МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Ведущий программист
ООО «ВОРТЕКСКОД»

_____ П.А. Михайлов

“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор

_____ Л.Б. Соколинский

“___”_____ 2020 г.

Разработка приложения для проверки подлинности дипломов с использованием технологии блокчейн

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.308-194.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ К.Ю. Никольская

Автор работы,
студент группы КЭ-403
_____ Д.А. Снегирева

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Снегиревой Дарьи Алексеевны,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)
Разработка приложения для проверки подлинности дипломов с использованием технологии блокчейн.
2. **Срок сдачи студентом законченной работы:** 06.06.2020.
3. **Исходные данные к работе**
3.1. Siraj Raval. Decentralized Applications Harnessing Bitcoin's Blockchain TechnologyAutomatic// O'Reilly Media, Inc. 2016.
4. **Перечень подлежащих разработке вопросов**
4.1. Провести обзор литературы и существующих аналогов, а также готовых решений для создания приложения для проверки подлинности дипломов с использованием технологии блокчейн.
4.2. Определить требования к веб-приложению и разработать варианты использования.
4.3. Определить требования к блокчейн приложению.
4.4. Разработать блокчейн приложение для хранения дипломов.
4.5. Разработать веб-приложение для проверки подлинности дипломов с использованием технологии блокчейн.
4.6. Провести тестирование блокчейн приложения.
4.7. Провести тестирование веб-приложения.
5. **Дата выдачи задания:** 03.02.2020.

Научный руководитель,
ст. преподаватель кафедры СП К.Ю. Никольская

Задание принял к исполнению Д.А. Снегирева

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5
ГЛОССАРИЙ 8
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9
1.1. Обзор научной литературы и аналогов 9
1.2. Обзор готовых решений для создания приложения для проверки подлинности дипломов 12
2. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕХНОЛОГИИ БЛОКЧЕЙН 19
2.1. Понятие, структура и механизм работы технологии блокчейн 19
2.2. Типы блокчейна 23
2.3. Понятие консенсуса 24
2.4. Алгоритмы консенсуса 25
3. ТРЕБОВАНИЯ К СИСТЕМЕ 30
3.1. Функциональные требования к системе 30
3.2. Нефункциональные требования к системе 31
3.3. Варианты использования системы 31
4. АРХИТЕКТУРА СИСТЕМЫ 34
4.1. Компоненты блокчейн приложения 35
4.1.1. Выбор платформы для реализации блокчейн приложения 35
4.1.2. Tendermint ABCI 36
4.2. Структура диплома 38
4.3. Схема базы данных 40
5. РЕАЛИЗАЦИЯ 43
5.1. Реализация блокчейн приложения 43
5.2. Реализация компонентов для создания, подписания, отзыва и проверки дипломов 50
5.3. Реализации веб-приложения 54
5.3.1. Реализация структуры базы данных 54
5.3.2. Реализация внутренней логики 55
5.4. Реализация внешнего интерфейса веб-приложения 57

6. ТЕСТИРОВАНИЕ 60
6.1. Тестирование веб-приложения 60
6.2. Тестирование блокчейн приложения 62
ЗАКЛЮЧЕНИЕ 66
ЛИТЕРАТУРА 67
ПРИЛОЖЕНИЯ 72
ПРИЛОЖЕНИЕ А – Код в файле state.go 72
ПРИЛОЖЕНИЕ Б – Код метода Query 74
ПРИЛОЖЕНИЕ В – Код метода newTendermint 75
ПРИЛОЖЕНИЕ Г – Код классов Profile и Template 77
ПРИЛОЖЕНИЕ Д – Диплом в JSON 79
ПРИЛОЖЕНИЕ Е – Код компонентов веб-приложения 81
ПРИЛОЖЕНИЕ Ж – Код функции index 85
ПРИЛОЖЕНИЕ И – Страница с проверенным дипломом 86
ПРИЛОЖЕНИЕ К– Код в файле genesis.json 87

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ТЕМЫ ИССЛЕДОВАНИЯ

На данный момент идет активное внедрение современных информационно-телекоммуникационных технологий во всех странах мира. Например, правительством Российской Федерации была сформирована национальная программа «Цифровая экономика Российской Федерации» в рамках которой определены основные направления развития цифрового общества [7]. Создание, хранение и использование электронных дубликатов (электронных образов) бумажных документов является одной из основных целей, определяемых программой развития национального цифрового пространства.

В итоге, если рассматривать систему образования, то становится ясной одна из важнейших задач – оцифровка документооборота образовательного учреждения. Очевидно, что в ближайшее время документы об образовании (аттестаты, дипломы, сертификаты, удостоверения и т.п.) трансформируются в электронный формат. В связи с этим встает вопрос о хранении электронных документов [8].

В случае бумажных документов об образовании выявлено множество инцидентов, связанных с их подделкой. Человек не окончивший вуз имеет возможность получить документ с ложной информацией об окончании им какого-либо высшего учебного заведения. Тем самым некомпетентный человек может устроиться на рабочее место, обманув своего работодателя. Несмотря на перевод бумажных документов в электронный вид данная проблема остается актуальной.

Исходя из вышесказанного можно сделать вывод, что хранилище документов об образовании должно удовлетворять некоторым требованиям. Во-первых, цифровой реестр должен обеспечить безопасное хранение электронных документов, исключить или снизить до минимума риск их подделки. Во-вторых, иметь механизм идентификации обеспечивающий доступ пользователей к документам, например, будущий работник должен иметь

возможность предоставить нанимателю, в случае устройства на рабочее место, свой подлинный диплом об образовании.

Впервые технология блокчейн была предложена в 2008 году [9]. Блокчейн можно рассматривать как публичный реестр, где вся информация хранится в списке блоков. Цепочка блоков растет по мере добавления в нее новых блоков. Для обеспечения безопасности пользователей и согласованности были использованы такие решения, как асимметричная криптография и распределенные алгоритмы консенсуса. Технология блокчейн, как правило, имеет такие ключевые характеристики как децентрализация, устойчивость, анонимность и проверяемость [3]. Благодаря этим характеристикам, технология блокчейн может быть использована для реализации системы хранения документов об образовании.

Однако, если рассматривать Российскую Федерацию, блокчейн имеет ряд проблем, которые влияют на распространение данной технологии в сфере образования, их можно разделить на три основные группы [10]:

1) технико-технологические проблемы (национальная платформа блокчейн пока что отсутствует);

2) правовые проблемы (цифровые образовательные документы в блокчейн не имеют юридической силы);

3) организационные проблемы (отсутствие централизованной управляющей структуры, координирующей «образовательный» блокчейн как единое цифровое информационное пространство).

ЦЕЛИ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка веб-приложения для проверки подлинности дипломов с использованием технологии блокчейн.

Для достижения данной цели необходимо решить следующие задачи, перечисленные ниже.

1. Провести обзор литературы и существующих аналогов, а также готовых решений для создания приложения для проверки подлинности дипломов с использованием технологии блокчейн.

2. Определить требования к веб-приложению и разработать варианты использования.

3. Определить требования к блокчейн приложению.

4. Разработать блокчейн приложение для хранения дипломов.

5. Разработать веб-приложение для проверки подлинности дипломов с использованием технологии блокчейн.

6. Провести тестирование блокчейн приложения.

7. Провести тестирование веб-приложения.

СОДЕРЖАНИЕ И СТРУКТУРА РАБОТЫ

Работа состоит из введения, шести глав и заключения. Объем работы составляет 88 страниц, объем библиографии – 49 источников.

В первой главе проводится анализ предметной области, приведен обзор существующей литературы, относящейся к теме работы, примеры аналогов, представлен обзор готовых решений для создания системы проверки подлинности дипломов.

Во второй главе приведена теоретическая часть по теме работы.

В третьей главе описаны требования к системе, а также варианты использования системы.

В четвертой главе представлена архитектура системы.

В пятой главе представлены результаты реализации системы и описаны используемые технологии.

В шестой главе описан процесс тестирования приложения для проверки дипломов и приведен протокол тестирования основных аспектов его работы.

В заключении приводятся основные результаты работы и рассматриваются дальнейшие пути развития приложения.

ГЛОССАРИЙ

1. *Алгоритм SHA-256* – представляет собой алгоритм, который принимает в качестве входных данных любые произвольные данные и создает серию уникальных цифр и букв [1].

2. *Алгоритм консенсуса* – это процесс получения согласованного результата группой участников [2].

3. *Блокчейн (цепочка блоков)* – это цифровой реестр, где все совершенные транзакции хранятся в списке блоков. Цепочка блоков растет по мере добавления новых блоков [3].

4. *Верификация* – процесс сравнения двух уровней спецификации средств вычислительной техники или автоматизированных систем на надлежащее соответствие [4].

5. *Идентификатор* – уникальный признак субъекта или объекта доступа [4].

6. *Идентификация* – присвоение субъектам и объектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов [4].

7. *Транзакция* – это атомарная операция над базой данных, то есть она либо завершается, либо не происходит вообще, но не может быть оставлена в промежуточном состоянии [5].

8. *Частный блокчейн* – это цепочка блоков, в которой, не каждый узел может участвовать в блокчейн, имеет строгое управление полномочиями по доступу к данным [6].

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор научной литературы и аналогов

По теме данной работы, ввиду своей актуальности, ведется большое количество разработок. Например, Джей Гупта (Jay Gupta) и Сваправа Натх (Swaprava Nath) из Индийского технологического института в Канпуре представили в своей статье образовательную платформу SkillCheck на основе блокчейна [11].

Фанк Эрик (Funk Eric), Ридделл Джефф (Riddell Jeff), Анкель Феликс (Ankel Felix) и Кабрера Даниэль (Cabrera Daniel) в своей статье рассмотрели применение технологии блокчейн в сфере профессионального медицинского образования, включая выдачу и хранение документов об образовании [12].

Лизкано Давид (Lizcano David), Лара Хуан А (Lara Juan A), Уайт Бебо (White Bebo) и Альджаварнех Шади (Aljawarneh Shadi) представили в своей работе применение технологии блокчейн в области высшего образования [13]. Основная идея состоит в регистрации и валидации знаний, приобретенных студентами, что облегчило бы процесс приема на рабочее место, так как все достижения претендента размещены в системе, которой можно доверять.

Абхишек Шривастава (Abhishek Srivastava), Проная Бхаттачарья (Pronaya Bhattacharya), Арундра Сингх (Arunendra Singh) и другие предложили глобальную доверенную образовательную систему, основанную на технологии блокчейн, среди различных заинтересованных сторон, таких как университеты, компании и другие высшие учебные заведения, которые соглашаются сотрудничать в рамках данной системы [14].

Джерина Греш (Jerinas Gresch), Бруно Родригес (Bruno Rodrigues), Эдер Шейд (Eder Scheid), Салил С. Канхере (Salil S. Kanhere) и Буркхард Стиллер (Burkhard Stiller) предложили в своей статье систему на основе технологии блокчейн для управления дипломами под названием UZHBC (University of ZuricH BlockChain) [15].

Система проверки дипломов Университета Никосии [16]

Еще в 2014 году руководство Университета Никосии (Кипр) поставило перед собой задачу решить проблему цифровых учетных данных с помощью технологии блокчейн. Исследователи университета придумали новый способ снять отпечатки с цифровых документов и закрепить их в блокчейне Bitcoin. С годами это решение было усовершенствовано до платформы Block.co.

Данное решение использует блокчейн Bitcoin [17], обосновывая это стабильностью и надежностью платформы. Также им удалось создать протокол поверх Bitcoin, который обеспечивает всю необходимую функциональность при выполнении всех действий в цепочке, чтобы решить проблему ограниченности и сложности работы в Bitcoin.

Документы публикуются в формате PDF (рисунок 1.1), так как большинству пользователей удобно их просматривать, а также большинство учреждений, которые выдают цифровые учетные данные предпочитают формат PDF. Для того, чтобы проверить документ, его прикрепляют к форме проверки, через несколько секунд система отображает результат (рисунок 1.2).

UNIVERSITY of NICOSIA

CERTIFICATE OF ACCOMPLISHMENT

This is to certify that

Alexis Nicolaou

has successfully completed the University of Nicosia MOOC

INTRODUCTION TO DIGITAL CURRENCIES

The MOOC provides an introductory understanding of decentralized digital currencies, such as Bitcoin, and blockchains (distributed ledger technology). The course is structured around four general topics: theoretical introduction to digital currencies (history, decentralized consensus, technical overview of Bitcoin, alternative uses of the blockchain), practical introduction to digital currencies (transactions, wallets, mining), financial and regulatory implications of digital currencies, innovation & future development.

Evaluation Score:

The student has completed at least 9 of 12 quizzes and has passed Examination with a grade of :

79.5/100

Professor Andreas Polemitis
Senior Vice-Rector

Date: 5 Dec 2017

This is a unique Digital Certificate that is timestamped into the Bitcoin blockchain. Detailed instructions of the process followed, and how to verify it can be found at https://digitalcurrency.unic.ac.cy/free-introductory-mooc/self-verifiable-certificates-on-the-bitcoin-blockchain/ or as otherwise distributed by the University of Nicosia. The OP_RETURN output of the Bitcoin transaction originating from address 1A96tDxxJtjPvoBCjCWe6GLUfT6BGTWaUq, which this certificate is part of, will include string "UNicDC". The University of Nicosia, has authenticated this individual's participation in the course and everyone presented with this digital certificate should be able to verify its authenticity on their own, without the need to trust the person that presented it, just by going to the link above, and doing a self-contained validation.

Рис. 1.1. Пример сертификата Университета Никосии

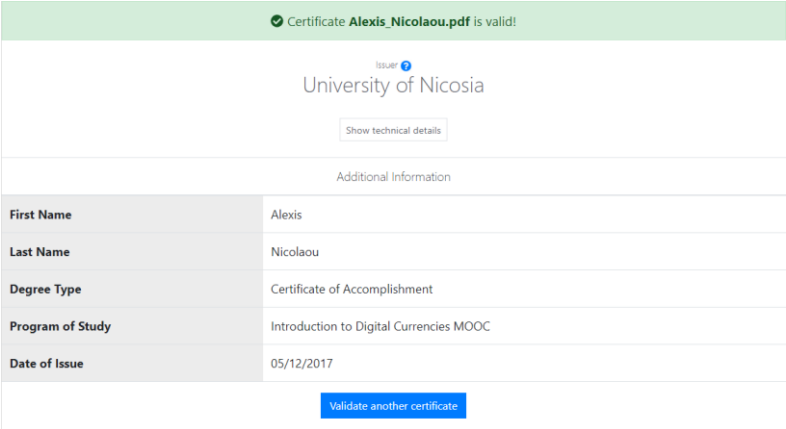

Рис. 1.2. Результат проверки подлинного сертификата

Проверка дипломов Финансового Университета в блокчейне [18]

Финансовый Университет внедрил в 2018 году в свой официальный сайт проверку подлинности дипломов в блокчейне (рисунок 1.3). Каждая отдельная запись подписывается университетом. Это защита от того, что записи будут созданы третьими лицами. Подобные подделки не будут иметь верификацию, а значит не будут признаваться действительными. Данные шифруются по алгоритму SHA-256 [1] и в результате для каждого диплома получается уникальный идентификатор – хэш диплома.

Минусом является то, что в форму ввода нельзя вводить хэш-функцию диплома. Система не использует один из главных плюсов хранения диплома в блокчейн – возможность хранения студентом одной небольшой хэш-функции, вместо полной версии документа.

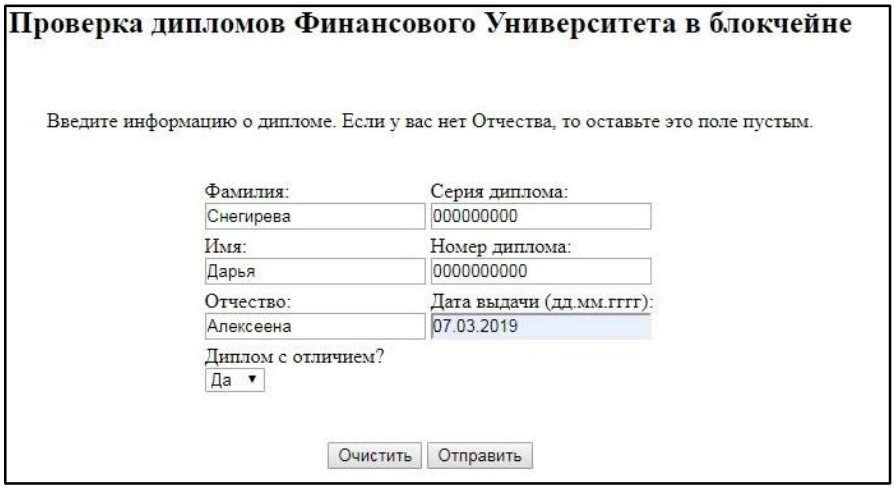

Рис. 1.3. Интерфейс диалогового окна

MIT Degree Verification [19]

MIT Degree Verification – сайт проверки сертификатов, где можно любому частному лицу провести независимую проверку диплома выданного в MIT (рисунок 1.4). Работодатели для проверки диплома могут вставить ссылку или загрузить файл с цифровым дипломом студента и затем сразу получить ответ от портала. Система использует блокчейн в качестве нотариуса, находит идентификатор транзакции в документе, проверяет ключи и подтверждает, что ничего не было изменено.

Данное решение построено с использованием платформы Blockcerts [20].

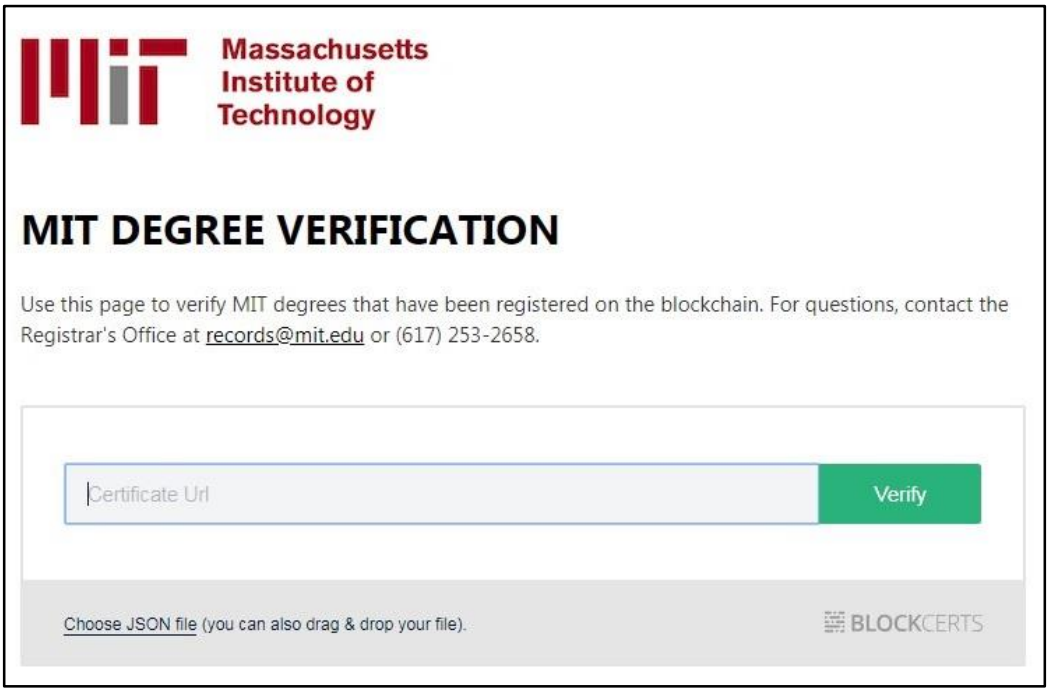

Рис. 1.4. Интерфейс портала MIT Degree Verification

1.2. Обзор готовых решений для создания приложения для проверки подлинности дипломов

Bitcoin Core [17]

В 2009 году анонимный разработчик программного обеспечения, известный как Сатоши Накамото, представил технологию на основе цепочки блоков в протоколе Bitcoin для решения проблемы наличия посредника в виде финансовых институтов в платежных системах [9].

