческим глазом и имеют высокую степень различимости между собой. Это делает Bech32 идеальным выбором для блокчейна на PoA, где точность и надежность важны для успешной обработки транзакций.

Кроме того, Bech32 обеспечивает эффективное использование места и уменьшает размер передаваемых данных (Рисунок 2.14). Это особенно важно для блокчейна на PoA, где меньший размер транзакций приводит к более быстрой обработке и уменьшает нагрузку на сеть.

Таким образом, Bech32 является идеальным выбором для блокчейна на PoA благодаря своей высокой надежности, эффективному использованию места и способности предотвращать ошибки при передаче данных (Рисунок 2.15).

```
1 usage   kesha1225
def generate_wallet() -> tuple[str, str, str, str]:
    mnemo = generate_mnemo_words()
    seed = to_seed(mnemo)
    unsigned_int = to_unsigned_list_int(seed)
    pk, sk = generate_pk_sk(unsigned_int)
    address = generate_wallet_address(pk, prefix=ChainConfig.address_prefix)
    return address, mnemo, pk, sk
```

Рисунок 2.14 – Функция генерации кошелька с префиксом poa с помощью bech32

```
[13, 15, 3, 20, 21, 24, 7, 16, 25, 1, 2, 19, 28, 31, 30, 25, 25,
 21, 7, 28, 2, 19,29, 27, 25, 13, 16, 22, 0, 13, 15, 3,5, 0, 19,
 19, 20, 25, 23, 29, 16, 11, 23, 30, 30, 15, 29, 9, 12, 11, 27, 0]

"poa1d0r54c8sepznul7ee48uznamedskqd0r9qnn5ehasth770afvtmqnp4sh3"
```

Рисунок 2.15 – Конвертированные байты из публичного ключа и адрес, получившийся из них путем закодирования с помощью bech32

Описание модуля блокчейна, представленное выше, подчеркивает ключевые аспекты его разработки и функционирования. Рассмотрены основные компоненты блокчейна, включая структуру блока, транзакции, ноды и веб-интерфейс. Архитектура проекта основана на использовании СУБД PostgreSQL, клиентского веб-интерфейса и нод, что обеспечивает надежную и эффективную работу блокчейна.

Использование инновационных технологий, таких как Bech32 для адресации и точное время до микросекунд для обеспечения уникальности транзакций, подчеркивает стремление к оптимизации и улучшению производительности блокчейна. Такой подход отражает не только технический аспект работы, но и его стремление к созданию надежной и эффективной системы для проведения трансграничных транзакций.

2.3.5 Модуль ноды

Модуль ноды – это сердце блокчейна. Он отвечает за создание, проверку и утверждение блоков, которые составляют основу цепи транзакций. Этот модуль также обеспечивает взаимодействие пользователей с блокчейном через веб-интерфейс.

Нода является ключевым элементом сети, так как именно здесь происходит процесс генерации новых блоков на основе подтвержденных транзакций. Кроме того, нода отвечает за поддержание согласованности и безопасности сети блокчейна.

Особенностью этого модуля является его распределенность. Ноды блокчейна могут быть развернуты в различных местах сети, что обеспечивает децентрализацию и надежность системы. Также нода способна взаимодействовать с другими узлами блокчейна для синхронизации данных и подтверждения блоков.

Таким образом, модуль ноды играет ключевую роль в функционировании блокчейна, обеспечивая его работу и взаимодействие с пользовательскими интерфейсами.

От владельца для запуска нода требует 4 секретных параметра:

– POSTGRES_URL: Ссылка на базу данных
– POSTGRES_URL_ALEMBIC: Ссылка на базу данных для миграции
– NODE_ID: Уникальный ключ-название ноды
– PRIVATE_KEY: Секретный ключ для подписывания блоков [22]

При запуске нода автоматически мигрирует возможные обновления в базе и начинает процесс предзапуска (Рисунок 2.16).

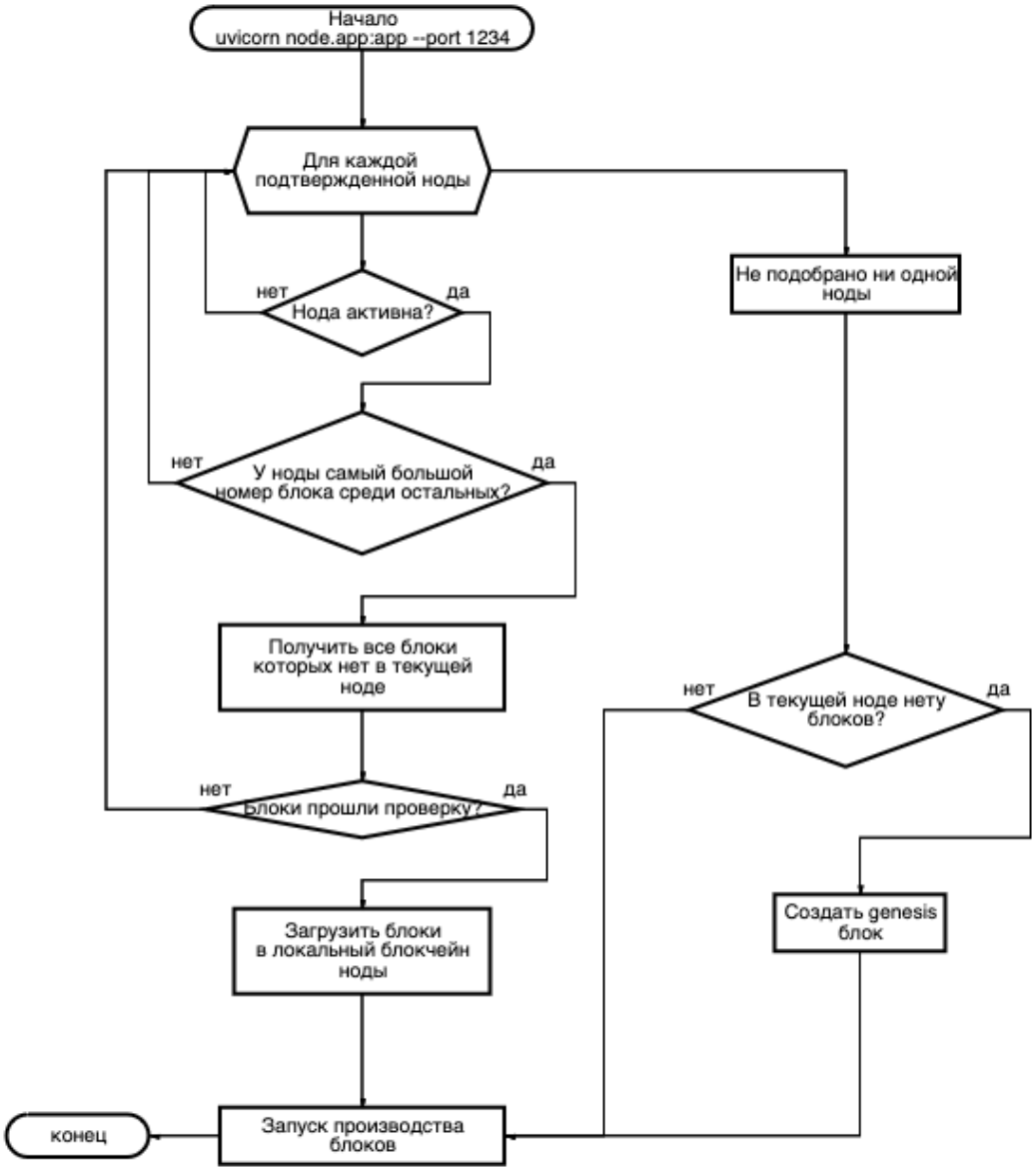

Рисунок 2.16 – Блок схема, описывающая процесс запуска ноды до момента запуска производства блоков

После запуска и синхронизации нода переходит к процессу, которым она и будет заниматься весь свой жизненный цикл – производству блоков.

Описание процесса производства блоков и управления статусами нод очень информативное. Описание процесса шаг за шагом:

Производство блоков: Ноды производят блоки с заданной периодичностью (например, каждые 10 секунд). Этот интервал времени выбирается для предотвращения

переизбытка данных блоков с пустыми транзакциями и для накопления транзакций для обработки.

Статусы нод: у нод есть два основных статуса - "готовность" и "ожидание". При запуске нода находится в статусе "ожидание". Готовые ноды могут принимать и генерировать блоки, только что запущенная нода является неготовой и может только принимать блоки. Если нет других активных нод, то нода становится "готовой", иначе она остается в режиме ожидания.

Создание блока: когда наступает время для создания блока, активная и готовая нода берет все неподтвержденные транзакции, созданные через нее, и формирует новый блок, вычисляя дерево Меркла на основе этих транзакций.

Рассылка уведомлений: Нода отправляет уведомление всем другим активным нодам о предстоящем выпуске нового блока, подписывая его собственным ключом.

Проверка уведомлений: Активные ноды получают уведомление, проверяют его подпись и отклоняют, если подпись неверна. Если проверка прошла успешно, они переходят к следующему шагу. Получившие уведомления неготовые ноды становятся готовыми.

Добавление блока в блокчейн: Блок добавляется в локальную копию блокчейна текущей ноды, и в него включаются данные о новом блоке и его транзакциях.

Рассылка блока: Новый блок рассылается всем остальным активным нодам с подписью.

Прием блока другими нодами: Другие активные ноды принимают блок, валидируют его подпись и хеши всех транзакций и добавляют его к себе в базу.

Переход к следующей ноде: Очередь переключается на следующую ноду по порядку, и весь процесс повторяется. После рассылки блока нода уходит в режим ожидания, а следующая за ней по списку нода начинает генерировать блок [25].

Этот процесс (Рисунок 2.17) позволяет поддерживать целостность и консистентность блокчейна, а также обеспечивает равномерное распределение обязанностей между нодами в сети.

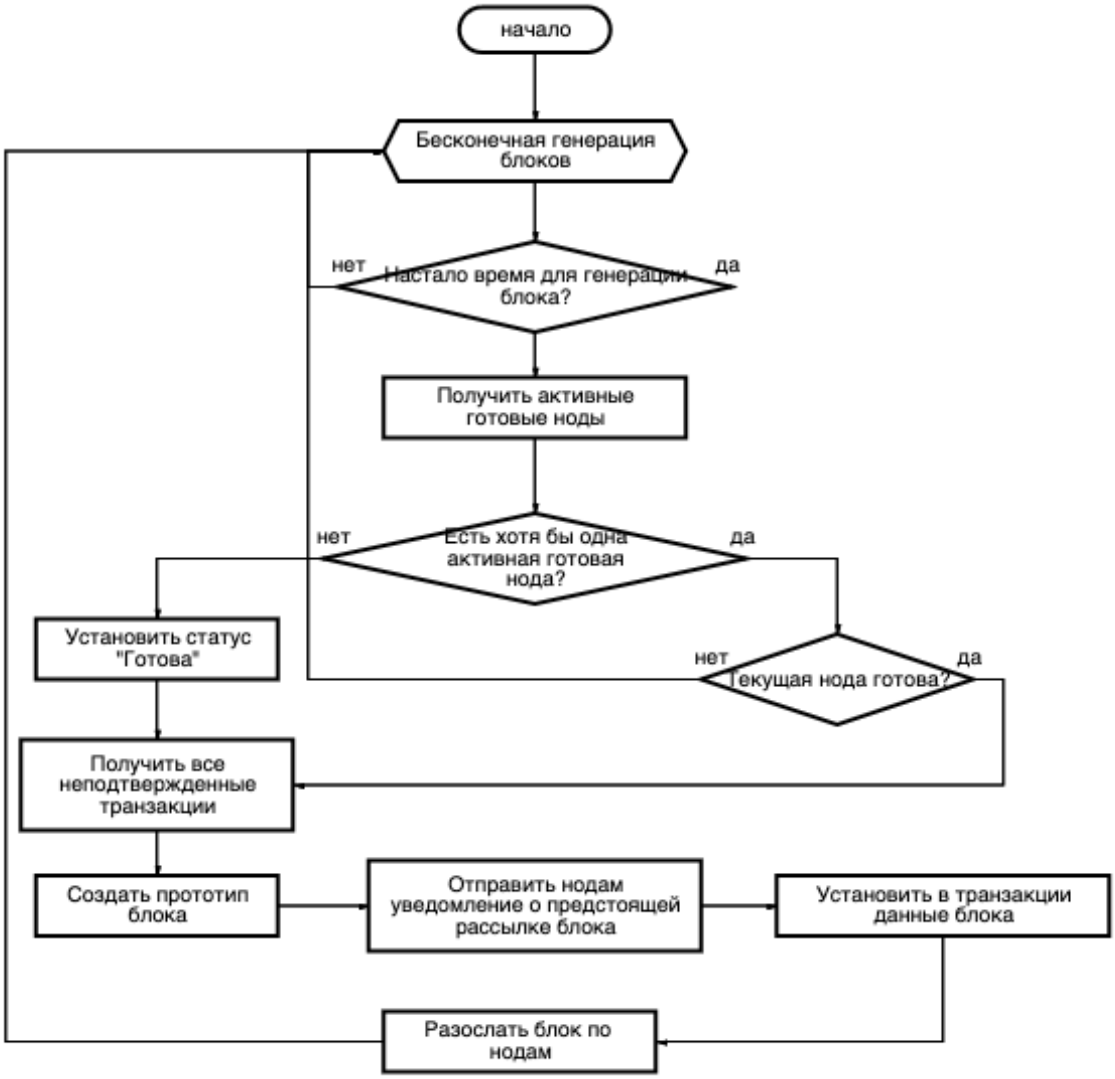

Рисунок 2.17 – Блок схема, описывающая процесс запуска ноды до момента запуска производства блоков

В данном блокчейне, использующем механизм Proof of Authority (PoA), проблема начальной эмиссии решена путем создания специализированного эмиссионного адреса, обладающего возможностью отправки неограниченных средств. Этот подход обладает важным преимуществом: он обеспечивает гибкость и контроль над процессом выпуска валюты, что критически важно для стабилизации и развития экономической модели блокчейна.

Такой эмиссионный адрес, управляемый центральной авторитетной фигурой или организацией, позволяет быстро и эффективно реагировать на меняющиеся

экономические условия и потребности сети. Это особенно важно в начальные стадии развития блокчейн-сетей, когда необходимо удовлетворить растущий спрос на токены для транзакций, управления и других операций внутри сети.

Существование такого адреса также минимизирует риски, связанные с перераспределением средств между участниками, поскольку контроль над эмиссией остается в руках уполномоченных лиц, что способствует поддержанию порядка и доверия в системе. Кроме того, это уменьшает вероятность инфляции, вызванной несанкционированным или необдуманным выпуском новых токенов, поскольку каждое решение о выпуске дополнительных средств может быть тщательно взвешено и контролируемо.

Таким образом, наличие адреса эмиссионера с возможностью отправки неограниченных средств в блокчейне на PoA является эффективным решением для управления начальной эмиссией и поддержания стабильности системы, что в свою очередь способствует укреплению доверия и привлекательности блокчейн-платформы для текущих и будущих пользователей.

Модуль ноды является центральным элементом блокчейна, ответственным за создание, проверку и утверждение блоков, а также за взаимодействие с пользовательскими интерфейсами через веб-интерфейс. Он играет ключевую роль в функционировании блокчейна, обеспечивая его работу и безопасность.

Особенностью этого модуля является его децентрализованность: ноды блокчейна могут быть развернуты в различных местах сети, что обеспечивает надежность и согласованность системы. Ноды также взаимодействуют между собой для синхронизации данных и подтверждения блоков.

Процесс производства блоков подразумевает создание новых блоков с определенной периодичностью, а также управление статусами нод для обеспечения равномерного распределения обязанностей между ними. Этот процесс позволяет поддерживать целостность и консистентность блокчейна.

Таким образом, модуль ноды является ключевым элементом блокчейна, обеспечивающим его нормальную работу и взаимодействие с внешними интерфейсами.

2.4 Руководство пользователя

Модуль веб-интерфейса является ключевой составляющей архитектуры нашего приложения, обеспечивающей взаимодействие между конечными пользователями и основной блокчейн-инфраструктурой. Этот модуль включает в себя веб-сайт и программный интерфейс приложения (API), которые играют роль посредника между фронтендом и блокчейн-нодами.

Веб-сайт предоставляет пользовательский интерфейс, доступный из любого браузера, который позволяет пользователям легко и интуитивно понятно взаимодействовать с различными функциями блокчейна. Этот аспект важен для обеспечения удобства использования и доступности системы для рядовых пользователей, не обладающих специальными техническими знаниями.

API, с другой стороны, представляет собой набор программных инструкций и протоколов, который позволяет фронтенду коммуницировать с нодами блокчейна. Он обеспечивает безопасную передачу данных и выполнение транзакций, что критично для поддержания целостности и безопасности всей блокчейн-системы. API служит мостом, по которому данные могут свободно и безопасно передаваться между пользовательским интерфейсом и блокчейном, обеспечивая выполнение требуемых операций без задержек и ошибок.

Таким образом, модуль веб-интерфейса не просто упрощает взаимодействие с блокчейном, но и играет важную роль в обеспечении функциональности, безопасности и доступности приложения. Он является витриной технологии блокчейна для конечного

пользователя, позволяя им эффективно использовать предоставляемые блокчейн-технологией возможности для своих нужд.

В процессе первого взаимодействия с блокчейн-платформой Poachain, начальный этап взаимодействия пользователя с системой начинается со страницы входа. Эта страница представляет собой критически важный элемент пользовательского интерфейса, который обеспечивает безопасность и удобство при входе в систему. На этой странице пользователь сталкивается с двумя основными опциями: ввод существующей мнемонической фразы или создание новой.

Мнемоническая фраза, или мнемофраза, является фундаментальным элементом криптографической безопасности в блокчейне. Это набор слов, сгенерированных в соответствии со стандартом BIP39 (Bitcoin Improvement Proposal), который представляет собой простой и понятный способ генерации криптографических ключей [9]. Каждая мнемофраза уникально связана с определенным кошельком и может восстанавливать доступ к криптовалютным активам в случае потери или повреждения устройства. Стандарт BIP39 включает в себя список из 2048 слов, из которых выбираются случайные слова для создания мнемонической фразы, обычно состоящей из 12, 18 или 24 слов.

Система Poachain предоставляет пользователям возможность либо ввести ранее созданную мнемофразу для доступа к уже существующему кошельку, либо сгенерировать новую мнемоническую фразу (Рисунок 2.). Создание новой мнемофразы инициирует процесс генерации уникальной последовательности слов, которая будет использоваться для создания и последующего доступа к новому блокчейн-кошельку. Этот процесс включает в себя использование криптографически безопасного генератора случайных чисел для обеспечения того, чтобы каждая сгенерированная мнемофраза была уникальной и не поддающейся предсказанию (Рисунок).

Важность мнемофразы в контексте безопасности блокчейн-кошелька не может быть недооценена. Она не только обеспечивает пользователю контроль над своими криптовалютными активами, но и является ключевым элементом в защите от

несанкционированного доступа. Пользователи должны обращать внимание на сохранность своей мнемофразы, избегая её разглашения и обеспечивая её безопасное хранение. Это обеспечивает надежную защиту от возможных кибератак и потери активов.

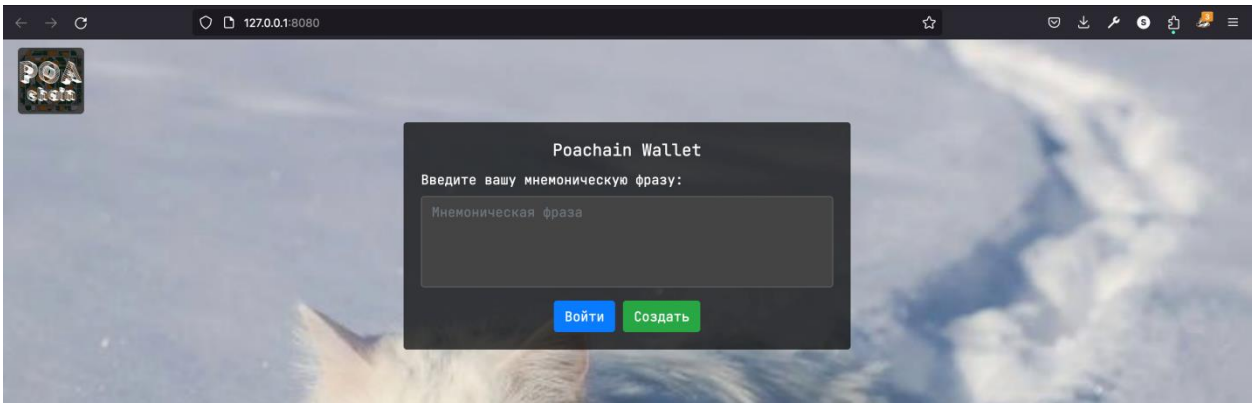

Рисунок 2.20 – Страница входа

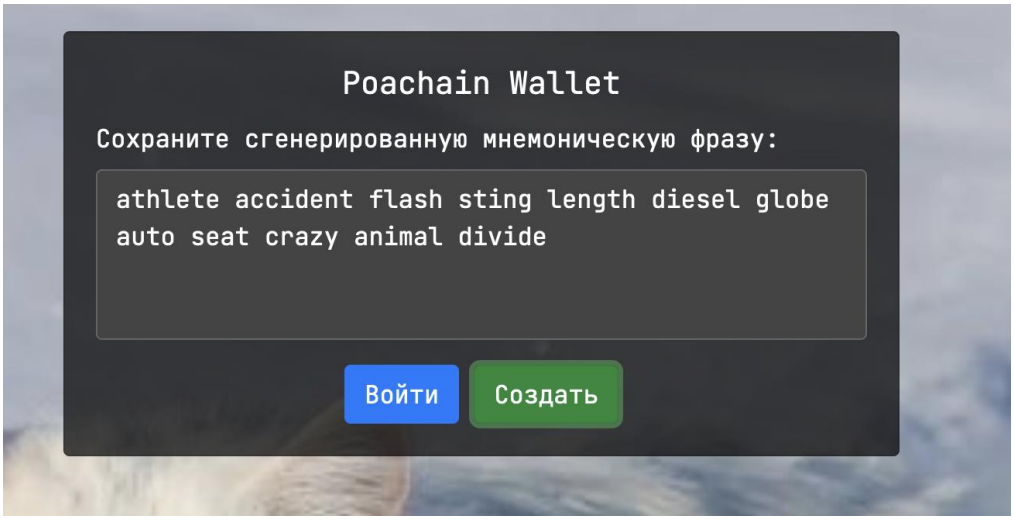

Рисунок 2.21 – Страница входа со свежесгенерированной мнемонической фразой

Основная страница, на которой пользователь проводит большую часть времени при использовании блокчейн-платформы Poachain, – это интерфейс кошелька (Рисунок 2.). Этот интерфейс кошелька является центральным элементом взаимодействия пользователя с блокчейном, предоставляя все необходимые инструменты и информацию

для эффективного управления криптовалютными активами.

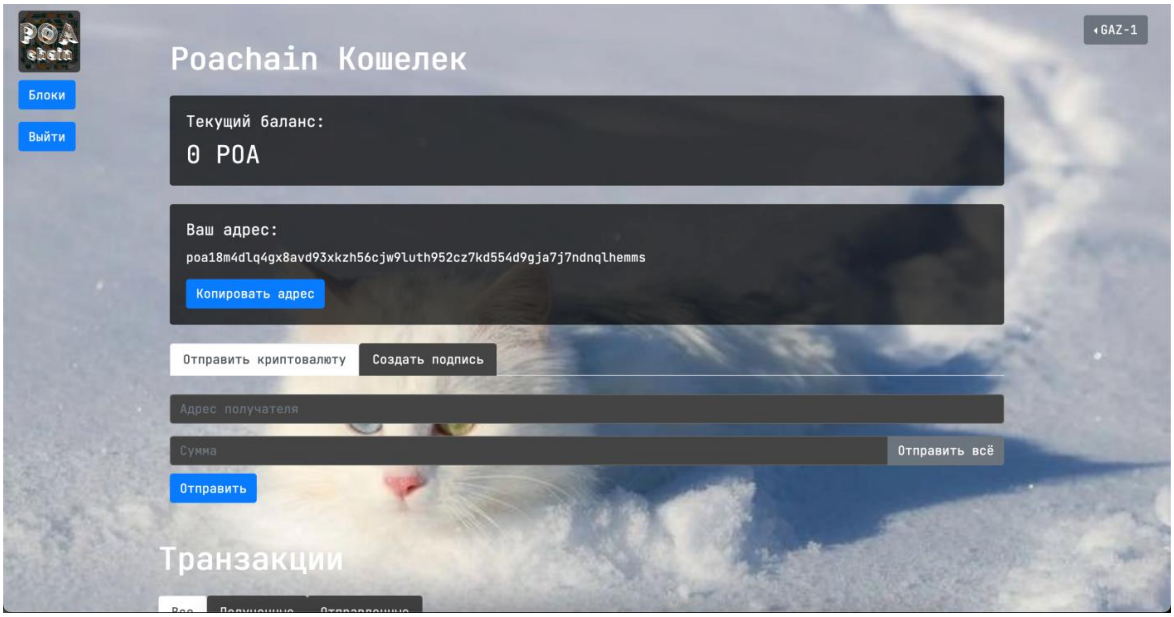

Рисунок 2.22 – Раздел кошелька целиком

В интерфейсе кошелька Poachain центральное место занимает отображение текущего баланса и адреса кошелька, причём адрес можно легко скопировать с помощью функции копирования. Это упрощает управление счетом и обмен данными между пользователями. В правом верхнем углу расположен модуль для обзора и выбора нод блокчейна (Рисунок 2.). Учитывая, что блокчейн включает несколько нод, пользователи имеют возможность выбирать ноду на основе различных критериев, таких как время задержки транзакций или предпочтение в отношении оператора ноды. Этот модуль также предоставляет информацию о статусе онлайн каждой ноды и текущей длине цепочки блоков, что позволяет пользователям делать информированный выбор и улучшает их взаимодействие с технологией блокчейн.