Решение, получившее название блокчейн, содержит цифровую валюту в виде последовательности цифровых (транзакции). Они упорядочиваются в партии (блоки), которые хэшируются и получают метку времени, притом, каждый последующий хэш содержит метку предыдущего блока, т.е. информацию о предыдущем блоке, тем самым выстраивается цепочка блоков, в которой сложность взлома растет с количеством новых блоков в цепи, т.к. замена одного блока требует также замены последующих.

В протоколе Bitcoin используется алгоритм консенсуса Proof of Work [9] основанный на вычислительных мощностях компьютеров.

Bitcoin Core – это бесплатное программное обеспечение с открытым исходным кодом, которое служит узлом Bitcoin и предоставляет кошелек, который полностью проверяет платежи. Он считается эталонной реализацией протокола Bitcoin.

Tendermint Core [21]

Tendermint Core – это программное обеспечение для безопасной и последовательной репликации блокчейн приложений, написанных пользователями, на множество компьютеров, состоящее из двух основных компонентов: механизма блокчейн, включающего алгоритм консенсуса и реализацию сети, и интерфейса ABCI (Application BlockChain Interface) (рисунок 1.5).

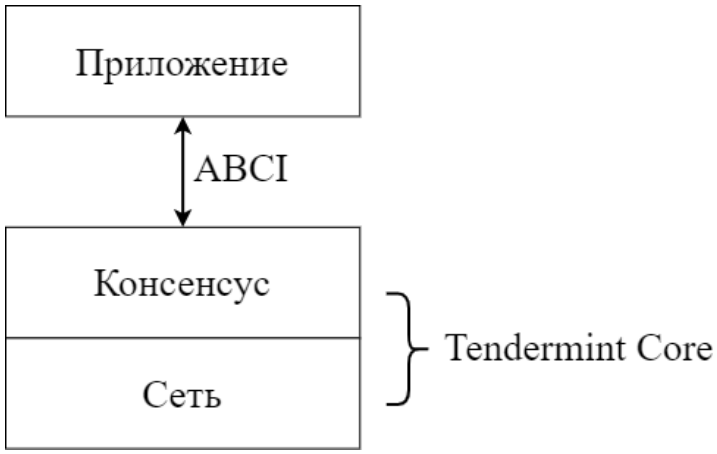

Рис. 1.5. Схема Tendermint Core

Подход Tendermint состоит в том, чтобы отделить механизм блок-

чейна от других компонентов, например, peer-to-peer (далее – P2P), путем абстрагирования деталей блокчейн приложения от интерфейса, который реализован в виде протокола сокетов. Таким образом, в программном обеспечении есть интерфейс Application BlockChain Interface (ABCI) и его основная реализация - протокол сокетов Tendermint (TSP или Teaspoon).

ABCI – это интерфейс между Tendermint (механизмом репликации конечного автомата) и приложением (фактическим конечным автоматом), написанным пользователем. Благодаря использованию Protocol Buffers [22] приложения можно писать на любом языке программирования. Тогда как ядро Tendermint Core написано на языке программирования Golang [23].

В Tendermint в качестве модели распределённых вычислений взят конечный автомат репликации. Конечный автомат – это широко используемая вычислительная модель, которая состоит из бесконечного или ограниченного набора состояний и входов, которые возбуждают переход из одного состояния в другое. Детерминированной конечный автомат должен находится в определенном состоянии в определенное время, т.е. последовательность состояний единственна. Репликация конечного автомата означает, что конечный автоматы, находящиеся в разных узлах, согласованны и являются копиями друг друга. Независимо от того, на какой узел пользователь отправляет транзакцию, все узлы в конечном итоге перейдут в новое согласованное состояние. Только детерминированный конечный автомат может быть использован как модель репликации конечного автомата для достижения распределенного консенсуса.

Когда узел получает запрос на добавление транзакции, ему сначала необходимо согласовать это с другими узлами, прийти к консенсусу и затем каждый узел должен обновить свое состояние.

При разработке типичного децентрализованного приложения, в дополнение к необходимости разработки сетевых приложений (таких как приложения ABCI), часто необходимо разрабатывать традиционные веб-прило-

жения, настольные или мобильные приложения, чтобы пользователю не требовалось разворачивать полноценный узел блокчейн. Для доступа к функциям узла из приложений, таким как отправка транзакций или запрос статуса узла, Tendermint Core предоставляет интерфейс RPC.

Алгоритм консенсуса Tendermint является формой алгоритма консенсуса, основанного на задаче о византийских генералах (BFT), представляющую собой BFT-ABC. Его особенностью является то, что в процессе принятия решения о включении блока в цепочку принимается группой специально выбранных валидаторов. Это значит, что участники консенсуса могут быть идентифицированы. Таким образом, алгоритм подходит для приватных и консорциумных блокчейнов.

Алгоритм консенсуса будет работать, если не более 1/3, или чуть меньше 1/3 валидаторов неисправны, т.е. все остальные должны быть онлайн и доступны.

Итан Бухман (Ethan Buchman) в своей работе [24] провел тесты программного обеспечения Tendermint, по итогам которых выяснилось, что на 64 узлах, распределенных по всему миру, легко достигается скорость в тысячу транзакций в секунду, что намного быстрее чем, например, Bitcoin со скоростью около 7 транзакций в секунду.

Exonum [25]

Exonum – это фреймворк для создания блокчейн-проектов. Ядро написано на языке системного программирования Rust [26]. Расширение Java Binding обеспечивает интерфейсы для разработки приложений на Java [27].

Exonum использует собственную модификацию византийского консенсуса (похожего на PBFT «Реализация протокола задачи византийских генералов» [28]), чтобы гарантировать, что в любой момент существует только одна согласованная версия блокчейна.

Алгоритм консенсуса Exonum также как и Tendermint [29] предполагает, что участники консенсуса могут быть идентифицированы. Таким образом, алгоритм подходит для приватных блокчейнов.

По сравнению с другими вариантами алгоритмами византийского консенсуса, алгоритм консенсуса Exonum имеет следующие отличительные особенности, перечисленные ниже.

1. Неограниченные раунды. Раунды имеют фиксированное время начала, но у них нет определенного времени окончания (раунд заканчивается только после принятия следующего блока). Это помогает уменьшить задержки при нестабильном сетевом соединении между валидаторами.

2. Разделение работы. Отложенная обработка транзакций снижает негативное влияние вредоносных узлов на пропускную способность и задержки в системе.

3. Алгоритм запросов. Алгоритм запросов позволяет валидатору восстановить любую консенсусную информацию с помощью других валидаторов. Это оказывает положительное влияние на жизнеспособность системы.

Алгоритм консенсуса должен работать при наличии сбоев, т.е. в условиях, когда участники сети могут вести себя аномально. Алгоритм консенсуса Exonum исходит из худших предположений: любой отдельный узел, или даже группа узлов в сети блокчейн, могут дать сбой или быть скомпрометированы. Алгоритм консенсуса справляется с ситуациями, когда до 1/3 валидаторов действуют злонамеренно, отключены или изолированы от сети, при этом остальные валидаторы должны быть доступны.

Exonum использует SHA-256 [1] для всех операций хеширования.

В глобальной распределенной сети на базе блокчейна Exonum можно обрабатывать в среднем 5 000 транзакций в секунду. В локальной сети с небольшим количеством узлов скорость обработки транзакций возрастает в два, три раза, максимальное количество транзакций до 15 000 в секунду.

Blockcerts [20]

Blockerts – это компоненты для создания, выдачи и проверки сертификатов блокчейне Bitcoin [17] или Ethereum [30].

Blockcerts состоит из библиотек с открытым исходным кодом, инструментов и мобильных приложений.

Выдача сертификата, представленная на рисунке 1.6 относительно проста: создается цифровой файл, который содержит основную информацию, такую как имя получателя, имя издателя (название университета), дата выпуска и т.д. Затем подписывается содержимое сертификата с использованием закрытого ключа, к которому имеет доступ только издатель.

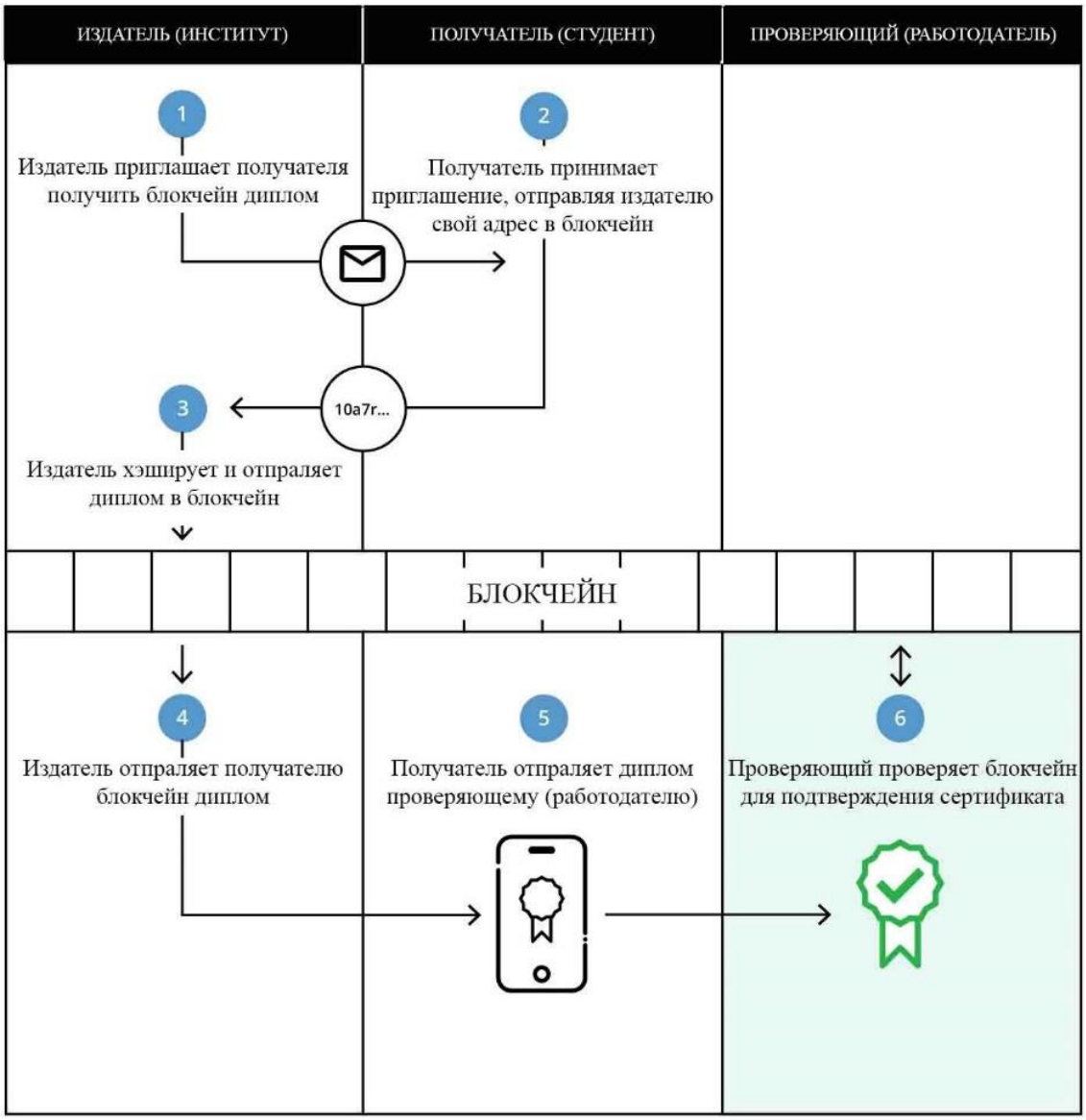

Рис. 1.6. Схема работы Blockcerts

Далее создается хеш-функция, которую можно использовать для проверки того, что сертификат не был изменен. и, наконец, снова используется закрытый ключ для создания записи в блокчейн, в которой

говорится, что данный сертификат выдан конкретному человеку в конкретный день. Система позволяет проверять, кому был выдан сертификат и его содержание.

Blockerts уже используется в нескольких проектах. Например, такие университеты как MIT [31] и Мельбурнский Университет [32] используют данное решение для выдачи студентам своих сертификатов. Также было подписано соглашение министерства образования и занятости Мальты с блокчейн-стартапом Learning Machine Technologies о разработке экспериментальной платформы, на основе Blockcerts [33].

Выводы по первому разделу

Анализ источников по теме работы показывает, что в данный момент ведутся активные исследования в области выдачи документов об образовании с использованием технологии блокчейн. Множество научно-исследовательских работ посвящены применению технологии блокчейн для безопасного хранения дипломов или сертификатов. Исходя из этого, можно заключить, что тема данной работы является актуальной.

Многие существующие решения поддерживают только блокчейн Bitcoin [17] или Ethereum [30], что ограничивает их в плане реализации и скорости. Можно сделать вывод, что разработка на основе блокчейна с более быстрой обработкой транзакций и с большими возможностями реализации является актуальной задачей.

2. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕХНОЛОГИИ БЛОКЧЕЙН

2.1. Понятие, структура и механизм работы технологии блокчейн

Понятие и структура

Блокчейн представляет собой последовательность блоков. На рисунке 2.1 показан пример общей структуры блокчейна. Каждый блок указывает на непосредственно предыдущий блок через ссылку, которая, по сути, является хэш-значением предыдущего блока, называемого родительским блоком. Первый блок цепочки называется блоком генезиса [34].

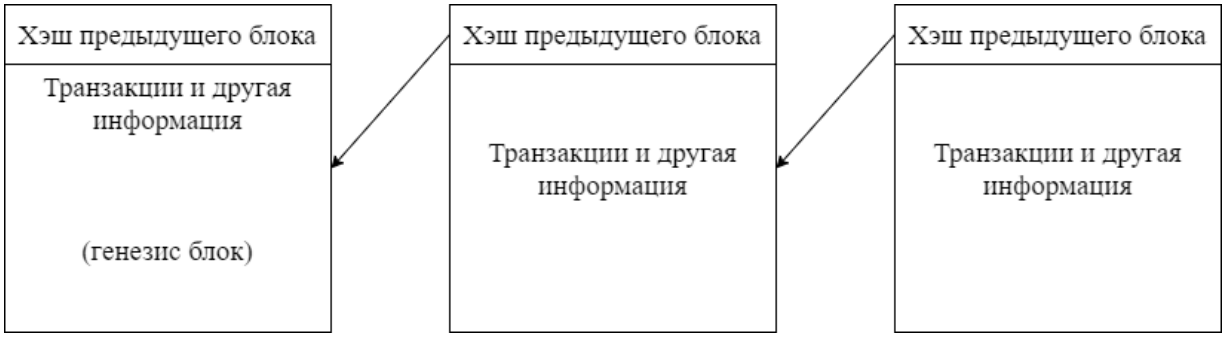

Рис. 2.1. Общая структура блокчейна

Транзакция может представлять собой различные действия в блокчейне, чаще всего это структура данных, содержащая логику перемещения валюты, адреса отправителя и получателя и другое [34].

Неподтвержденные транзакции хранятся в мемпул (mempool) [34].

Адреса – это уникальные идентификаторы, используемые в блокчейн-транзакции для обозначения отправителей и получателей. Адрес обычно является открытым ключом или производным от него [34].

Для обеспечения безопасности экосистемы блокчейна используется множество различных криптографических примитивов, таких как хэш-функция, симметричная криптография, ассиметричная криптография и цифровые подписи [34].

Хэш функции используют для создания битовой строки фиксированной длины из входных данных произвольной длины. Они не имеют ключей, и их функция состоит в том, чтобы обеспечить целостность данных [34].

Симметричная криптография относится к типу криптографии, где ключ, который используется для шифрования данных, также используется для расшифровки данных [34].

Асимметричная криптография относится к типу криптографии, где ключ, который используется для шифрования данных, отличается от ключа, который используется для дешифрования данных. В данном случае используется как открытый, так и закрытый ключ [34].

Закрытый ключ – это случайным образом сгенерированное число, которое хранится в секрете от других [34].

Открытый ключ – это ключ, который общедоступен, он публикуется владельцем закрытого ключа [34].

Любой, кто хочет отправить зашифрованное сообщение данному пользователю, может зашифровать сообщение с помощью открытого ключа и быть уверенным, что только владелец закрытого ключа его расшифрует [34].

Блок можно представить, как некий контейнер, который объединяет транзакции для включения в блокчейн. Он обычно состоит из заголовка, содержащего метаданные и тела со списком транзакций [35].

В различных блокчейнах набор метаданных может различаться.

Например, в блокчейне Bitcoin заголовок блока состоит из трех наборов метаданных:

1) ссылка на предыдущий хеш блока, которая соединяет нынешний блок с предыдущим;

2) временная метка и случайное число (nonce или соль);

3) корень дерева Меркла – структуры данных, используемой для суммирования всех транзакций в блок с целью уменьшения места занимаемого транзакциями [35].

Дерево Меркла или двоичное хеш-дерево, представляет собой структуру данных, используемую для эффективного обобщения и проверки целостности больших наборов данных. Термин «дерево» используется в ин-

форматике для описания структуры ветвящихся данных, но эти деревья обычно отображаются вверх дном с «корнем» вверху и «листьями» внизу диаграммы (рисунок 2.2) [35].

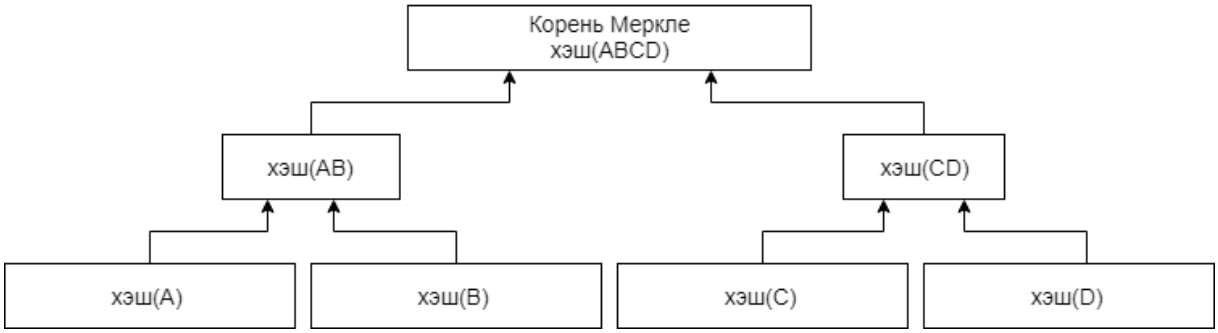

Рис. 2.2. Дерево Меркла

Деревья Меркла используются в блокчейне для суммирования всех транзакций в блоке, создавая общий цифровой отпечаток всего набора транзакций, обеспечивая очень эффективный процесс проверки того, включена ли транзакция в блок. Дерево Меркла создается рекурсивно хеширующими парами узлов, пока не будет только один хеш, называемый корнем или корнем Меркла [35].

Peer-to-peer (далее P2P) – это топология сети, в которой компьютеры, участвующие в сети, являются одноранговыми друг с другом, это означает, что все они равны, нет «специальных» узлов и все они совместно предоставляют сетевые услуги. Узлы в одноранговой сети взаимно предоставляют и потребляют услуги, что выступает стимулом для участия в такой сети [35].

Узел в сети блокчейна выполняет различные функции в зависимости от роли, которую он берет на себя. Он может предлагать и проверять транзакции и выполнять работу по обеспечению консенсуса и безопасности блокчейна. Консенсус достигается за счет применения алгоритма консенсуса (подробнее в пункте 2.3). Узлы также могут выполнять другие функции, такие как простая проверка платежей (облегченные узлы), проверка подлинности, подписание транзакции и многое другое в зависимости от используемого блокчейна и роли, назначенной узлу [34].

Скрипты или программы выполняют различные операции с транзакциями. Например, в Bitcoin сценарии транзакций предопределены на языке под названием Script, который состоит из наборов команд, позволяющих узлам передавать криптовалюту с одного адреса на другой. Script является ограниченным языком, он допускает только существенные операции, необходимые для выполнения транзакций, но не допускает разработки собственных функций [34].

Таким образом, язык Script нельзя назвать полным по Тьюренгу. Говоря простыми словами, если язык программирования полный по Тьюренгу, то он может выполнять любые вычисления. Например, язык Solidity Ethereum является полным по Тьюренгу [34].

Для запуска полного по Тьюренгу кода, например, смарт-контракты, используется виртуальная машина, при этом сценарий транзакций все еще остается ограниченным. Различные блокчейны используют виртуальные машины, например, в блокчейне Ethereum используется Ethereum Virtual Machine (EVM) [34].

Смарт контракты – это программы, которые работают поверх блокчейна и инкапсулируют бизнес-логику, которая заложена в ней [34].

Блокчейн также можно рассматривать как конечный атомат, его состояния изменяются от начального состояния до следующего и затем до конечного в результате выполнения, проверки и включения транзакции [34].

В итоге, блокчейн – это одноранговая распределенная бухгалтерская книга, которая криптографически защищена, доступна только для добавления, неизменяема и может обновляться только на основе консенсуса или соглашения между одноранговыми узлами [34].

Термин распределенная бухгалтерская книга означает, что каждый узел в сети содержит копию блоков блокчейна [34].

Понимание данной технологии крайне важно для понимания технологии блокчейн, так как она лежит в ее основе.

Механизм работы

Рассмотрим общую схему создания блоков.

1. Узел запускает транзакцию, сначала создавая, а затем подписывая ее своим закрытым ключом.

2. Транзакция распространяется по сети. Другие узлы проверяют полученную транзакцию по правилам, заданным в блокчейне.

3. Если транзакция правильная, она включается в блок, который затем распространяется по сети. В этот момент транзакция становится подтвержденной.

4. Новый блок становится частью блокчейна, следующий после него блок свяжет себя с ним, включив его хэш с свой заголовок.

5. Затем транзакция повторно подтверждается каждый раз при создании нового блока и после определенного числа подтверждений, например, в блокчейне Bitcon их 6, она будет считаться оконченной [34].

2.2. Типы блокчейна

Блокчейн делится на три типа, которые представленны ниже.

1. Публичный блокчейн – это блокчейн, к которому может получить доступ любой человек в мире, то есть он может отправлять транзакции и ожидать их включения, если они действительны, а также участвовать в консенсусе – процессе определения того, какие блоки добавляются в цепочку, а также текущего состояния блокчейна.

В качестве замены централизованного управления, используемого в обычных экономических системах, публичный блокчейн децентрализован, за счет использования принципов криптоэкономики, которые основаны на сочетании экономических стимулов и криптографической проверки с использованием таких механизмов, как доказательство работы (Proof of Work) или доказательство доли владения (Proof of Stake), следуя общему принципу, согласно которому степень, в которой кто-либо может повлиять на процесс консенсуса, пропорциональна количеству экономических ресурсов,

которые он может привнести в процесс консенсуса.

Преимущество публичного блокчейна в том, что каждая транзакция является общедоступной.

Недостатки заключаются в достаточно медленной работе и ощутимых затратах электроэнергии и памяти, т. к. нужна синхронизация со всей цепочкой блоков начиная с первого блока.

2. Блокчейн, принадлежащий консорциуму – это блокчейн, в котором процесс достижения консенсуса контролируется предварительно выбранным набором узлов. Виталий Бутерин в качестве примера взял консорциум, состоящий из пятнадцати финансовых учреждений, каждое из которых управляет узлом. Для того, чтобы блок был признан действительным и его добавили в блокчейн, десять из них должны его подтвердить.

Данный блокчейн может считаться «частично децентрализованным».

Консорциумный блокчейн обладает такими же преимуществами, как и частный блокчейн, но работает под руководством группы, а не единой сущности.

3. Частный блокчейн – это блокчейн, в котором одна организация имеет право на запись новых блоков. Право на чтение данных может быть у всех, либо ограничено в той или иной степени [36].

Преимущества в том, что транзакции в частной цепочке блоков выполняются значительно быстрее чем в публичном блокчейне.

Недостаток частного блокчейна в том, что он не предлагает такую же децентрализованную безопасность, как его публичный аналог.

2.3. Понятие консенсуса

Алгоритм консенсуса – это процесс получения согласованного результата группой участников [2]. Достижение консенсуса между ненадежными узлами является проблемой византийских генералов (англ. Byzantine fault tolerance (BFT)) для распределенных систем, которая была описана в работе Лесли Лэмпорт (Leslie Lamport), Роберта Шостак (Robert Shostak), и

Маршалла Пиз (Marshall Pease) [37]. Устройства компьютерной системы абстрактно представляют византийской армией, с несколькими дивизиями, которые стоят лагерями за пределами вражеского города, и каждой дивизией командует свой генерал. Генералы могут общаться друг с другом только через посыльных. Понаблюдав за противником, они должны выработать общий план действий. Однако некоторые генералы могут оказаться предателями, пытаясь помешать лояльным генералам прийти к соглашению. Генералы должны иметь алгоритм, гарантирующий, что все лояльные генералы принимают решение об одном и том же плане действий и малое число предателей не может заставить верных генералов принять плохой план.

Так как данная проблема затрагивает распределенные системы, она является актуальной для технологии блокчейн, которая основана на распределенных вычислениях.

2.4. Алгоритмы консенсуса

Существует несколько типов алгоритмов консенсуса. У каждого есть свои преимущества и недостатки, при попытке сбалансирования безопасности, с функциональностью и масштабируемостью. Каждый их них может использован в блокчейнах разного типа.

Далее приведены несколько из них.

Proof of Work (PoW)

Алгоритм консенсуса Proof of Work или «доказательство выполнения работы» используется в протоколах Bitcoin [9] и Ethereum [38]. Данный алгоритм является самым первым алгоритмом достижения консенсуса, который начал использоваться в блокчейне. Данный алгоритм используется для криптовалюты, т.е. создать блокчейн приложение без использования криптовалют, используя данный алгоритм невозможно.

В PoW каждый узел сети вычисляет хэш-значение постоянно меняющегося заголовка блока. Консенсус требует, чтобы вычисленное значение было равно или меньше определенного заданного значения [39]. В

децентрализованной сети все участники должны вычислять хэш-значение непрерывно, перебирая значения итерируемого поля-добавки (nonce) в блоке данных до тех пор, пока цель не будет достигнута. Когда один узел получает соответствующее значение, все остальные узлы должны взаимно подтвердить правильность найденного значения. Только после этого блок помещается в блокчейн. Содержимое включенного блока нельзя изменить, не выполнив заново всех вычислений, включая перевычисление всех блоков следующих за ним [9].

Узлы, которые вычисляют хэши, называются майнерами, а процедура поиска значения хэша – майнингом. Поскольку вычисления тратят время, ресурсы компьютера и электроэнергию, введен механизм стимулирования, например, предоставление майнеру небольшой части криптовалюты, которая берется из транзакций в виде комиссий.

Алгоритм PoW используется в публичном блокчейне.

При использовании PoW существует риск атаки 51%.

Proof of Stake (PoS)

Алгоритм Proof of Stake или «доказательство доли владения» является энергосберегающей альтернативой алгоритма PoW.

Вместо нахождения «nonce» на неограниченном пространстве, алгоритм требует, чтобы пользователи доказали право собственности на количество валюты, потому что считается, что люди с большим количеством валюты были бы менее склонны к атакам на сеть. Так как выбор на основе баланса учетной записи довольно несправедлив, потому что самый богатый человек будет доминировать в сети. Вместо этого берутся другие способы отбора, которые могут отличаться в разных блокчейнах [39].

При использовании PoS снижается риск атаки 51%, так как случится обвал курса криптовалют, а злоумышленник, который будет держать более половины монет сам станет жертвой, т.е. данная атака является экономичес-

ки невыгодной.

Алгоритм используется и в публичном и в частном блокчейне.

Proof of Activity (PoA)

Алгоритм Proof of activity (PoA) или «доказательство активности» является объединением преимуществ алгоритмов PoS и PoW. Для того, чтобы блок был помещен в блокчейн, он должен быть подписан определяемым в каждом блоке количеством пользователей [40]. Таким образом, если существует владелец 50% всех монет, то он не может самостоятельно контролировать создание новых блоков [39].

Proof of Capacity (PoC)

В алгоритме Proof of Capaciy или «доказательство ресурсов» для добычи блока майнер должен выделить свободное место на своем жестком диске. От количества выделенного места напрямую зависит вычислительная мощность, т.е. скорость выполнения операций, связанных с вычислением блока [3].

Delegated Proof of Stake (DPoS)

В алгоритме Delegated Proof of Stake или «делегированное доказательство доли» подобно PoS майнеры получают приоритет для создания блоков в соответствии с их ставкой. Основное различие между PoS и DPoS заключается в том, что PoS – это прямая демократия, а DPoS – представительная демократия, т.е. право подтверждения блоков передается определенной группе людей, которые выбираются с помощью голосования. При значительно меньшем количестве узлов для валидации блока, блок может быть быстро подтвержден, что делает данные операции быстрыми. В то же время параметры сети, такие как размер блока и интервалы между блоками, могут быть настроены. Кроме того, пользователям не нужно беспокоиться о недобросовестных валидаторах, потому что голосования проводятся регулярно, что дает стимул валидаторам быть честными, чтобы их выбрали повторно. Алгоритм DPoS является основой блокчейна Bitshares [41].

Используется в публичном и в частном блокчейне.

Tendermint

Алгоритм консенсуса tendermint основан на задаче о византийских генералах (Byzantine Fault Tolerance) [37]. Особенностью данного алгоритма является то, что решения о включении блока в цепочку принимаются специальными участниками, которых называют валидаторами. Это создаёт некоторую централизацию, что может повлиять на доверие пользователей к такой системе [42].

Чтобы создать новый блок и проголосовать за него используется алгоритм из трех шагов, которые предсталены ниже.

1. Алгоритм консенсуса делится на раунды, начало которых определяется каждым валидатором на основе его собственного источника времени. Для каждого раунда существует предопределенный валидатор-лидер, который определяется на основе числа раундов и высоты цепочки. Лидер создает предложение блока (proposal) и отправляет его другим валидаторам. Другие валидаторы проверяют это предложение, и, если оно правильное, голосуют за него, передавая сообщения типа prevote другим валидаторам.

2. Если валидатор собирает сообщения типа prevote для одного и того же предложения от абсолютного большинства валидаторов, он выполняет транзакции в предложении, создает сообщение типа precommit с новым состоянием хранилища данных и отправляет его валидаторам.

3. Наконец, если валидатор получает сообщения типа precommit от абсолютного большинства валидаторов для одного и того же предложения, предложение становится новым блоком и фиксируется в локальном хранилище валидатора.

Так как на основе Tendermint можно создавать произвольные приложения, то можно либо определять валюту и право голоса валидатора в этой валюте, либо не определять. Когда право голоса выражено в национальной валюте, систему часто определяют, как PoS.

На рисунке 2.3 представлена схема алгоритма консенсуса Tendermint.

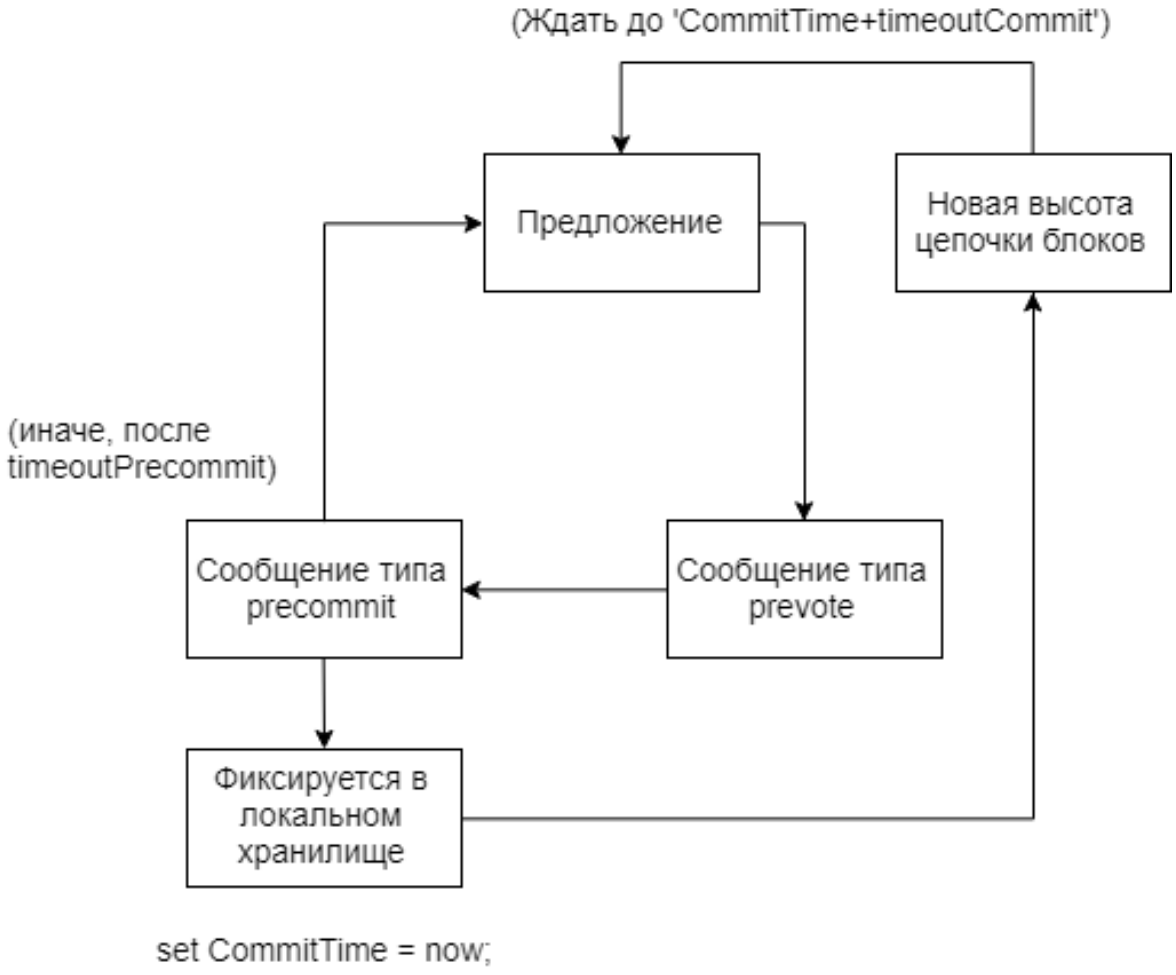

Рис. 2.3. Алгоритм консенсуса Tendermint

В соответствии с логикой приложения, валидаторы могут быть вынуждены «связать» свои валютные запасы в гарантийном депозите, который может быть уничтожен, если обнаружится, что валидатор злонамеренный.

Алгоритм используется в частном блокчейне и в блокчейне, принадлежащему консорциуму.

Выводы по второму разделу

В данном разделе были рассмотрена основная теория по технологии блокчейн, виды блокчейн и некоторые алгоритмы консенсуса.

3. ТРЕБОВАНИЯ К СИСТЕМЕ

Система будет представлять собой веб-приложение, обеспечивающее создание дипломов, публикацию и проверку их в блокчейне, а также отзыв. С помощью веб-приложения, пользователь сможет создать шаблон диплома, для этого ему нужно будет ввести данные своей организации, выпускающей дипломы. Пользователь сможет выбрать созданный шаблон и прикрепить файл со списком получателей. На основе них веб-приложение создаст партию дипломов, отправит их в блокчейн и вернет пользователю архив с файлами дипломов. Для того, чтобы проверить диплом, пользователь сможет прикрепить файл диплома. Веб-приложение проверит его наличие в блокчейне и выведет сам диплом и результат проверки пользователю. Кроме этого, можно будет отозвать диплом через веб-приложение.

Для обработки хранения дипломов будет использовано блокчейн приложение. Дипломы будут содержать личные данные сотрудников и получателей. Для того, чтобы она не была общедоступна, нужно использовать частный блокчейн для их хранения.

3.1. Функциональные требования к системе

Можно определить следующий набор функциональных требований к системе.

1. Веб-приложение должно обеспечивать аутентификации и авторизацию пользователя.

2. Веб-приложение должно позволять пользователю вводить данные о своей организации.

3. Веб-приложение должно позволять прикреплять файлы в формате CSV и JSON.

4. Веб-приложение должно поддерживать создание шаблона диплома, партии дипломов на основе данного шаблона, проверку и просмотр диплома.

5. Блокчейн приложение должно позволять пользователю задавать

параметры при запуске.

3.2. Нефункциональные требования к системе

Можно определить следующий набор нефункциональных требований к системе.

1. Веб-приложение должно быть доступно в браузерах Google Chrome, FireFox последних версий.

2. Веб-приложение должно быть написано на языке программирования Python 3.8 [43].

3. Блокчейн приложение должно работать на операционной системе Linux Ubuntu 18.04.

4. Блокчейн приложение должно быть основано на частном блокчейне.

5. Блокчейн приложение не должно использовать валюту.

6. Блокчейн приложение должно быть написано на языке программирования Golang [23].

3.3. Варианты использования системы

С системой взаимодействует три актера (рисунок 3.1).