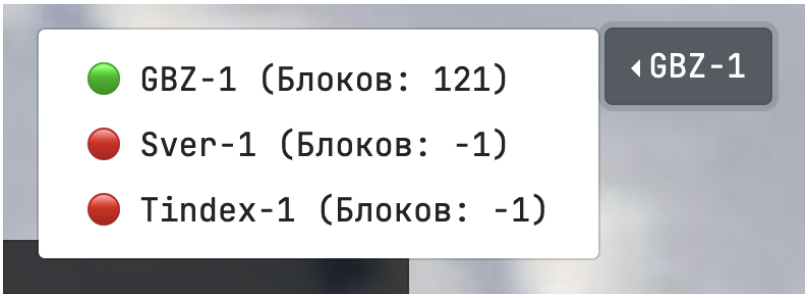

Рисунок 2.23 – Интерфейс обзора и выбора нод

Под основным блоком интерфейса, отображающим адрес и баланс пользователя, расположена функция подписи (Рисунок 2.). Эта функция критически важна, так как она позволяет пользователю подтверждать свои права на кошелек перед сторонними ресурсами или другими пользователями. Использование цифровой подписи в этом контексте является средством аутентификации, обеспечивающим безопасное подтверждение владения без необходимости раскрывать конфиденциальные данные.

Рисунок 2.24 - Интерфейс подписи

Расположенная рядом с кнопкой подписи, функция отправки криптовалюты является одним из ключевых элементов интерфейса кошелька. Эта функция позволяет пользователю инициировать транзакции, отправляя криптовалютные средства на другие адреса. Например, как показано на рисунке 18, адрес эмиссии переводит 100 единиц условной валюты POA на другой адрес, который был сгенерирован ранее (Рисунок 2.).

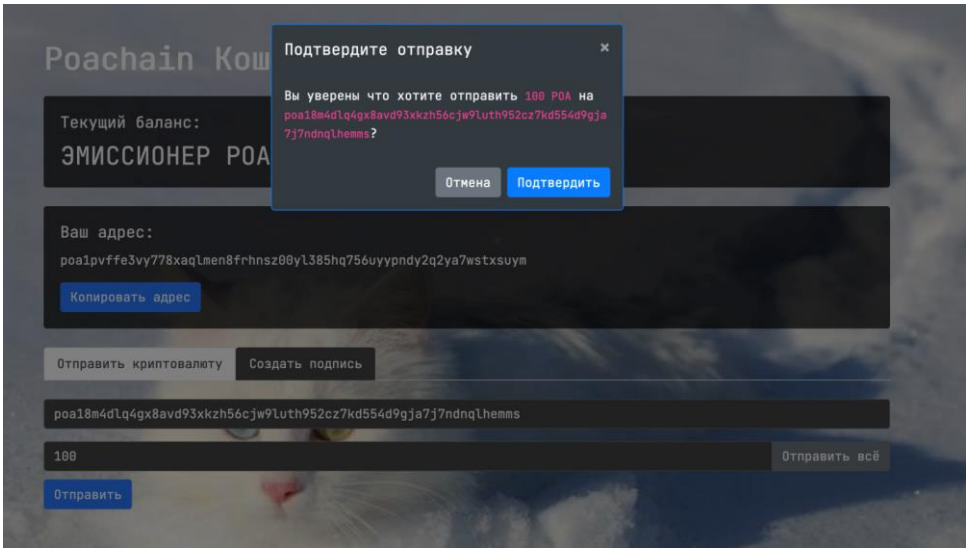

Рисунок 2.25 – Интерфейс отправки криптовалюты

После выполнения транзакции изменения в балансах немедленно отражаются в системе: сумма списывается с баланса отправителя и появляется на балансе получателя, при условии, что оба пользователя находятся в сети одной и той же ноды. Если же транзакция была направлена на адрес, который обслуживается другой нодой, изменения в балансе будут видны только после того, как эта нода сгенерирует новый блок и включит в него данную транзакцию.

В нижней части страницы (Рисунок 2.18) расположен блок для просмотра транзакций, где пользователи могут сортировать их по типу: входящие или исходящие. Каждая транзакция в списке снабжена кликабельным адресом отправителя или получателя, а также кнопкой, позволяющей перейти к подробному обзору каждой транзакции. Это удобство позволяет пользователям легко отслеживать и анализировать свои финансовые операции в блокчейн-сети, обеспечивая прозрачность и контролируемость денежных потоков.

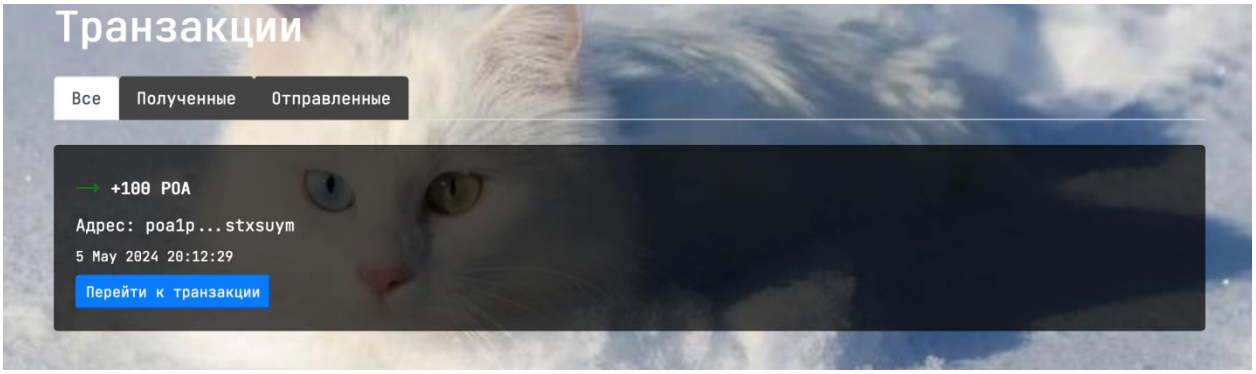

Рисунок 2.18 - Интерфейс обзора транзакций, в котором видно только что полученную транзакцию на 100 POA

При клике на адрес в списке транзакций в блокчейн-системе Poachain, пользователь активирует просмотрщик адреса. Это интерфейс, аналогичный основному окну кошелька, но с некоторыми ограничениями: в нём недоступны функции подписи или перевода средств. Однако, пользователь по-прежнему имеет доступ ко всем остальным данным, аналогичным тем, что представлены в основном кошельке. Это включает в себя информацию о балансе и истории транзакций связанного с адресом. Таким образом, просмотрщик адреса предоставляет комплексный обзор активности и

баланса без возможности инициировать транзакции, обеспечивая тем самым безопасный и информативный способ изучения транзакционной активности адресов в сети (Рисунок 2.19).

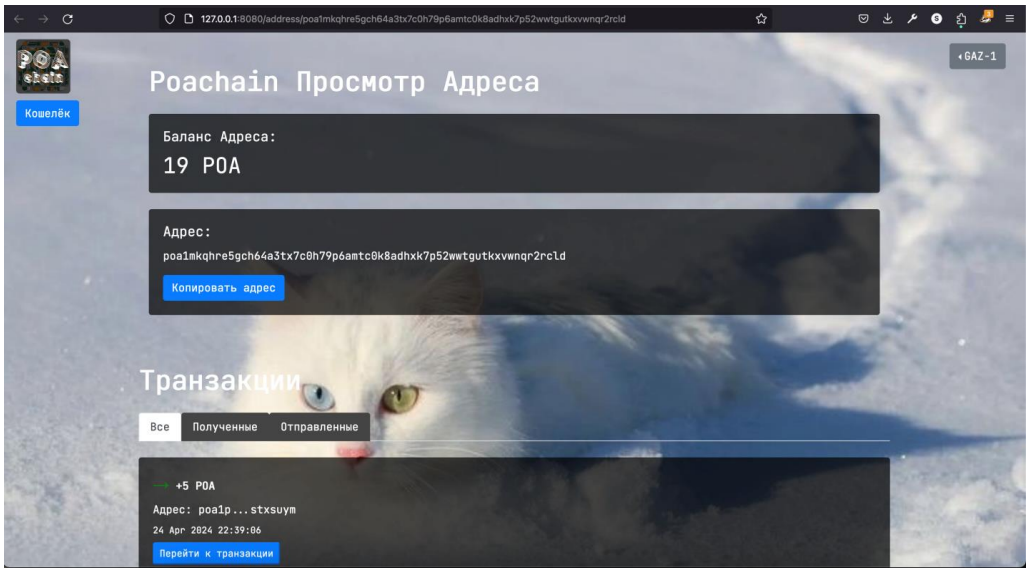

Рисунок 2.19 - Страница просмотра адреса. В адресной строке видно, как она формируется.

Клик по кнопке «Перейти к транзакции» в интерфейсе кошелька Poachain активирует детальное окно просмотра транзакции (Рисунок 2.20). Это окно предоставляет обширную информацию о конкретной транзакции, облегчая глубокий анализ и проверку данных для пользователей. В окне отображаются такие параметры, как хеш транзакции, который является уникальным идентификатором и позволяет отслеживать её в блокчейне, источник (адрес отправителя) и получатель (адрес получателя), что указывает на участников транзакции. Также здесь указывается сумма перевода, номер блока, в котором транзакция была зафиксирована, идентификатор ноды, создавшей этот блок, и точное время проведения транзакции.

Такое подробное представление данных важно для обеспечения прозрачности и возможности верификации всех аспектов транзакции, что является критически важным для поддержания доверия и безопасности в рамках блокчейн-системы. Эти данные помогают пользователям не только проследить за динамикой своих средств, но и оценить надежность и статус узлов сети.

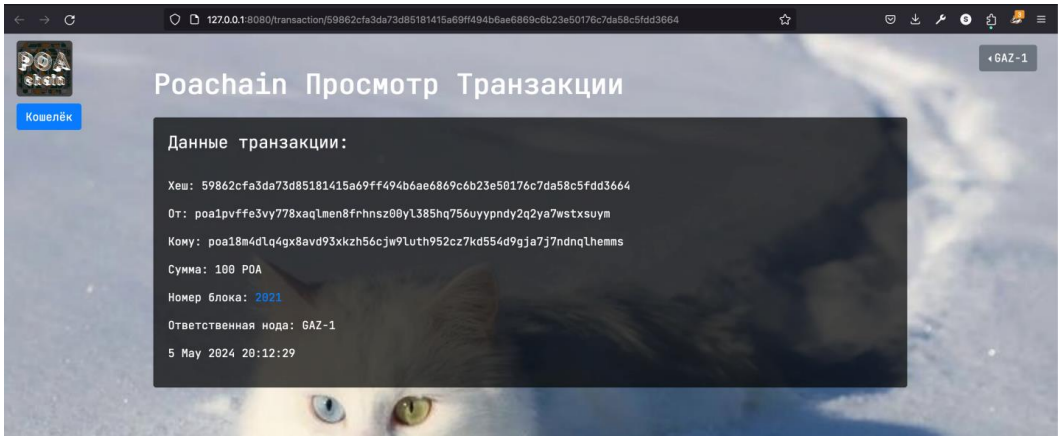

Рисунок 2.20 - Страница просмотра транзакции. В адресной строке видно, как она формируется.

На рисунке 21 в интерфейсе кошелька Poachain показано, что номер блока является кликабельным элементом. При нажатии на него пользователи получают доступ к дополнительному окну, которое предоставляет глубокие детали о блоке, содержащем выбранную транзакцию. Это окно обзора блока включает в себя не только номер и хеш блока, обеспечивая уникальный идентификатор блока в блокчейне, но и указывает на ноду-издателя, которая была ответственна за создание данного блока.

Кроме того, представлен предыдущий хеш блока, который также является кликабельным и позволяет пользователям последовательно переходить к предшествующим блокам, тем самым просматривая историю блокчейна. Эта функциональность добавляет прозрачности и позволяет отслеживать последовательность блоков назад во времени. Дата выпуска блока также отображается, предоставляя информацию о времени его генерации.

В нижней части окна обзора блока отображаются все транзакции, входящие в данный блок, что позволяет пользователям увидеть полную картину всех операций, произведенных в конкретный момент времени (Рисунок 2.21). Эта функция обеспечивает высокую степень подробности и прозрачности, что критически важно для аудита и верификации действий в блокчейне.

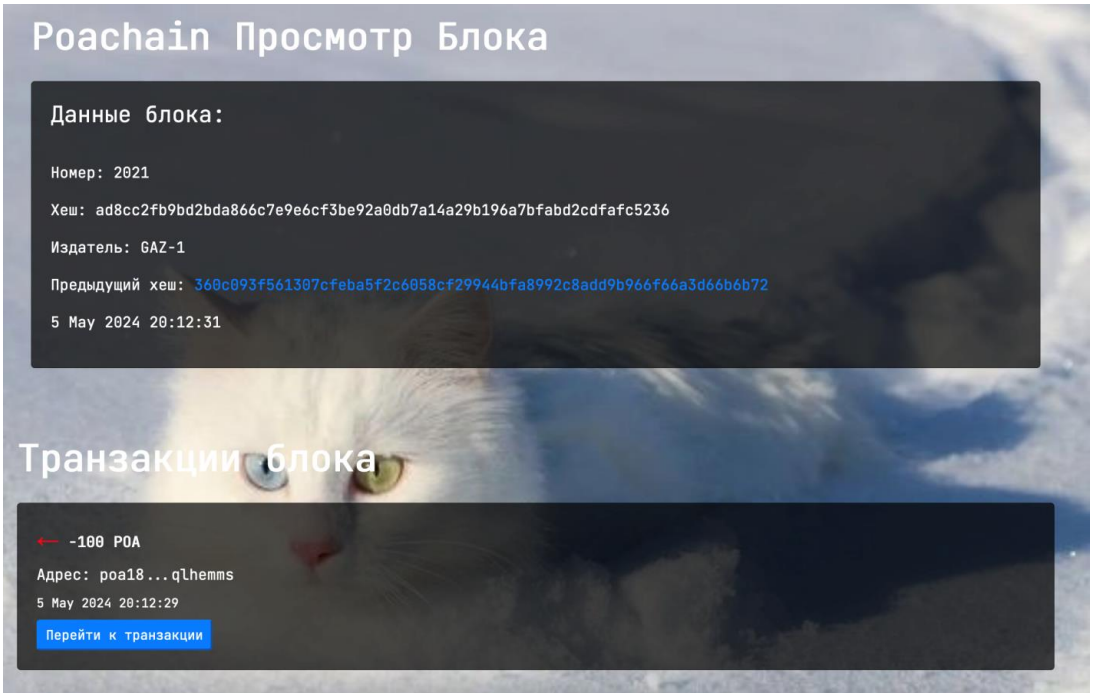

Рисунок 2.21 - Страница просмотра блока

В левом верхнем углу интерфейса кошелька Poachain расположена кнопка «Блоки». Эта кнопка является порталом в обзорщик блоков (Рисунок 2.), где пользователи могут в режиме реального времени наблюдать за генерацией блоков в блокчейне. Это позволяет пользователям оставаться в курсе всех актуальных изменений и операций, происходящих в сети.

В обзорщике блоков представлена информация о каждом сгенерированном блоке, включая базовые данные, такие как номер блока, хеш, нода-издатель, и дата выпуска. Пользователи могут нажать на кнопку «Подробнее», чтобы перейти в окно обзора блока, где доступна более детальная информация о содержимом блока, включая список всех транзакций, которые он содержит.

На нижней части страницы обзорщика блоков расположена система пагинации. Это удобный инструмент, который позволяет пользователям переходить между страницами и просматривать блоки за всю историю существования ноды. Такая возможность не только обеспечивает глубокий аналитический доступ к данным блокчейна, но и улучшает навигацию и исследование блокчейн-сети, делая процесс поиска и верификации информации более простым и доступным для всех пользователей.

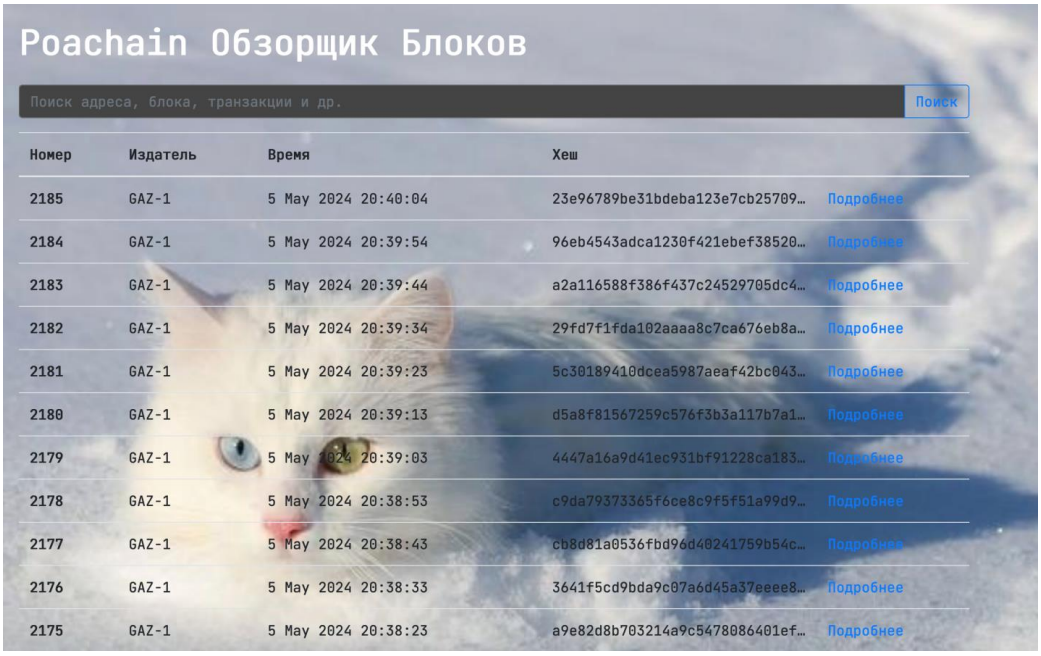

Рисунок 2.30 – Обзорщик блоков

В обзорщике блоков Poachain внедрена функция поиска, позволяющая пользователям искать информацию по адресу, номеру блока или хешу транзакции. Этот инструмент значительно упрощает навигацию по блокчейну, предоставляя быстрый доступ к необходимым данным. Введение конкретных параметров в строку поиска автоматически перенаправляет пользователя на страницу с подробным обзором соответствующего блока, адреса или транзакции.

Для улучшения пользовательского опыта и удобства взаимодействия с блокчейн-платформой Poachain, веб-сайт реализует механизм хранения зашифрованных данных пользователя в локальном хранилище браузера. Это решение позволяет избежать необходимости повторного входа в систему при каждом новом посещении сайта, ускоряя доступ к функционалу платформы.

Защита и шифрование данных: Использование шифрования для данных, таких как публичные и приватные ключи, адрес кошелька, а также информация о текущей выбранной ноде, обеспечивает безопасность конфиденциальной информации. Шифрование данных перед их сохранением в локальное хранилище помогает предотвратить их утечку и несанкционированный доступ, так как даже в случае

получения доступа к локальному хранилищу данные останутся защищёнными благодаря их зашифрованному виду.

Быстрый доступ к нужной информации: поскольку данные уже сохранены и доступны локально, время на загрузку и обработку информации существенно сокращается. Пользователи могут мгновенно получить доступ к своему кошельку и начать операции с криптовалютой, без задержек на процесс аутентификации.

Управление выбором ноды: Хранение информации о выбранной ноде позволяет системе автоматически подключать пользователя к последней использованной ноде, улучшая опыт взаимодействия и устраняя необходимость в ручной настройке при каждом входе.

Эта функциональность значительно повышает удобство использования платформы Poachain, делая процесс управления блокчейном более гладким и менее затратным в плане времени. Однако при этом крайне важно обеспечить, чтобы механизмы шифрования и обработки данных были выполнены на высочайшем уровне, чтобы исключить любые возможности для утечек или взлома данных.

Веб-интерфейс блокчейн-платформы Poachain представляет собой высокофункциональный, пользовательский и безопасный элемент в архитектуре системы. Он обеспечивает не только эффективное управление криптовалютными активами через кошелек, но и предлагает глубокую интеграцию с технологической инфраструктурой блокчейна через различные модули и функции. Ключевые аспекты включают детальный просмотр транзакций, блоков, и возможность легкой навигации и анализа через поиск по адресу, номеру блока или хешу транзакции.

Интеграция механизмов шифрования для безопасного хранения ключевых данных в локальном хранилище улучшает пользовательский опыт, минимизируя необходимость повторного ввода данных и ускоряя доступ к ресурсам. Также важным аспектом является возможность сохранения и автоматической загрузки предпочтений пользователя, включая выбор ноды, что делает взаимодействие с системой более интуитивно понятным и персонализированным.

В целом, веб-интерфейс Poachain служит важной связующей звеном между пользователем и сложной инфраструктурой блокчейна, облегчая доступ к передовым технологиям безопасности и обеспечивая надёжное и эффективное управление блокчейн-операциями. Это способствует не только удобству, но и поддерживает высокий уровень безопасности и доверия, что является критически важным для успеха любой блокчейн-системы.

Заключение

В рамках данной работы была разработана система для проведения трансграничных транзакций с использованием технологии блокчейна на основе консенсуса Proof of Authority (PoA). Работа включала глубокий анализ существующих аналогов и консенсус-алгоритмов, выбор среды разработки и проектирование комплексного приложения, способного обеспечить быстрые и безопасные транзакции между пользователями в разных странах.

Цель выпускной квалификационной работы – упростить и ускорить трансграничные платежи, минимизировать риски и издержки, связанные с традиционными банковскими системами и их регуляторными ограничениями. Использование блокчейна на PoA позволило достичь высокой производительности и масштабируемости системы, а также обеспечить необходимый уровень безопасности благодаря механизмам шифрования и аутентификации.

В ходе работы были рассмотрены ключевые аспекты проектирования и разработки приложения, включая создание архитектуры блокчейна, разработку интерфейса пользователя и реализацию логики работы блокчейна. Особое внимание уделено модулям ноды и веб-интерфейса, которые играют важную роль в обеспечении взаимодействия пользователей с системой.

Результаты данной работы показывают, что блокчейн на PoA предоставляет значительные преимущества для трансграничных транзакций, включая снижение времени транзакции, уменьшение комиссий и повышение безопасности данных. В дальнейшем планируется развитие системы путём интеграции дополнительных инструментов для анализа и управления транзакциями, а также расширения функционала веб-интерфейса для обеспечения более глубокой интеграции с другими блокчейн-сетями и платформами.

Разработанное приложение не только демонстрирует жизнеспособность использования блокчейна для трансграничных платежей, но и открывает новые перспективы для его применения в различных сферах, где требуется быстрая, надёжная и прозрачная обработка транзакций.

При внедрении системы трансграничных транзакций на базе блокчейна с консенсусом Proof of Authority в России, основным преимуществом становится возможность существенного снижения затрат и времени на проведение транзакций. Блокчейн на PoA способен упростить и ускорить международные платежи, минимизируя зависимость от посредников и банковских структур, что традиционно связано с высокими комиссиями и длительным временем обработки платежей.

В контексте российской финансовой системы, где регулятивные требования кросс-бордерных транзакций строги, использование блокчейна может обеспечить повышенную прозрачность и безопасность операций. Блокчейн технологии способны обеспечить неизменность и непрерывность записей, что крайне важно для соответствия нормативным требованиям и защиты данных.

Кроме того, блокчейн может способствовать расширению возможностей российских предприятий на международные рынки, предоставляя надежную инфраструктуру для безопасной и эффективной коммерческой деятельности. Это особенно актуально в свете стремления России усилить свои экспортные позиции и расширить географию внешнеэкономических связей.

Таким образом, применение блокчейна для трансграничных транзакций в России представляет собой перспективное направление, способное принести значительные экономические выгоды и укрепить позиции страны на мировом финансовом рынке. Но успешная реализация такой системы потребует тщательного учета правовых аспектов и создания надежного механизма взаимодействия с международными платежными системами.

Список источников информации

1. Bitcoin Whitepaper. - Текст : электронный // [сайт]. - 2023. - URL: https://bitcoin.org/files/bitcoin-paper/bitcoin_ru.pdf (дата обращения: 13.02.2024).

2. Типы консенсус-механизмов блокчейна. - Текст : электронный // [сайт]. - URL: https://vc.ru/crypto/509865-tipy-konsensus-mehanizmov-blokcheyna (дата обращения: 09.06.2024).

3. Сравнение гетерогенных блокчейнов (Cosmos, Polkadot, Avalanche). - Текст : электронный // [сайт]. - URL: https://habr.com/ru/articles/673110/ (дата обращения: 08.06.2024).

4. Что такое слои блокчейна L0, L1, L2 и L3 и зачем они нужны. - Текст : электронный // [сайт]. - URL: https://habr.com/ru/articles/688076/ (дата обращения: 08.06.2024).

5. Проблемы безопасности блокчейна. - Текст : электронный // [сайт]. - URL: https://www.h-x.technology/ru/blog-ru/blockchain-security-issues-ru (дата обращения: 06.06.2024).

6. Антонопулос, А.М. Mastering Bitcoin. - Москва : ДМК Пресс, 2018. - 428 с. - ISBN 978-5-94074-965-3. - Текст : непосредственный.

7. Сонг, Джимми. Python для программирования криптовалют. - Москва : Диалектика, 2020. - 368 с. - ISBN 78-5-907144-82-8. - Текст : непосредственный.

8. Маршрутизация в сети Lightning. - Текст : электронный // [сайт]. - URL: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf (дата обращения: 15.03.2024).

9. Автоматическая кластеризация адресов в сети Биткоин. - Текст : электронный // [сайт]. - URL: https://bitfury.com/content/downloads/clustering_whitepaper.pdf (дата обращения: 15.02.2024).

10. Деревья Меркла в биткойн сети. - Текст : электронный // [сайт]. - URL: https://habr.com/ru/companies/bitfury/articles/346398/ (дата обращения: 15.02.2024).

11. Кроме криптовалют: для чего еще используется блокчейн. - Текст : электронный // [сайт]. - URL: https://habr.com/ru/companies/bitfury/articles/353350/ (дата обращения: 20.04.2024).

12. Росс, К. Дж., Компьютерные сети. Нисходящий подход. - Москва : Э, 2016. - 912 с. - Текст : непосредственный.

13. Bitcoin исходный код. - Текст : электронный // [сайт]. - URL: https://github.com/bitcoin/bitcoin (дата обращения: 12.02.2024).

14. Технология Proof of Authority. - Текст : электронный // [сайт]. - URL: https://en.wikipedia.org/wiki/Proof_of_authority (дата обращения: 13.04.2024).

15. Лутц, М. Изучаем Python. 5-е издание. Том 1. - Санкт-Петербург : Диалектика, 2019. - 832 с. - ISBN 978-5-907144-52-1. - Текст : непосредственный.

16. Massias, H., Avila, X.S., Quisquater, J.-J. Design of a secure timestamping service with minimal trust requirements. - Текст : электронный // [сайт]. - 20-й симпозиум по теории информации; Бенилюкс, 1999. - URL: https://www.researchgate.net/publication/2570020_Design_Of_A_Secure_Timestamping_Service_With_Minimal_Trust_Requirement (дата обращения: 01.02.2024).