1. *Администратор* – это суперпользователь, который может просматривать поступившую от других пользователей информацию, создавать, удалять или изменять аккаунты сотрудников.

2. *Сотрудник* – это авторизованный пользователь, способный просматривать, создавать, удалять шаблоны дипломов, создавать и подписывать партии дипломов, проверять их.

3. *Гость* – это неавторизованный пользователь, способный проверить диплом.

Данные актеры могут реализовать следующие варианты использования системы.

1. Администратор может *просмотреть список сотрудников*.

2. Администратор может *просмотреть*, *создать*, *изменить* или *удалить аккаунт сотрудника*.

3. Сотрудник может *просмотреть список шаблонов*, которые уже существуют.

4. Сотрудник может *создать шаблон*, введя информацию о дипломе и об организации его выпускающей. Информацией о дипломе является: название, описание. Информацией об организации является: название, сайт, изображение логотипа, печати и почтовый ящик.

5. Сотрудник может *просмотреть шаблон*.

6. Сотрудник может *удалить шаблон*.

7. Сотрудник может *создать партию дипломов* на основе существующего шаблона диплома.

8. Сотрудник может *отозвать диплом*, если он существует.

9. Сотрудник и гость могут *проверить диплом*.

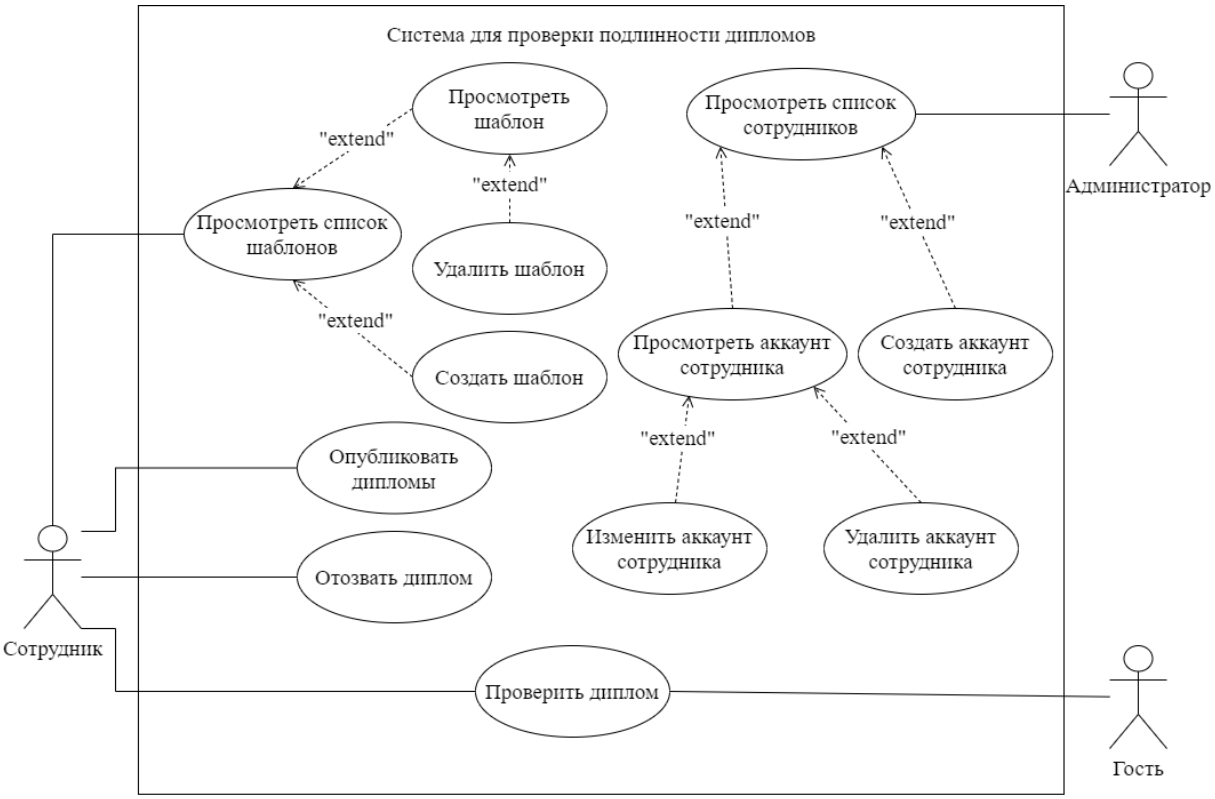

Рис. 3.1. Диаграмма вариантов использования системы

Выводы по третьему разделу

В данном разделе были составлены функциональные и нефункциональные требования к системе. Кроме этого была составлены диаграммы вариантов использования, для наглядного представления вариантов использования системы.

4. АРХИТЕКТУРА СИСТЕМЫ

На рисунке 4.1 представлена диаграмма компонентов системы. Она состоит из двух частей: веб приложения и блокчейн приложения.

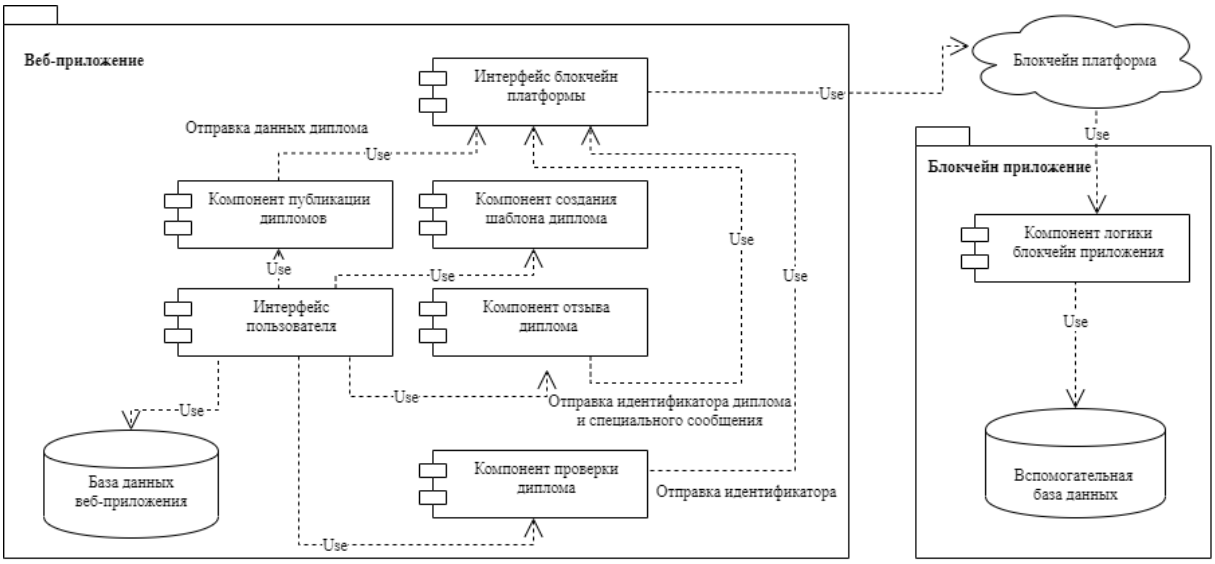

Рис. 4.1. Диаграмма компонентов

Блокчейн приложение представляет собой хранилище для пар ключ значение, где ключом является идентификатор диплома, а значением хэш файла диплома. Оно состоит из следующих компонентов:

1) компонент логики приложения реализует логику хранилища пар ключ значение поверх блокчейна сторонней платформы. Данный компонент взаимодействует со вспомогательной базой данных для организации логики получения хэша диплома по его идентификтору;

2) блокчейн платформа является сторонней платформой, обеспечивающей сам блокчейн и необходимые компоненты для его работы.

Веб-приложение представляет собой веб-сервис, который позволяет создать дипломы, опубликовать, проверить в блокчейне и отозвать. Оно состоит из следующих компонентов:

1) интерфейс пользователя инкапсулирует взаимодействие пользователя с веб-приложением;

2) компонент создания шаблона диплома отвечает за создание шаблона диплома в виде структуры данных, которая дальше будет помещена в

файл формата json;

3) компонент публикации диплома отвечает за создание партии новых дипломов и отправку их идентификаторов и хэшей в блокчейн;

4) компонент проверки диплома отвечает за сравнение данных диплома и данных полученных из блокчейна;

5) компонент отзыва диплома отвечает за отзыв диплома;

6) интерфейс блокчейн платформы инкапсулирует взаимодействие веб-приложения и блокчейн платформы.

4.1. Компоненты блокчейн приложения

4.1.1. Выбор платформы для реализации блокчейн приложения

Многие блокчейн решения (например, Bitcoin [17]) имеют монолитный дизайн. То есть они представляют собой отдельную программу, которая обрабатывает все части децентрализованного распределенного реестра, такие как: подключение peer-to-peer, трансляцию транзакций в mempool, работу консенсус алгоритма, умные контракты, разрешения на уровне пользователя и т. д [44]. Использование монолитной архитектуры обычно является плохой практикой, т. к. это затрудняет повторное использование компонентов кода.

Другая проблема с монолитным дизайном состоит в том, что он ограничивает разработчика языком программирования программного обеспечения блокчейн решения. В случае Ethereum [30], который поддерживает полную по Тьюрингу виртуальную машину со своим байт-кодом (bytecode), он ограничивает разработчика языками, которые компилируются до этого байт-кода, сейчас это Snake и Solid.

Tendermint Core соответствует требованиям, представленным к системе.

1. Алгоритм консенсуса Tendermint можно использовать в частных блокчейнах.

2. В приложениях на основе Tendermint Core можно не определять валюту.

Кроме того, в Tendermint Core отделен механизм блокчейна от других компонентов и можно создавать приложения на разных языках программирования, поэтому данная платформа удобна для реализации приложения.

В итоге платформа Tendermint Core была выбрана для реализации блокчейн приложения.

4.1.2. Tendermint ABCI

Интерфейс ABCI состоит из множества методов. Tendermint вызывает методы ABCI в блокчейн приложении, отправляя запросы в блокчейн приложения и получая от него ответ.

В блокчейн-приложении будут реализованы методы, представленные ниже.

1. Info – обрабатывает сообщения запроса и ответа Info. Когда tendermint в первый раз устанавливает соединение с ABCI приложением, он отправляет запрос Info, чтобы получить в ответ высоту последнего блока, хэш состояния и другую информацию, соответствующую состоянию приложения [29]. На рисунке 4.2 представлена структура запроса RequestInfo и ответа ResponseInfo.

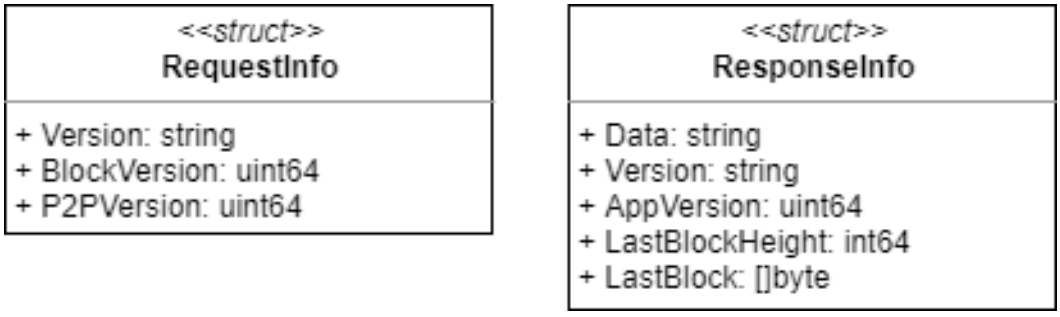

Рис. 4.2. Структура RequestInfo и ResponseInfo

2. DeliverTx – обрабатывает сообщения запроса и ответа DeliverTx для каждой транзакции в блоке, когда обновляется состояние конечного автомата приложения [29].

На рисунке 4.3 представлена структура запроса RequestDeliverTx и ответа ResponseDeliverTx.

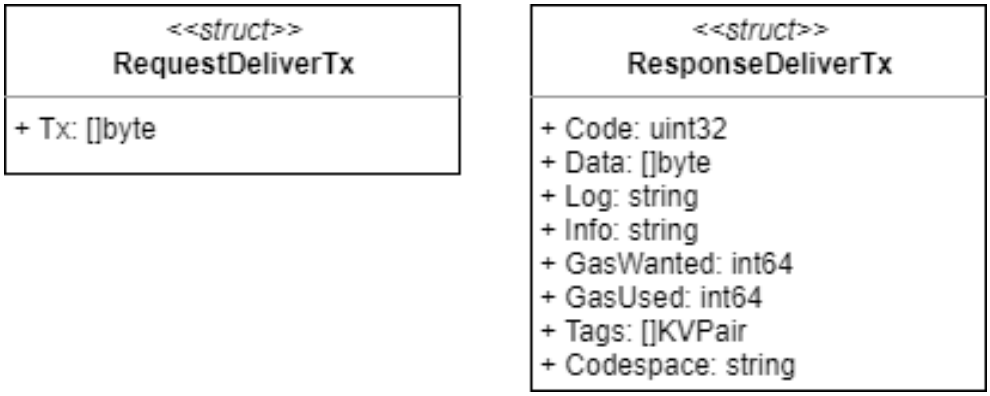

Рис. 4.3. Структура RequestDeliverTx и ResponseDeliverTx

3. CheckTx – обрабатывает сообщения запроса и ответа CheckTx. Когда tendermint получает новую транзакцию от интерфейса RPC или через соединение P2P, он сначала передает ее в ABCI приложение сообщением запроса CheckTx для проверки правильности транзакции. Только подтвержденные транзакции войдут в пул транзакций tendermint и будут ожидать включение в блок [29]. На рисунке 4.4 представлена структура запроса RequestCheckTx и ответа ResponseCheckTx.

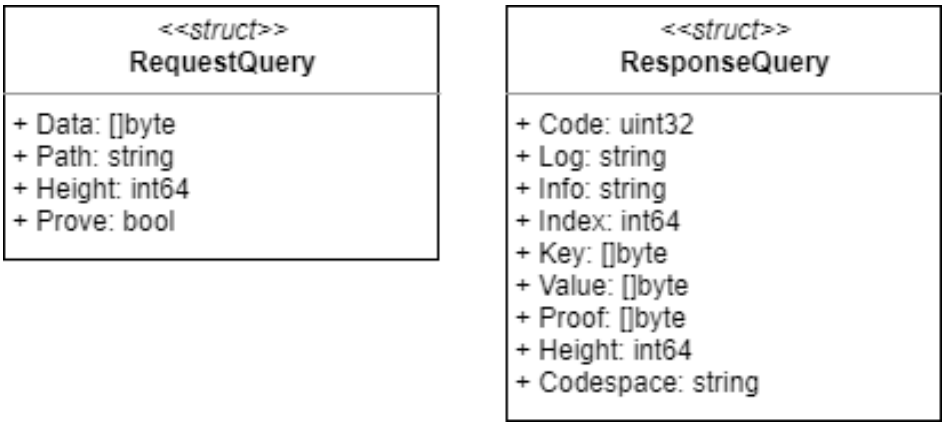

Рис. 4.4. Структура RequestCheckTx и ResponseCheckTx

4. Query – обрабатывает сообщения запроса и ответа DeliverTx для каждой транзакции в блоке, когда обновляется состояние конечного автомата приложения [29]. На рисунке 4.5 представлена структура запроса RequestQuery и ответа ResponseQuery.

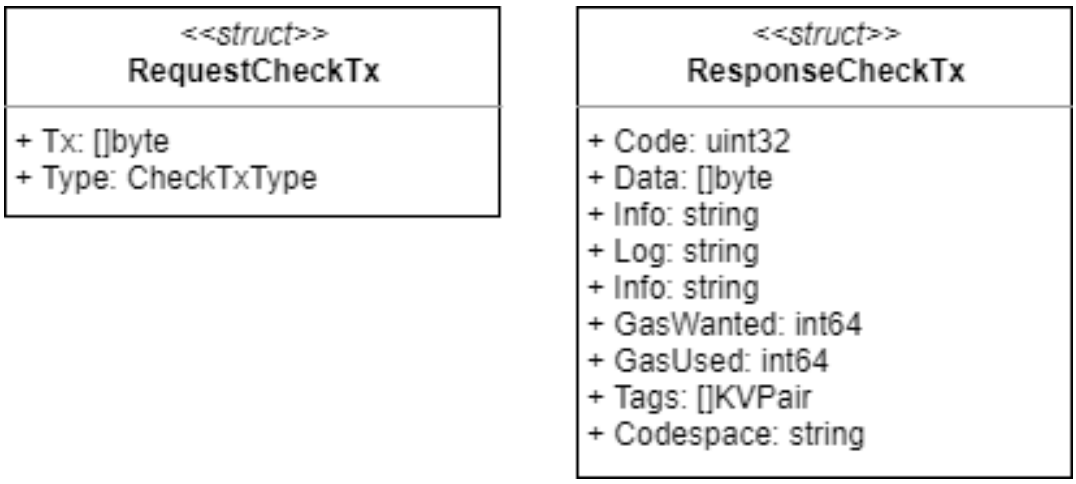

Рис. 4.5. Структура RequestQuery и ResponseQuery

5. Commit – обрабатывает сообщение ответа Commit. Последним шагом добавления транзакции в блокчейн является отправление tendermint запроса Commit в ABCI приложение, т.е. запрос на фиксацию транзакций в блокчейне [29]. На рисунке 4.6 представлена структура ответа ResponseCommit.

Рис. 4.6. Структура ResponseCommit

4.2. Структура диплома

Диплом состоит из самого диплома и квитанции, содержащей хэш диплома.

Документ с дипломом будет храниться и предоставляться пользователям в формате JSON [45], т. к. он удобен для чтения как человеку, так и компьютеру, в документ данного формата можно легко заносить информацию используя любой язык программирования, т. к. для каждого из них есть библиотека для обработки формата JSON, например, для языка программирования Python [43] существует библиотека json.

Для того, чтобы диплом обладал определенной структурой, возьмем схему OpenBadges 2.0 [46], использующую JSON-LD [47] контекст.

В таблице 4.1 представлены поля файла диплома и их описание.

Таблица 4.1. Описание полей файла

Поле	Описание
issuedOn	Метка времени, когда диплом был опубликован в блокчейн.
recipientProfile	Служит для идентификации получателя.
recipientProfile.name	Имя получателя.
recipientProfile.type	Тип определенный в файле контекста JSON-LD.
type	Тип определенный в файле контекста JSON-LD.
badge.signatureLines	Поле необходимое для представления сотрудника опубликовавшего диплом.
badge.signatureLines.jobTitle	Должность сотрудника.
badge.signatureLines.image	Изображение подписи сотрудника.
badge.signatureLines.type	Тип определенный в файле контекста JSON-LD.
badge.signatureLines.name	ФИО сотрудника.
badge.image	Изображение печати организации, выдающей диплом.
badge.name	Название диплома.
badge.type	Тип определенный в файле контекста JSON-LD.
badge.id	Уникальный идентификатор документа. Содержит префикс «urn:uuid» и сам идентификатор.
badge.issuer	Описывает организацию, которая выдает диплом.
badge.issuer.name	Название организации
badge.issuer.url	Сайт организации.
badge.issuer.image	Изображение логотипа.
badge.issuer.id	Уникальный идентификатор документа. Содержит префикс «urn:uuid» и сам идентификатор.
badge.issuer.type	Тип определенный в файле контекста JSON-LD.
badge.issuer.email	Электронная почта.
@contexte	Определяет контекст документа.
recipient	Описывает получателя документа.