17. Haber, S., Stornetta, W.S. How to time-stamp a digital document. - Текст : электронный // Journal of Cryptology. - выпуск 3, №2, страницы 99-111, 1991. - URL: http://www.staroceans.org/e-book/Haber_Stornetta.pdf (дата обращения: 10.03.2024).

18. Bayer, D., Haber, S., Stornetta, W.S. Improving the efficiency and reliability of digital time-stamping. - В разделе II: Методы связи, безопасности и информатики, страницы 329–334, 1993.

19. Haber, S., Stornetta, W.S. Secure names for bit-strings. - В материалах 4-й конференции ACM по компьютерной и коммуникационной безопасности, страницы 28–35, апрель 1997 г.

20. Гивакс, Дж. Дж. Паттерны проектирования API. - Санкт-Петербург : Питер, 2021. - 512 с. - ISBN 978-5-4461-1984-4. - Текст : непосредственный.

21. Back, A. Hashcash - a denial of service counter-measure. - Текст : электронный // [сайт]. - URL: http://www.hashcash.org/papers/hashcash.pdf (дата обращения: 12.03.2024).

22. Merkle, R.C. Protocols for public key cryptosystems. - В учебном симпозиуме по безопасности и конфиденциальности, Компьютерное общество IEEE, страницы 122–133, апрель 1980 г.

23. Feller, W. An introduction to probability theory and its applications. - Текст : электронный // [сайт]. - 1957. - URL: https://bitcoinwords.github.io/assets/papers/an-introduction-to-probability-theory-and-its-applications.pdf (дата обращения: 15.04.2024).

24. Принципы работы Proof of Authority. - Текст : электронный // [сайт]. - URL: https://www.sciencedirect.com/topics/computer-science/proof-of-authority (дата обращения: 12.05.2024).

25. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D., Wattenhofer, R. On Scaling Decentralized Block-chains. - Текст : электронный // Journal of Computer and System Sciences. - выпуск 81, №7, страницы 1044-1059, 2016. - URL: https://eprint.iacr.org/2016/106.pdf (дата обращения: 12.05.2024).

26. Buterin, V. A Next-Generation Smart Contract and Decentralized Application Platform. - Текст : электронный // White Paper, 2014. - URL: https://ethereum.org/whitepaper/ (дата обращения: 17.05.2024).

27. Vasin, P. BlackCoin’s Proof-of-Stake Protocol v2. - Текст : электронный // [сайт]. - URL: https://blackcoin.org/blackcoin-pos-protocol-v2-whitepaper.pdf (дата обращения: 29.05.2024).

28. Могайар, У. Блокчейн для бизнеса. - Москва : Бомбора, 2018. - 320 с. - ISBN 978-5-699-98499-2. - Текст : печатный.

29. Акулич, М. Блокчейн и логистика. - Москва : Издательские решения, 2018. - 480 с. - ISBN 9785449050502. - Текст : печатный.

30. Proof of Work vs Proof of Stake: Basic Mining Guide. - Текст : электронный // [сайт]. - URL: https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/ (дата обращения: 01.05.2024).

Приложение А. Программный код

Файл: ./chain_config.py

```
import os

from dotenv import load_dotenv

from node_constants import ALL_NODES

load_dotenv(dotenv_path=os.getenv("DOTENV_FILE"))


class PostgresConfig:
    db_url = os.getenv("POSTGRES_URL")
    db_url_alembic = os.getenv("POSTGRES_URL_ALEMBIC")


class WebConfig:
    encryption_key = os.getenv("ENCRYPT_KEY")


class ChainConfig:
    address_prefix = "poa"


class NodeConfig:
    title_id = os.getenv("NODE_ID")
    private_key = os.getenv("PRIVATE_KEY")
    money_issuer_address = (
        "poa1pvffe3vy778xaqlmen8frhnsz00yl385hq756uyypndy2q2ya7wstxsuym"
    )
    block_notify_message = "notify_about_block_{random_data}"
```

Файл: ./collect_code.py

```
import os


def collect_code(base_dir, output_file, extensions, ignore_dirs, ignore_files):
    """
    Собирает код из файлов с заданными расширениями в один файл, игнорируя указанные директории.

    :param base_dir: Директория для поиска файлов.
    :param output_file: Файл, в который будет записан результат.
    :param extensions: Список расширений файлов, которые нужно включить.
    :param ignore_dirs: Список директорий, которые нужно игнорировать.
    """
    with open(output_file, 'w') as outfile:
        for dirpath, dirnames, filenames in os.walk(base_dir):
            # Удаление игнорируемых директорий из списка dirnames
            dirnames[:] = [d for d in dirnames if d not in ignore_dirs]
            for filename in filenames:
                if filename in ignore_files:
                    continue
                if any(filename.endswith(ext) for ext in extensions):
                    file_path = os.path.join(dirpath, filename)
                    outfile.write(f"\nФайл: {file_path}\n\n")
                    with open(file_path, 'r') as file:
                        outfile.write(file.read() + "\n")
# Пример использования
base_directory = '.'
output_filename = 'all_code.txt'
extensions_to_include = ['.py', '.js', ".html", ".css"]
ignore_directories = ['__pycache__', '.idea', ".venv"]
ignore_files = ['test.py', 'test1.py','test2.py','test3.py','test4.py']

collect_code(base_directory, output_filename, extensions_to_include, ignore_directories, ignore_files)
```

Файл: ./node_constants.py

```
import dataclasses
from operator import attrgetter


@dataclasses.dataclass
class NodeConstant:
    url: str
    public_key: str
    title_id: str


def sort_nodes(nodes: list[NodeConstant]) -> list[NodeConstant]:
    return sorted(nodes, key=attrgetter("title_id", "url", "public_key"))


YANDEX_NODE = NodeConstant(
    title_id="Yandex-1",
    url="http://127.0.0.1:1234",
    public_key="3076124a6d30a1125e916bbbc99a8470f25ced4c777c1fd9c14d68fb275a3641",
)
SBER_NODE = NodeConstant(
    title_id="Sber-1",
    url="http://127.0.0.1:3456",
    public_key="b0b9d00d6a948e70c05173f2888150ae3945ae619526b85e0bba796ba68c5f92",
)
GAZ_NODE = NodeConstant(
    title_id="GAZ-1",
    url="http://127.0.0.1:7890",
    public_key="b49a1e105f4e8921d980d6f65791a7d843724b92f69d92065243e917c19427f4",
)

ALL_NODES = sort_nodes([YANDEX_NODE, SBER_NODE, GAZ_NODE])
```

Файл: ./crypto/bech32.py

```
from enum import Enum


class Encoding(Enum):
    """Enumeration type to list the various supported encodings."""

    BECH32 = 1
    BECH32M = 2


CHARSET = "qpzry9x8gf2tvdw0s3jn54khce6mua7l"
BECH32M_CONST = 0x2BC830A3


def bech32_polymod(values):
    """Internal function that computes the Bech32 checksum."""
    generator = [0x3B6A57B2, 0x26508E6D, 0x1EA119FA, 0x3D4233DD, 0x2A1462B3]
    chk = 1
    for value in values:
        top = chk >> 25
        chk = (chk & 0x1FFFFFF) << 5 ^ value
        for i in range(5):
            chk ^= generator[i] if ((top >> i) & 1) else 0
    return chk


def bech32_hrp_expand(hrp):
    """Expand the HRP into values for checksum computation."""
    return [ord(x) >> 5 for x in hrp] + [0] + [ord(x) & 31 for x in hrp]


def bech32_verify_checksum(hrp, data):
    """Verify a checksum given HRP and converted data characters."""
    const = bech32_polymod(bech32_hrp_expand(hrp) + data)
```

```
    if const == 1:
        return Encoding.BECH32
    if const == BECH32M_CONST:
        return Encoding.BECH32M
    return None


def bech32_create_checksum(hrp, data, spec):
    """Compute the checksum values given HRP and data."""
    values = bech32_hrp_expand(hrp) + data
    const = BECH32M_CONST if spec == Encoding.BECH32M else 1
    polymod = bech32_polymod(values + [0, 0, 0, 0, 0, 0]) ^ const
    return [(polymod >> 5 * (5 - i)) & 31 for i in range(6)]


def bech32_encode(hrp, data, spec) -> str:
    """Compute a Bech32 string given HRP and data values."""
    combined = data + bech32_create_checksum(hrp, data, spec)
    return hrp + "1" + "".join([CHARSET[d] for d in combined])


def bech32_decode(bech):
    """Validate a Bech32/Bech32m string, and determine HRP and data."""
    if (any(ord(x) < 33 or ord(x) > 126 for x in bech)) or (
        bech.lower() != bech and bech.upper() != bech
    ):
        return None, None, None
    bech = bech.lower()
    pos = bech.rfind("1")
    if pos < 1 or pos + 7 > len(bech) or len(bech) > 90:
        return None, None, None
    if not all(x in CHARSET for x in bech[pos + 1 :]):
        return None, None, None
    hrp = bech[:pos]
    data = [CHARSET.find(x) for x in bech[pos + 1 :]]
    spec = bech32_verify_checksum(hrp, data)
    if spec is None:
        return None, None, None
    return hrp, data[:-6], spec


def convertbits(data, frombits, tobits, pad=True) -> list[int] | None:
    """General power-of-2 base conversion."""
    acc = 0
    bits = 0
    ret = []
    maxv = (1 << tobits) - 1
    max_acc = (1 << (frombits + tobits - 1)) - 1
    for value in data:
        if value < 0 or (value >> frombits):
            return None
        acc = ((acc << frombits) | value) & max_acc
        bits += frombits
        while bits >= tobits:
            bits -= tobits
            ret.append((acc >> bits) & maxv)
    if pad:
        if bits:
            ret.append((acc << (tobits - bits)) & maxv)
    elif bits >= frombits or ((acc << (tobits - bits)) & maxv):
        return None
    return ret


def decode(hrp, addr):
    """Decode a segwit address."""
    hrpgot, data, spec = bech32_decode(addr)
    if hrpgot != hrp:
        return None, None
    decoded = convertbits(data[1:], 5, 8, False)
    if decoded is None or len(decoded) < 2 or len(decoded) > 40:
        return None, None
    if data[0] > 16:
```

```
        return None, None
    if data[0] == 0 and len(decoded) != 20 and len(decoded) != 32:
        return None, None
    if (
        data[0] == 0
        and spec != Encoding.BECH32
        or data[0] != 0
        and spec != Encoding.BECH32M
    ):
        return None, None
    return data[0], decoded


def encode(hrp, witver, witprog) -> str | None:
    """Encode a segwit address."""
    spec = Encoding.BECH32 if witver == 0 else Encoding.BECH32M
    ret = bech32_encode(hrp, [witver] + convertbits(witprog, 8, 5), spec)
    if decode(hrp, ret) == (None, None):
        return None
    return ret
```

Файл: ./crypto/converter.py

```
from base64 import b64decode

from chain.constants import (
    PUBLIC_KEY_LENGTH,
    AMOUNT_LENGTH,
    TIMESTAMP_LENGTH,
)
from chain.transaction import calculate_transaction_hash
from crypto.transfer import public_key_to_address
from node.models.transaction import TransactionModel


def hex_to_int_list(hex_string: str) -> list[int]:
    return [int(i) for i in bytes.fromhex(hex_string)]


def extract_address_from_transaction(transaction: list[int]) -> tuple[list[int], str]:
    data = transaction[:PUBLIC_KEY_LENGTH]
    transaction[:] = transaction[PUBLIC_KEY_LENGTH:]
    return data, public_key_to_address(data)


def extract_amount_from_transaction(transaction: list[int]) -> int:
    data = transaction[:AMOUNT_LENGTH]
    transaction[:] = transaction[AMOUNT_LENGTH:]
    return int.from_bytes(bytes(data), "big")


def extract_timestamp_from_transaction(transaction: list[int]) -> int:
    data = transaction[:TIMESTAMP_LENGTH]
    transaction[:] = transaction[TIMESTAMP_LENGTH:]

    return int.from_bytes(bytes(data), "big")


def normalize_transaction(transaction: str) -> list[int]:
    return list(b64decode(transaction))


def expand_transaction_from_request(transaction: list[int]) -> TransactionModel:
    from_pub_key, from_address = extract_address_from_transaction(
        transaction=transaction
    )
    to_pub_key, to_address = extract_address_from_transaction(transaction=transaction)
    amount = extract_amount_from_transaction(transaction=transaction)
    timestamp = extract_timestamp_from_transaction(transaction=transaction)

    transaction_model = TransactionModel(
        sender_address=from_address,
```

```
        recipient_address=to_address,
        amount=amount,
        timestamp=timestamp,
    )
    transaction_model.transaction_hash = calculate_transaction_hash(
        transaction=transaction_model
    )
    return transaction_model
```

Файл: ./crypto/__init__.py

Файл: ./crypto/generate_wallet.py

```
from hashlib import pbkdf2_hmac, sha256
from mnemonic import Mnemonic

from nacl.bindings import crypto_sign_seed_keypair

from chain_config import ChainConfig
from crypto import bech32
from crypto.converter import hex_to_int_list


def generate_mnemo_words() -> str:
    mnemo = Mnemonic("english")
    return mnemo.generate(strength=256)


def to_seed(words: str) -> bytes:
    return pbkdf2_hmac("sha512", words.encode(), "mnemonic".encode(), 2048)


def to_unsigned_list_int(seed: bytes) -> list[int]:
    to_sha256 = sha256(seed).hexdigest()
    return [int(to_sha256[i : i + 2], 16) for i in range(0, len(to_sha256), 2)]


def generate_pk_sk(unsigned_list_int: list[int]) -> tuple[str, str]:
    public_key, secret_key = crypto_sign_seed_keypair(bytes(unsigned_list_int))
    return public_key.hex(), secret_key.hex()


def generate_wallet_address(public_key: str, prefix: str) -> str:
    convert_bits = bech32.convertbits(hex_to_int_list(public_key), 8, 5, True)
    return bech32.bech32_encode(prefix, convert_bits, 0)


def generate_wallet() -> tuple[str, str, str, str]:
    mnemo = generate_mnemo_words()
    seed = to_seed(mnemo)
    unsigned_int = to_unsigned_list_int(seed)
    pk, sk = generate_pk_sk(unsigned_int)
    address = generate_wallet_address(pk, prefix=ChainConfig.address_prefix)
    return address, mnemo, pk, sk


def restore_wallet(mnemonic: str) -> tuple[str, str, str, str]:
    seed = to_seed(mnemonic)
    unsigned_int = to_unsigned_list_int(seed)
    pk, sk = generate_pk_sk(unsigned_int)
    address = generate_wallet_address(pk, prefix=ChainConfig.address_prefix)
    return address, mnemonic, pk, sk
```

Файл: ./crypto/transfer.py

```
import time
from typing import Union
from base64 import b64encode
from random import random
```

```
from nacl.bindings.crypto_sign import crypto_sign

from chain.timestamps import get_current_accurate_timestamp
from chain_config import ChainConfig
from crypto import bech32
from crypto.bech32 import Encoding


def address_to_public_key(address: str) -> list[int]:
    prefix, list_int, encoding = bech32.bech32_decode(address)
    public_key = bech32.convertbits(list_int, 5, 8, False)
    return public_key


def public_key_to_address(public_key: list[int]) -> str:
    list_int = bech32.convertbits(public_key, 8, 5, True)
    address = bech32.bech32_encode(
        ChainConfig.address_prefix, list_int, Encoding.BECH32
    )
    return address


def to_2(value: int) -> list[int]:
    return [(value >> 8) & 0xFF, value & 0xFF]


def to_4(value: int) -> list[int]:
    res = [
        (value & 0xFF000000) >> 24,
        (value & 0x00FF0000) >> 16,
        (value & 0x0000FF00) >> 8,
        (value & 0x000000FF) >> 0,
    ]
    return res


def to_8(value: int) -> list[int]:
    res = [
        (value & 0xFF00000000000000) >> 56,
        (value & 0x00FF000000000000) >> 48,
        (value & 0x0000FF0000000000) >> 40,
        (value & 0x000000FF00000000) >> 32,
        (value & 0x00000000FF000000) >> 24,
        (value & 0x0000000000FF0000) >> 16,
        (value & 0x000000000000FF00) >> 8,
        (value & 0x00000000000000FF) >> 0,
    ]
    return res


def set_list(trx: list[int], address: list[int]) -> None:
    trx += address


def set_amount(trx: list[int], amount: Union[int, float]) -> None:
    amount_to_bytes: bytes = int(amount).to_bytes(8, "big")
    to_list: list[int] = [int(i) for i in amount_to_bytes]

    trx += to_list


def sign(trx: list[int], sk: list[int]) -> None:
    signature = crypto_sign(bytes(trx), bytes(sk))
    trx += signature[:64]


def sign_transaction(trx: list[int], sk: list[int]) -> None:
    timestamp = get_current_accurate_timestamp()

    trx += to_8(timestamp)
    trx += to_4(int(random() * 9999))
```

```
    trx.append(0)
    sign(trx, sk)


def transfer_coins(
    public_key: list[int],
    private_key: list[int],
    target_address: str,
    amount: Union[int, float],
) -> str:
    trx: list[int] = []

    set_list(trx, public_key)

    to_pk = address_to_public_key(target_address)
    set_list(trx, to_pk)
    set_amount(trx, amount)

    sign_transaction(trx, private_key)

    return b64encode(bytes(trx)).decode()
```

Файл: ./crypto/poa_mnemonic.py

```
import dataclasses
from mnemonic import Mnemonic

from crypto.generate_wallet import restore_wallet


@dataclasses.dataclass
class PoaKeys:
    private_key: str
    public_key: str
    address: str


def get_data_from_mnemonic(mnemonic_phrase: str) -> PoaKeys:
    address, mnemonic, pk, sk = restore_wallet(mnemonic=mnemonic_phrase)
    return PoaKeys(private_key=sk, public_key=pk, address=address)


def is_valid_mnemonic(mnemonic_phrase: str) -> bool:
    return Mnemonic().check(mnemonic=mnemonic_phrase)
```

Файл: ./crypto/sign.py

```
import base64

from cryptography.hazmat.primitives.asymmetric import ed25519
from nacl.bindings import crypto_sign, crypto_sign_open
from nacl.exceptions import BadSignatureError

from chain.constants import PUBLIC_KEY_LENGTH, TRANSACTION_PURE_LENGTH
from crypto.converter import hex_to_int_list


def sign_message(message: str, private_key: str) -> str:
    encoded_message = message.encode()
    result = crypto_sign(
        message=encoded_message, sk=bytes(hex_to_int_list(private_key))
    )
    return base64.b64encode(result[: -len(encoded_message)]).decode()


def text_sign_verify(signature: str, original_message: str, public_key: str) -> bool:
    public_key = bytes.fromhex(public_key)
    base64_sig = base64.b64decode(signature)

    base64_sig += original_message.encode()
    try:
```

```
        result = crypto_sign_open(signed=base64_sig, pk=public_key)
    except BadSignatureError:
        return False

    return result.decode() == original_message


def verify_transaction_sign(transaction: list[int]) -> bool:
    public_key = transaction[:PUBLIC_KEY_LENGTH]
    original_msg = transaction[:TRANSACTION_PURE_LENGTH]
    sign = transaction[TRANSACTION_PURE_LENGTH:]

    pk2 = ed25519.Ed25519PublicKey.from_public_bytes(bytes(public_key))

    try:
        pk2.verify(bytes(sign), bytes(original_msg))
        return True
    except Exception as e:
        return False
```

Файл: ./web/encryption.py

```
from cryptography.fernet import Fernet

from chain_config import WebConfig


def encrypt_text(text: str) -> str:
    cipher_suite = Fernet(bytes.fromhex(WebConfig.encryption_key))
    encrypted_text = cipher_suite.encrypt(text.encode())
    return encrypted_text.decode()


def decrypt_text(encrypted_text: str) -> str:
    cipher_suite = Fernet(bytes.fromhex(WebConfig.encryption_key))
    decrypted_text = cipher_suite.decrypt(encrypted_text).decode()
    return decrypted_text
```

Файл: ./web/constants.py

```
HTML_PATH = "web/static/html/"
```

Файл: ./web/__init__.py

Файл: ./web/utils.py

```
def format_balance(balance: int) -> float:
    return round(balance / 100, 2)
```

Файл: ./web/file_response.py

```
from web.constants import HTML_PATH


def get_html_file_data(filename: str) -> str:
    with open(f"{HTML_PATH}{filename}", "r") as f:
        return f.read()
```

Файл: ./web/app.py

```
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles

from .routers import (
    index_router,
    mnemonic_router,
```

```
    wallet_router,
    sign_router,
    transaction_router,
    address_watcher_router,
    blocks_watcher_router,
)


app = FastAPI()
app.include_router(index_router)
app.include_router(mnemonic_router)
app.include_router(wallet_router)
app.include_router(sign_router)
app.include_router(transaction_router)
app.include_router(address_watcher_router)
app.include_router(blocks_watcher_router)
app.mount("/css", StaticFiles(directory="web/static/css/"), name="css")
app.mount("/html", StaticFiles(directory="web/static/html/"), name="html")
app.mount("/js", StaticFiles(directory="web/static/js/"), name="js")
app.mount("/images", StaticFiles(directory="web/static/images/"), name="images")


"""
Это все использовать в описании.

Есть список авторитетных нод. Они записаны в каждой ноде.
Когда юзер отправляет транзакцию то она высылается в ноду по загруженности или удаленности.
Распределением транзакций занимается балансировщик, он тупа перенаправляет запросы.
Также транзакцию можно послать в ноду напрямую.

При запуске ноды она идет во все остальные ноды и запрашивает у них данные блоков и транзакций
и синхронизируется. При запуске ноды выбирается нода с самой длинной цепочкой блоков и каждая транза
проверяется на валидность, чтобы отсеять жуликов. Если найдена ошибка ищется нода дальше.

Ноды в порядке очереди генерируют блоки и высылают их на проверку другим нодам вместе с транзакциями.
Пока не пришел подтвержденный блок от предыдущей ноды в списке генерация блока не начинается.

Перед рассылкой блока нода отправляет всем уведомление мол скоро разошлю блок, и ждет ответа
от всех доступных, дальше рассылает блок всем. Нода становится готовой после того как она
получила уведомление о предстоящей рассылке блока.
Нода проверяет каждую другую на готовность и если какая то не готова или недоступна,
то от нее блок не ждется а ждется от предыдущей в списке итд.

(че делать с жуликами которые сказали что готовы а сами блок не генерируют все же встанет)
(видимо решать ручными проверками и исключениями из списка нод)

Перед отправкой блока все транзы перепровряются на баланс юзера, чтобы юзер не мог с двух нод отпраивть параллельно
Ноды принимают блок и проверяют там все транзы чтобы не было жульничества.
Если ноду все устраивает она добавляет блок себе.
Если есть ошибки то блок нода не добавляет.
И очередь двигается дальше.
Получается если нода сжульничала то нормальные ноды ее блок не примут, а плохие примут.
из за чего их блокчейны разойдутся (они не смогут принять в дальнейшем блоки друг друга)
но в этом и смысл останется выбор, а в битке например
если 51% то все никуда не деться все примут блок.
Пока нода не готова она сама генерировать блоки не начинает.
Это нужно чтобы обе ноды подряд не начали генерить блок после получения его от предыдущей ноды.
При перезапуске ноды она пересинхронизируется.

В блок попадают неподтвержденные транзакции в конкретной ноде.

Также есть обычные ноды которые синхронизируются с нодами валидаторами и могут отправлять транзакции, отправляя
их в балансировщик или ноды валидаторы. Для децентрализации!
"""


Файл: ./web/routers/address_watcher.py

from fastapi import APIRouter
from starlette.requests import Request
from starlette.responses import HTMLResponse

from web.file_response import get_html_file_data
```

```
router = APIRouter()


@router.get("/address/{address}")
async def watch_address(request: Request):
    return HTMLResponse(content=get_html_file_data("address.html"))
```

Файл: ./web/routers/index.py

```
from fastapi import APIRouter
from starlette.responses import HTMLResponse, Response

from web.file_response import get_html_file_data

router = APIRouter()


@router.get("/")
async def index():
    return HTMLResponse(content=get_html_file_data("login.html"))


@router.get("/favicon.ico")
async def favicon():
    with open("web/static/images/favicon.ico", "rb") as f:
        return Response(content=f.read(), media_type="image/png")
```

Файл: ./web/routers/transaction.py

```
import base64

import aiohttp
import cryptography
from fastapi import APIRouter
from starlette.requests import Request
from starlette.responses import HTMLResponse

from crypto.converter import hex_to_int_list
from crypto.transfer import transfer_coins
from node.api.block import get_last_block_number_from_node
from node.api.status import is_node_online
from node.api.transaction import (
    send_transaction_to_mempool,
    get_transactions_from_node,
    get_transaction_from_node,
    get_transactions_by_block_from_node,
)
from node.utils import get_node_by_id
from node_constants import ALL_NODES
from web.encryption import decrypt_text
from web.file_response import get_html_file_data

router = APIRouter()


@router.post("/create_transaction")
async def create_transaction(request: Request):
    transaction_data = await request.json()

    public_key = list(
        bytes(hex_to_int_list(decrypt_text(transaction_data["publicKey"])))
    )
    private_key = list(
        bytes(hex_to_int_list(decrypt_text(transaction_data["privateKey"])))
    )

    try:
        encoded_transaction = transfer_coins(
            public_key=public_key,
            private_key=private_key,
```

```
            target_address=transaction_data["address"],
            amount=int(float(transaction_data["amount"]) * 100),
        )
    except Exception as e:
        return {"status": False, "message": str(e)}

    return {"status": True, "encoded_transaction": encoded_transaction}


@router.post("/send_transaction")
async def send_transaction(request: Request):
    transaction_data = await request.json()

    encoded_transaction = transaction_data["data"]
    node = transaction_data["node"]

    node = get_node_by_id(node_id=node)

    session = aiohttp.ClientSession()
    is_online = await is_node_online(url=node.url, session=session)

    if not is_online:
        await session.close()
        return {
            "status": False,
            "description": "Node not online",
        }

    return {
        "status": True,
        "result": await send_transaction_to_mempool(
            url=node.url, session=session, transaction_data=encoded_transaction
        ),
    }


@router.post("/get_transactions")
async def get_transactions(request: Request):
    request_data = await request.json()

    node = request_data["node"]

    try:
        address = decrypt_text(request_data["address"])
    except cryptography.fernet.InvalidToken:
        address = request_data["address"]
    transaction_type = request_data["type"]
    node = get_node_by_id(node_id=node)

    session = aiohttp.ClientSession()
    is_online = await is_node_online(url=node.url, session=session)

    if not is_online:
        await session.close()
        return {
            "transactions": [],
        }

    res = {"transactions": []}
    transactions = (
        await get_transactions_from_node(
            session=session,
            url=node.url,
            address=address,
            transaction_type=transaction_type,
        )
    )["transactions"]

    for tr in transactions:
        tr["is_income"] = tr["recipient_address"] == address
        res["transactions"].append(tr)

    await session.close()
```

```
    return res


@router.post("/get_transactions_by_block")
async def get_transactions_by_block_handler(request: Request):
    request_data = await request.json()

    node = request_data["node"]
    block_hash = request_data["block_hash"]

    node = get_node_by_id(node_id=node)

    session = aiohttp.ClientSession()
    is_online = await is_node_online(url=node.url, session=session)

    if not is_online:
        await session.close()
        return {
            "transactions": [],
        }

    res = {"transactions": []}
    transactions = (
        await get_transactions_by_block_from_node(
            session=session,
            url=node.url,
            block_hash=block_hash,
        )
    )["transactions"]

    for tr in transactions:
        res["transactions"].append(tr)

    await session.close()
    return res


@router.get("/transaction/{transaction}")
async def watch_transaction(request: Request):
    return HTMLResponse(content=get_html_file_data("transaction.html"))


@router.post("/get_transaction_data")
async def get_transaction_data_handler(request: Request):
    request_data = await request.json()
    node = request_data["node"]
    transaction = request_data["transaction"]

    node = get_node_by_id(node_id=node)
    session = aiohttp.ClientSession()

    if node is None:
        is_online = False
    else:
        is_online = await is_node_online(url=node.url, session=session)

    nodes = []

    for _node in ALL_NODES:
        _is_node_online = await is_node_online(url=_node.url, session=session)
        if _is_node_online:
            blocks_count = await get_last_block_number_from_node(
                url=_node.url, session=session
            )
        else:
            blocks_count = -1

        nodes.append(
            {
                "title_id": _node.title_id,
                "is_online": _is_node_online,
                "blocks_count": blocks_count,
            }
```

```
        )

    if not is_online:
        await session.close()
        return {
            "status": False,
            "node_none": node is None,
            "description": "Node not online",
            "transaction_data": None,
            "balance": "Ошибка подключения",
            "nodes": nodes,
        }

    transaction_data = await get_transaction_from_node(
        url=node.url, session=session, transaction_hash=transaction
    )

    await session.close()
    return {
        "status": True,
        "node_none": node is None,
        "transaction_data": transaction_data,
        "nodes": nodes,
    }
```

Файл: ./web/routers/__init__.py

```
from .index import router as index_router
from .mnemonic import router as mnemonic_router
from .wallet import router as wallet_router
from .sign import router as sign_router
from .transaction import router as transaction_router
from .address_watcher import router as address_watcher_router
from .blocks_watcher import router as blocks_watcher_router
```

Файл: ./web/routers/blocks_watcher.py

```
import aiohttp
from fastapi import APIRouter
from starlette.requests import Request
from starlette.responses import HTMLResponse

from node.api.block import (
    get_blocks_from_node,
    get_last_block_number_from_node,
    get_block_from_node,
    get_block_by_number_from_node,
)
from node.api.status import is_node_online
from node.utils import get_node_by_id
from node_constants import ALL_NODES
from web.file_response import get_html_file_data

router = APIRouter()


@router.get("/blocks")
async def watch_blocks(request: Request):
    return HTMLResponse(content=get_html_file_data("blocks.html"))


@router.post("/blocks_latest")
async def latest_blocks(request: Request):
    request_data = await request.json()
    node = request_data["node"]
    limit = request_data.get("limit", 100)
    offset = request_data.get("offset", 0)

    node = get_node_by_id(node_id=node)
    session = aiohttp.ClientSession()
```

```
    if node is None:
        await session.close()
        return {"blocks": [], "total_count": 0}

    if not await is_node_online(url=node.url, session=session):
        await session.close()
        return {"blocks": [], "total_count": 0}

    result_blocks = await get_blocks_from_node(
        url=node.url, session=session, limit=limit, offset=offset
    )
    await session.close()
    return result_blocks


@router.get("/block/{block_hash}")
async def watch_block(request: Request):
    return HTMLResponse(content=get_html_file_data("block.html"))


@router.get("/transaction/{transaction}")
async def watch_transaction(request: Request):
    return HTMLResponse(content=get_html_file_data("transaction.html"))


@router.post("/get_block_data")
async def get_block_data_handler(request: Request):
    request_data = await request.json()
    node = request_data["node"]
    block_hash = request_data["block"]

    node = get_node_by_id(node_id=node)
    session = aiohttp.ClientSession()

    if node is None:
        is_online = False
    else:
        is_online = await is_node_online(url=node.url, session=session)

    nodes = []

    for _node in ALL_NODES:
        _is_node_online = await is_node_online(url=_node.url, session=session)
        if _is_node_online:
            blocks_count = await get_last_block_number_from_node(
                url=_node.url, session=session
            )
        else:
            blocks_count = -1

        nodes.append(
            {
                "title_id": _node.title_id,
                "is_online": _is_node_online,
                "blocks_count": blocks_count,
            }
        )

    if not is_online:
        await session.close()
        return {
            "status": False,
            "node_none": node is None,
            "description": "Node not online",
            "block_data": None,
            "balance": "Ошибка подключения",
            "nodes": nodes,
        }

    transaction_data = await get_block_from_node(
        url=node.url, session=session, block_hash=block_hash
    )
```

```
    await session.close()
    return {
        "status": True,
        "node_none": node is None,
        "block_data": transaction_data,
        "nodes": nodes,
    }


@router.post("/get_block_by_number")
async def get_block_data_by_number_handler(request: Request):
    request_data = await request.json()
    node = request_data["node"]
    block_number = request_data["block_number"]

    node = get_node_by_id(node_id=node)
    session = aiohttp.ClientSession()

    if node is None:
        is_online = False
    else:
        is_online = await is_node_online(url=node.url, session=session)

    nodes = []

    for _node in ALL_NODES:
        _is_node_online = await is_node_online(url=_node.url, session=session)
        if _is_node_online:
            blocks_count = await get_last_block_number_from_node(
                url=_node.url, session=session
            )
        else:
            blocks_count = -1

        nodes.append(
            {
                "title_id": _node.title_id,
                "is_online": _is_node_online,
                "blocks_count": blocks_count,
            }
        )

    if not is_online:
        await session.close()
        return {
            "status": False,
            "node_none": node is None,
            "description": "Node not online",
            "block_data": None,
            "balance": "Ошибка подключения",
            "nodes": nodes,
        }

    transaction_data = await get_block_by_number_from_node(
        url=node.url, session=session, block_number=block_number
    )

    await session.close()
    return {
        "status": True,
        "node_none": node is None,
        "block_data": transaction_data,
        "nodes": nodes,
    }


Файл: ./web/routers/mnemonic.py

from mnemonic import Mnemonic
from fastapi import APIRouter

from chain.constants import PUBLIC_KEY_LENGTH
from crypto.poa_mnemonic import get_data_from_mnemonic, is_valid_mnemonic
```

```
from web.encryption import encrypt_text
from web.models.mnemonic import UserInputMnemonic

router = APIRouter()


@router.get("/new_mnemonic")
async def generate_mnemonic():
    return {"mnemonic": Mnemonic().generate()}


@router.post("/get_data_from_mnemonic")
async def get_data_from_mnemonic_handler(user_mnemonic: UserInputMnemonic):
    mnemonic_phrase = user_mnemonic.mnemonic

    if not is_valid_mnemonic(mnemonic_phrase=mnemonic_phrase):
        return {"ok": False, "error": "Invalid mnemonic phrase"}

    mnemonic_data = get_data_from_mnemonic(mnemonic_phrase=mnemonic_phrase)

    return {
        "ok": True,
        "wallet_data": {
            "private_key": encrypt_text(mnemonic_data.private_key),
            "public_key": encrypt_text(mnemonic_data.public_key),
            "address": encrypt_text(mnemonic_data.address),
        },
    }
```

Файл: ./web/routers/sign.py

```
from fastapi import APIRouter
from starlette.requests import Request

from crypto.sign import sign_message
from web.encryption import decrypt_text

router = APIRouter()


@router.post("/sign")
async def sign_handler(request: Request):
    request_data = await request.json()

    return {
        "result": sign_message(
            message=request_data["message"],
            private_key=decrypt_text(request_data["private_key"]),
        )
    }
```

Файл: ./web/routers/wallet.py

```
import aiohttp
import cryptography
from cryptography.fernet import InvalidToken
from fastapi import APIRouter
from starlette.requests import Request
from starlette.responses import HTMLResponse

from node.api.block import get_last_block_number_from_node
from node.api.status import is_node_online
from node.api.user import get_address_balance
from node.utils import get_node_by_id
from node_constants import ALL_NODES
from web.encryption import decrypt_text
from web.file_response import get_html_file_data


router = APIRouter()
```

```
@router.get("/wallet")
async def wallet_handler():
    return HTMLResponse(content=get_html_file_data("wallet.html"))


@router.post("/get_wallet_data")
async def get_wallet_data_handler(request: Request):
    request_data = await request.json()
    node = request_data["node"]

    try:
        address = decrypt_text(request_data["address"])
    except cryptography.fernet.InvalidToken:
        address = request_data["address"]

    node = get_node_by_id(node_id=node)

    session = aiohttp.ClientSession()

    if node is None:
        is_online = False
    else:
        is_online = await is_node_online(url=node.url, session=session)

    nodes = []

    for _node in ALL_NODES:
        _is_node_online = await is_node_online(url=_node.url, session=session)
        if _is_node_online:
            blocks_count = await get_last_block_number_from_node(
                url=_node.url, session=session
            )
        else:
            blocks_count = -1

        nodes.append(
            {
                "title_id": _node.title_id,
                "is_online": _is_node_online,
                "blocks_count": blocks_count,
            }
        )

    if not is_online:
        await session.close()
        return {
            "status": False,
            "node_none": node is None,
            "description": "Node not online",
            "address": address,
            "balance": "Ошибка подключения",
            "nodes": nodes,
        }

    balance = await get_address_balance(url=node.url, session=session, address=address)
    if isinstance(balance, int):
        balance = round(balance / 100, 2)

    await session.close()
    return {
        "status": True,
        "node_none": node is None,
        "address": address,
        "balance": balance,
        "nodes": nodes,
    }
```

Файл: ./web/models/__init__.py

Файл: ./web/models/mnemonic.py

```
import pydantic


class UserInputMnemonic(pydantic.BaseModel):
    mnemonic: str
```

Файл: ./web/static/css/wallet.css

```
.nav-link {
    background: #444;
    color: white;
}

.nav-link:hover {
    color: white;
}

.error {
    border: 1px solid red;
}

.modal-content {
    max-width: 90%;
    word-wrap: break-word;
}


.transaction-type-arrow {
    font-size: 24px; /* Размер стрелочки */
}

.transaction-amount {
    font-weight: bold; /* Жирный шрифт для суммы */
}

.transaction-date {
    font-size: smaller; /* Уменьшить размер шрифта для даты */
}

.green-arrow {
    color: green; /* Зеленый цвет для стрелочки */
}

.red-arrow {
    color: red; /* Зеленый цвет для стрелочки */
}
```

Файл: ./web/static/css/index.css

```
body {
    background-color: #121212;
    color: #fff;
    background-image: url('../images/logback.jpg');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: center;
}

.wallet-container {
    margin-top: 100px;
}

.card {
    background-color: rgba(0, 0, 0, 0.7);
}

.card-title {
    color: #fff;
```

```
}

.form-control {
    background-color: #444;
    color: #fff;
    border-color: #666;
}

.btn-primary {
    background-color: #007bff;
    border-color: #007bff;
}

.btn-primary:hover {
    background-color: #0056b3;
    border-color: #0056b3;
}

.btn-success {
    background-color: #28a745;
    border-color: #28a745;
}

.form-control:focus {
    background-color: #444 !important;
    color: white;
}


html * {
    font-family: "JetBrains Mono";
}


.top {
    position: absolute;
    top: 10px;
    left: 20px;
}

.corner-image {
    margin-bottom: 10px;
    width: 80px;
    border-radius: 5px;
}

#sendlink, #signlink, #allTrans, #sent, #receive {
    cursor: pointer;
}


.toast{
    z-index: 9999999;
}
```

Файл: ./web/static/js/transaction.js

```
const globalTransaction = window.location.pathname.replace("/transaction/", "")


async function setDataTrans() {
    let nodes = document.getElementById("nodes")

    let currentNode = localStorage.getItem(uniqueKey("node"))

    let nodeButton = document.getElementById("dropdownMenuButton")

    let transactionData = await (await fetch("/get_transaction_data", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
            node: currentNode,
```

```
            transaction: globalTransaction
        })
    })).json()

    if (!transactionData["status"] && !transactionData["node_none"]) {
        document.getElementById("errorNodeText").innerText = `Не получилось подключиться к ноде ${currentNode}. Выберите другую.`
        $("#errorPopupNode").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#errorPopupNode").toast('hide'); // Скрываем popup через 2 секунды
        }, 2000);
    }

    if (transactionData["status"]) {
        document.getElementById("successNodeText").innerText = `Успешно подключились к ноде ${currentNode}`
        $("#successChangeNode").toast('show');
        setTimeout(function () {
            $("#successChangeNode").toast('hide');
        }, 2000);
    }

    if (!currentNode) {
        let active_nodes = []
        for (let node of transactionData["nodes"]) {
            if (node["is_online"]) {
                active_nodes.push(node)
            }
        }

        let randomNode;
        if (active_nodes.length > 0) {
            randomNode = active_nodes[Math.floor(Math.random() * active_nodes.length)];
        } else {
            randomNode = transactionData["nodes"][Math.floor(Math.random() * transactionData["nodes"].length)];
        }
        nodeButton.textContent = randomNode["title_id"]
        localStorage.setItem(uniqueKey("node"), randomNode["title_id"])
        await setDataTrans()
    } else {
        nodeButton.textContent = currentNode
        nodes.innerHTML = ""

        for (const node of transactionData["nodes"]) {
            let status = node["is_online"] ? "" : ""
            nodes.innerHTML += `<a class="dropdown-item" href="#"
        onclick="setNode('${node["title_id"]}')">${status} ${node["title_id"]} (Блоков: ${node["blocks_count"]})</a>`
        }
    }

    if (!transactionData["transaction_data"]) {
        return
    }

    let transactionObj = transactionData["transaction_data"]["transaction"]

    console.log(transactionObj)
    let transactionTitleText = document.getElementById("transTitle")
    if (!transactionObj) {
        transactionTitleText.innerText = "Транзакция не найдена"
    } else {
        let transactionText = document.getElementById("transData")

        transactionTitleText.innerText = "Данные транзакции:"

        console.log(transactionObj)
        transactionText.innerHTML = `
        <br>
        <p>Хеш: ${transactionObj['transaction_hash']}</p>
        <p>От: <a style="color: white" href="/address/${transactionObj["sender_address"]}">
${transactionObj["sender_address"]}</a></p>
        <p>Кому: <a style="color: white" href="/address/${transactionObj["recipient_address"]}"
        >${transactionObj["recipient_address"]}</a></p>
        <p>Сумма: ${transactionObj['amount'] / 100} POA</p>
        <p>
```

```
        Номер блока: <a href="${transactionObj['block_hash'] ?
            '/block/' + transactionObj['block_hash'] : '#'}">
${transactionObj['block_number']}</a></p>
        <p>Ответственная нода: ${transactionObj['authority_id']}</p>
        <p>${timeConverter(transactionObj['timestamp'])}</p>
        `
    }

}

async function setNode(node) {
    let nodeButton = document.getElementById("dropdownMenuButton")
    nodeButton.textContent = node
    localStorage.setItem(uniqueKey("node"), node)

    await setDataTrans()
}
```

Файл: ./web/static/js/index.js

```
if (!sessionStorage.tabId) {
    sessionStorage.tabId = Date.now();
}

function uniqueKey(key) {
    return '_' + key;
}

async function createNew() {
    let mnemonicInput = document.getElementById("mnemonic")
    let mnemonicInputLabel = document.getElementById("mnemid")

    mnemonicInput.value = (await (await fetch("/new_mnemonic")).json())["mnemonic"]
    mnemonicInputLabel.textContent = "Сохраните сгенерированную мнемоническую фразу:"
}

async function login() {
    let mnemonicInput = document.getElementById("mnemonic").value.trim()
    let res = await fetch("/get_data_from_mnemonic",
        {
            method: 'POST', headers: {
                'Content-Type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({mnemonic: mnemonicInput})
        })

    res = await res.json()

    if (!res["ok"]) {
        badLogin()
        return
    }
    res = res["wallet_data"]

    localStorage.setItem(uniqueKey('privateKey'), res["private_key"]);
    localStorage.setItem(uniqueKey('publicKey'), res["public_key"]);
    localStorage.setItem(uniqueKey('address'), res["address"]);
    window.location.replace("/wallet");

}


async function logout() {
    localStorage.removeItem(uniqueKey('privateKey'))
    localStorage.removeItem(uniqueKey('publicKey'))
    localStorage.removeItem(uniqueKey('address'))
    window.location.replace("/");
}

function badLogin() {
    $("#errorPopup").toast('show');
    setTimeout(function () {
        $("#errorPopup").toast('hide');
```

```
    }, 2000);
}

function onloadAdaptive() {
    if (window.innerWidth < 900) {
        document.getElementById("logoIMG").hidden = true
        document.getElementById("setAllBTN").className = "btn btn-secondary btn-sm"
        document.getElementById("sendlink").className = "nav-link active btn-sm"
        document.getElementById("signlink").className = "nav-link btn-sm"

        document.getElementById("allTrans").className = "nav-link active btn-sm"
        document.getElementById("receive").className = "nav-link btn-sm"
        document.getElementById("sent").className = "nav-link btn-sm"

        document.getElementById("bigScreenP").hidden = true
    }
}


function timeConverter(UNIX_timestamp) {
    let a = new Date(UNIX_timestamp / 1000);
    let months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
    let year = a.getFullYear();
    let month = months[a.getMonth()];
    let date = a.getDate();
    let hour = a.getHours().toString();
    if (hour.length < 2){
        hour = "0" + hour
    }
    let min = a.getMinutes().toString();
    if (min.length < 2){
        min = "0" + min
    }
    let sec = a.getSeconds().toString();
    if (sec.length < 2){
        sec = "0" + sec
    }
    let time = date + ' ' + month + ' ' + year + ' ' + hour + ':' + min + ':' + sec;
    return time;
}

function createTransaction(transaction) {
    let arrow = "&larr;"
    let amount = `-${transaction['amount'] / 100} POA`
    let address = truncateAddress(transaction['recipient_address'], 10)
    let fullAddress = transaction['recipient_address']
    let arrowClass = "red-arrow"
    if (transaction["is_income"]) {
        arrow = "&rarr;"
        amount = `+${transaction['amount'] / 100} POA`
        address = truncateAddress(transaction["sender_address"], 10)
        fullAddress = transaction['sender_address']
        arrowClass = "green-arrow"
    }

    let code = `<div class="card mb-1">
            <div class="card-body">
                <div class="container">
                    <div class="row mb-2">
                        <p class="card-text">
                            <span class="transaction-type-arrow ${arrowClass}">${arrow}</span>
                            <span class="transaction-amount">${amount}</span>
                        </p>
                    </div>
                    <div class="row mb-2">
                        <p class="card-text">
                            <span class="transaction-address">
                            <a style="color: white" href="/address/${fullAddress}">Адрес: ${address}</a>
                            </span>

                        </p>
                    </div>
                    <div class="row mb-2">
```

```
                    <p class="card-text transaction-date">${timeConverter(transaction['timestamp'])}</p>
                </div>
                <div class="row">
                    <p class="card-text"><button
                     class="btn-sm btn-primary">
                     <a style="color: white"
                     href="/transaction/${transaction['transaction_hash']}">
                     Перейти к транзакции</a></button></p>
                </div>
            </div>
        </div>

    </div>`

    const newTrans = document.createElement('div');
    newTrans.class = "card mb-1"
    newTrans.innerHTML = code
    return newTrans
}

function truncateAddress(address) {
    return address.substring(0, 5) + '...' + address.substring(address.length - 7, address.length);
}
```

Файл: ./web/static/js/blocks.js

```
let updateAllBlocks = window.setInterval(async function () {
    await blocksLoad()
}, 3000);

let blocksOnPage = 15

function range(size, startAt = 0) {
    return [...Array(size).keys()].map(i => i + startAt);
}

function getCurrentPage() {
    let currentPage = window.location.search.replace("?page=", "")
    if (!currentPage || currentPage === "1") {
        currentPage = 1
    }
    return parseInt(currentPage)
}

async function updatePages(blocksCount) {
    let pagesCount = Math.floor(blocksCount / blocksOnPage)
    let pagesObject = document.getElementById("pagesData")
    pagesObject.innerHTML = ""

    let currentPage = getCurrentPage()
    if (currentPage === 1) {
        pagesObject.innerHTML += `<li class="page-item disabled">
            <a class="page-link" href="#" tabIndex="-1">&laquo;</a>
        </li>`
        pagesObject.innerHTML += `<li class="page-item disabled">
            <a class="page-link" href="#" tabIndex="-1">Предыдущая</a>
        </li>`
    } else {
        pagesObject.innerHTML += `<li class="page-item">
            <a class="page-link" href="/blocks?page=1" tabIndex="-1">&laquo;</a>
        </li>`
        pagesObject.innerHTML += `<li class="page-item">
            <a class="page-link" href="/blocks?page=${currentPage - 1}" tabIndex="-1">Предыдущая</a>
        </li>`
    }

    let startPage = Math.max(1, currentPage - 3);
    let endPage = Math.min(pagesCount, startPage + 5);

    let pages = range(pagesCount + 2).slice(startPage, endPage + 1)

    for (const x of pages) {
        if (x === currentPage) {
```

```
            pagesObject.innerHTML +=
                `<li class="page-item disabled"><a class="page-link" href="/blocks?page=${x}">${x}</a></li>`
        } else {
            pagesObject.innerHTML +=
                `<li class="page-item"><a class="page-link" href="/blocks?page=${x}">${x}</a></li>`
        }

    }


    if (currentPage === pagesCount) {
        pagesObject.innerHTML += `<li class="page-item disabled">
            <a class="page-link" href="#">Следующая</a>
        </li>`
        pagesObject.innerHTML += `<li class="page-item disabled">
            <a class="page-link" href="#" tabIndex="-1">&raquo;</a>
        </li>`
    } else {
        pagesObject.innerHTML += `<li class="page-item">
            <a class="page-link" href="/blocks?page=${currentPage + 1}">Следующая</a>
        </li>`
        pagesObject.innerHTML += `<li class="page-item">
            <a class="page-link" href="/blocks?page=${pagesCount}" tabIndex="-1">&raquo;</a>
        </li>`
    }

}


async function blocksLoad(force = false) {
    let currentNode = localStorage.getItem(uniqueKey("node"))

    let blocksData = await fetch("/blocks_latest",
        {
            method: 'POST', headers: {
                'Content-Type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                node: currentNode,
                limit: blocksOnPage,
                offset: blocksOnPage * getCurrentPage()
            })
        })

    let response = (await blocksData.json())
    blocksData = response["blocks"]

    let blocksObject = document.getElementById("blockTable")

    blocksObject.innerHTML = ""
    for (let block of blocksData) {
        let code = `
            <th scope="row">${block['block_number']}</th>
            <td>${block['authority_id']}</td>
            <td>${timeConverter(block['timestamp'])}</td>
            <td style="max-width: 200px; overflow: hidden; text-overflow: ellipsis;">
${block['block_hash']}</td>
            <td><a href="/block/${block['block_hash']}">Подробнее</a></td>`
        let newBlock = document.createElement('tr');
        newBlock.innerHTML = code
        blocksObject.prepend(newBlock)
    }

    await updatePages(response["total_count"])

    let nodeButton = document.getElementById("dropdownMenuButton")
    let nodes = document.getElementById("nodes")

    let transactionData = await (await fetch("/get_transaction_data", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
```

```
            node: currentNode,
            transaction: ""
        })
    })).json()

    if (force) {
        if (!transactionData["status"] && !transactionData["node_none"]) {
            document.getElementById("errorNodeText").innerText = `Не получилось подключиться к ноде ${currentNode}. Выберите другую.`
            $("#errorPopupNode").toast('show'); // Показываем popup
            setTimeout(function () {
                $("#errorPopupNode").toast('hide'); // Скрываем popup через 2 секунды
            }, 2000);
        }

        if (transactionData["status"]) {
            document.getElementById("successNodeText").innerText = `Успешно подключились к ноде ${currentNode}`
            $("#successChangeNode").toast('show');
            setTimeout(function () {
                $("#successChangeNode").toast('hide');
            }, 2000);
        }
    }

    if (!currentNode) {
        let active_nodes = []
        for (let node of transactionData["nodes"]) {
            if (node["is_online"]) {
                active_nodes.push(node)
            }
        }

        let randomNode;
        if (active_nodes.length > 0) {
            randomNode = active_nodes[Math.floor(Math.random() * active_nodes.length)];
        } else {
            randomNode = transactionData["nodes"][Math.floor(Math.random() * transactionData["nodes"].length)];
        }
        nodeButton.textContent = randomNode["title_id"]
        localStorage.setItem(uniqueKey("node"), randomNode["title_id"])
        await blocksLoad(true)
    } else {
        nodeButton.textContent = currentNode
        nodes.innerHTML = ""

        for (const node of transactionData["nodes"]) {
            let status = node["is_online"] ? "" : ""
            nodes.innerHTML += `<a class="dropdown-item" href="#"
        onclick="setNode('${node["title_id"]}')">${status} ${node["title_id"]} (Блоков: ${node["blocks_count"]})</a>`
        }
    }
}

function is_numeric(str) {
    return /^\d+$/.test(str);
}

async function searchAll() {
    let searchQuery = document.getElementById("searchData").value
    let currentNode = localStorage.getItem(uniqueKey("node"))

    if (!searchQuery) {
        return
    }

    if (searchQuery.length === 62 && searchQuery.startsWith("poa")) {
        window.open(`/address/${searchQuery}`, '_blank');
    }

    if (searchQuery.length === 64) {
        let transactionData = await (await fetch("/get_transaction_data", {
            method: 'POST', headers: {
                'Content-Type': 'application/json;charset=utf-8'
            },
```

```
            body: JSON.stringify({
                node: currentNode,
                transaction: searchQuery
            })
        })).json()

        console.log(transactionData)

        if (transactionData["transaction_data"]["transaction"]) {
            window.open(`/transaction/${searchQuery}`,
                "_blank")
            return
        }


        let blockData = await (await fetch("/get_block_data", {
            method: 'POST', headers: {
                'Content-Type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                node: currentNode,
                block: searchQuery
            })
        })).json()

        if (blockData["block_data"]["block"]) {
            window.open(`/block/${blockData['block_data']['block']['block_hash']}`, "_blank")
            return
        }

        return
    }

    if (is_numeric(searchQuery)) {
        let blockData = await (await fetch("/get_block_by_number", {
            method: 'POST', headers: {
                'Content-Type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                node: currentNode,
                block_number: searchQuery
            })
        })).json()

        if (!blockData["block_data"]["block"]) {
            return
        }

        window.open(`/block/${blockData['block_data']['block']['block_hash']}`, "_blank")
    }
}


async function setNode(node) {
    let nodeButton = document.getElementById("dropdownMenuButton")
    nodeButton.textContent = node
    localStorage.setItem(uniqueKey("node"), node)

    await blocksLoad(true)
}
```