Окончание таблицы 4.1

Поле	Описание
recipient.type	Тип определенный в файле контекста JSON-LD.
recipient.hashed	Хэшируется ли значение recipient.identity.
recipient.identity	Электронная почта получателя.
id	Уникальный идетификатор документа. Содержит префикс «urn:uuid» и уникальный идентификатор.
signature	Описывает информацию о транзакции.
signature.type	Тип определенный в файле контекста JSON-LD.
signature.targetHash	Хэш документа для сравнения значения в блокчейне.

Хэшироваться и являться дипломом будет содержимое файла в формате JSON без поля signature, т. к. данное поле будет являться квитанцией, с которой будет сверяться хэш диплома при его проверке.

4.3. Схема базы данных

Исходя из требований к системе была разработана схема базы данных представленная на рисунке 4.7. На этапе проектирования были описаны названия всех таблиц и полей данных.

Рассмотрим подробнее таблицы и поля базы данных.

Таблица *User* содержит информацию о пользователе и включает в себя следующие поля:

1) *id* – уникальный идентификатор пользователя;
2) *username* – логин пользователя;
3) *password* – пароль пользователя;
4) *first_name* – имя пользователя;
5) *last_name* – фамилия пользователя;
6) *email* – почта пользователя.

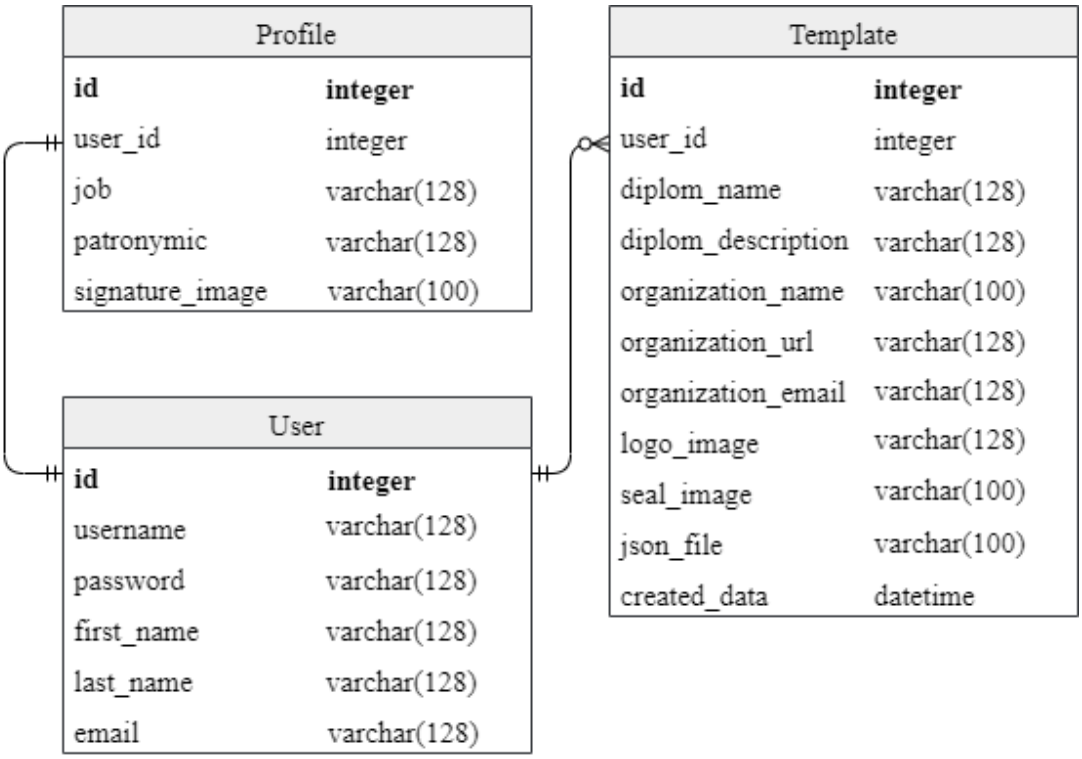

Рис. 4.7. Схема базы данных

Таблица *Profile* содержит дополнительную информацию о пользователе и включает в себя следующие поля:

1) *id* – уникальный идентификатор профиля;

2) *user_id* – поле внешнего ключа, уникальный идентификатор пользователя, к которому относится данный профиль;

3) *job* – должность пользователя;

4) *patronymic* – отчество пользователя;

5) *signature_image* – изображение подписи пользователя.

Таблица *Template* содержит информацию о шаблоне диплома и включает в себя следующие поля:

1) id – уникальный идентификатор шаблона;

2) *user_id* – поле внешнего ключа, уникальный идентификатор пользователя, к которому относится данный шаблон;

3) diplom_name – название диплома;

4) diplom_description – описание диплома;

5) organization_name – название организации, выдающей диплом;

6) organization_url – сайт организации, выдающей диплом;

7) organization_email – почта организации, выдающей диплом;

8) logo_image – изображение логотипа организации, выдающей диплом;

9) seal_image – изображение печати организации, выдающей диплом;

10) json_file – файл шаблона диплома в формате JSON;

11) created_data – дата создания шаблона.

Выводы по четвертому разделу

В данном разделе были смоделированы диаграмма компонентов системы и схема базы данных. Были описаны методы блокчейн приложения и структура файла диплома.

5. РЕАЛИЗАЦИЯ

5.1. Реализация блокчейн приложения

Для реализации блокчейн приложения был выбран язык программирования Golang [23], что позволяет скомпилировать приложение и ядро Tendermint Core в один исполняемый бинарный файл, а в качестве среды разработки Goland.

На рисунке 5.1 представлены структуры, разработанные для блокчейн приложения Application и State, структура базового ABCI приложения и интерфейсы Application и DB принадлежащие tendermint.

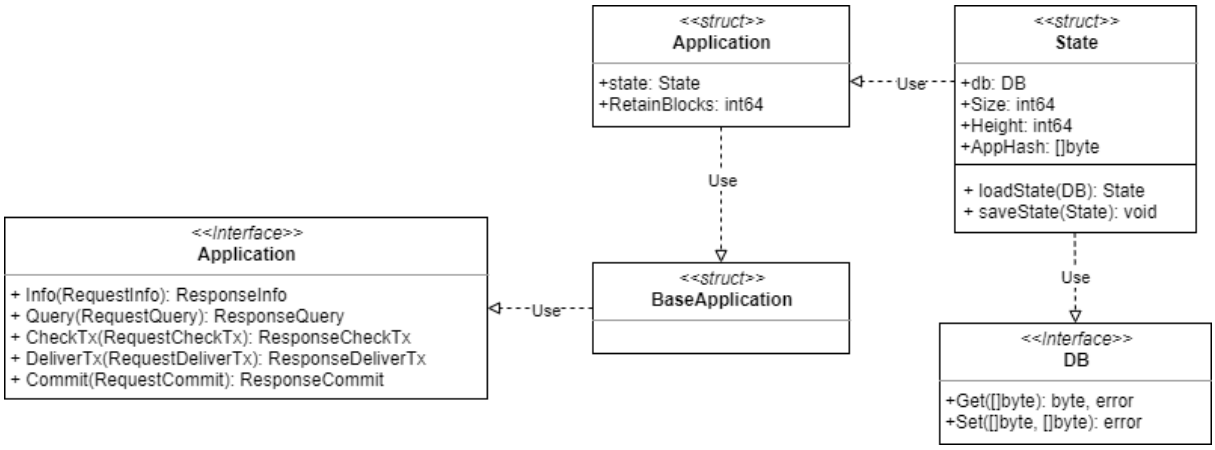

Рис. 5.1. Структура блокчейн приложения

Структура BaseApplication является базовым блокчейн приложением. Данную структуру предоставляет tendermint для доступа к интерфейсу Application.

Интерфейс DB является интерфейсом tendermint и отвечает за работу с базой данных. Метод Get предназначен для получения данных из базы данных по ключу. На вход подается значение ключа. Результатом является значение из базы данных и значение ошибки (пустое, если успешно). Метод Set предназначен для установки значения для данного ключа в базу данных, если оно уже установлено, то происходит замена. На вход подается ключ и значение. Результатом является значение ошибки (пустое, если успешно).

Структура State отвечает за состояние блокчейн приложения. Метод loadState предназначен для загрузки состояния из базы данных. Метод

saveState предназначен для сохранения состояния блокчейн приложения в базу данных. Листинг файла state,go, содержащий структуру State и методы loadState и saveState представлен в приложении А.

Структура Application является основной для блокчейн приложения. В листинге 5.1 представлена реализация структуры Application.

Листинг 5.1. Реализация структуры Application

```
type Application struct {
	types.BaseApplication

	state        State
	RetainBlocks int64 // blocks to retain after commit
}
```

Интерфейс Application является интерфейсом tendermint и отвечает за логику блокчейн приложения. Каждый метод соответствует определенному методу ABCI.

В методе Info реализован возврат сообщения ResponseInfo, содержащее информацию о блокчейн приложении, например, высоту и хэш последнего блока. В листинге 5.2 приведен код метода Info.

Листинг 5.2. Код метода Info

```
func (app *Application) Info(req types.RequestInfo) (resInfo
types.ResponseInfo) {
	return types.ResponseInfo{
		Data:             fmt.Sprintf("{\"size\":%v}", app.state.Size),
		Version:          version.ABCIVersion,
		AppVersion:       ProtocolVersion.Uint64(),
		LastBlockHeight:  app.state.Height,
		LastBlockAppHash: app.state.AppHash,
	}
}
```

В методе DeliverTx реализована обработка входящего сообщения RequestDeliverTx, включающая в себя разделение входящей транзакции на две части (ключ и значение), и возврат ResponseDeliveTx.

В листинге 5.3 приведен код метода DeliverTx.

Листинг 5.3. Код метода DeliverTx

```
func (app *Application) DeliverTx(req types.RequestDeliverTx)
types.ResponseDeliverTx {
	var key, value []byte
	parts := bytes.Split(req.Tx, []byte("="))
	if len(parts) == 2 {
		key, value = parts[0], parts[1]
	} else {
		key, value = req.Tx, req.Tx
	}

	app.state.db.Set(prefixKey(key), value)
	app.state.Size++

	events := []types.Event{
		{
			Type: "app",
			Attributes: []kv.Pair{
				{Key: []byte("key"), Value: key},
				{Key: []byte("value"), Value: value},
			},
		},
	}

	return types.ResponseDeliverTx{Code: code.CodeTypeOK, Events: events}
}
```

В методе CheckTx реализована обработка входящего сообщения RequestCheck, включающая в себя проверку правильности транзакции с помощью вызова метода isValid, и возврат ResponseDeliveTx. В листинге 5.4 приведен код методов CheckTx и isValid.

Листинг 5.4. Код методов CheckTx и isValid

```
func (app *Application) isValid(tx []byte) (code uint32) {
	// check format
	parts := bytes.Split(tx, []byte("="))
	if len(parts) != 2 {
		return 1
```

```
	}
	key, value := parts[0], parts[1]
	// check if the same key=value already exists
	val, err := app.state.db.Get(prefixKey(key))
	if err != nil {
		panic(err)
	}
	if bytes.Equal(val, value){
		code = 2
	}

	return code
}


func	(app	*Application)	CheckTx(req	types.RequestCheckTx)
types.ResponseCheckTx {
	code := app.isValid(req.Tx)
	return types.ResponseCheckTx{Code: code, GasWanted: 1}
}
```

В методе Query реализована проверка наличия в базе данных значения из сообщения RequestQuery. Возвращаемое сообщение ResponseQuey содержит результат проверки. В приложении Б приведен код метода Query

И наконец, в методе Commit реализовано изменение состояния блокчейн приложения и возврат ResponseCommit. В листинге 5.5 приведен код метода Commit.

Для того, чтобы была возможность создания нового экземпляра приложения был реализован метод NewApplication. В листинге 5.6 представлен код метода NewApplication.

Кроме экземпляра самого приложения, необходимо иметь возможность создать новый узел tendermint, поэтому был реализован метод newTendermint. В данном методе присутствует чтение файла конфигурации, его изменение в соответствии с параметрами, заданными пользователями, и на основе данных параметров создание нового узла. В приложении В приведен код метода.

Листинг 5.5. Код метода Commit

```
func (app *Application) Commit() types.ResponseCommit {
	appHash := make([]byte, 8)
	binary.PutVarint(appHash, app.state.Size)
	app.state.AppHash = appHash
	app.state.Height++
	saveState(app.state)

	resp := types.ResponseCommit{Data: appHash}
	if app.RetainBlocks > 0 && app.state.Height >= app.RetainBlocks {
		resp.RetainHeight = app.state.Height - app.RetainBlocks + 1
	}
	return resp
}
```

Листинг 5.6. Код метода NewApplication

```
func NewApplication(dbDir string) *Application {
	name := "dstore"
	db, err := dbm.NewGoLevelDB(name, dbDir)
	if err != nil {
		panic(err)
	}

	state := loadState(db)
	return &Application{state: state}
}
```

Блокчейн приложение, скомпилированное в один бинарный файл с помощью команды go build, было размещено в двух облачных хранилищах Amazon EC2 под управлением виртуального окружения Ubuntu Server 18.04 LTS.

Кроме самого приложения были также размещены файлы конфигурации, которые были созданы с помощью бинарного файла tendermint и опции init.

Конфигурационные файлы содержатся в двух каталогах: data и config.

В каталоге data размещается файл priv_validator_key.json, как следует

из названия, он содержит закрытый ключ валидатора.

В каталоге config содержатся следующие файлы конфигурации:

1) addrbook.json – файл, содержащий адреса для связи по P2P;

2) config.toml – файл конфигурации узла;

3) node_key.json – файл, содержащий ключ узла, используемый для шифрования P2P связи;

4) genesis.json – файл конфигурации генезис блока, включающий в себя информацию о времени запуска блокчейна, его идентификатор, адреса и публичные ключи изначально заданных вадидаторов и параметры консенсуса.

На рисунке 5.2 показана структура директории, содержащей файлы блокчейн приложения.

```
ubuntu@ip-172-31-59-249:~/blockchain_app$ tree
.
├── MyDiplomProject
└── appdata
    ├── config
    │   ├── addrbook.json
    │   ├── config.toml
    │   ├── genesis.json
    │   ├── node_key.json
    │   └── priv_validator_key.json
    └── data
        └── priv_validator_state.json

3 directories, 7 files
```

Рис. 5.2.Файлы блокчейн приложения

В процессе работы в каталоге с бинарным файлом приложения Tendermint Core создаст в каталоге data следующие файлы формата leveldb:

1) blockstore.db – хранит блоки, фиксации блоков и метаданные блоков, каждый из которых проиндексирован по высоте. Используется для синхронизации новых пиров;

2) evidence.db – хранит все доказательства плохого поведения валидатора;

3) state.db – хранит текущее состояние блокчейна, изменяется только

в случае изменения параметров консенсуса или валидаторов, также используется для временного хранения промежуточных результатов во время обработки блока;

4) tx_index.db – хранит данные об индексах транзакций [29].

Также Tendermint Core использует файл предзаписи cs.wal [29].

Для того, чтобы настроить узел, нужно изменить файл config.toml, чтобы постоянно его не переписывать в приложение были добавлены опции, которые изменяют стандартную конфигурацию (рисунок 5.3).

```
daria@daria-A6GMV:~/go/src/BlockchainDiplomProject$ ./MyDiplomProject -h
Usage of ./MyDiplomProject:
  -config string
        Path to config.toml (default "$HOME/config/config.toml")
  -empty
        Set this to false to only produce blocks when there are txs or when the
AppHash changes (default true)
  -per_peer string
        Set persistent peers
  -rpc_adr string
        Set listen address for rpc (default "tcp://127.0.0.1:26657")
```

Рис. 5.3. Опции блокчейн приложения

Опция -config нужна для указания расположения файлов конфигурации и данных на машине.

По умолчанию в Tendermint создаются пустые блоки, которые не содержат транзакций, с интервалом, заданным в параметре create_empty_blocks_interval, для того, чтобы иметь возможность создавать только блоки содержащие транзакции была добавлена опция empty.

Опция –per_peer нужна для указания валидатору адресов других валидаторов для связи по P2P.

Узел с блокчейн приложением должен прослушивать адрес машины, на которой расположено веб-приложение, по умолчанию используется локальный адрес и порт 26657. Для того, чтобы задавать собственный адрес была добавлена опция –rpc_adr.

На рисунке 5.4 представлен терминал с запущенным блокчейн приложением.

```
ubuntu@ip-172-31-50-159: ~/blockchain_app
I[2020-06-04|16:30:44.361] Committed state                          module=s
tate height=14 txs=1 appHash=2E00000000000000
I[2020-06-04|16:30:45.714] Executed block                           module=s
tate height=15 validTxs=0 invalidTxs=0
I[2020-06-04|16:30:45.723] Committed state                          module=s
```

Рис. 5.4. Терминал с запущенным блокчейн приложением

5.2. Реализация компонентов для создания, подписания, отзыва и проверки дипломов

Реализация компонента создания диплома

Для данного компонента был разработан класс TemplateJson. Он отвечает за создание шабона диплома. Метод create_diplom_template принимает данные необходимые для создания шаблона диплома. Данный метод генерирует идентификаторы для диплома и вызывает методы для создания секций диплома. В итоге он возвращает данные шаблона в виде словаря Python.

Компонент создания диплома был реализован как класс для того, чтобы в дальнейшем не запутаться при введении данных шаблона. Код данного класса приведен в листинге 5.7.

Листинг 5.7. Код класса TemplateJson

```
class TemplateJson (object):
    def __index__(self, user_name, user_jobtitle, signature_image,
                  diplom_name, diplom_descr, orgn_name,
                  org_url, org_email, seal_image, logo_image,):
        self.user_name = user_name
        self.user_jobtitle = user_jobtitle
        self.signature_image = signature_image
        self.diplom_name = diplom_name
        self.diplom_descr = diplom_descr
        self.org_name = orgn_name
        self.org_url = org_url
        self.org_email = org_email
        self.logo_image = logo_image
        self.seal_image = seal_image

    def create_diplom_template(self):
        t_uuid = str(uuid.uuid4())
```

```
        ss_uuid = str(uuid.uuid4())
        diplom = create_diplom_section(t_uuid,self.diplom_name,
self.diplom_descr, ss_uuid,self.org_name,
self.org_url,self.org_email, self.user_jobtitle, self.user_name,
self.seal_image, self.logo_image, self.signature_image)
        assertion = create_assertion_section()
        recipient = create_recipient_section()
        recipient_profile = create_recipient_profile_section()
        assertion['recipient'] = recipient
        assertion['recipientProfile'] = recipient_profile
        assertion['badge'] = diplom
        return assertion
```

Реализация компонентов публикации и отзыва диплома

Для данного компонента была разработана функция sign_diploms. Код данной функции приведен в листинге 5.8.

Листинг 5.8. Код функции sign_diploms

```
def sign_diploms(roster, template_file):
    diploms = create_unsigned_diploms_from_file(roster, template_file)
    messages = {}
    for uid in diploms.keys():
        diplom_json = json.dumps(diploms[uid])
        diplom_json_byte = diplom_json.encode()
        hashed = hash_byte_array(diplom_json_byte)
        result, err = broadcast_tx(uid, hashed)
        if err == True:
            messages[uid] = result
        else:
            messages[uid] = 'Диплом успешно добавлен'
        diploms[uid]['signature'] = {
            "type": ['Proof', 'Extension'],
            "targetHash": hashed,
        }
    return diploms, messages
```

Функция sign_diplom принимает на вход путь к двум файлам: список получателей и шаблон диплома, а на выход возвращает список опубликованных дипломов и сообщения для пользователя.

Данная функция вызывает другие функции необходимые для создания опубликованных дипломов, одной из них является функция create_unsigned_diploms_from_file, которая принимает на вход путь к двум файлам: список получателей и шаблон диплома, а на выходе возвращает список неопубликованных дипломов. Код функции create_unsigned_diploms_from_file приведен в листинге 5.9.

Листинг 5.9. Код функции create_unsigned_diploms_from_file

```
def create_unsigned_diploms_from_file(roster, template_file):
    recipients = get_recipients_from_roster(roster)
    template = get_template(template_file)
    issued_on = create_iso8601_tz()
    diploms = {}
    for recipient in recipients:
        uid = str(uuid.uuid4())
        cert = copy.deepcopy(template)
        instantiate_assertion(cert, uid, issued_on)
        instantiate_recipient(cert, recipient)
        diploms[uid] = cert
    return diploms
```

Другой важной функцией является broadcast_tx, т. к. она связывается с блокчейн приложением по протоколу RPC. Функция принимает на вход идентификатор диплома и его хэш, а на выход возвращает в случае успеха хэш, в случае неудачи сообщение об ошибке.

Код приведен в листинге 5.10.

Листинг 5.10. Код функции broadcast_tx

```
def broadcast_tx(uid, hashed):
    url = "http://"+ val_url+":26657"
    message = uid + '=' + hashed
    message_bytes = message.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')
    payload = {
        "id": -1,
        "jsonrpc": "2.0",
        "method": "broadcast_tx_sync",
```

```
        "params": [base64_message]
    }
    response = requests.post(url, json=payload).json()
    try:
        response["error"]
    except KeyError:
        return response["result"]["hash"], False
    else:
        return response["error"]["data"], True
```

Для компонента отзыва диплома была использована функция broadcast_tx, только вместо хеша диплома на вход подается строка «deleted».

Реализация компонента проверки диплома

Для данного компонента была разработана функция verify_diplom. Код функции приведен в листинге 5.11.

Листинг 5.11. Код функции verify_diplom

```
def verify_diplom(diplom_json):
    diplom = json.loads(diplom_json)
    targetHash = diplom['signature']['targetHash']
    del diplom['signature']
    diplom_no_sign = json.dumps(diplom)
    diplom_no_sign_byte = diplom_no_sign.encode()
    hashed = hash_byte_array(diplom_no_sign_byte)
    di_id = diplom['id']
    di_id = di_id.replace("urn:uuid:", "")
    log_response, value_response = check_tx(di_id)
    if log_response == 'exists':
        message1 = 'exist'
    elif log_response == 'recalled':
        message1 = 'recalled'
    else:
        message1 = 'not exist'
    if hashed == value_response == targetHash:
        message2 = 'same'
    else:
        message2 = 'not same'
    return message1, message2
```

Функция verify_diplom принимает на вход путь к файлу диплома, а на выход возвращает два сообщения с результатами проверки.

Данная функция вызывает функцию chech_tx, логика которой идентична функции broadcast_tx за исключением используемого метода. В данном случает – это метод abci_query.

5.3. Реализации веб-приложения

Для разработки веб-приложения был выбран фреймворк Django [48], использующий шаблон проектирования, подобный шаблону MVC, так как на нем можно быстро написать свое приложение, данный фреймворк серьезно относится к безопасности, имеет заранее встроенную систему аутентификации и панель для администратора.

Так как Django написан на языке Python [43], то для разработки был выбран язык программирования Python 3.8, а средой для разработки и тестирования был выбран PyCharm.

В рамках создания веб-приложения для проверки подлинности дипломов необходимо реализовать структуру базы данных, обработку данных, по ступающих от пользователей, и внешний вид страниц сайта.

5.3.1. Реализация структуры базы данных

В качестве базы данных была использована СУБД SQLite [49], которая в проекте приложения представлена одним файлом db.sqlite3.

В соответствии со схемой, приведенной в проектировании, был создан файл models.py, содержащий представления данных в базе данных. Он содержит два класса – Profile и Template. Класс Profile является представлением данных профиля сотрудника учебной организации, а Template – данных шаблона диплома, принадлежащего сотруднику. Код данных классов представлен в приложении Г.

В качестве реализации таблицы User из разработанной в проектировании схемы был взят класс User предоставляемый Django.

5.3.2. Реализация внутренней логики

В качестве интерфейса пользователя были реализованы формы, используемые для приема данных от пользователей веб-приложения.

Формы для создания шаблона, подписания и отзыва диплома были реализованы с использованием класса Dlango Form. Данный метод упрощает генерацию формы в HTML, очистку и валидацию данных. Для всех трех форм были определены поля для получения данных от пользователей. Код форм находится в файле forms.py.

Для обработки данных форм были созданы три функции представлений: create_template, sign_template и recall_template. Код данных функций находится в views.py.

Функция create_template при обработке запроса POST получает проверенные Django данные из формы TemplateCreateForm и информацию о пользователе, затем создает новый экземпляр класса TemplateJson, который необходим для генерации файла диплома. Экземпляр класса вызывает функцию create_diplom_template для получения объекта диплома, который в дальнейшем генерируется в JSON строку и записывается в файл. И, наконец, вся полученная информация и JSON файл с дипломом сохраняются в базу данных. Пример JSON файла диплома представлен в приложении Д.

Функция sign_template при обработке POST запроса получает проверенные Django данные из формы TemplateSignForm и информацию о пользователе, проверяет наличие соединения с блокчейн приложением, с помощью функции sign_diplom получает объекты опубликованных в блокчейн дипломов, генерирует JSON файлы и добавляет их в ZIP архив, который передается пользователя в ответном сообщении.

Функция recall_template при обработке запроса POST получает проверенные Django данные из формы TemplateRecallForm, проверяет наличие соединения с блокчейн приложением, обрабатывает файлы и с помощью функции broadcast_tx отправляет транзакцию, состоящую из идентификатора диплома и строки «deleted» и возвращает пользователю результат.

Код функций create_template, sign_template и recall_template представлен в приложении Е.

У всех функций обработка GET запроса одинакова, она состоит в возврате пользователю самого запроса и сгенерированной страницы веб-приложения.

В каждой функции есть взаимодействие с файлами, поэтому были написаны функции обработчики. Например, функция handle_upload_json служит для чтения файла и переработки данных JSON в объекты Python. В листинге 5.12 представлен код данной функции.

Листинг 5.12. Код функции handle_upload_json

```
def handle_upload_json(f):
    diplom_json = f.read()
    diplom = json.loads(diplom_json)
    return diplom
```

Формы для просмотра шаблона диплома, списка шаблонов и их удаления были реализованы на основе базовых классов Dlango Form DetailView, ListView и DeleteView соответственно. Они автоматически генерируются на основе существующих классов-моделей и указанных полей.

Для обработки данных форм не требуется создания функций представлений.

Форма для проверки диплома была реализована как HTML форма. Для ее обработки была реализована функция представления index. Данная функция, при обработке POST запроса, сначала проверяет соединение с блокчейн приложением, затем получает данные из HTML формы, проверяет с помощью функции verify_diplom, полученный от пользователя, диплом. И, наконец, возвращает диплом, отформатированный с помощью функции get_formatted_award_and_verification_info, и результат проверки. Код функции index представлен в приложении Ж.

Для ограничения доступа неавторизованных пользователей в функциях представления был применен декоратор @login_requred. Если пользо-

ватель не авторизован, то его перенаправит на URL адрес страницы аутентификации. Для авторизованного пользователя будет выполняться код представления. В классах на основе базовых классов Dlango Form был добавлен класс LoginMixedMixin для перенаправления запросов неавторизованных пользователей на страницу аутентификации.

Django предоставляет почти все, что нужно для создания страниц аутентификации входа, выхода из системы, поэтому реализовывать внутреннюю логику входа и выхода из приложения не потребовалось.

5.4. Реализация внешнего интерфейса веб-приложения

Фреймворк Django динамически генерирует html файлы с использованием подхода, основанного на шаблонах. В html документах кроме самого языка гипертекстовой разметки html используется язык шаблонов. Шаблон содержит переменные, которые будут заменены значениями при выполнении шаблона, и теги, которые управляют логикой шаблона.

Для внешнего интерфейса веб-приложения были написаны html шаблоны, стили CSS и функции на языке программирования JavaScript к ним.

Для входа и выхода пользователя были также созданы html шаблоны, т. к. они не включены в Django.

На рисунке 5.5 представлена реализованная главная страница с формой для проверки диплома для авторизованного пользователя, главная страница для гостей выглядит также, но исключены пункты меню «Шаблоны», «Опубликовать» и «Отозвать» и пункт «Вход» заменен на «Выход». На рисунке 5.6 – страница со списком шаблонов, на рисунке 5.7 – страница для создания шаблона.

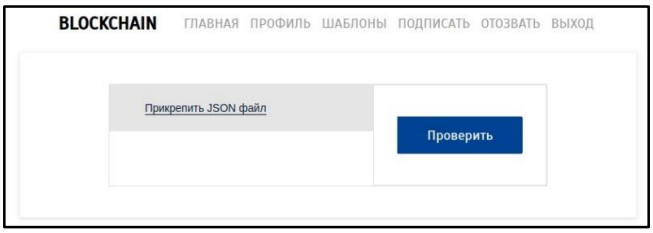

Рис. 5.5. Главная страница

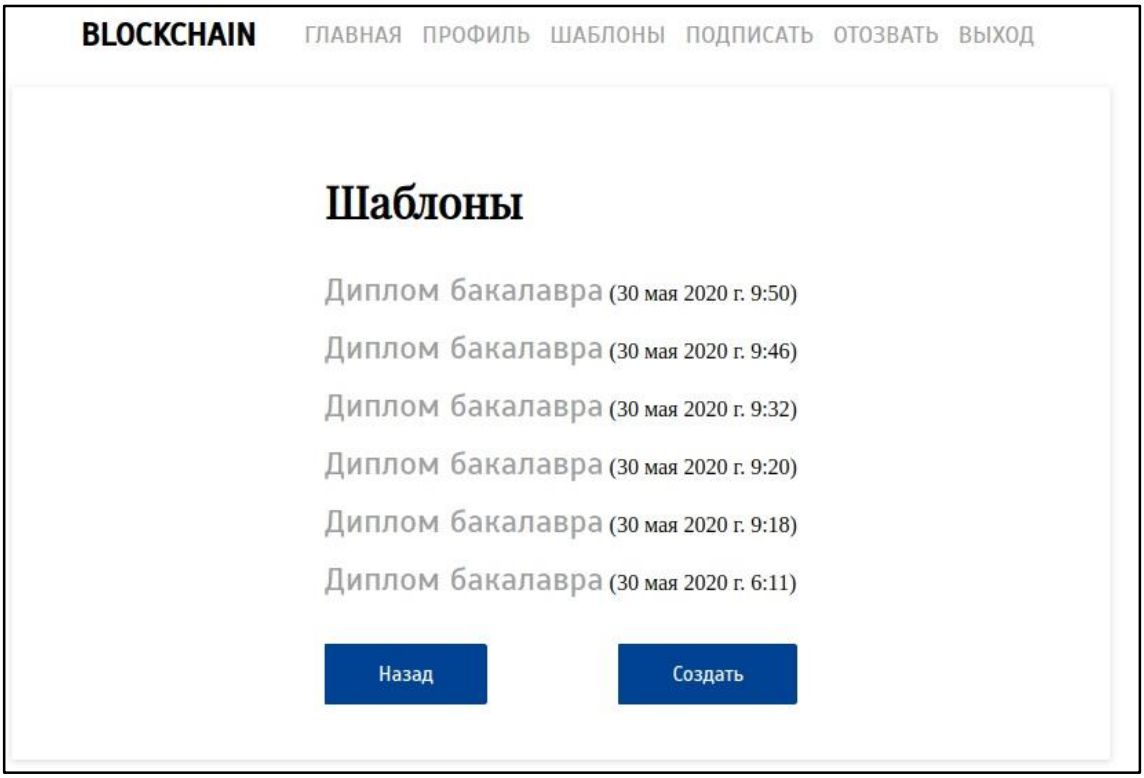

Рис. 5.6. Страница со списком шаблонов

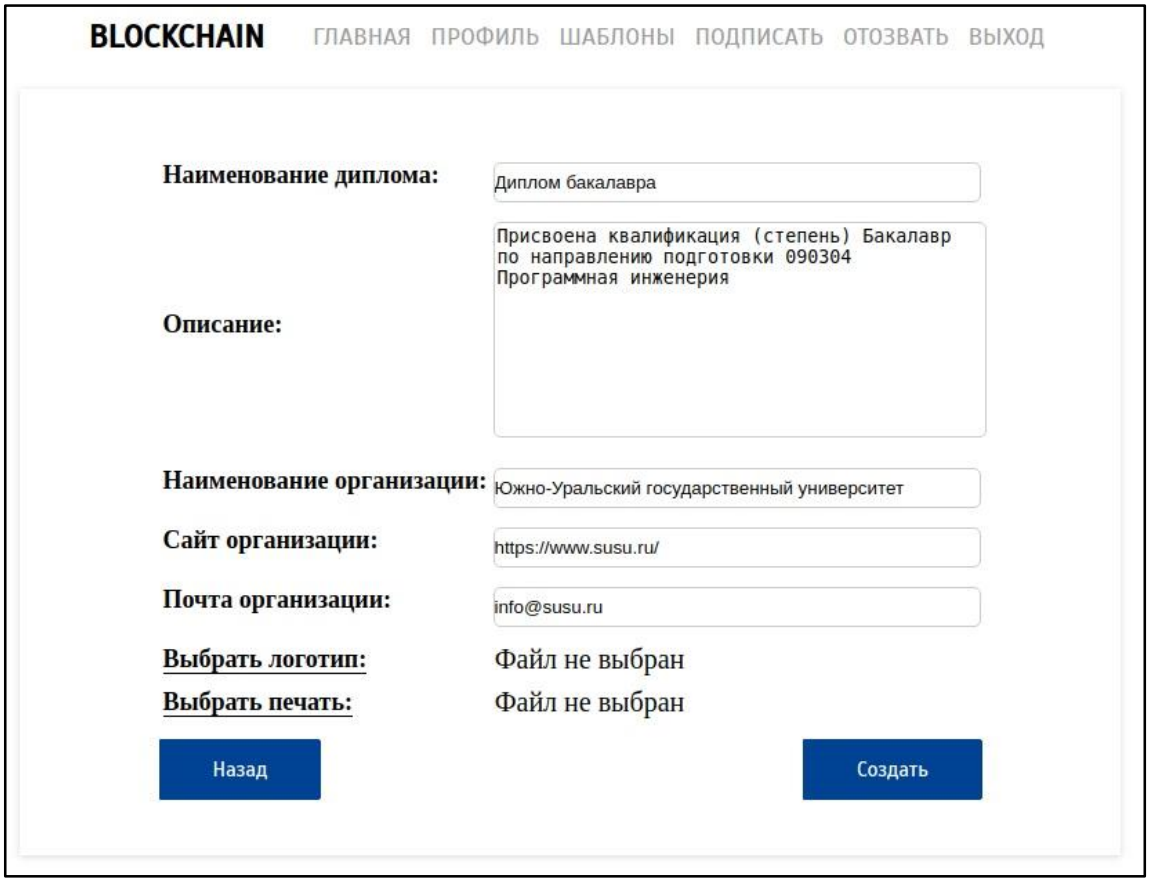

Рис. 5.7. Страница с формой для создания шаблона

На рисунке 5.8 представлены страницы для публикации диплома, а на рисунке 5.9 для отзыва диплома. Страница с проверенным дипломом представлена в приложении И.

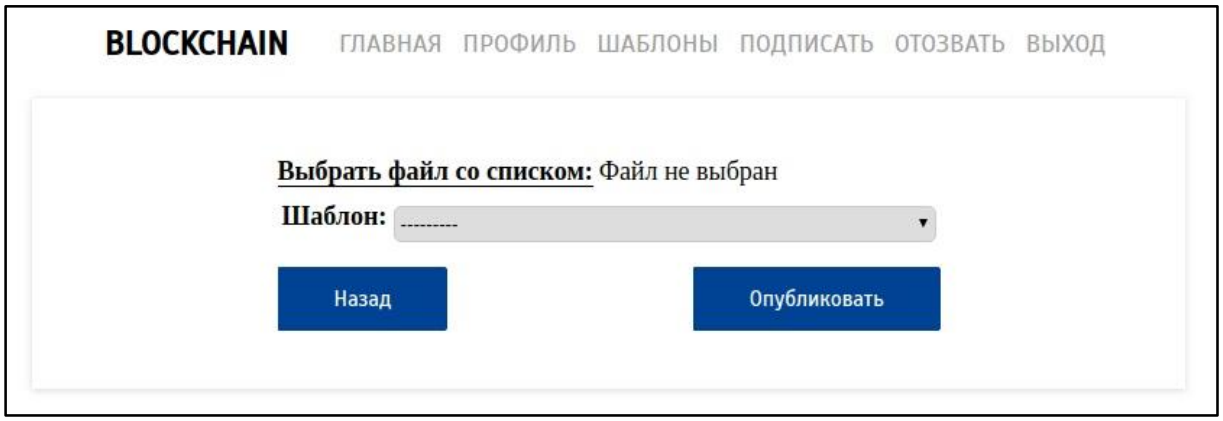

Рис. 5.8. Страница с формой для публикации диплома

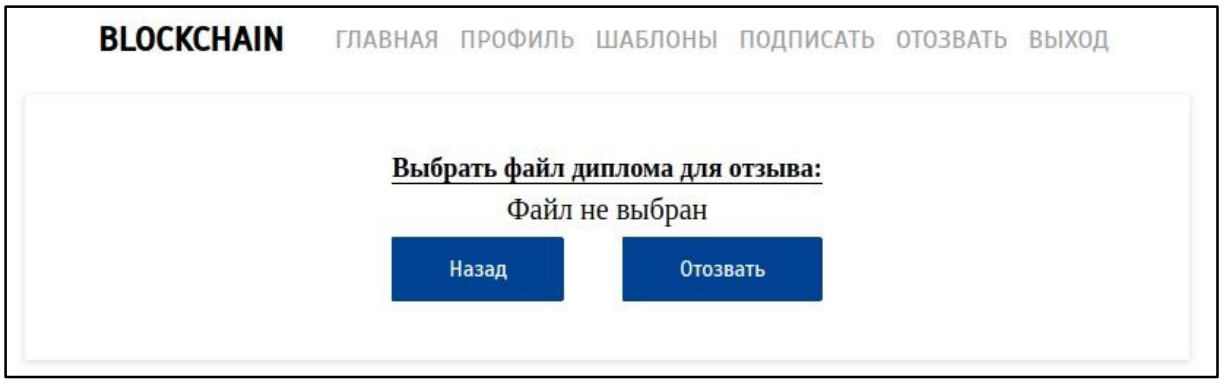

Рис. 5.9. Страница с формой для отзыва диплома

Стандартный интерфейс администратора для добавления, изменения и удаления объектов в/из базы данных предоставляется платформой Django. Для управления аккаунтами сотрудников не потребовалось ничего дополнительно реализовывать.