Файл: ./web/static/js/block.js

```
const globalBlock = window.location.pathname.replace("/block/", "")
let addedHashes = []


async function setDataBlock() {
    let nodes = document.getElementById("nodes")

    let currentNode = localStorage.getItem(uniqueKey("node"))

    let nodeButton = document.getElementById("dropdownMenuButton")
```

```
let blockData = await (await fetch("/get_block_data", {
    method: 'POST', headers: {
        'Content-Type': 'application/json;charset=utf-8'
    },
    body: JSON.stringify({
        node: currentNode,
        block: globalBlock
    })
})).json()


if (!blockData["status"] && !blockData["node_none"]) {
    document.getElementById("errorNodeText").innerText = `Не получилось подключиться к ноде ${currentNode}. Выберите другую.`
    $("#errorPopupNode").toast('show'); // Показываем popup
    setTimeout(function () {
        $("#errorPopupNode").toast('hide'); // Скрываем popup через 2 секунды
    }, 2000);
}

if (blockData["status"]) {
    document.getElementById("successNodeText").innerText = `Успешно подключились к ноде ${currentNode}`
    $("#successChangeNode").toast('show');
    setTimeout(function () {
        $("#successChangeNode").toast('hide');
    }, 2000);
}

if (!currentNode) {
    let active_nodes = []
    for (let node of blockData["nodes"]) {
        if (node["is_online"]) {
            active_nodes.push(node)
        }
    }

    let randomNode;
    if (active_nodes.length > 0) {
        randomNode = active_nodes[Math.floor(Math.random() * active_nodes.length)];
    } else {
        randomNode = blockData["nodes"][Math.floor(Math.random() * blockData["nodes"].length)];
    }
    nodeButton.textContent = randomNode["title_id"]
    localStorage.setItem(uniqueKey("node"), randomNode["title_id"])
    await setDataTrans()
} else {
    nodeButton.textContent = currentNode
    nodes.innerHTML = ""

    for (const node of blockData["nodes"]) {
        let status = node["is_online"] ? "" : ""
        nodes.innerHTML += `<a class="dropdown-item" href="#"
    onclick="setNode('${node["title_id"]}')">${status} ${node["title_id"]} (Блоков: ${node["blocks_count"]})</a>`
    }
}

if (!blockData["block_data"]) {
    return
}

let blockObj = blockData["block_data"]["block"]

let transactionTitleText = document.getElementById("transTitle")
if (!blockObj) {
    transactionTitleText.innerText = "Блок не найден"
} else {
    let transactionText = document.getElementById("transData")

    transactionTitleText.innerText = "Данные блока:"

    transactionText.innerHTML = `
    <br>
    <p>Номер: ${blockObj['block_number']}</p>
```

```
        <p>Хеш: ${blockObj['block_hash']}</p>
        <p>Издатель: ${blockObj['authority_id']}</p>
        <p>Предыдущий хеш: <a href="/block/${blockObj['previous_hash']}">${blockObj['previous_hash']}</a></p>
        <p>${timeConverter(blockObj['timestamp'])}</p>
        `
    }

    await createTransactions(currentNode)
}


async function setNode(node) {
    let nodeButton = document.getElementById("dropdownMenuButton")
    nodeButton.textContent = node
    localStorage.setItem(uniqueKey("node"), node)

    await setDataBlock()
}


async function createTransactions(currentNode) {
    let transactions = (await (await fetch("/get_transactions_by_block", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
            node: currentNode,
            block_hash: globalBlock,
        })
    })).json())["transactions"]

    let transactionsObject = document.getElementById("transactions")

    for (const transactionObj of transactions) {
        if (addedHashes.includes(transactionObj["transaction_hash"])) {
            continue
        }
        transactionsObject.prepend(createTransaction(transactionObj))
        addedHashes.push(transactionObj["transaction_hash"])
    }
}
```

Файл: ./web/static/js/wallet.js

```
let updateAll = window.setInterval(async function () {
    await setData(true)
}, 3000);


let addedHashes = []
let chosenState = "all"

function setAll() {
    document.getElementById("amount").value =
        document.getElementById("balance").innerText.split(" ")[0]
}

function isvalidAddress(address) {
    return (address.startsWith("poa") && address.length === 62
        && address !== document.getElementById("address").textContent);
}

function isvalidAmount(amount) {
    return (!isNaN(amount) && parseFloat(amount) >= 0.01);
}

function confirmSend() {
    let targetAddress = document.getElementById("recipient")
    let targetAmount = document.getElementById("amount")

    let targetAddressValue = targetAddress.value
    let targetAmountValue = targetAmount.value
```

```
    if (!targetAddressValue || !isvalidAddress(targetAddressValue)) {
        targetAddress.classList.add("error");
        setTimeout(function () {
            targetAddress.classList.remove("error");
        }, 2000);
        $("#errorPopupAddress").toast('show');
        setTimeout(function () {
            $("#errorPopupAddress").toast('hide');
        }, 2000);
        return
    }

    if (!targetAmountValue || !isvalidAmount(targetAmountValue)) {
        targetAmount.classList.add("error");
        setTimeout(function () {
            targetAmount.classList.remove("error");
        }, 2000);
        $("#errorPopupAmount").toast('show');
        setTimeout(function () {
            $("#errorPopupAmount").toast('hide');
        }, 2000);
        return;
    }

    document.getElementById("transactionData").innerHTML = `Вы уверены что хотите отправить
    <code>${targetAmountValue} POA</code> на <code>${targetAddressValue}</code>?`
    $('#confirmModal').modal('show');
}


async function sendTransaction() {
    $('#confirmModal').modal('hide');
    let targetAddress = document.getElementById("recipient").value
    let targetAmount = document.getElementById("amount").value

    let transaction = (await (await fetch("/create_transaction", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
            address: targetAddress, amount: targetAmount,
            publicKey: localStorage.getItem(uniqueKey("publicKey")),
            privateKey: localStorage.getItem(uniqueKey("privateKey"))
        })
    })).json())

    if (!transaction["status"]){
        document.getElementById("errorNodeDesc").innerText = "Не удалось отправить транзакцию. Неверные данные."
        $("#errorPopupNodeSend").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#errorPopupNodeSend").toast('hide'); // Скрываем popup через 2 секунды
        }, 2000);
        return
    }


    transaction = transaction["encoded_transaction"]

    let currentNode = localStorage.getItem(uniqueKey("node"))

    let response = await (await fetch("/send_transaction", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({data: transaction, node: currentNode})
    })).json()

    if (!response["status"]) {
        document.getElementById("errorNodeDesc").innerText = "Не удалось отправить транзакцию. Нода недоступна."
        $("#errorPopupNodeSend").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#errorPopupNodeSend").toast('hide'); // Скрываем popup через 2 секунды
```

```
        }, 2000);
    } else if (!response["result"]["status"]) {
        document.getElementById("errorNodeDesc").innerText = response["result"]["description"]

        $("#errorPopupNodeSend").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#errorPopupNodeSend").toast('hide'); // Скрываем popup через 2 секунды
        }, 2000);
    } else {
        document.getElementById("recipient").value = ""
        document.getElementById("amount").value = ""


        $("#successPopupNodeSend").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#successPopupNodeSend").toast('hide'); // Скрываем popup через 2 секунды
        }, 2000);
        await setData(true)
    }

}


async function openTab(tabName) {
    let openTab = document.getElementById(tabName)
    let address = getAddress()

    let currentNode = localStorage.getItem(uniqueKey("node"))
    let closeTab, openLink, closeLink, closeLink2;
    if (openTab.id === "send") {
        closeTab = document.getElementById("sign")
        openLink = document.getElementById("sendlink")
        closeLink = document.getElementById("signlink")
    } else if (openTab.id === "sign") {
        closeTab = document.getElementById("send")
        openLink = document.getElementById("signlink")
        closeLink = document.getElementById("sendlink")
    } else if (openTab.id === "allTrans") {
        openLink = document.getElementById("allTrans")
        closeLink = document.getElementById("receive")
        closeLink2 = document.getElementById("sent")
        await createTransactions(address, currentNode, "all", true)
        chosenState = "all"
    } else if (openTab.id === "receive") {
        openLink = document.getElementById("receive")
        closeLink = document.getElementById("allTrans")
        closeLink2 = document.getElementById("sent")
        await createTransactions(address, currentNode, "to", true)
        chosenState = "to"
    } else if (openTab.id === "sent") {
        openLink = document.getElementById("sent")
        closeLink = document.getElementById("allTrans")
        closeLink2 = document.getElementById("receive")
        await createTransactions(address, currentNode, "from", true)
        chosenState = "from"
    }

    let isMobile = window.innerWidth < 900

    if (openTab) {
        openTab.hidden = false
    }

    openLink.className = "nav-link active"

    if (closeTab) {
        closeTab.hidden = true
    }
    closeLink.className = "nav-link"

    if (closeLink2) {
        closeLink2.className = "nav-link"
    }
```

```
    if (isMobile) {
        openLink.className = "nav-link active btn-sm"
        closeLink.className = "nav-link btn-sm"
        closeLink2.className = "nav-link btn-sm"
    }
}


function getAddress() {
    let address = localStorage.getItem(uniqueKey("address"))

    if (window.location.pathname.includes("/address/")) {
        address = window.location.pathname.replace("/address/", "")
    }
    return address
}

function copyAddress() {
    let addressElement = document.getElementById("address");
    let address = addressElement.textContent;
    navigator.clipboard.writeText(address)
        .then(function () {
            $("#successPopup").toast('show'); // Показываем popup
            setTimeout(function () {
                $("#successPopup").toast('hide'); // Скрываем popup через 2 секунды
            }, 2000);
        })
        .catch(function (error) {
        });
}


function isWallet() {
    if (window.location.pathname.includes("/blocks")) {
        return false
    }
    return true
}

async function createTransactions(address, currentNode, type = "all", full_update = false) {
    if (!isWallet()) {
        return
    }

    let transactions = (await (await fetch("/get_transactions", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
            node: currentNode,
            address: address,
            type: type
        })
    })).json())["transactions"]

    let transactionsObject = document.getElementById("transactions")

    if (full_update) {
        addedHashes = []
        transactionsObject.innerHTML = ""
        transactionsObject.scrollTo(0, 0);
    }
    for (const transactionObj of transactions) {
        if (addedHashes.includes(transactionObj["transaction_hash"])) {
            continue
        }
        transactionsObject.prepend(createTransaction(transactionObj))
        addedHashes.push(transactionObj["transaction_hash"])
    }
}
```

```
async function setData(forUpdate = false) {
    let addressText = document.getElementById("address")
    let balanceText = document.getElementById("balance")
    let nodes = document.getElementById("nodes")

    let currentNode = localStorage.getItem(uniqueKey("node"))

    if (!localStorage.getItem(uniqueKey("privateKey")) ||
        !localStorage.getItem(uniqueKey("publicKey")) ||
        !localStorage.getItem(uniqueKey("address"))) {
        await logout()
        return
    }

    let address = getAddress()

    let walletData = await (await fetch("/get_wallet_data", {
        method: 'POST', headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify({
            node: currentNode,
            address: address
        })
    })).json()


    let nodeButton = document.getElementById("dropdownMenuButton")

    if (!walletData["status"] && !walletData["node_none"] && !forUpdate) {
        document.getElementById("errorNodeText").innerText = `Не получилось подключиться к ноде ${currentNode}. Выберите другую.`
        $("#errorPopupNode").toast('show'); // Показываем popup
        setTimeout(function () {
            $("#errorPopupNode").toast('hide'); // Скрываем popup через 2 секунды
        }, 2000);
    }

    if (walletData["status"] && !forUpdate) {
        document.getElementById("successNodeText").innerText = `Успешно подключились к ноде ${currentNode}`
        $("#successChangeNode").toast('show');
        setTimeout(function () {
            $("#successChangeNode").toast('hide');
        }, 2000);
    }

    if (isWallet()) {
        addressText.textContent = walletData["address"]
        balanceText.textContent = `${walletData["balance"]} POA`
    }


    if (!currentNode) {
        let active_nodes = []
        for (let node of walletData["nodes"]) {
            if (node["is_online"]) {
                active_nodes.push(node)
            }
        }

        let randomNode;
        if (active_nodes.length > 0) {
            randomNode = active_nodes[Math.floor(Math.random() * active_nodes.length)];
        } else {
            randomNode = walletData["nodes"][Math.floor(Math.random() * walletData["nodes"].length)];
        }
        nodeButton.textContent = randomNode["title_id"]
        localStorage.setItem(uniqueKey("node"), randomNode["title_id"])
        await setData()
    } else {
        nodeButton.textContent = currentNode
        nodes.innerHTML = ""

        for (const node of walletData["nodes"]) {
```

```
            let status = node["is_online"] ? "" : ""
            nodes.innerHTML += `<a class="dropdown-item" href="#"
        onclick="setNode('${node["title_id"]}')">${status} ${node["title_id"]} (Блоков: ${node["blocks_count"]})</a>`
        }
        await createTransactions(address, currentNode, chosenState)
    }

}

async function setNode(node) {
    let nodeButton = document.getElementById("dropdownMenuButton")
    nodeButton.textContent = node
    localStorage.setItem(uniqueKey("node"), node)

    await setData()
}


async function signMessage() {
    let message = document.getElementById("message")

    let result = (await (await fetch("/sign", {
        method: "POST",
        body: JSON.stringify({
            message: message.value,
            private_key: localStorage.getItem(uniqueKey("privateKey"))
        })
    })).json())["result"]

    let signText = document.getElementById("resultSign")

    signText.hidden = false
    signText.value = result

    document.getElementById("signButton").hidden = false
}


async function copySign() {
    await signMessage()
    let signElement = document.getElementById("resultSign");
    let sign = signElement.value;
    navigator.clipboard.writeText(sign)
        .then(function () {
            $("#successPopupSign").toast('show'); // Показываем popup
            setTimeout(function () {
                $("#successPopupSign").toast('hide'); // Скрываем popup через 2 секунды
            }, 2000);
        })
        .catch(function (error) {
        });
}

async function copyTransAddress(address) {
    navigator.clipboard.writeText(address)
        .then(function () {
            $("#successPopup").toast('show'); // Показываем popup
            setTimeout(function () {
                $("#successPopup").toast('hide'); // Скрываем popup через 2 секунды
            }, 2000);
        })
        .catch(function (error) {
        });
}
```

Файл: ./web/static/html/login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Wallet</title>
```

```
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link href="../css/index.css" rel="stylesheet">
    <script src="../js/index.js"></script>
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
</head>
<body onload=" onloadAdaptive()">
<div class="container">
    <div class="row justify-content-center wallet-container">
        <div class="col-md-6">
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title text-center">Poachain Wallet</h5>
                    <form id="wallet-form">
                        <div class="form-group">
                            <label for="mnemonic" id="mnemid">Введите вашу мнемоническую фразу:</label>

                            <textarea type="text" class="form-control" id="mnemonic"
                                      placeholder="Мнемоническая фраза" rows="4"
                                      style="color: #fff; resize: none;" required></textarea>
                        </div>
                        <div class="text-center">
                            <button type="button" class="btn btn-primary" onclick="login()">Войти</button>
                            <button type="button" class="btn btn-success"
                                    id="create-wallet" onclick="createNew()">Создать
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>


<div class="top">
    <a href="/wallet" id="logoIMG">
        <img src="../images/logo.png" alt="Главная" class="corner-image">
    </a> <br>
</div>


<div id="errorPopup" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка входа!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверная мнемоническая фраза.
    </div>
</div>

<!-- Bootstrap JS -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Файл: ./web/static/html/address.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Wallet</title>
    <!-- Bootstrap CSS -->
```

```
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link href="../css/index.css" rel="stylesheet">
    <link href="../css/wallet.css" rel="stylesheet">
    <script src="../js/index.js"></script>
    <script src="../js/wallet.js"></script>
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
</head>
<body onload="setData(); onloadAdaptive()">
<div class="top" id="logo">
    <div id="logoIMG">
        <a href="/wallet">
            <img src="../images/logo.png" alt="Главная" class="corner-image">
        </a> <br>
    </div>
    <a href="/wallet"><button type="button" class="btn btn-primary">Кошелёк</button></a>
</div>

<div class="container">
    <div>
        <div>
            <div class="container mt-5">
                <h1 class="mb-4" style="margin-top: 1%"><b>Poachain Просмотр Адреса</b></h1>

                <div class="card mb-4">
                    <div class="card-body">
                        <h5 class="card-title">Баланс Адреса:</h5>
                        <p class="card-text h2" id="balance"></p>
                    </div>
                </div>
                <div class="card mb-4">
                    <div class="card-body">
                        <h5 class="card-title">Адрес:</h5>
                        <p class="card-text" id="address">Адрес будет здесь</p>
                        <button class="btn btn-primary" onclick="copyAddress()">Копировать адрес</button>
                    </div>
                </div>
            </div>
        </div>

        <div style="height: 700px; overflow: hidden;">

            <h1 class="mb-4 mt-5"><b>Транзакции</b></h1>
            <ul class="nav nav-tabs mb-4">
                <li class="nav-item">
                    <a class="nav-link active" id="allTrans" onclick="openTab('allTrans')">Все</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" id="receive" onclick="openTab('receive')">Полученные</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" id="sent" onclick="openTab('sent')">Отправленные</a>
                </li>
            </ul>
            <div style="overflow-y: auto; height: 70%;" id="transactions">
            </div>
        </div>

    </div>
</div>


<div class="modal fade" id="confirmModal" tabindex="-1" role="dialog" aria-labelledby="confirmModalLabel"
     aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content bg-dark text-light border border-primary">
            <div class="modal-header border-0">
                <h5 class="modal-title" id="confirmModalLabel">Подтвердите отправку</h5>
                <button type="button" class="close text-light" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body" id="transactionData">
```

```
            </div>
            <div class="modal-footer border-0">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Отмена</button>
                <button type="button" class="btn btn-primary" onclick="sendTransaction()">Подтвердить</button>
            </div>
        </div>
    </div>
</div>
<div id="successPopup" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Адрес был скопирован в буфер обмена.
    </div>
</div>
<div id="successPopupSign" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Подпись была скопирована в буфер обмена.
    </div>
</div>
<div id="successChangeNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Нода успешно подключена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="successNodeText">
    </div>
</div>
<div id="errorPopupNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка подключения!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="errorNodeText">

    </div>
</div>
<div id="errorPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка отправки транзакции!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="errorNodeDesc">
    </div>
</div>
<div id="successPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Транзакция успешно отправлена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
```

```
    </div>
    <div class="toast-body" style="color: black" id="hashData">
        Транзакция принята мемпулом ноды.
    </div>
</div>
<div id="errorPopupAddress" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверный адрес.
    </div>
</div>
<div id="errorPopupAmount" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверная сумма.
    </div>
</div>
<div class="btn-group dropleft" style="position: absolute; top: 2%; right: 2%;">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
        Выбор ноды
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton" id="nodes">
    </div>
</div>

<!-- Bootstrap JS -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Файл: ./web/static/html/wallet.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Кошелек</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link href="../css/index.css" rel="stylesheet">
    <link href="../css/wallet.css" rel="stylesheet">
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
    <script src="../js/wallet.js"></script>
    <script src="../js/index.js"></script>
</head>
<body onload="setData(); onloadAdaptive()">

<div class="top" id="logo">
    <div id="logoIMG">
        <a href="/wallet">
            <img src="../images/logo.png" alt="Главная" class="corner-image">
```

```
    </a> <br>
  </div>
  <a href="/blocks"><button type="button" class="btn btn-primary">Блоки</button></a>
  <p id="bigScreenP"></p>
  <button type="button" class="btn btn-primary" onclick="logout()">Выйти</button>
</div>


<div class="container">
  <div>
    <div>
      <div class="container mt-5">
        <h1 class="mb-4" style="margin-top: 1%"><b>Poachain Кошелек</b></h1>

        <div class="card mb-4">
          <div class="card-body">
            <h5 class="card-title">Текущий баланс:</h5>
            <p class="card-text h2" id="balance"></p>
          </div>
        </div>
        <div class="card mb-4">
          <div class="card-body">
            <h5 class="card-title">Ваш адрес:</h5>
            <p class="card-text" id="address">Ваш адрес будет здесь</p>
            <button class="btn btn-primary" onclick="copyAddress()">Копировать адрес</button>
          </div>
        </div>
        <ul class="nav nav-tabs mb-4">
          <li class="nav-item">
            <a class="nav-link active" id="sendlink" onclick="openTab('send')">Отправить
              криптовалюту</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" id="signlink" onclick="openTab('sign')">Создать подпись</a>
          </li>
        </ul>
        <div id="send" class="tab-content">
          <form onsubmit="confirmSend(); return false;">
            <div class="form-group">
              <input type="text" class="form-control" id="recipient" name="recipient"
                  placeholder="Адрес получателя"
                  autocomplete="off">
            </div>
            <div class="input-group" style="margin-bottom: 1%">
              <input type="text" class="form-control" id="amount" name="amount" placeholder="Сумма"
                  autocomplete="off">
              <div class="input-group-append">
                <button class="btn btn-secondary" type="button" id="setAllBTN"
                    onclick="setAll()">Отправить всё
                </button>
              </div>
            </div>
            <button type="submit" class="btn btn-primary">Отправить</button>
          </form>
        </div>
        <div id="sign" class="tab-content" hidden>
          <form>
            <div class="form-group">
              <input type="text" class="form-control" id="message" name="message" placeholder="Сообщение">
            </div>
            <div class="form-group">
              <input type="text" class="form-control" id="resultSign" name="message" hidden></input>
            </div>

            <button type="button" class="btn btn-primary" onclick="signMessage()">Создать подпись</button>
            <button id="signButton" type="button" class="btn btn-primary" onclick="copySign()" hidden>
              Скопировать
              подпись
            </button>
          </form>
        </div>
      </div>
    </div>
```

```
        <div style="height: 700px; overflow: hidden;">
          <h1 class="mb-4 mt-5"><b>Транзакции</b></h1>
          <ul class="nav nav-tabs mb-4">
            <li class="nav-item">
              <a class="nav-link active" id="allTrans" onclick="openTab('allTrans')">Все</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" id="receive" onclick="openTab('receive')">Полученные</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" id="sent" onclick="openTab('sent')">Отправленные</a>
            </li>
          </ul>
          <div style="overflow-y: auto; height: 70%;" id="transactions">
          </div>
        </div>

    </div>
</div>

<div class="modal fade" id="confirmModal" tabindex="-1" role="dialog" aria-labelledby="confirmModalLabel"
    aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content bg-dark text-light border border-primary">
            <div class="modal-header border-0">
                <h5 class="modal-title" id="confirmModalLabel">Подтвердите отправку</h5>
                <button type="button" class="close text-light" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body" id="transactionData">

            </div>
            <div class="modal-footer border-0">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Отмена</button>
                <button type="button" class="btn btn-primary" onclick="sendTransaction()">Подтвердить</button>
            </div>
        </div>
    </div>
</div>
<div id="successPopup" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
    style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Адрес был скопирован в буфер обмена.
    </div>
</div>
<div id="successPopupSign" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
    style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Подпись была скопирована в буфер обмена.
    </div>
</div>
<div id="successChangeNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
    style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Нода успешно подключена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
```

```
    <div class="toast-body" style="color: black" id="successNodeText">
    </div>
</div>
<div id="errorPopupNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка подключения!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="errorNodeText">

    </div>
</div>
<div id="errorPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка отправки транзакции!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="errorNodeDesc">
    </div>
</div>
<div id="successPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Транзакция успешно отправлена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="hashData">
        Транзакция принята мемпулом ноды.
    </div>
</div>
<div id="errorPopupAddress" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверный адрес.
    </div>
</div>
<div id="errorPopupAmount" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверная сумма.
    </div>
</div>

<div class="btn-group dropleft" style="position: absolute; top: 2%; right: 2%;">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
        Выбор ноды
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton" id="nodes">
    </div>
</div>