Выводы по пятому разделу

В данном разделе были реализованы все компоненты системы, включая блокчейн приложение, компоненты для создания, публикации, проверки и отзыва дипломов и внешний интерфейс веб-приложения.

6. ТЕСТИРОВАНИЕ

6.1. Тестирование веб-приложения

Для тестирования разработанного веб-приложения применялось функциональное тестирование. В таблице 6.1 приведен протокол тестирования веб-приложения.

Таблица 6.1. Функциональное тестирование веб-приложения

№	Аспект работы	Действия	Результат	Тест пройден?
1.	Авторизация.	1. Пользователь заходит на страницу авторизации. 2. Вводит логин и пароль. 3. Нажимает кнопку «Войти».	Пользователь авторизуется, ему становятся доступными действия по созданию, просмотру, удалению шаблонов, их публикация и отзыв.	Да
2.	Добавление нового шаблона.	1. Пользователь заходит на страницу со списком шаблонов. 2. Нажимает кнопку «Создать». Заполняет поля и нажимает кнопку «Создать».	Новый шаблон создан, пользователь перенаправляется на страницу со списком дипломов.	Да
3.	Публикация дипломов.	1. Пользователь заходит на страницу публикации дипломов. 2. Прикрепляет файл с получателями. 3. Выбирает шаблон. 3. Нажимает кнопку «Опубликовать».	Дипломы опубликованы, пользователю предлагается скачать архив, содержащий файлы дипломов.	Да
4.	Проверка диплома .	Пользователь прикрепляет файл с правильным дипломом. Нажимает кнопку «проверить».	Пользователя перенаправляется на страницу с дипломом и с результатом проверки «Диплом правильный».	Да

Окончание таблицы 6.1

№	Аспект работы	Действия	Результат	Тест пройден?
5.	Отзыв диплома.	1. Пользователь переходит на страницу отзыва диплома. 2. Прикрепляет файл диплома. Нажимает кнопку «Отозвать».	Диплом отозван.	Да
6.	Проверка отозванного диплома.	1. Пользователь прикрепляет файл с отозванным дипломом. Нажимает кнопку «проверить».	Пользователя перенаправляется на страницу с дипломом и с результатом проверки «Диплом отозван».	Да
7.	Удаление шаблона.	1. Пользователь заходит на страницу со списком шаблонов. 2. Нажимает на ссылку диплома. 3. Нажимает кнопку «Удалить». 2. На новой странице пользователь нажимает кнопку «Да, удалить».	Шаблон удален. Пользователь перенаправляется на страницу со списком дипломов.	Да
8.	Запрет доступа по ссылкам без авторизации.	Пользователь нажимает на кнопку «Выход».	Пользователю не доступны действия с шаблонами и дипломами	Да
9.	Вывод имени файла при прикреплении.	Пользователь прикрепляет файл.	Выводится имя файла.	Да
10.	Работа нескольких пользователей.	К веб-приложению подключаются несколько пользователей.	Веб-приложение работает стабильно.	Да
11.	Обработка ошибки связи с блокчейном.	Пользователь подписывает, проверяет или отзывает диплом.	Пользователя перенаправляет на страницу с сообщением «Нет связи с сервером».	Да

6.2. Тестирование блокчейн приложения

Для тестирования блокчейн приложения, оно было развернуто на двух виртуальных машинах Amazon EC2 со следующими характеристиками:

1) 1 виртуальное ядро (2.5 ГГц);
2) 1 Гб оперативной памяти;
3) 8 Гб дискового пространства (SSD);
4) ОС Ubuntu 18.04 LTS.

Для настройки двух валидаторов в файл genesis.json были добавлены адреса и публичные ключи двух валидаторов. Данный файл представлен в приложении К.

Первая и вторая машины были запущены с параметрами, показанными на рисунке 6.1 и 6.2.

Ошибка в выводе первого валидатора появляется из-за того, что пока не был запущен второй.

Для того, чтобы проверить работу блокчейн приложения, с помощью curl был отправлен запрос на добавление транзакции по протоколу RPC в одно из хранилищ (рисунок 6.3). В итоге транзакция была включена в блок, и два валидатора обновили свои состояния (рисунок 6.4).

```
ubuntu@ip-172-31-50-159:~/blockchain_app$ sudo ./MyDiplomProject -config "./appd
ata/config/config.toml" -empty=false -per_peer "7d36fcf1366df4cc712461d3f5f7dc80
0e5b1fdb@52.91.182.222:26656" -rpc_adr "tcp://0.0.0.0:26657"
E[2020-06-04|15:55:05.557] dialing failed (attempts: 1): dial tcp 52.91.182.222:
26656: connect: connection refused module=pex addr=7d36fcf1366df4cc712461d3f5f7d
c800e5b1fdb@52.91.182.222:26656
E[2020-06-04|15:55:06.871] Error dialing peer                          module=p
2p err="dial tcp 52.91.182.222:26656: connect: connection refused"
I[2020-06-04|15:55:09.195] Executed block                              module=s
tate height=1 validTxs=0 invalidTxs=0
I[2020-06-04|15:55:09.205] Committed state                             module=s
tate height=1 txs=0 appHash=0000000000000000
I[2020-06-04|15:55:10.459] Executed block                              module=s
tate height=2 validTxs=0 invalidTxs=0
I[2020-06-04|15:55:10.468] Committed state                             module=s
tate height=2 txs=0 appHash=0000000000000000
```

Рис. 6.1. Первый запущенный валидатор

```
ubuntu@ip-172-31-59-249:~/blockchain_app$ sudo ./MyDiplomProject -config "./appd
ata/config/config.toml" -empty=false -per_peer "0b0615b0b1248fdfeb7fb48615d405d2
8f8540cf@52.3.253.174:26656" -rpc_adr "tcp://0.0.0.0:26657"
I[2020-06-04|15:55:09.106] Executed block                               module=s
tate height=1 validTxs=0 invalidTxs=0
I[2020-06-04|15:55:09.116] Committed state                              module=s
tate height=1 txs=0 appHash=0000000000000000
I[2020-06-04|15:55:10.512] Executed block                               module=s
tate height=2 validTxs=0 invalidTxs=0
I[2020-06-04|15:55:10.521] Committed state                              module=s
tate height=2 txs=0 appHash=0000000000000000
```

Рис. 6.2. Второй запущенный вадидатор

```
daria@daria-A6GMV:~$ curl -s '54.90.39.144:26657/broadcast_tx_sync?tx="e45365b2-a1d8-4d17-b9a0-a24c
6f2d4e09=6734befce7727e0b30abef52aff08acdcc1535031d26ef147eecdfacb85a4fgd7"'
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "code": 0,
    "data": "",
    "log": "",
    "codespace": "",
    "hash": "328E48360B810E68F5F91541965BB91406C6AAC770D238F6D74E2D2C7519E249"
  }
```

Рис. 6.3. Запрос на добавление информации

```
ubuntu@ip-172-31-50-159: ~/blockchain_app
I[2020-06-04|15:58:07.856] Executed block                               module=s
tate height=3 validTxs=1 invalidTxs=0
I[2020-06-04|15:58:07.865] Committed state                              module=s
tate height=3 txs=1 appHash=0200000000000000
ubuntu@ip-172-31-59-249: ~/blockchain_app
I[2020-06-04|15:58:07.770] Executed block                               module=s
tate height=3 validTxs=1 invalidTxs=0
I[2020-06-04|15:58:07.780] Committed state                              module=s
tate height=3 txs=1 appHash=0200000000000000
```

Рис. 6.4. Состояния валидаторов после запроса

С помощью запроса информации о наличии транзакции по протоколу RPC во второе хранилище было получено подтверждение, доказывающее, что транзакция была добавлена в блок в обоих валидаторах (рисунок 6.5). Данные приходят в кодировке base64, после раскодирования получились те же данные, что были отправлены ранее.

```
daria@daria-A6GMV:~$ url -s '34.201.165.93:26657/abci_query?data="e45365b2-a1d8-4d17-b9a0-a24c6f2d
4e09"'
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "response": {
      "code": 0,
      "log": "exists",
      "info": "",
      "index": "0",
      "key": "ZTQ1MzY1YjItYTFkOC00ZDE3LWI5YTAtYTI0YzZmMmQ0ZTA5",
      "value": "NjczNGJlZmNlNzcyN2UwYjMwYWJlZjUyYWZmMDhhY2RjYzE1MzUwMzFkMjZlZjE0N2VlY2RmYWNiODVhNGZ
hZDc=",
      "proof": null,
      "height": "8",
      "codespace": ""
    }
  }
```

Рис. 6.5. Запрос информации

Затем с помощью веб-приложения были подписаны двадцать дипломов и проверено состояние узлов. В итоге выяснилось, что дипломы действительно фиксируются в блокчейне (рисунок 6.6 и 6.7).

```
ubuntu@ip-172-31-59-249: ~/blockchain_app
I[2020-06-04|16:21:51.322] Executed block                                  module=s
tate height=7 validTxs=1 invalidTxs=0
I[2020-06-04|16:21:51.335] Committed state                                 module=s
tate height=7 txs=1 appHash=0600000000000000
I[2020-06-04|16:21:52.543] Executed block                                  module=s
tate height=8 validTxs=3 invalidTxs=0
I[2020-06-04|16:21:52.552] Committed state                                 module=s
tate height=8 txs=3 appHash=0C00000000000000
I[2020-06-04|16:21:53.928] Executed block                                  module=s
tate height=9 validTxs=5 invalidTxs=0
I[2020-06-04|16:21:53.938] Committed state                                 module=s
tate height=9 txs=5 appHash=1600000000000000
I[2020-06-04|16:21:55.396] Executed block                                  module=s
tate height=10 validTxs=4 invalidTxs=0
I[2020-06-04|16:21:55.406] Committed state                                 module=s
tate height=10 txs=4 appHash=1E00000000000000
I[2020-06-04|16:21:56.724] Executed block                                  module=s
tate height=11 validTxs=4 invalidTxs=0
I[2020-06-04|16:21:56.736] Committed state                                 module=s
tate height=11 txs=4 appHash=2600000000000000
I[2020-06-04|16:21:58.071] Executed block                                  module=s
tate height=12 validTxs=3 invalidTxs=0
I[2020-06-04|16:21:58.081] Committed state                                 module=s
tate height=12 txs=3 appHash=2C00000000000000
I[2020-06-04|16:21:59.425] Executed block                                  module=s
tate height=13 validTxs=0 invalidTxs=0
I[2020-06-04|16:21:59.437] Committed state                                 module=s
tate height=13 txs=0 appHash=2C00000000000000
```

Рис. 6.6. Вывод первого валидатора после публикации дипломов

```
ubuntu@ip-172-31-50-159: ~/blockchain_app
tate height=7 validTxs=1 invalidTxs=0
I[2020-06-04|16:21:51.188] Committed state                          module=s
tate height=7 txs=1 appHash=0600000000000000
I[2020-06-04|16:21:52.546] Executed block                           module=s
tate height=8 validTxs=3 invalidTxs=0
I[2020-06-04|16:21:52.556] Committed state                          module=s
tate height=8 txs=3 appHash=0C00000000000000
I[2020-06-04|16:21:53.992] Executed block                           module=s
tate height=9 validTxs=5 invalidTxs=0
I[2020-06-04|16:21:54.002] Committed state                          module=s
tate height=9 txs=5 appHash=1600000000000000
I[2020-06-04|16:21:55.394] Executed block                           module=s
tate height=10 validTxs=4 invalidTxs=0
I[2020-06-04|16:21:55.404] Committed state                          module=s
tate height=10 txs=4 appHash=1E00000000000000
I[2020-06-04|16:21:56.716] Executed block                           module=s
tate height=11 validTxs=4 invalidTxs=0
I[2020-06-04|16:21:56.725] Committed state                          module=s
tate height=11 txs=4 appHash=2600000000000000
I[2020-06-04|16:21:58.064] Executed block                           module=s
tate height=12 validTxs=3 invalidTxs=0
I[2020-06-04|16:21:58.073] Committed state                          module=s
tate height=12 txs=3 appHash=2C00000000000000
I[2020-06-04|16:21:59.429] Executed block                           module=s
tate height=13 validTxs=0 invalidTxs=0
```

Рис. 6.7. Вывод второго валидатора после публикации дипломов

Выводы по шестому разделу

В данном разделе был составлен протокол тестирования разработанного веб-приложения, было проведено тестирование по данному протоколу. Все тесты были пройдены. Также была проверена работа блокчейн-приложения.

ЗАКЛЮЧЕНИЕ

В данной работе было разработано веб-приложение для проверки подлинности дипломов с использованием технологии блокчейн.

При этом были решены задачи, приведенные ниже.

1. Проведен обзор литературы и существующих аналогов, а также готовых решений для создания приложения для проверки подлинности дипломов с использованием технологии блокчейн.

2. Определены требования к веб-приложению и разработать варианты использования.

3. Определены требования к блокчейн приложению.

4. Разработано блокчейн приложение для хранения дипломов.

5. Разработано веб-приложение для проверки подлинности дипломов с использованием технологии блокчейн.

6. Проведено тестирование блокчейн приложения.

7. Проведено тестирование веб-приложения.

Планируется дальнейшее развитие проекта, где планируется реализация функций, приведенных ниже.

1. Добавление в веб-приложение управления партиями дипломов.

2. Добавление в веб-приложение данных о получателях.

3. Развертывание блокчейн приложения на большем числе машин.

ЛИТЕРАТУРА

1. Zhu S., Zhu C., Wang W. A new image encryption algorithm based on chaos and secure hash SHA-256 // Entropy. 2018.

2. Консенсус - Документация [Электронный ресурс]. URL: https://exonum.com/ru/doc/architecture/consensus/ (дата обращения: 19.03.2020).

3. Zheng Z., Xie S., Dai H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends // Proc. - 2017 IEEE 6th Int. Congr. Big Data, BigData Congr. 2017. 2017. С. 557–564.

4. Руководящий документ. Решение председателя Гостехкомиссии России от 30 марта 1992 г. - ФСТЭК России [Электронный ресурс]. URL: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/114-spetsialnye-normativnye-dokumenty/386-rukovodyashchij-dokument-reshenie-predsedatelya-gostekhkomissii-rossii-ot-30-marta-1992-g3 (дата обращения: 19.03.2020).

5. Gray J. Transaction Concept: Virtues and Limitations. // Very Large Data Bases, Int. Conf. Very Large Data Bases. 1981. № 1. С. 144–154.

6. Lin I.C., Liao T.C. A survey of blockchain security issues and challenges // Int. J. Netw. Secur. 2017. Т. 19, № 5. С. 653–659.

7. Национальная программа «Цифровая экономика Российской Федерации». Утверждена протоколом заседания президиума Совета при Президенте Российской Федерации по стратегическому развитию и национальным проектам от 4 июня 2019 г. № 7.

8. Novikov S. и др. Digital registry of professional competences of the population drawing on distributed registries and smart contracts technologies // Bus. Informatics. National Research University, Higher School of Economics (HSE), 2018. Т. 2018, № 4. С. 43–53.

9. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System.

10. Shamsutdinova T.M. Application of the blockchain technology for digital diplomas: problems and prospects // Open Educ. Plekhanov Russian University of Economics (PRUE), 2019. Т. 22, № 6. С. 51–58.

11. Gupta J., Nath S. SkillCheck: An Incentive-based Certification System using Blockchains.

12. Funk E., Riddell J., Ankel F. Blockchain technology: A data framework to improve validity, trust, and accountability of information exchange in health professions education // Acad. Med. 2018. Т. 93, № 12. С. 1791–1794.

13. Lizcano D., Lara J.A., White B. Blockchain-based approach to create a model of trust in open and ubiquitous higher education // J. Comput. High. Educ. Springer, 2020. Т. 32, № 1. С. 109–134.

14. Srivastava A., Bhattacharya P., Singh A. A Distributed Credit Transfer Educational Framework based on Blockchain // Proc. - 2018 2nd Int. Conf. Adv. Comput. Control Commun. Technol. IAC3T 2018. IEEE, 2019. С. 54–59.

15. Gresch J., Rodrigues B., Scheid E. The proposal of a blockchain-based architecture for transparent certificate handling // Lecture Notes in Business Information Processing. Springer Verlag, 2019. Т. 339. С. 185–196.

16. Block.Co Design Principles & Future Directions - Block.co - Medium [Электронный ресурс]. URL: https://medium.com/@Blockdotco/block-co-design-principles-future-directions-df60fcb5bd2c (дата обращения: 18.03.2020).

17. Биткойн - P2P деньги с Открытым кодом [Электронный ресурс]. URL: https://bitcoin.org/ru/ (дата обращения: 17.05.2020).

18. Проверка диплома выпускника в блокчейне [Электронный ресурс]. URL: http://www.fa.ru/checkdiploma_blockchain/Pages/Home.aspx (дата обращения: 19.03.2020).

19. Massachusetts Institute of Technology - Verification Portal [Электронный ресурс]. URL: https://credentials.mit.edu/ (дата обращения: 19.03.2020).

20. Blockcerts : The Open Standard for Blockchain Credentials [Электронный ресурс]. URL: https://www.blockcerts.org/ (дата обращения: 18.03.2020).

21. Tendermint Core | Tendermint [Электронный ресурс]. URL: https://tendermint.com/core/ (дата обращения: 17.05.2020).

22. Protocol Buffers | Google Developers [Электронный ресурс]. URL: https://developers.google.com/protocol-buffers (дата обращения: 17.05.2020).

23. The Go Programming Language [Электронный ресурс]. URL: https://golang.org/ (дата обращения: 17.05.2020).

24. Buchman E. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. 2016.

25. Exonum Documentation [Электронный ресурс]. URL: https://exonum.com/doc/version/latest/ (дата обращения: 19.03.2020).

26. Rust Programming Language [Электронный ресурс]. URL: https://www.rust-lang.org/ (дата обращения: 17.05.2020).

27. Java Programming Language [Электронный ресурс]. URL: https://docs.oracle.com/javase/7/docs/technotes/guides/language/ (дата обращения: 17.05.2020).

28. Castro M., Liskov B. Practical Byzantine Fault Tolerance // OSDI ’99 Proc. Third Symp. Oper. Syst. Des. Implement. SE - OSDI ’99. 1999. № February. С. 173–186.