</body>
```

```
</html>
```

Файл: ./web/static/html/blocks.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Wallet</title>
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link href="../css/index.css" rel="stylesheet">
    <link href="../css/wallet.css" rel="stylesheet">
    <script src="../js/index.js"></script>
    <script src="../js/blocks.js"></script>
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
</head>
<body onload="blocksLoad(); onloadAdaptive()">
<div class="top" id="logo">
    <div id="logoIMG">
        <a href="/wallet">
            <img src="../images/logo.png" alt="Главная" class="corner-image">
        </a> <br>
    </div>
    <a href="/wallet">
        <button type="button" class="btn btn-primary">Кошелёк</button>
    </a>
</div>

<div class="container">
    <div>
        <div>
            <div class="container mt-5">
                <h1 class="mb-4" style="margin-top: 1%"><b>Poachain Обзорщик Блоков</b></h1>
                <form onsubmit="searchAll(); return false">
                    <div class="input-group mb-3">
                        <input type="text" class="form-control" id="searchData"
                               placeholder="Поиск адреса, блока, транзакции и др."
                               aria-label="Search" aria-describedby="button-search">
                        <div class="input-group-append">
                            <button class="btn btn-outline-primary" type="button"
                                    id="button-search" onclick="searchAll()">Поиск</button>
                        </div>
                    </div>
                </form>
                <table class="table">
                    <thead>
                    <tr>
                        <th scope="col">Номер</th>
                        <th scope="col">Издатель</th>
                        <th scope="col">Время</th>
                        <th scope="col">Хеш</th>
                        <th scope="col"></th>
                    </tr>
                    </thead>
                    <tbody id="blockTable">
                    </tbody>
                </table>
            </div>
        </div>

    </div>
</div>

<nav aria-label="Page navigation example">
    <ul class="pagination justify-content-center" id="pagesData">
        <li class="page-item disabled">
            <a class="page-link" href="#" tabindex="-1">Предыдущая</a>
        </li>
        <li class="page-item"><a class="page-link" href="#">1</a></li>
        <li class="page-item"><a class="page-link" href="#">2</a></li>
```

```
        <li class="page-item"><a class="page-link" href="#">3</a></li>
        <li class="page-item">
          <a class="page-link" href="#">Следующая</a>
        </li>
    </ul>
</nav>

<div id="successChangeNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Нода успешно подключена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="successNodeText">
    </div>
</div>
<div id="errorPopupNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка подключения!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black" id="errorNodeText">

    </div>
</div>
<div class="btn-group dropleft" style="position: absolute; top: 2%; right: 2%;">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
        Выбор ноды
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton" id="nodes">
    </div>
</div>

<!-- Bootstrap JS -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Файл: ./web/static/html/block.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Wallet</title>
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link href="../css/index.css" rel="stylesheet">
    <link href="../css/wallet.css" rel="stylesheet">
    <script src="../js/index.js"></script>
    <script src="../js/block.js"></script>
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
</head>
<body onload="setDataBlock(); onloadAdaptive()">
<div class="top" id="logo">
    <div id="logoIMG">
        <a href="/wallet">
            <img src="../images/logo.png" alt="Главная" class="corner-image">
        </a> <br>
    </div>
    <a href="/wallet">
        <button type="button" class="btn btn-primary">Кошелёк</button>
```

```
    </a>
</div>

<div class="container">
    <div class="container mt-5">
        <h1 class="mb-4" style="margin-top: 1%"><b>Poachain Просмотр Блока</b></h1>

        <div class="card mb-4">
            <div class="card-body">
                <h4 class="card-title" id="transTitle"></h4>
                <div id="transData">

                </div>
            </div>
        </div>
    </div>

    <div style="height: 700px; overflow: hidden;">
        <h1 class="mb-4 mt-5"><b>Транзакции блока</b></h1>
        <div style="overflow-y: auto; height: 70%;" id="transactions">
        </div>
    </div>

</div>


<div class="modal fade" id="confirmModal" tabindex="-1" role="dialog" aria-labelledby="confirmModalLabel"
     aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content bg-dark text-light border border-primary">
            <div class="modal-header border-0">
                <h5 class="modal-title" id="confirmModalLabel">Подтвердите отправку</h5>
                <button type="button" class="close text-light" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body" id="transactionData">

            </div>
            <div class="modal-footer border-0">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Отмена</button>
                <button type="button" class="btn btn-primary" onclick="sendTransaction()">Подтвердить</button>
            </div>
        </div>
    </div>
</div>
<div id="successPopup" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Адрес был скопирован в буфер обмена.
    </div>
</div>
<div id="successPopupSign" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Подпись была скопирована в буфер обмена.
    </div>
</div>
<div id="successChangeNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-success text-white">
```

```
        <strong class="mr-auto">Нода успешно подключена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="successNodeText">
      </div>
    </div>
    <div id="errorPopupNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка подключения!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="errorNodeText">

      </div>
    </div>
    <div id="errorPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка отправки транзакции!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="errorNodeDesc">
      </div>
    </div>
    <div id="successPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Транзакция успешно отправлена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="hashData">
        Транзакция принята мемпулом ноды.
      </div>
    </div>
    <div id="errorPopupAddress" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black">
        Неверный адрес.
      </div>
    </div>
    <div id="errorPopupAmount" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black">
        Неверная сумма.
      </div>
    </div>
    <div class="btn-group dropleft" style="position: absolute; top: 2%; right: 2%;">
      <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">
        Выбор ноды
      </button>
      <div class="dropdown-menu" aria-labelledby="dropdownMenuButton" id="nodes">
```

```
    </div>
</div>

<!-- Bootstrap JS -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Файл: ./web/static/html/transaction.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Poachain Wallet</title>
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <link href="../css/index.css" rel="stylesheet">
    <link href="../css/wallet.css" rel="stylesheet">
    <script src="../js/index.js"></script>
    <script src="../js/transaction.js"></script>
    <link href='https://fonts.googleapis.com/css?family=JetBrains Mono' rel='stylesheet'>
</head>
<body onload="setDataTrans(); onloadAdaptive()">
<div class="top" id="logo">
    <div id="logoIMG">
        <a href="/wallet">
            <img src="../images/logo.png" alt="Главная" class="corner-image">
        </a> <br>
    </div>
    <a href="/wallet"><button type="button" class="btn btn-primary">Кошелёк</button></a>
</div>

<div class="container">
    <div>
        <div>
            <div class="container mt-5">
                <h1 class="mb-4" style="margin-top: 1%"><b>Poachain Просмотр Транзакции</b></h1>

                <div class="card mb-4">
                    <div class="card-body">
                        <h4 class="card-title" id="transTitle"></h4>
                        <div id="transData">

                        </div>
                    </div>
                </div>
            </div>
        </div>

    </div>
</div>


<div class="modal fade" id="confirmModal" tabindex="-1" role="dialog" aria-labelledby="confirmModalLabel"
     aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content bg-dark text-light border border-primary">
            <div class="modal-header border-0">
                <h5 class="modal-title" id="confirmModalLabel">Подтвердите отправку</h5>
                <button type="button" class="close text-light" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body" id="transactionData">

            </div>
            <div class="modal-footer border-0">
```

```
            <button type="button" class="btn btn-secondary" data-dismiss="modal">Отмена</button>
            <button type="button" class="btn btn-primary" onclick="sendTransaction()">Подтвердить</button>
          </div>
        </div>
      </div>
    </div>
    <div id="successPopup" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black">
        Адрес был скопирован в буфер обмена.
      </div>
    </div>
    <div id="successPopupSign" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Успешно скопировано!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black">
        Подпись была скопирована в буфер обмена.
      </div>
    </div>
    <div id="successChangeNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Нода успешно подключена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="successNodeText">
      </div>
    </div>
    <div id="errorPopupNode" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка подключения!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="errorNodeText">

      </div>
    </div>
    <div id="errorPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Ошибка отправки транзакции!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="errorNodeDesc">
      </div>
    </div>
    <div id="successPopupNodeSend" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
      style="position: fixed; bottom: 10px; right: 10px;">
      <div class="toast-header bg-success text-white">
        <strong class="mr-auto">Транзакция успешно отправлена!</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="toast-body" style="color: black" id="hashData">
```

```
        Транзакция принята мемпулом ноды.
    </div>
</div>
<div id="errorPopupAddress" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверный адрес.
    </div>
</div>
<div id="errorPopupAmount" class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-delay="2000"
     style="position: fixed; bottom: 10px; right: 10px;">
    <div class="toast-header bg-danger text-white">
        <strong class="mr-auto">Неверные данные.</strong>
        <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="toast-body" style="color: black">
        Неверная сумма.
    </div>
</div>
<div class="btn-group dropleft" style="position: absolute; top: 2%; right: 2%;">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
        Выбор ноды
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton" id="nodes">
    </div>
</div>

<!-- Bootstrap JS -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Файл: ./chain/timestamps.py

```
import time


def get_current_accurate_timestamp() -> int:
    return int(time.time() * 1000000)
```

Файл: ./chain/transaction.py

```
import hashlib

from sqlalchemy import func, select, delete
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.sql.operators import and_, or_

from chain.db import Transaction, Block
from chain.db.session import db_session
from chain_config import NodeConfig
from node.models.transaction import TransactionModel


@db_session
async def create_transaction(
    session: AsyncSession,
    transaction: TransactionModel,
    block_id: int | None = None,
```

```
    block_number: int | None = None,
    with_commit: bool = True,
) -> Transaction:
    new_transaction = Transaction(
        sender_address=transaction.sender_address,
        recipient_address=transaction.recipient_address,
        amount=transaction.amount,
        timestamp=transaction.timestamp,
        transaction_hash=transaction.transaction_hash,
        block_id=block_id,
        block_number=block_number,
    )
    session.add(new_transaction)

    if with_commit:
        await session.commit()

    return new_transaction


@db_session
async def calculate_balance(
    session: AsyncSession, address: str, exclude_hash: str | None = None
) -> int | float:
    if address == NodeConfig.money_issuer_address:
        return float("inf")

    sent_amount_query = select(func.sum(Transaction.amount)).where(
        and_(
            Transaction.sender_address == address,
            Transaction.transaction_hash != exclude_hash,
        ),
    )
    sent_amount = await session.scalar(sent_amount_query)

    received_amount_query = select(func.sum(Transaction.amount)).where(
        and_(
            Transaction.recipient_address == address,
            Transaction.transaction_hash != exclude_hash,
        ),
    )
    received_amount = await session.scalar(received_amount_query)

    balance = (received_amount or 0) - (sent_amount or 0)

    return balance


@db_session
async def get_block_transactions(
    session: AsyncSession, block_id: int
) -> list[Transaction]:
    block_transactions = (
        await session.execute(
            select(Transaction).where(Transaction.block_id == block_id)
        )
    ).fetchall()

    return [transaction[0] for transaction in block_transactions]


@db_session
async def get_unconfirmed_transactions(session: AsyncSession) -> list[TransactionModel]:
    unconfirmed_transactions = (
        await session.execute(select(Transaction).where(Transaction.block_id.is_(None)))
    ).fetchall()

    return [transaction[0] for transaction in unconfirmed_transactions]


@db_session
async def get_transactions_by_address(
    session: AsyncSession, address: str, transactions_type: str
```

```
) -> list[TransactionModel]:

    if transactions_type == "all":
        query = or_(
            Transaction.sender_address == address,
            Transaction.recipient_address == address,
        )
    elif transactions_type == "from":
        query = Transaction.sender_address == address
    else:
        query = Transaction.recipient_address == address

    transactions = (
        await session.execute(
            select(Transaction).where(query).order_by(Transaction.timestamp)
        )
    ).fetchall()
    return [transaction[0] for transaction in transactions]


@db_session
async def get_transactions_by_block_hash(
    session: AsyncSession, block_hash: str
) -> list[TransactionModel]:

    block = (
        await session.execute(select(Block).where(Block.block_hash == block_hash))
    ).first()

    if block is None:
        return []

    transactions = (
        await session.execute(
            select(Transaction).where(Transaction.block_number == block[0].block_number)
        )
    ).fetchall()
    return [transaction[0] for transaction in transactions]


@db_session
async def get_transaction_by_hash(
    session: AsyncSession, transaction_hash: str
) -> TransactionModel | None:

    transaction = (
        await session.execute(
            select(Transaction).where(Transaction.transaction_hash == transaction_hash)
        )
    ).fetchone()

    if transaction is None:
        return

    transaction = transaction[0]

    return transaction


@db_session
async def delete_transaction(session: AsyncSession, transaction_id: int) -> None:
    await session.execute(delete(Transaction).where(Transaction.id == transaction_id))
    await session.commit()


def calculate_block_merkle_root(
    transactions: list[Transaction | TransactionModel],
) -> str | None:
    merkle_tree = [tx.transaction_hash for tx in transactions]
    while len(merkle_tree) > 1:
        merkle_tree = [
            hashlib.sha256(
                merkle_tree[i].encode() + merkle_tree[i + 1].encode()
```

```
            ).hexdigest()
            for i in range(0, len(merkle_tree), 2)
        ]

    return merkle_tree[0] if merkle_tree else None


def calculate_transaction_hash(transaction: Transaction | TransactionModel) -> str:
    transaction_str = (
        f"{transaction.sender_address}{transaction.recipient_address}"
        f"{transaction.amount}{transaction.timestamp}"
    )
    return hashlib.sha256(transaction_str.encode()).hexdigest()
```

Файл: ./chain/constants.py

```
GENESIS_BLOCK_PREVIOUS_HASH = "genesis"
NO_BLOCK_PREVIOUS_HASH = "NONE"

PUBLIC_KEY_LENGTH = 32
AMOUNT_LENGTH = 8
TIMESTAMP_LENGTH = 8
RANDOM_DATA_LENGTH = 4
SIGN_LENGTH = 64
TRANSACTION_PURE_LENGTH = 85
MIN_BLOCK_RELEASE_TIME = 10 * 1000000
```

Файл: ./chain/__init__.py

Файл: ./chain/block.py

```
import hashlib

from sqlalchemy import select, update, func
from sqlalchemy.ext.asyncio import AsyncSession

from chain.constants import NO_BLOCK_PREVIOUS_HASH
from chain.db import Block, Transaction
from chain.db.session import db_session
from chain.transaction import create_transaction
from node.models.block import NewBlocksModel, BlockModel


@db_session
async def get_last_block(session: AsyncSession) -> Block | None:
    last_block = (
        await session.execute(
            select(Block).order_by(Block.block_number.desc()).limit(1)
        )
    ).first()

    if last_block is None:
        return None

    return last_block[0]


@db_session
async def get_block_by_hash(session: AsyncSession, block_hash: str) -> Block | None:
    block = (
        await session.execute(select(Block).where(Block.block_hash == block_hash))
    ).first()

    if block is None:
        return None

    return block[0]
```

```
@db_session
async def get_block_by_number(session: AsyncSession, block_number: int) -> Block | None:
    block = (
        await session.execute(
            select(Block).where(Block.block_number == int(block_number))
        )
    ).first()

    if block is None:
        return None

    return block[0]


@db_session
async def get_last_block_number(session: AsyncSession) -> int:
    last_block = await get_last_block(session=session)
    if last_block is None:
        return -1

    return last_block.block_number


@db_session
async def get_last_block_timestamp(session: AsyncSession) -> int:
    last_block = await get_last_block(session=session)
    if last_block is None:
        return -1

    return last_block.timestamp


async def get_last_block_previous_hash() -> str:
    last_block = await get_last_block()

    if last_block is None:
        return NO_BLOCK_PREVIOUS_HASH

    return last_block.previous_hash


@db_session
async def get_blocks_until_previous_hash(
    session: AsyncSession, last_block_previous_hash: str
) -> list[Block]:
    last_block = await get_last_block()

    blocks = []
    while last_block_previous_hash != last_block.previous_hash:
        blocks.append(last_block)

        last_block = (
            await session.execute(
                select(Block).where(Block.block_hash == last_block.previous_hash)
            )
        ).fetchone()

        if last_block is None:
            break

        last_block = last_block[0]

    return list(reversed(blocks))


@db_session
async def create_block(
    session: AsyncSession, block: BlockModel, with_commit: bool = True
) -> Block:
    new_block = Block(
        block_number=block.block_number,
        block_hash=block.block_hash,
        previous_hash=block.previous_hash,
```

```
        merkle_root=block.merkle_root,
        authority_id=block.authority_id,
        timestamp=block.timestamp,
    )

    session.add(new_block)

    if with_commit:
        await session.commit()

    return new_block


async def add_new_blocks_from_node(
    new_blocks: NewBlocksModel, with_commit: bool = True
) -> None:
    for block in new_blocks.blocks:
        new_block = await create_block(block=block, with_commit=with_commit)

        for transaction in block.transactions:
            await create_transaction(
                transaction=transaction,
                block_id=new_block.id,
                block_number=new_block.block_number,
                with_commit=with_commit,
            )


def calculate_block_hash(block: BlockModel) -> str:
    block_data = (
        f"{block.block_number}{block.previous_hash}{block.authority_id}"
        f"{block.merkle_root}{block.timestamp}"
    )
    return hashlib.sha256(block_data.encode()).hexdigest()


@db_session
async def update_transactions_for_block(
    session: AsyncSession,
    block_id: int,
    block_number: int,
    transactions: list[Transaction],
) -> None:
    await session.execute(
        update(Transaction)
        .values(block_id=block_id, block_number=block_number)
        .where(Transaction.id.in_([transaction.id for transaction in transactions]))
    )

    await session.commit()


@db_session
async def get_blocks(session: AsyncSession, limit: int, offset: int) -> list[Block]:
    last_blocks = await session.execute(
        select(Block).order_by(-Block.block_number).offset(offset).limit(limit)
    )

    return list(reversed([block[0] for block in last_blocks.all()]))


@db_session
async def get_blocks_count(session: AsyncSession) -> int:
    all_blocks = await session.execute(select(func.count()).select_from(Block))
    return all_blocks.scalar()
```

Файл: ./chain/db/transaction.py

```
from sqlalchemy import Column, BIGINT, String, BigInteger, ForeignKey, Integer
from sqlalchemy.orm import relationship

from .base import Base
```

```
class Transaction(Base):
    __tablename__ = "transactions"

    id = Column(Integer, primary_key=True)
    sender_address = Column(String, nullable=False, index=True)
    recipient_address = Column(String, nullable=False, index=True)
    amount = Column(BIGINT, nullable=False)
    timestamp = Column(BigInteger, nullable=False)
    transaction_hash = Column(String, unique=True, nullable=False)
    block_id = Column(Integer, ForeignKey("block.id"))
    block_number = Column(Integer)
    block = relationship("Block", back_populates="transactions")

    def dict(self) -> dict:
        return {
            "sender_address": self.sender_address,
            "recipient_address": self.recipient_address,
            "amount": self.amount,
            "timestamp": self.timestamp,
            "transaction_hash": self.transaction_hash,
        }
```

Файл: ./chain/db/session.py

```
from sqlalchemy.ext.asyncio import async_sessionmaker
from sqlalchemy.ext.asyncio import create_async_engine
from functools import wraps

from chain_config import PostgresConfig

engine = create_async_engine(PostgresConfig.db_url)
async_session = async_sessionmaker(engine, expire_on_commit=False)


def db_session(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        if "session" in kwargs:
            return await func(*args, **kwargs)

        async with async_session() as session:
            result = await func(session, *args, **kwargs)
        return result

    return wrapper
```

Файл: ./chain/db/__init__.py

```
from .block import Block
from .transaction import Transaction

from .base import Base
```

Файл: ./chain/db/block.py

```
import hashlib
from sqlalchemy import Column, Integer, String, BigInteger
from sqlalchemy.orm import relationship

from node.models.transaction import TransactionModel
from .transaction import Transaction
from .base import Base


class Block(Base):
    __tablename__ = "block"

    id = Column(Integer, primary_key=True)
```

```
    block_number = Column(Integer, unique=True, nullable=False)

    block_hash = Column(String, unique=True, nullable=False)
    previous_hash = Column(String, unique=True, nullable=False)
    merkle_root = Column(String, unique=True, nullable=True)
    authority_id = Column(String, nullable=False)

    timestamp = Column(BigInteger, nullable=False)

    transactions = relationship("Transaction", back_populates="block")

    def to_dict(self, transactions: list[Transaction | TransactionModel]) -> dict:
        return {
            "block_number": self.block_number,
            "block_hash": self.block_hash,
            "previous_hash": self.previous_hash,
            "merkle_root": self.merkle_root,
            "authority_id": self.authority_id,
            "timestamp": self.timestamp,
            "transactions": [transaction.dict() for transaction in transactions],
        }
```

Файл: ./chain/db/base.py

```
from sqlalchemy.ext.declarative import declarative_base


Base = declarative_base()
```

Файл: ./node/__init__.py

Файл: ./node/utils.py

```
from node_constants import ALL_NODES, NodeConstant


def get_node_by_id(node_id: str) -> NodeConstant | None:
    for node in ALL_NODES:
        if node.title_id == node_id:
            return node

    return None
```

Файл: ./node/app.py

```
import asyncio
import logging

import aiohttp
from fastapi import FastAPI

from .blockchain.block_processing import process_blocks_forever
from .blockchain.startup import start_node, after_start_node
from .routers import (
    alive_router,
    block_router,
    receive_transaction_router,
    receive_block_router,
    user_router,
    transactions_router,
)


app = FastAPI()


async def _startapp():
    session = aiohttp.ClientSession()
```

```
    await start_node(session=session)
    await after_start_node()
    await session.close()


@app.on_event("startup")
async def startup_event():
    app.is_ready = False
    app.is_waiting = True
    asyncio.create_task(_startapp())
    asyncio.create_task(process_blocks_forever(app=app))


logging.basicConfig(level="INFO")
app.include_router(alive_router)
app.include_router(block_router)
app.include_router(receive_transaction_router)
app.include_router(receive_block_router)
app.include_router(user_router)
app.include_router(transactions_router)
```

Файл: ./node/routers/user.py

```
from fastapi import APIRouter
from starlette.requests import Request

from chain.transaction import calculate_balance

router = APIRouter()


@router.post("/get_balance")
async def get_balance_handler(request: Request):
    address = (await request.json())["address"]

    balance = await calculate_balance(address=address)
    if balance == float("inf"):
        balance = "ЭМИССИОНЕР"
    return {"balance": balance}
```

Файл: ./node/routers/transactions.py

```
from fastapi import APIRouter
from starlette.requests import Request

from chain.block import get_block_by_hash, get_block_by_number
from chain.transaction import (
    get_transactions_by_address,
    get_transaction_by_hash,
    get_transactions_by_block_hash,
)

router = APIRouter()


@router.post("/get_transactions")
async def get_transactions_handler(request: Request):
    request_data = await request.json()
    address = request_data["address"]
    transactions_type = request_data["transaction_type"]

    return {
        "transactions": await get_transactions_by_address(
            address=address, transactions_type=transactions_type
        )
    }


@router.post("/get_transactions_by_block")
async def get_transactions_by_block_handler(request: Request):
    request_data = await request.json()
```

```
    block_hash = request_data["block_hash"]

    return {"transactions": await get_transactions_by_block_hash(block_hash=block_hash)}


@router.post("/get_transaction")
async def get_transaction_handler(request: Request):
    request_data = await request.json()
    transaction_hash = request_data["transaction_hash"]

    data = await get_transaction_by_hash(transaction_hash=transaction_hash)
    if data is None:
        return {"transaction": None}

    result = data.dict()

    if data.block_number:
        block = await get_block_by_number(block_number=data.block_number)
        result["authority_id"] = block.authority_id
        result["block_number"] = block.block_number
        result["block_hash"] = block.block_hash
    else:
        result["block_number"] = "Не подтверждено."
        result["authority_id"] = "Не подтверждено."
        result["block_hash"] = ""

    return {"transaction": result}
```

Файл: ./node/routers/__init__.py

```
from .alive import router as alive_router
from .block import router as block_router
from .receive_transaction import router as receive_transaction_router
from .receive_block import router as receive_block_router
from .user import router as user_router
from .transactions import router as transactions_router
```

Файл: ./node/routers/alive.py

```
from fastapi import APIRouter

router = APIRouter()


@router.get("/is_alive")
async def is_alive():
    return {"alive": True}
```

Файл: ./node/routers/receive_transaction.py

```
from fastapi import APIRouter
from starlette.requests import Request

from chain.timestamps import get_current_accurate_timestamp
from chain.transaction import calculate_balance, create_transaction
from crypto.converter import expand_transaction_from_request, normalize_transaction
from crypto.sign import verify_transaction_sign


router = APIRouter()


@router.post("/mempool")
async def add_to_mempool_handler(request: Request):
    transaction_data = normalize_transaction((await request.json())["data"])

    is_valid = verify_transaction_sign(transaction=transaction_data)

    if not is_valid:
        return {
```

```
            "status": False,
            "code": "Bad signature",
            "description": "Неверная подпись",
        }

    transaction_model = expand_transaction_from_request(transaction=transaction_data)

    if transaction_model.amount < 1:
        return {
            "status": False,
            "code": "Bad amount",
            "description": "Слишком маленькая сумма.",
        }

    if transaction_model.sender_address == transaction_model.recipient_address:
        return {
            "status": False,
            "code": "Bad address",
            "description": "Нельзя отправить монеты самому себе.",
        }

    sender_balance = await calculate_balance(
        address=transaction_model.sender_address,
        exclude_hash=transaction_model.transaction_hash,
    )

    if sender_balance < transaction_model.amount:
        return {
            "status": False,
            "code": "Insufficient funds",
            "description": "Недостаточно средств.",
        }

    if transaction_model.timestamp > get_current_accurate_timestamp():
        return {
            "status": False,
            "code": "Invalid timestamp",
            "description": "Ошибка. Транзакция из будущего.",
        }

    await create_transaction(transaction=transaction_model)
    return {"status": True, "hash": transaction_model.transaction_hash}


Файл: ./node/routers/block.py

from fastapi import APIRouter
from starlette.requests import Request

from chain.block import (
    get_last_block_number,
    get_blocks_until_previous_hash,
    get_blocks,
    get_blocks_count,
    get_block_by_hash,
    get_block_by_number,
)
from chain.transaction import get_block_transactions

router = APIRouter()


@router.get("/get_last_block_number")
async def get_last_block_number_handler():
    return {"last_block_number": await get_last_block_number()}


@router.post("/get_blocks_until_hash")
async def get_blocks_until_hash(request: Request):
    last_block_previous_hash = (await request.json())["last_block_previous_hash"]
    return {
        "blocks": [
            block.to_dict(transactions=await get_block_transactions(block_id=block.id))
```

```
            for block in await get_blocks_until_previous_hash(
                last_block_previous_hash=last_block_previous_hash
            )
        ]
    }


@router.post("/get_blocks")
async def get_blocks_handler(request: Request):
    limit = request.query_params.get("limit", 100)
    offset = request.query_params.get("offset", 0)

    return {
        "blocks": [
            block.to_dict(transactions=await get_block_transactions(block_id=block.id))
            for block in await get_blocks(limit=limit, offset=offset)
        ],
        "total_count": await get_blocks_count(),
    }


@router.post("/get_block")
async def get_block_handler(request: Request):
    request_data = await request.json()
    block_hash = request_data["block_hash"]

    return {"block": await get_block_by_hash(block_hash=block_hash)}


@router.post("/get_block_by_number")
async def get_block_by_numberhandler(request: Request):
    request_data = await request.json()
    block_number = request_data["block_number"]

    return {"block": await get_block_by_number(block_number=block_number)}
```

Файл: ./node/routers/receive_block.py

```
import aiohttp
from fastapi import APIRouter
from starlette.requests import Request

from chain.block import add_new_blocks_from_node
from chain.db.session import async_session
from crypto.sign import text_sign_verify
from node.blockchain.balancer import is_previous_node
from node.blockchain.validate import validate_block_with_transactions
from node.models.block import BlockModel
from node.utils import get_node_by_id

router = APIRouter()


@router.get("/is_ready")
async def is_ready_handler(request: Request):
    return {"is_ready": request.app.is_ready}


@router.post("/receive_notify")
async def receive_notify_handler(request: Request):
    notify_data = (await request.json())["notify_data"]
    node_id = notify_data["node_id"]
    node_message = notify_data["message"]
    node_sign = notify_data["sign"]

    node = get_node_by_id(node_id=node_id)

    if node is None:
        return {"status": False, "description": "Node not found"}

    verify_result = text_sign_verify(
        signature=node_sign, original_message=node_message, public_key=node.public_key
```

```
    )

    if not verify_result:
        return {"status": False, "description": "Bad sign"}

    request.app.is_ready = True
    return {"status": True}


@router.post("/receive_block")
async def receive_block_handler(request: Request):
    block_data_raw = (await request.json())["block_data"]
    block_data = BlockModel(**block_data_raw["block_data"])
    node_sign = block_data_raw["sign"]

    node = get_node_by_id(node_id=block_data.authority_id)

    if node is None:
        return {"status": False, "description": "Node not found"}

    verify_result = text_sign_verify(
        signature=node_sign,
        original_message=str(block_data.dict()),
        public_key=node.public_key,
    )

    if not verify_result:
        return {"status": False, "description": "Bad sign"}

    async with async_session() as sql_session:
        verify_result = await validate_block_with_transactions(
            session=sql_session, block=block_data
        )

        sql_session.expunge_all()

    if await is_previous_node(session=aiohttp.ClientSession(), node=node):
        request.app.is_waiting = False

    if not verify_result.status:
        return {"status": False, "description": "Bad verify"}

    await add_new_blocks_from_node(new_blocks=verify_result.new_blocks)

    return {"status": True}
```

Файл: ./node/models/transaction.py

```
import datetime
import hashlib

import pydantic


class TransactionModel(pydantic.BaseModel):
    sender_address: str
    recipient_address: str
    amount: int
    timestamp: int
    transaction_hash: str | None = None
    block_number: int | None = None
```

Файл: ./node/models/__init__.py

Файл: ./node/models/block.py

```
import pydantic

from chain.db import Block
```

```
from node.models.transaction import TransactionModel


class BlockModel(pydantic.BaseModel):
    block_number: int
    block_hash: str | None = None
    previous_hash: str
    merkle_root: str | None = None
    authority_id: str
    timestamp: int
    transactions: list[TransactionModel] | None = None


class NewBlocksModel(pydantic.BaseModel):
    blocks: list[BlockModel]
    total_count: int | None = None


Файл: ./node/blockchain/startup.py

import logging

import aiohttp
from sqlalchemy.ext.asyncio import AsyncSession

from chain.block import (
    get_last_block_previous_hash,
    create_block,
    add_new_blocks_from_node,
    get_blocks_count,
    calculate_block_hash,
)
from chain.db.session import async_session
from chain.transaction import create_transaction
from chain_config import NodeConfig
from node.api.block import get_blocks_until_hash_from_node
from node.blockchain.balancer import get_suitable_node_url
from node.blockchain.validate import validate_transaction, validate_block
from node.models.block import BlockModel
from node.structs.block import BlocksVerifyResult


async def process_latest_blocks(
    session: AsyncSession,
    suitable_node_url: str,
    http_session: aiohttp.ClientSession,
    last_block_previous_hash: str,
) -> BlocksVerifyResult:
    latest_blocks = await get_blocks_until_hash_from_node(
        url=suitable_node_url,
        session=http_session,
        last_block_previous_hash=last_block_previous_hash,
    )

    for block in latest_blocks.blocks:
        if not await validate_block(session=session, block=block):
            return BlocksVerifyResult(
                status=False,
                suitable_node_url=suitable_node_url,
            )

        for transaction in block.transactions:
            if not await validate_transaction(session=session, transaction=transaction):
                return BlocksVerifyResult(
                    status=False,
                    suitable_node_url=suitable_node_url,
                )

            await create_transaction(
                session=session, transaction=transaction, with_commit=False
            )

        await create_block(session=session, block=block, with_commit=False)
```

```
    logging.info(f"got {len(latest_blocks.blocks)} new blocks after sync")
    return BlocksVerifyResult(
        status=True, suitable_node_url=suitable_node_url, new_blocks=latest_blocks
    )


async def start_node(session: aiohttp.ClientSession):
    success = False

    bad_urls = []
    while not success:
        suitable_node_url = await get_suitable_node_url(
            session=session, exclude_urls=bad_urls
        )
        logging.info(f"Trying sync with {suitable_node_url}")

        if suitable_node_url is None:
            # мы одни синкать нечего
            logging.info("No suitable node for sync")
            return

        # мы не одни начинаем качать блоки
        last_block_previous_hash = await get_last_block_previous_hash()

        async with async_session() as sql_session:
            # получили все новые блоки с транзакциями
            verify_result = await process_latest_blocks(
                session=sql_session,
                last_block_previous_hash=last_block_previous_hash,
                http_session=session,
                suitable_node_url=suitable_node_url,
            )

            sql_session.expunge_all()

        if verify_result.status:
            await add_new_blocks_from_node(new_blocks=verify_result.new_blocks)
            return

        bad_urls.append(verify_result.suitable_node_url)


async def after_start_node():
    blocks_count = await get_blocks_count()
    if blocks_count > 0:
        return

    genesis_block = BlockModel(
        block_number=1,
        previous_hash="genesis",
        authority_id=NodeConfig.title_id,
        timestamp=1,
    )
    genesis_block.block_hash = calculate_block_hash(block=genesis_block)

    await create_block(block=genesis_block)
```

Файл: ./node/blockchain/block_processing.py

```
import asyncio
import logging

import aiohttp
from fastapi import FastAPI

from chain.block import (
    get_last_block_timestamp,
    get_last_block,
    calculate_block_hash,
    create_block,
    update_transactions_for_block,
```

```
)
from chain.constants import MIN_BLOCK_RELEASE_TIME
from chain.timestamps import get_current_accurate_timestamp
from chain.transaction import (
    get_unconfirmed_transactions,
    calculate_block_merkle_root,
    delete_transaction,
)
from chain_config import NodeConfig
from node.api.distributor import send_block_release_notify, send_block_release_created
from node.blockchain.balancer import get_active_ready_nodes
from node.blockchain.validate import validate_transaction
from node.models.block import BlockModel
from node.models.transaction import TransactionModel


async def is_time_for_release():
    current_timestamp = get_current_accurate_timestamp()
    last_block_timestamp = await get_last_block_timestamp()
    time_from_last_block = current_timestamp - last_block_timestamp
    return time_from_last_block >= MIN_BLOCK_RELEASE_TIME


async def get_valid_unconfirmed_transactions() -> list[TransactionModel]:
    unconfirmed_transactions = await get_unconfirmed_transactions()
    valid_unconfirmed_transactions = []

    for unconfirmed_transaction in unconfirmed_transactions:
        is_valid = await validate_transaction(transaction=unconfirmed_transaction)
        if not is_valid:
            await delete_transaction(transaction_id=unconfirmed_transaction.id)
            continue
        valid_unconfirmed_transactions.append(unconfirmed_transaction)

    return valid_unconfirmed_transactions


async def release_block(session: aiohttp.ClientSession):
    unconfirmed_transactions = await get_valid_unconfirmed_transactions()

    last_block = await get_last_block()
    new_block = BlockModel(
        block_number=last_block.block_number + 1,
        previous_hash=last_block.block_hash,
        authority_id=NodeConfig.title_id,
        timestamp=get_current_accurate_timestamp(),
    )

    new_block.merkle_root = calculate_block_merkle_root(
        transactions=unconfirmed_transactions
    )
    new_block.block_hash = calculate_block_hash(block=new_block)

    logging.info(f"releasing block {new_block}")

    await send_block_release_notify(session=session)
    new_block = await create_block(block=new_block)
    await update_transactions_for_block(
        block_id=new_block.id,
        block_number=new_block.block_number,
        transactions=unconfirmed_transactions,
    )
    await send_block_release_created(
        session=session, block=new_block, block_transactions=unconfirmed_transactions
    )


async def process_blocks_forever(app: FastAPI):
    await asyncio.sleep(3)
    session = aiohttp.ClientSession()
    while True:
        await asyncio.sleep(0.1)
```

```
        if not await is_time_for_release():
            continue

        active_ready_nodes = await get_active_ready_nodes(session=session)
        if not active_ready_nodes:
            app.is_ready = True
            await release_block(session=session)
            continue

        if not app.is_ready:
            continue

        if not app.is_waiting:
            await release_block(session=session)
            app.is_waiting = True
            continue
```

Файл: ./node/blockchain/__init__.py

Файл: ./node/blockchain/validate.py

```
from sqlalchemy.ext.asyncio import AsyncSession

from chain.block import (
    get_last_block,
    calculate_block_hash,
    create_block,
    get_block_by_hash,
)
from chain.constants import GENESIS_BLOCK_PREVIOUS_HASH
from chain.db.session import db_session
from chain.timestamps import get_current_accurate_timestamp
from chain.transaction import (
    calculate_balance,
    create_transaction,
    calculate_transaction_hash,
)
from node.models.block import BlockModel, NewBlocksModel
from node.models.transaction import TransactionModel
from node.structs.block import BlocksVerifyResult


@db_session
async def validate_transaction(
    session: AsyncSession, transaction: TransactionModel
) -> bool:
    if transaction.transaction_hash != calculate_transaction_hash(
        transaction=transaction
    ):
        return False

    if transaction.sender_address == transaction.recipient_address:
        return False

    if transaction.timestamp > get_current_accurate_timestamp():
        return False

    sender_balance = await calculate_balance(
        session=session,
        address=transaction.sender_address,
        exclude_hash=transaction.transaction_hash,
    )

    if sender_balance < transaction.amount:
        return False

    return True


@db_session
```

```
async def validate_block(session: AsyncSession, block: BlockModel):
    is_block_exists = await get_block_by_hash(block_hash=block.block_hash) is not None
    if is_block_exists:
        return False

    previous_block = await get_last_block(session=session)

    is_genesis = block.previous_hash == GENESIS_BLOCK_PREVIOUS_HASH

    if not is_genesis and block.previous_hash != previous_block.block_hash:
        return False

    if not is_genesis and block.block_number - 1 != previous_block.block_number:
        return False

    if block.timestamp > get_current_accurate_timestamp():
        return False

    if not is_genesis and block.timestamp < previous_block.timestamp:
        return False

    if block.block_hash != calculate_block_hash(block=block):
        return False

    return True


async def validate_block_with_transactions(
    session: AsyncSession,
    block: BlockModel,
) -> BlocksVerifyResult:
    if not await validate_block(session=session, block=block):
        return BlocksVerifyResult(
            status=False,
        )

    for transaction in block.transactions:
        if not await validate_transaction(session=session, transaction=transaction):
            return BlocksVerifyResult(status=False)

        await create_transaction(
            session=session, transaction=transaction, with_commit=False
        )

    await create_block(session=session, block=block, with_commit=False)

    return BlocksVerifyResult(status=True, new_blocks=NewBlocksModel(blocks=[block]))
```

Файл: ./node/blockchain/balancer.py

```
import aiohttp

from chain_config import NodeConfig
from node.api.status import is_node_online, is_node_ready
from node.api.block import get_last_block_number_from_node
from node.utils import get_node_by_id
from node_constants import ALL_NODES, NodeConstant, sort_nodes


async def get_suitable_node_url(
    session: aiohttp.ClientSession, exclude_urls: list[str]
) -> str | None:
    biggest_block_number = -1
    suitable_node_url = None

    for node in ALL_NODES:
        if node.title_id == NodeConfig.title_id:
            continue

        if node.url in exclude_urls:
            continue
```

```
        if not await is_node_online(url=node.url, session=session):
            continue

        last_node_block_number = await get_last_block_number_from_node(
            url=node.url, session=session
        )

        if last_node_block_number > biggest_block_number:
            biggest_block_number = last_node_block_number
            suitable_node_url = node.url

    return suitable_node_url


async def get_active_nodes(session: aiohttp.ClientSession) -> list[NodeConstant]:
    result = []

    for node in ALL_NODES:
        if node.title_id == NodeConfig.title_id:
            continue

        if not await is_node_online(url=node.url, session=session):
            continue

        result.append(node)

    return sort_nodes(result)


async def get_active_ready_nodes(session: aiohttp.ClientSession) -> list[NodeConstant]:
    result = []

    for node in ALL_NODES:
        if node.title_id == NodeConfig.title_id:
            continue

        if not await is_node_online(url=node.url, session=session):
            continue

        if not await is_node_ready(url=node.url, session=session):
            continue

        result.append(node)

    return sort_nodes(result)


async def is_previous_node(session: aiohttp.ClientSession, node: NodeConstant) -> bool:
    active_ready_nodes = await get_active_ready_nodes(session=session)

    current_node = get_node_by_id(node_id=NodeConfig.title_id)
    active_ready_nodes.append(current_node)
    active_ready_nodes = sort_nodes(active_ready_nodes)

    current_node_index = active_ready_nodes.index(current_node)
    previous_node = active_ready_nodes[current_node_index - 1]

    return node.title_id == previous_node.title_id
```

Файл: ./node/structs/__init__.py

Файл: ./node/structs/block.py

```
import dataclasses

from node.models.block import NewBlocksModel


@dataclasses.dataclass
class BlocksVerifyResult:
```

```
    status: bool
    suitable_node_url: str | None = None
    new_blocks: NewBlocksModel | None = None
```

Файл: ./node/api/user.py

```
import aiohttp
from aiohttp import ClientConnectorError


async def get_address_balance(
    url: str, session: aiohttp.ClientSession, address: str
) -> bool:
    try:
        response = await session.post(f"{url}/get_balance", json={"address": address})
    except ClientConnectorError:
        return False
    if response.status != 200:
        return False

    return (await response.json())["balance"]
```

Файл: ./node/api/distributor.py

```
import asyncio
import random

import aiohttp

from chain.db import Block, Transaction
from chain_config import NodeConfig
from crypto.sign import sign_message
from node.blockchain.balancer import get_active_nodes, get_active_ready_nodes
from node.models.block import BlockModel
from node.models.transaction import TransactionModel


async def send_block_release_notify(session: aiohttp.ClientSession) -> None:
    active_nodes = await get_active_nodes(session=session)
    message = str(NodeConfig.block_notify_message.format(random_data=random.random()))

    tasks = []

    for node in active_nodes:
        tasks.append(
            session.post(
                f"{node.url}/receive_notify",
                json={
                    "notify_data": {
                        "node_id": NodeConfig.title_id,
                        "message": message,
                        "sign": sign_message(
                            message=message, private_key=NodeConfig.private_key
                        ),
                    }
                },
            )
        )

    await asyncio.gather(*tasks)


async def send_block_release_created(
    session: aiohttp.ClientSession,
    block: Block,
    block_transactions: list[TransactionModel],
) -> None:
    active_nodes = await get_active_ready_nodes(session=session)
    block_data = BlockModel(**block.to_dict(transactions=block_transactions))

    tasks = []
```

```
    for node in active_nodes:
        tasks.append(
            session.post(
                f"{node.url}/receive_block",
                json={
                    "block_data": {
                        "block_data": block_data.dict(),
                        "sign": sign_message(
                            message=str(block_data.dict()),
                            private_key=NodeConfig.private_key,
                        ),
                    }
                },
            )
        )

    result = await asyncio.gather(*tasks)
```

Файл: ./node/api/transaction.py

```
import aiohttp
from aiohttp import ClientConnectorError


async def send_transaction_to_mempool(
    url: str, session: aiohttp.ClientSession, transaction_data: str
) -> bool:
    try:
        response = await session.post(f"{url}/mempool", json={"data": transaction_data})
    except ClientConnectorError:
        return False
    if response.status != 200:
        return False

    return await response.json()


async def get_transactions_from_node(
    url: str, session: aiohttp.ClientSession, address: str, transaction_type: str
) -> dict:
    try:
        response = await session.post(
            f"{url}/get_transactions",
            json={"address": address, "transaction_type": transaction_type},
        )
    except ClientConnectorError:
        return {"transactions": []}
    if response.status != 200:
        return {"transactions": []}

    return await response.json()


async def get_transactions_by_block_from_node(
    url: str, session: aiohttp.ClientSession, block_hash: str
) -> dict:
    try:
        response = await session.post(
            f"{url}/get_transactions_by_block",
            json={"block_hash": block_hash},
        )
    except ClientConnectorError:
        return {"transactions": []}
    if response.status != 200:
        return {"transactions": []}

    return await response.json()


async def get_transaction_from_node(
    url: str, session: aiohttp.ClientSession, transaction_hash: str
```

```
) -> dict:
    try:
        response = await session.post(
            f"{url}/get_transaction",
            json={"transaction_hash": transaction_hash},
        )
    except ClientConnectorError:
        return {"transaction": None}
    if response.status != 200:
        return {"transaction": None}

    return await response.json()
```

Файл: ./node/api/__init__.py

Файл: ./node/api/block.py

```
import aiohttp
from aiohttp import ClientConnectorError

from node.models.block import NewBlocksModel


async def get_last_block_number_from_node(
    url: str, session: aiohttp.ClientSession
) -> int:
    response = await session.get(f"{url}/get_last_block_number")

    if response.status != 200:
        return -1

    return (await response.json())["last_block_number"]


async def get_blocks_until_hash_from_node(
    url: str, session: aiohttp.ClientSession, last_block_previous_hash: str
) -> NewBlocksModel:
    response = await session.post(
        f"{url}/get_blocks_until_hash",
        json={"last_block_previous_hash": last_block_previous_hash},
    )

    if response.status != 200:
        return NewBlocksModel(blocks=[])

    return NewBlocksModel(**await response.json())


async def get_blocks_from_node(
    url: str, session: aiohttp.ClientSession, limit: int, offset: int
) -> NewBlocksModel:
    response = await session.post(
        f"{url}/get_blocks",
        params={"limit": limit, "offset": offset},
    )

    if response.status != 200:
        return NewBlocksModel(blocks=[])

    return NewBlocksModel(**await response.json())


async def get_block_from_node(
    url: str, session: aiohttp.ClientSession, block_hash: str
) -> dict:
    try:
        response = await session.post(
            f"{url}/get_block",
            json={"block_hash": block_hash},
```

```
        )
    except ClientConnectorError:
        return {"block": None}
    if response.status != 200:
        return {"block": None}

    return await response.json()


async def get_block_by_number_from_node(
    url: str, session: aiohttp.ClientSession, block_number: str
) -> dict:
    try:
        response = await session.post(
            f"{url}/get_block_by_number",
            json={"block_number": block_number},
        )
    except ClientConnectorError:
        return {"block": None}
    if response.status != 200:
        return {"block": None}

    return await response.json()
```

Файл: ./node/api/status.py

```
import aiohttp
from aiohttp import ClientConnectorError


async def is_node_online(url: str, session: aiohttp.ClientSession) -> bool:
    try:
        response = await session.get(f"{url}/is_alive")
    except ClientConnectorError:
        return False
    if response.status != 200:
        return False

    return (await response.json())["alive"]


async def is_node_ready(url: str, session: aiohttp.ClientSession) -> bool:
    try:
        response = await session.get(f"{url}/is_ready")
    except ClientConnectorError:
        return False
    if response.status != 200:
        return False

    return (await response.json())["is_ready"]
```

Файл: ./migration/env.py

```
from logging.config import fileConfig

from sqlalchemy import engine_from_config
from sqlalchemy import pool

from alembic import context

from chain_config import PostgresConfig

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
if config.config_file_name is not None:
    fileConfig(config.config_file_name)
```

```
from chain.db import Base

target_metadata = Base.metadata

config.set_main_option("sqlalchemy.url", PostgresConfig.db_url_alembic)


def run_migrations_offline() -> None:
    """Run migrations in 'offline' mode.

    This configures the context with just a URL
    and not an Engine, though an Engine is acceptable
    here as well.  By skipping the Engine creation
    we don't even need a DBAPI to be available.

    Calls to context.execute() here emit the given string to the
    script output.

    """
    url = config.get_main_option("sqlalchemy.url")
    context.configure(
        url=url,
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
        context.run_migrations()


def run_migrations_online() -> None:
    """Run migrations in 'online' mode.

    In this scenario we need to create an Engine
    and associate a connection with the context.

    """
    connectable = engine_from_config(
        config.get_section(config.config_ini_section, {}),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
        context.configure(connection=connection, target_metadata=target_metadata)

        with context.begin_transaction():
            context.run_migrations()


if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Файл: ./migration/versions/dd7cc63002ed_.py

```
"""empty message

Revision ID: dd7cc63002ed
Revises: 004766e0c634
Create Date: 2024-04-02 14:09:21.012825

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa
```

```
# revision identifiers, used by Alembic.
revision: str = "dd7cc63002ed"
down_revision: Union[str, None] = "004766e0c634"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column("block", sa.Column("authority_id", sa.String(), nullable=False))
    op.drop_constraint("block_nonce_key", "block", type_="unique")
    op.drop_column("block", "nonce")
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column(
        "block", sa.Column("nonce", sa.VARCHAR(), autoincrement=False, nullable=False)
    )
    op.create_unique_constraint("block_nonce_key", "block", ["nonce"])
    op.drop_column("block", "authority_id")
    # ### end Alembic commands ###
```

Файл: ./migration/versions/004766e0c634_.py

```
"""empty message

Revision ID: 004766e0c634
Revises: 9f31208dc33e
Create Date: 2024-03-26 14:07:52.255050

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "004766e0c634"
down_revision: Union[str, None] = "9f31208dc33e"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_index(
        op.f("ix_transactions_recipient_address"),
        "transactions",
        ["recipient_address"],
        unique=False,
    )
    op.create_index(
        op.f("ix_transactions_sender_address"),
        "transactions",
        ["sender_address"],
        unique=False,
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_index(op.f("ix_transactions_sender_address"), table_name="transactions")
    op.drop_index(op.f("ix_transactions_recipient_address"), table_name="transactions")
    # ### end Alembic commands ###
```

Файл: ./migration/versions/7477f995b497_initial.py

```
"""initial

Revision ID: 7477f995b497
Revises:
Create Date: 2024-03-26 13:48:15.091904

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "7477f995b497"
down_revision: Union[str, None] = None
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table(
        "block",
        sa.Column("id", sa.Integer(), nullable=False),
        sa.Column("block_number", sa.Integer(), nullable=False),
        sa.Column("block_hash", sa.String(), nullable=False),
        sa.Column("previous_hash", sa.String(), nullable=False),
        sa.Column("nonce", sa.String(), nullable=False),
        sa.Column("merkle_root", sa.String(), nullable=False),
        sa.Column("timestamp", sa.BigInteger(), nullable=False),
        sa.PrimaryKeyConstraint("id"),
        sa.UniqueConstraint("block_hash"),
        sa.UniqueConstraint("block_number"),
        sa.UniqueConstraint("merkle_root"),
        sa.UniqueConstraint("nonce"),
        sa.UniqueConstraint("previous_hash"),
    )
    op.create_table(
        "transactions",
        sa.Column("id", sa.Integer(), nullable=False),
        sa.Column("sender_address", sa.String(), nullable=False),
        sa.Column("recipient_address", sa.String(), nullable=False),
        sa.Column("amount", sa.DECIMAL(precision=2), nullable=False),
        sa.Column("timestamp", sa.BigInteger(), nullable=False),
        sa.Column("transaction_hash", sa.String(), nullable=False),
        sa.Column("block_id", sa.Integer(), nullable=True),
        sa.ForeignKeyConstraint(
            ["block_id"],
            ["block.id"],
        ),
        sa.PrimaryKeyConstraint("id"),
        sa.UniqueConstraint("transaction_hash"),
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table("transactions")
    op.drop_table("block")
    # ### end Alembic commands ###
```

Файл: ./migration/versions/8da8420576da_.py

```
"""empty message

Revision ID: 8da8420576da
Revises: efc133a3c757
```

Create Date: 2024-03-26 13:57:34.347806

"""

```
from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "8da8420576da"
down_revision: Union[str, None] = "efc133a3c757"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    pass
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    pass
    # ### end Alembic commands ###
```

Файл: ./migration/versions/efc133a3c757_.py

"""empty message

Revision ID: efc133a3c757
Revises: 7477f995b497
Create Date: 2024-03-26 13:49:49.929138

"""

```
from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "efc133a3c757"
down_revision: Union[str, None] = "7477f995b497"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column(
        "transactions",
        "amount",
        existing_type=sa.NUMERIC(precision=2, scale=0),
        type_=sa.Numeric(precision=2, scale=2),
        existing_nullable=False,
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column(
        "transactions",
        "amount",
        existing_type=sa.Numeric(precision=2, scale=2),
        type_=sa.NUMERIC(precision=2, scale=0),
        existing_nullable=False,
    )
```

```
    # ### end Alembic commands ###
```

Файл: ./migration/versions/81396b4c6d04_.py

```
"""empty message

Revision ID: 81396b4c6d04
Revises: ee3fd4166e49
Create Date: 2024-04-23 21:07:56.789869

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "81396b4c6d04"
down_revision: Union[str, None] = "ee3fd4166e49"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column(
        "transactions", sa.Column("block_number", sa.Integer(), nullable=True)
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_column("transactions", "block_number")
    # ### end Alembic commands ###
```

Файл: ./migration/versions/9f31208dc33e_.py

```
"""empty message

Revision ID: 9f31208dc33e
Revises: 8da8420576da
Create Date: 2024-03-26 14:01:37.751028

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "9f31208dc33e"
down_revision: Union[str, None] = "8da8420576da"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column(
        "transactions",
        "amount",
        existing_type=sa.NUMERIC(precision=2, scale=2),
        type_=sa.BIGINT(),
        existing_nullable=False,
    )
    # ### end Alembic commands ###
```

```
def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column(
        "transactions",
        "amount",
        existing_type=sa.BIGINT(),
        type_=sa.NUMERIC(precision=2, scale=2),
        existing_nullable=False,
    )
    # ### end Alembic commands ###
```

Файл: ./migration/versions/ee3fd4166e49_.py

```
"""empty message

Revision ID: ee3fd4166e49
Revises: dd7cc63002ed
Create Date: 2024-04-10 23:40:17.977040

"""

from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = "ee3fd4166e49"
down_revision: Union[str, None] = "dd7cc63002ed"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column("block", "merkle_root", existing_type=sa.VARCHAR(), nullable=True)
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column("block", "merkle_root", existing_type=sa.VARCHAR(), nullable=False)
    # ### end Alembic commands ###
```