29. Tendermint Documentation [Электронный ресурс]. URL: https://docs.tendermint.com/master/ (дата обращения: 19.03.2020).

30. Ethereum.org [Электронный ресурс]. URL: https://ethereum.org/ru/ (дата обращения: 21.05.2020).

31. Digital Diploma debuts at MIT | MIT News [Электронный ресурс]. URL: http://news.mit.edu/2017/mit-debuts-secure-digital-diploma-using-bitcoin-blockchain-technology-1017 (дата обращения: 18.03.2020).

32. University of Melbourne to issue recipient-owned blockchain records [Электронный ресурс]. URL: https://about.unimelb.edu.au/newsroom/news/2017/october/university-of-melbourne-to-issue-recipient-owned-blockchain-records (дата обращения: 19.03.2020).

33. Malta's Government Is Putting Academic Certificates on a Blockchain - CoinDesk [Электронный ресурс]. URL: https://www.coindesk.com/maltas-government-putting-academic-certificates-blockchain (дата обращения: 18.03.2020).

34. Bashir I. Mastering Blockchain Second Edition // Packt Publishing. 2018.

35. Mastering Bitcoin, 2nd Edition [Book] [Электронный ресурс]. URL: https://www.oreilly.com/library/view/mastering-bitcoin-2nd/9781491954379/ (дата обращения: 23.05.2020).

36. On Public and Private Blockchains | Ethereum Foundation Blog [Электронный ресурс]. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ (дата обращения: 25.03.2020).

37. Lamport L., Shostak R., Pease M. The Byzantine Generals Problem. 1982.

38. White Paper Ethereum [Электронный ресурс]. URL: https://github.com/ethereum/wiki/wiki/White-Paper (дата обращения: 11.03.2020).

39. Zheng Z., Xie S., Dai H.N. Blockchain challenges and opportunities: A survey // Int. J. Web Grid Serv. 2018. Т. 14, № 4. С. 352–375.

40. Bentov I., Lee C., Mizrahi A. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake // Cryptol. ePrint Arch. 2014.

41. BitShares 4.0 | OSS Solutions for finance and business management based on blockchain technology [Электронный ресурс]. URL: https://bitshares.org/ (дата обращения: 22.05.2020).

42. Герасимов И.Ю., Чижов И.В. Алгоритм консенсуса платформы Tendermint и механизм Proof Of Lock Change. 2019. Т. 7, № 6.

43. Python.org [Электронный ресурс]. URL: https://www.python.org/ (дата обращения: 23.05.2020).

44. What is Tendermint? | Tendermint [Электронный ресурс]. URL: https://docs.tendermint.com/master/introduction/what-is-tendermint.html (дата обращения: 22.05.2020).

45. JSON [Электронный ресурс]. URL: https://www.json.org/json-ru.html (дата обращения: 27.05.2020).

46. Open Badges v2.0 [Электронный ресурс]. URL: https://www.imsglobal.org/sites/default/files/Badges/OBv2p0Final/index.html (дата обращения: 27.05.2020).

47. JSON-LD - JSON for Linking Data [Электронный ресурс]. URL: https://json-ld.org/ (дата обращения: 27.05.2020).

48. Django documentation [Электронный ресурс]. URL: https://docs.djangoproject.com/en/3.0/ (дата обращения: 24.05.2020).

49. SQLite Home Page [Электронный ресурс]. URL: https://www.sqlite.org/index.html (дата обращения: 06.06.2020).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А – Код в файле state.go

Листинг. А.1. Код в файле state.go

```
package diplomstore

import (
	"encoding/json"
	"github.com/tendermint/tendermint/version"
	dbm "github.com/tendermint/tm-db"
)

var (
	stateKey        = []byte("stateKey")
	kvPairPrefixKey = []byte("kvPairKey:")

	ProtocolVersion version.Protocol = 0x1
)

type State struct {
	db      dbm.DB
	Size    int64  `json:"size"`
	Height  int64  `json:"height"`
	AppHash []byte `json:"app_hash"`
}

func loadState(db dbm.DB) State {
	var state State
	state.db = db
	stateBytes, err := db.Get(stateKey)
	if err != nil {
		panic(err)
	}
	if len(stateBytes) == 0 {
		return state
	}
	err = json.Unmarshal(stateBytes, &state)
	if err != nil {
		panic(err)
	}
	return state
}
```

```
func saveState(state State) {
	stateBytes, err := json.Marshal(state)
	if err != nil {
		panic(err)
	}
	state.db.Set(stateKey, stateBytes)
}
```

ПРИЛОЖЕНИЕ Б – Код метода Query

Листинг. Б.1. Код метода Query

```
func (app *Application) Query(reqQuery types.RequestQuery) (resQuery
types.ResponseQuery) {
	if reqQuery.Prove {
		value, err := app.state.db.Get(prefixKey(reqQuery.Data))
		if err != nil {
			panic(err)
		}
		if value == nil {
			resQuery.Log = "does not exist"
		} else if string(value) =="deleted" {
			resQuery.Log = "recalled"
		} else {
			resQuery.Log = "exists"
		}
		resQuery.Index = -1 // TODO make Proof return index
		resQuery.Key = reqQuery.Data
		resQuery.Value = value
		resQuery.Height = app.state.Height

		return
	}

	resQuery.Key = reqQuery.Data
	value, err := app.state.db.Get(prefixKey(reqQuery.Data))
	if err != nil {
		panic(err)
	}
	if value == nil {
		resQuery.Log = "does not exist"
	} else if string(value) =="deleted" {
		resQuery.Log = "recalled"
	} else {
		resQuery.Log = "exists"
	}
	resQuery.Value = value
	resQuery.Height = app.state.Height

	return resQuery
}
```

ПРИЛОЖЕНИЕ В – Код метода newTendermint

Листинг В.1. Код метода newTendermint

```
func newTendermint(app abci.Application, configFile string) (*nm.Node, error)
{
	// read config
	config := cfg.DefaultConfig()
	config.RootDir = filepath.Dir(filepath.Dir(configFile))
	viper.SetConfigFile(configFile)
	if err := viper.ReadInConfig(); err != nil {
		return nil, errors.Wrap(err, "viper failed to read config file")
	}
	if err := viper.Unmarshal(config); err != nil {
		return nil, errors.Wrap(err, "viper failed to unmarshal config")
	}
	if err := config.ValidateBasic(); err != nil {
		return nil, errors.Wrap(err, "config is invalid")
	}
	config.TxIndex.IndexAllKeys = true
	config.Consensus.CreateEmptyBlocks = no_empty
	config.P2P.PersistentPeers = perpeer
	config.RPC.ListenAddress = rpcadress
	if no_empty == false {
		seconds, err := time.ParseDuration("30s")
		if err != nil {
			return nil, errors.Wrap(err, "failed to parse duration")
		}
		config.Consensus.CreateEmptyBlocksInterval = seconds
	}
	//config.P2P.AddrBook =
	// create logger
	logger := log.NewTMLogger(log.NewSyncWriter(os.Stdout))
	var err error
	logger,	err	=	tmflags.ParseLogLevel(config.LogLevel,	logger,
cfg.DefaultLogLevel())
	if err != nil {
		return nil, errors.Wrap(err, "failed to parse log level")
	}
	// read private validator
	pv := privval.LoadFilePV(
		config.PrivValidatorKeyFile(),
		config.PrivValidatorStateFile(),
```

```
	)
	// read node key
	nodeKey, err := p2p.LoadNodeKey(config.NodeKeyFile())
	if err != nil {
		return nil, errors.Wrap(err, "failed to load node's key")
	}
	// create node
	node, err := nm.NewNode(
		config,
		pv,
		nodeKey,
		proxy.NewLocalClientCreator(app),
		nm.DefaultGenesisDocProviderFunc(config),
		nm.DefaultDBProvider,
		nm.DefaultMetricsProvider(config.Instrumentation),
		logger)
	if err != nil {
		return nil, errors.Wrap(err, "failed to create new Tendermint
node")
	}
	return node, nil
}
```

ПРИЛОЖЕНИЕ Г – Код классов Profile и Template

Листинг Г.1. Код классов Profile и Template

```
class Template(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    diplom_name =  models.CharField(max_length=200, default="Имя диплома")
    diplom_description =  models.CharField(max_length=200, de-
fault="Описание диплома")
    organization_name = models.CharField(max_length=200, default="имя
организации")
    organization_url = models.CharField(max_length=200, de-
fault="https://www.susu.ru/")
    organization_email = models.CharField(max_length=200, de-
fault="info@susu.ru")
    logo_image = models.ImageField(upload_to='logo/%Y/%m/%d', blank=True)
    seal_image = models.ImageField(upload_to='diplom/%Y/%m/%d', blank=True)
    json_file = models.FileField(upload_to='json/', blank=True)
    created_data = models.DateTimeField(auto_now_add=True, ver-
bose_name='Создан')

    def __str__(self):
        return '{0} ({1})'.format(self.diplom_name, self.created_data)

    class Meta:
        verbose_name = 'Шаблон'
        verbose_name_plural = 'Шаблоны'
        ordering = ['-created_data']

class Profile(models.Model):
    user_id=models.OneToOneField(User, on_delete=models.CASCADE)
    job = models.CharField(max_length=200, default="Сотрудник")
    patronymic = models.CharField(max_length=200, default="Ивановна")
    signature_image = models.ImageField(upload_to='sign_image/%Y/%m/%d',
blank=True)

    @receiver(post_save, sender=User)
    def create_user_profile(sender, instance, created, **kwargs):
        if created:
            Profile.objects.create(user=instance)

    @receiver(post_save, sender=User)
    def save_user_profile(sender, instance, **kwargs):
```

```
        instance.profile.save()

    def __str__(self):
        return '{0} {1} {2} ({3})'.for-
mat(self.user_id.last_name,self.user_id.first_name,self.patronymic,
self.job)

    class Meta:
        verbose_name = 'Профиль'
        verbose_name_plural = 'Профили'
```

ПРИЛОЖЕНИЕ Д – Диплом в JSON

Листинг Д.1. Диплом в JSON


```
{
  "@context": [
    "https://w3id.org/openbadges/v2",
    {
      "displayHtml": {
        "@id": "schema:description"
      }
    }
  ],
  "type": "Assertion",
  "displayHtml": "no",
  "issuedOn": "2020-06-02T01:57:57.134438+00:00",
  "id": "urn:uuid:6df1a84f-de56-469d-bd43-7fa921463fd6",
  "recipient": {
    "type": "email",
    "identity": "sha256$1658e9287b9eb9efb9453097e062d4541a4a72b9bb5ab-
feb134fd2a8cf7e3891",
    "hashed": true,
    "salt": "d9097bfd470d4018986d48241463e1ea"
  },
  "recipientProfile": {
    "type": [
      "RecipientProfile",
      "Extension"
    ],
    "name": "Снегирева Дарья Алексеевна"
  },
  "badge": {
    "type": "BadgeClass",
    "id": "urn:uuid:65c3c980-40fd-4bf7-90df-1dca45581e35",
    "name": "Диплом бакалавра",
    "description": "Присвоена квалификация (степень) Бакалавр по направле-
нию подготовки 090304 Программная инженерия",
    "image": "",
    "issuer": {
      "id": "urn:uuid:9f1fd873-1d03-48f3-982e-fe7fe806379a",
      "type": "Profile",
      "name": "Южно-Уральский государственный университет",
      "url": "https://www.susu.ru/",
```

```
      "email": "info@susu.ru",
      "image": ""
    },
    "criteria": {
      "narrative": "k1"
    },
    "signatureLines": [
      {
        "type": [
          "SignatureLine",
          "Extension"
        ],
        "jobTitle": "Администратор",
        "image": "",
        "name": "Снегирева Дарья Алексеевна"
      }
    ]
  },
  "signature": {
    "type": [
      "Proof",
      "Extension"
    ],
    "targetHash":
"1d940d2d4fd66b7eaae063182128f0b7ecd22da3c7596e7987ddbb8afd15ed48"
  }
}
```

ПРИЛОЖЕНИЕ Е – Код компонентов веб-приложения

Листинг Е.1. Код функции create_template

```
@login_required
def create_template(request):
    if request.method == 'POST':
        form = TemplateCreateForm(request.POST, request.FILES)
        if form.is_valid():
            user_id=request.user.id
            diplom_name = form.cleaned_data.get("diplom_name")
            diplom_descr= form.cleaned_data.get("diplom_description")
            seal_image                                                    =
handle_upload_file_image(request.FILES['seal_image'], user_id)
            org_name = form.cleaned_data.get("organization_name")
            org_url = form.cleaned_data.get("organization_url")
            org_email = form.cleaned_data.get("organization_email")
            logo_image                                                    =
handle_upload_file_image(request.FILES['logo_image'], user_id)
            user_jobtitle = request.user.profile.job
            user_name     =     request.user.last_name         +"     "     +
request.user.first_name + " " + request.user.profile.patronymic
            signature_image = request.user.profile.signature_image.path
            template = TemplateJson()
            template.diplom_name = diplom_name
            template.diplom_descr = diplom_descr
            template.seal_image = seal_image
            template.org_name = org_name
            template.org_url = org_url
            template.org_email = org_email
            template.logo_image = logo_image
            template.user_jobtitle = user_jobtitle
            template.user_name = user_name
            template.signature_image= signature_image
            ditemp = template.create_diplom_template()
            json_data = load_file_json(ditemp, user_id)
            bd_temp = Template(user_id=request.user,diplom_name=diplom_name,
diplom_description=diplom_descr,                organization_name=org_name,
organization_url=org_url,                     organization_email=org_email,
seal_image=request.FILES['seal_image'],
logo_image=request.FILES['logo_image'],json_file=json_data)
            bd_temp.save()
            del(logo_image)
```

```
            del(seal_image)
            return HttpResponseRedirect(reverse('templates'))
    else:
        form = TemplateCreateForm()
    return render(
        request,
        'create_template.html',
        {'form': form}
    )
```

Листинг Е.2. Код функции sign_template

```
@login_required
def sign_template(request):
    if request.method == 'POST':
        form = TemplateSignForm( request.user, request.POST, request.FILES)
        if form.is_valid():
            user_id = request.user.id
            if connection_rpc_on() == False:
                message = 'Нет связи с сервером'
                return render(
                    request,
                    'info_page.html',
                    {'message': message}
                )
            template = request.POST['user_template']
            diplom_template = Template.objects.get(pk=template)
            dest = handle_upload_file(request.FILES['roster'], user_id)
            path = './media'
            index = 0
            s = diplom_template.json_file.name
            s = s[:index] + s[index+1:]
            with open(path + s) as f:
                json_data = f.read()
                print(type(json_data))
            diploms, messages = sign_diploms(dest, json_data)
            response = HttpResponse(content_type='application/zip')
            z = zipfile.ZipFile(response, 'w')
            for diplom in diploms:
                diplom_id = diploms[diplom]['id'].replace(PREFIX_UID, "")
                rec_name = diploms[diplom]['recipientProfile']['name']
```

```
                dest_json,                  file_json_name                  =
handle_file_json(diploms[diplom],diplom_id, rec_name)
                z.write(dest_json,file_json_name )
                os.remove(dest_json)
            os.remove(dest)
            response['Content-Disposition']         =         f'attachment;
filename={ZIPFILE_NAME}'
            return response
    else:
        form = TemplateSignForm(request.user)
    return render(
        request,
        'sign_template.html',
        {'form': form}
    )
```

Листинг Е.3. Код функции recall_template

```
@login_required
def recall_template(request):
    if request.method == 'POST':
        form = TemplateRecallForm(request.POST, request.FILES)
        if form.is_valid():
            if connection_rpc_on() == False:
                message = 'Нет связи с сервером'
                return render(
                    request,
                    'info_page.html',
                    {'message': message}
                )
            diplom = handle_upload_json(request.FILES['diplom_file'])
            diplom_id = diplom['id'].replace(PREFIX_UID, "")
            result, error = broadcast_tx(diplom_id, 'deleted')
            if error == True:
                printed_res = result
            else:
                printed_res = 'Диплом успешно отозван'
            return render(
                request,
                'info_page.html',
                {'message': printed_res}
            )
```

```
    else:
        form = TemplateRecallForm()
    return render(
        request,
        'recall_diplom.html',
        {'form': form}
    )
```

ПРИЛОЖЕНИЕ Ж – Код функции index

Листинг Ж.1. Код функции index

```
def index(request):
    if request.method == 'POST':
        if connection_rpc_on() == False:
            message = 'Нет связи с сервером'
            return render(
                request,
                'info_page.html',
                {'message': message}
            )
        user_id = request.user.id
        file = request.FILES['filesjson']
        json_byte_file = file.read()
        json_file = json_byte_file.decode()
        m1, m2 = verify_diplom(json_file)
        if m1 == 'exist' and m2 =='same':
            verify_response = 1
        elif m1 == 'recalled':
            verify_response = 2
        else:
            verify_response = 0
        award     =     get_formatted_award_and_verification_info(json_file,
verify_response, user_id)
        return render(
            request,
            'award.html',
            award
        )
    return render(
        request,
        'index.html'
    )
```

ПРИЛОЖЕНИЕ И – Страница с проверенным дипломом

BLOCKCHAIN

ГЛАВНАЯ ПРОФИЛЬ ШАБЛОНЫ ПОДПИСАТЬ ОТОЗВАТЬ ВЫХОД

Южно-Уральский государственный университет
Национальный исследовательский университет
1943

Снегирева Дарья Алексеевна

Диплом бакалавра

Южно-Уральский государственный университет

Присвоена квалификация (степень) Бакалавр по направлению подготовки 090304 Программная инженерия

Выдан: 04.06.2020

Подписал: Сотрудник Иванова Николь Ивановна

Южно-Уральский государственный университет подписал в цифровом виде диплом и выпустил его в блокчейне .

Диплом правильный

Сайт: https://www.susu.ru/ **Почта:** info@susu.ru

Проверить другой диплом

Рис. И.1. Страница с проверенным дипломом

ПРИЛОЖЕНИЕ К– Код в файле genesis.json

Листинг К.1. Файл genesis.json

```
{
  "genesis_time": "2020-06-01T12:06:14.754378055Z",
  "chain_id": "test-chain-4SJXx5",
  "consensus_params": {
    "block": {
      "max_bytes": "22020096",
      "max_gas": "-1",
      "time_iota_ms": "1000"
    },
    "evidence": {
      "max_age_num_blocks": "100000",
      "max_age_duration": "172800000000000"
    },
    "validator": {
      "pub_key_types": [
        "ed25519"
      ]
    }
  },
  "validators": [
    {
      "address": "C7943A9DD27E46969019E45BCD660DB82E766013",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "I7UeMLVmJtATQ+mlXKuntvQLjf+RcCKu/XpHUM6dlAM="
      },
      "power": "10",
      "name": ""
    },
    {
      "address": "0629CA5AF78A83F43FD1762B338F2FE9E35099DA",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "LA8pz3//kok9HCky9zzthwrNumMV4m5kB88quKq0JH4="
    },
      "power": "10",
      "name": ""
    }
  ],
```

```
  "app_hash": ""
}
```