Приложение Б. Скриншоты программы

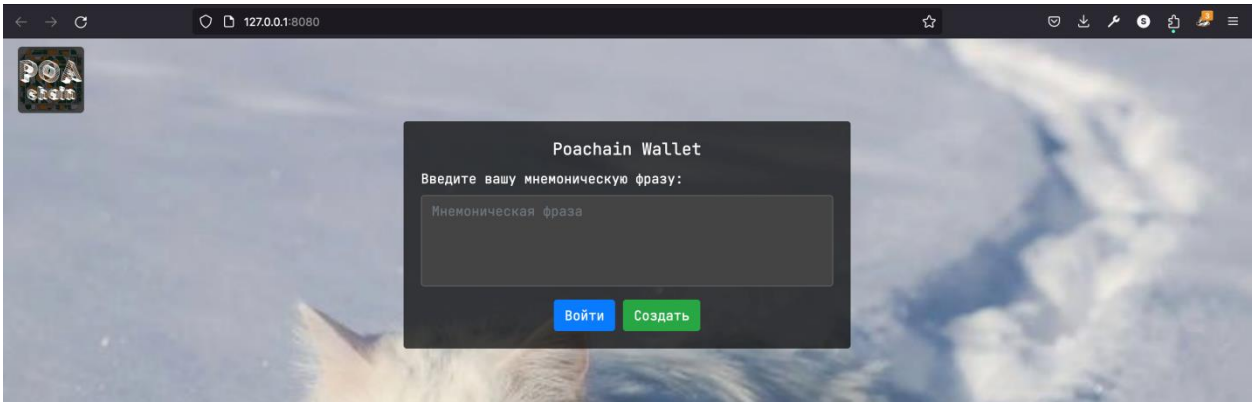

Рисунок Б.1 – Страница входа

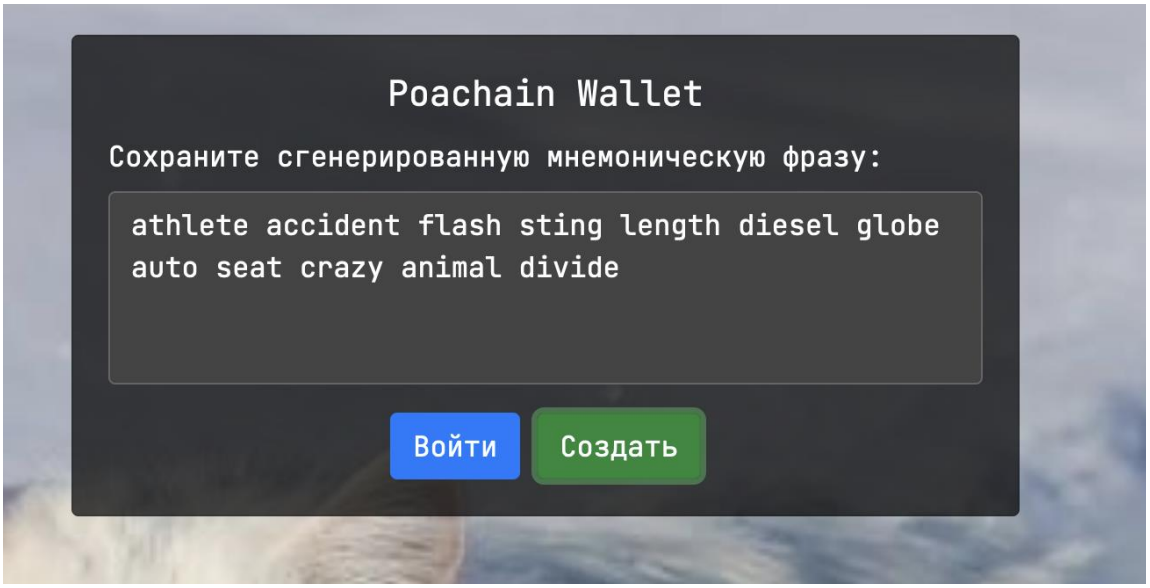

Рисунок Б.2 – Страница входа со свежесгенерированной мнемонической фразой

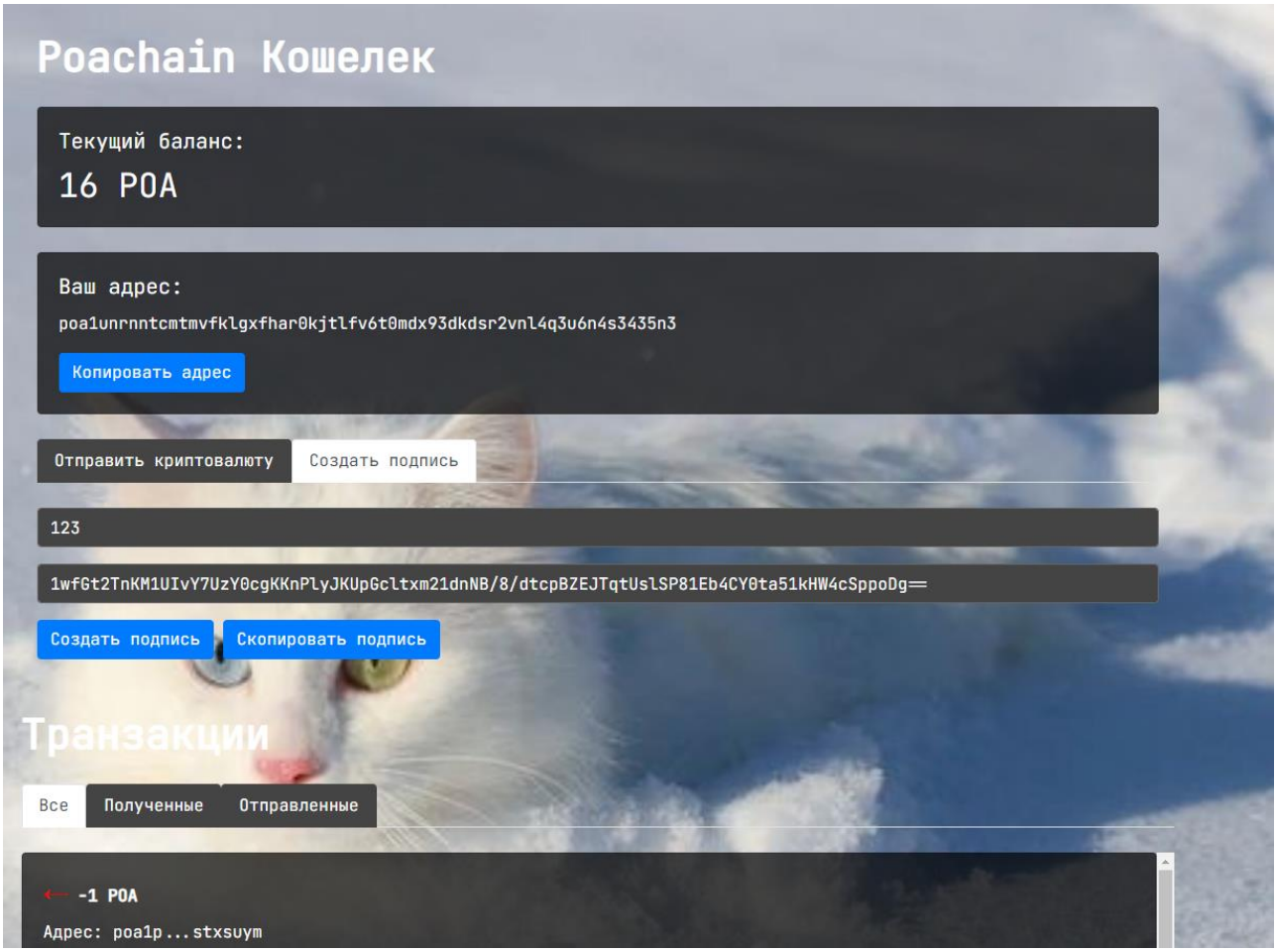

Рисунок Б.3 – Раздел кошелька целиком

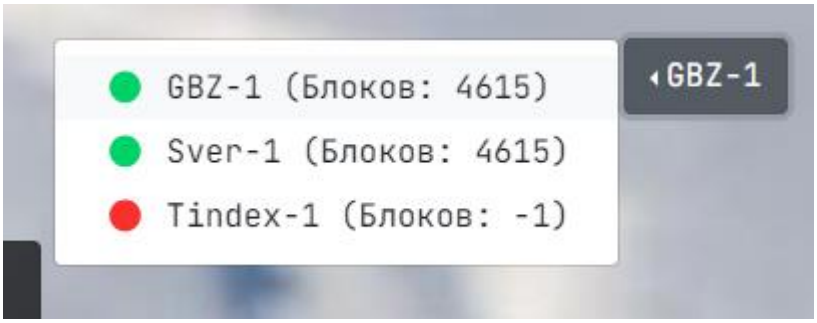

Рисунок Б.4 – Интерфейс обзора и выбора нод

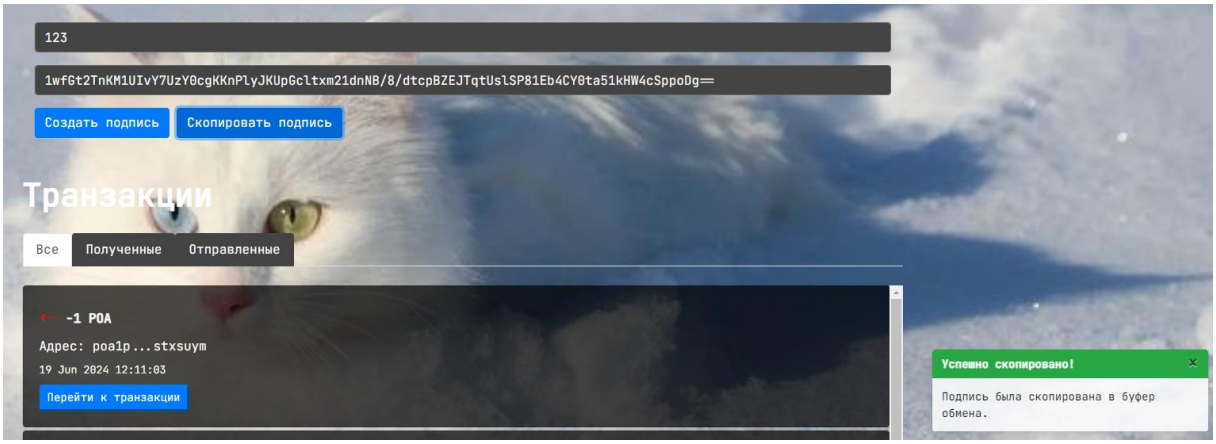

Рисунок Б.5 - Интерфейс подписи

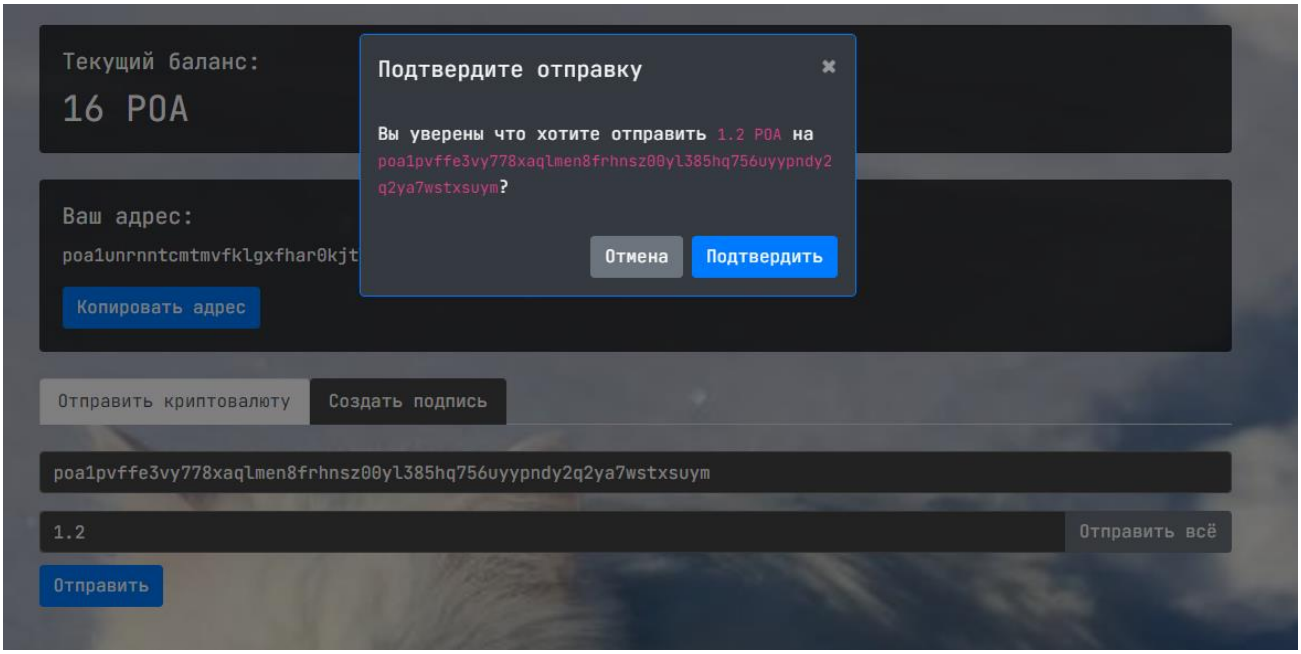

Рисунок Б.6 – Интерфейс отправки криптовалюты

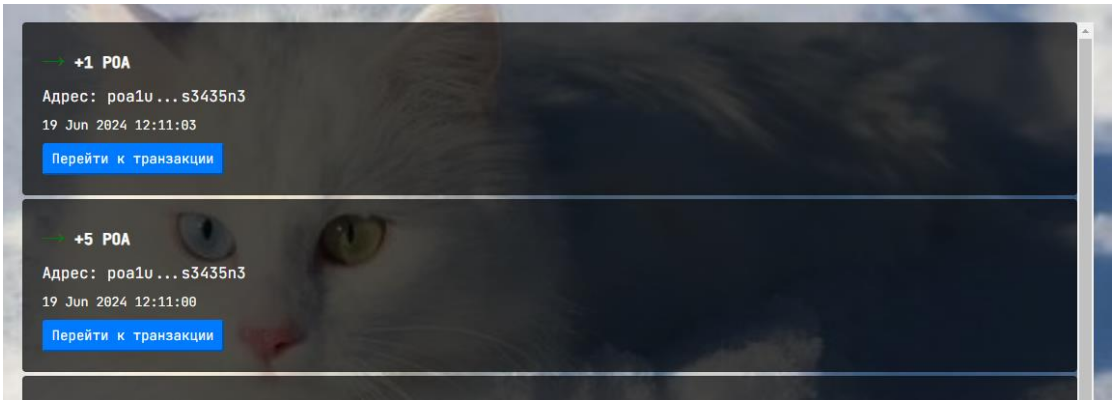

Рисунок Б.7 - Интерфейс обзора транзакций, в котором видно полученные транзакции

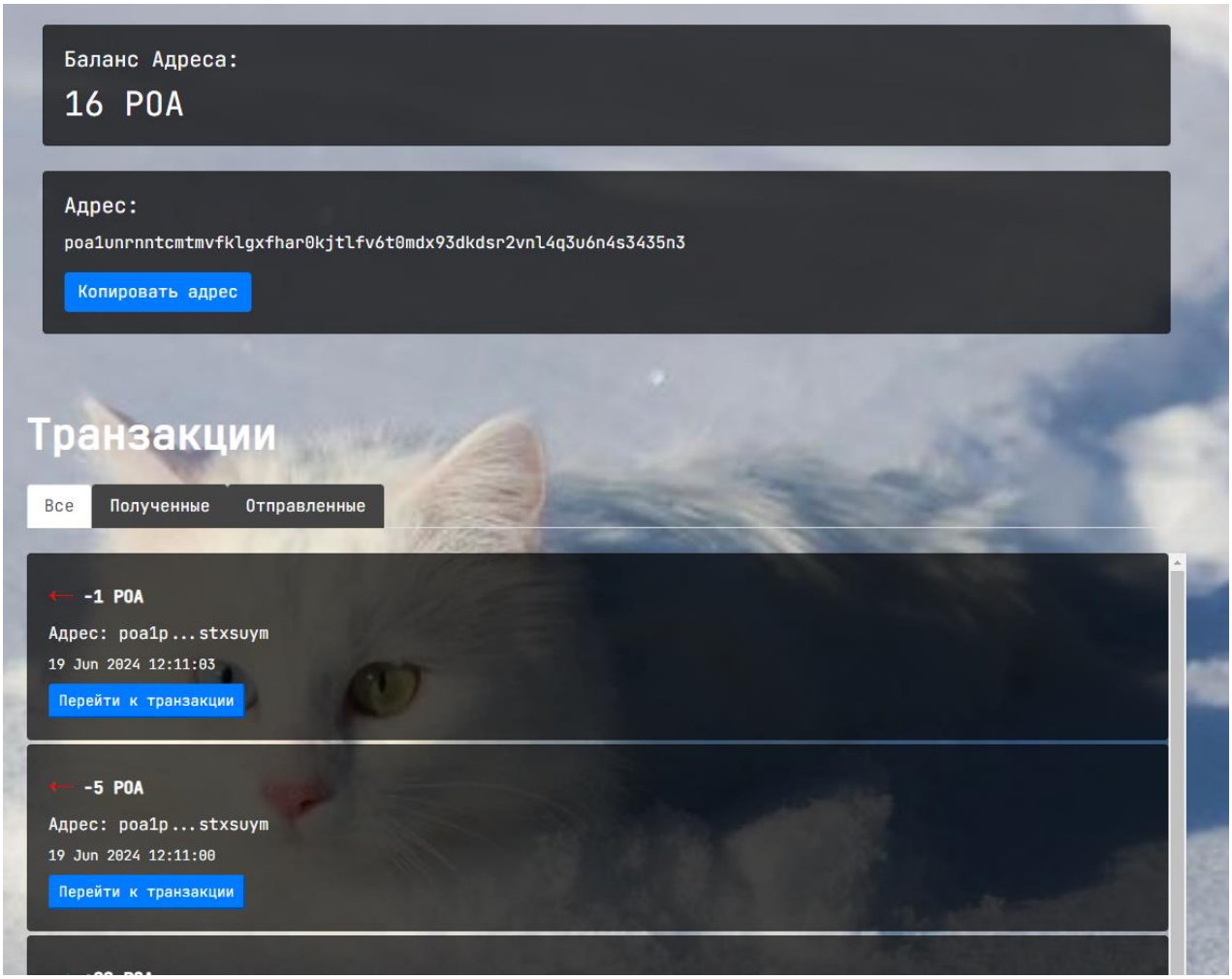

Рисунок Б.8 - Страница просмотра адреса

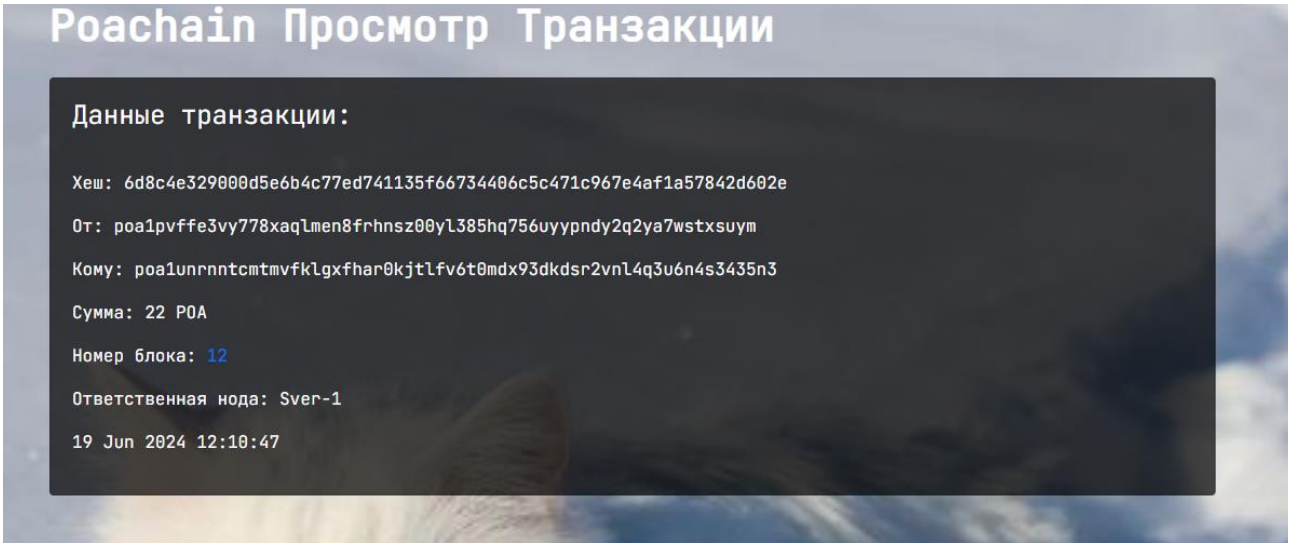

Рисунок Б.9 - Страница просмотра транзакции. В адресной строке видно, как она формируется

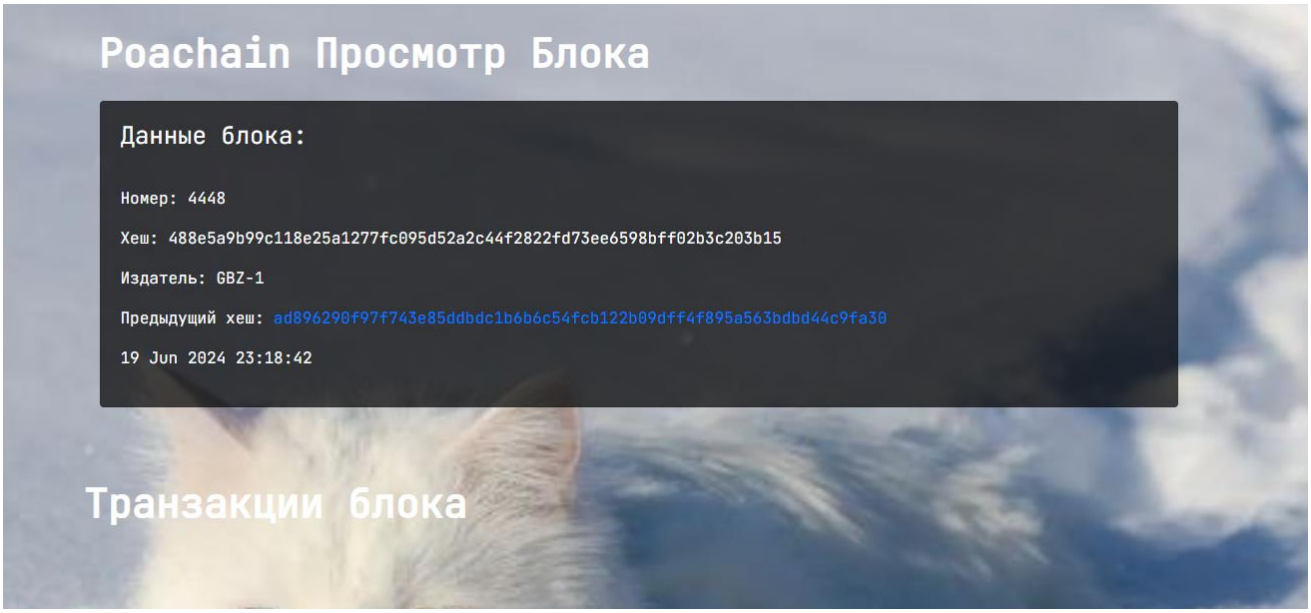

Рисунок Б.10 - Страница просмотра блока без транщакций

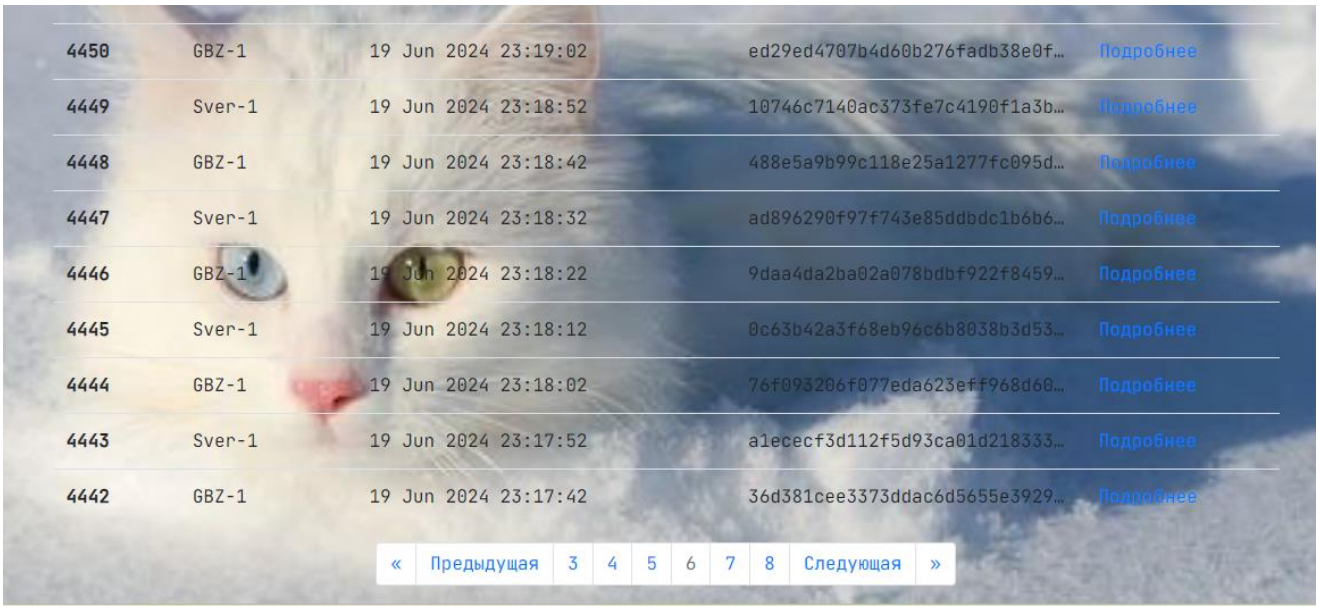

Рисунок Б.11 – Обзорщик блоков с пагинацией

Приложение В. Список файлов на внешнем носителе

1. СамедовНЮ_ВКР_ПояснительнаяЗаписка.docx – файл пояснительной записки к выпускной квалификационной работе.

2. СамедовНЮ_ВКР_ПрограммныйПродукт/ – папка с файлами программного продукта.

3. СамедовНЮ_ВКР_Презентация.pptx – файл презентации выпускной квалификационной работы.

4. СамедовНЮ_ВКР_Доклад.txt – файл доклада к защите выпускной квалификационной работы.