**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ОТЧЕТ

по производственной практике

«Реализация блокчейн модуля системы электронного документооборота»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Студент | | | | |  | |  |  | | Гриднев А.Д. | | |  |
|  |  | | | | |  | | подпись, дата |  | | фамилия, инициалы | | |  |
|  | Группа |  | М-АС-17 | | | |  | | |  | |  |
|  |  | | |  |  | | | |  |  | | | |  |
|  | Руководитель | | |  |  | | | |  |  | | | |  |
|  | доц. | | |  |  | | | |  | Овчинников В.В. | | | |  |
|  | ученая степень, звание | | |  | подпись, дата | | | |  | фамилия, инициалы | | | |  |

Липецк 2018г.

**Аннотация**

С. 31. Ил. 32. Литература 7 назв.

Настоящий отчет является пояснительной запиской по производственной практике на тему «Реализация блокчейн модуля системы электронного документооборота».

В ходе работы была реализована блокчейн платформа, направленная на хранение информации о договорах различных форматов. Реализованный проект состоит из двух частей. Back-end часть представляет собой непосредственно блокчейн платформу, тогда как front-end часть служит для визуального взаимодействия с реализованной платформой. Разработанный модуль в совокупности направлен на хранение информации, подтверждающей факт хранения подлинного документа в системе, а также подтверждение подлинности имеющегося у пользователя документа.

**Содержание**

[Введение 4](#_Toc518398737)

[1 Информация о компании Фьюжнсофт 5](#_Toc518398738)

[2 Техническое задание 10](#_Toc518398739)

[2.1 Постановка задачи 10](#_Toc518398740)

[2.2 Требования к системе 11](#_Toc518398741)

[2.2.1 Функциональные требования 11](#_Toc518398742)

[2.2.2 Нефункциональные требования 11](#_Toc518398743)

[2.2.3 Варианты использования 12](#_Toc518398744)

[3 Архитектура системы 13](#_Toc518398745)

[3.1 Структурные диаграммы 13](#_Toc518398746)

[3.2 Диаграмма деятельности 15](#_Toc518398747)

[3.3 Диаграмма развертывания 16](#_Toc518398748)

[5 Программные средства 16](#_Toc518398749)

[6 Результаты работы 18](#_Toc518398750)

[6.1 Back-end часть системы 18](#_Toc518398751)

[6.2 Front-end часть системы 23](#_Toc518398752)

[Вывод 29](#_Toc518398753)

[Список источников 30](#_Toc518398754)

# **Введение**

Полностью одноранговое устройство системы электронных транзакций позволяет совершать их между участниками напрямую, минуя любые финансовые институты. Частично, эту задачу решает использование цифровых подписей, но необходимость доверенного лица для контроля за двойной тратой лишает этот подход основных преимуществ. Предлагается децентрализованное решение проблемы двойной траты с использованием одноранговой (пиринговой) сети. Сеть ставит метки времени на транзакции, соединяя их в цепочку доказательств проделанной работы на основе хэширования.

Сформированные таким образом записи невозможно изменить, не выполнив заново всего объема вычислений. Самая длинная версия цепочки служит не только подтверждением очередности событий, но и доказывает, что над ней произвел работу самый большой вычислительный сегмент сети. До тех пор, пока большая часть вычислительных мощностей контролируется узлами, не объединенными с целью атаковать сеть, они будут генерировать самую длинную цепочку, опережая любых злоумышленников.

Устройство самой сети очень простое: сообщения рассылаются на основе принципа «наименьших затрат», а узлы могут покидать сеть и снова подключаться к ней в любой момент, принимая самую длинную версию цепочки для восстановления пропущенной истории транзакций.

1. **Информация о компании Фьюжнсофт**

Компания ООО «Фьюжнсофт» специализируется на создании программного обеспечения различной степени сложности. Она предлагает услуги по созданию web-приложений, мобильных приложений, а также комбинированных систем, удовлетворяющих требованиям заказчика.

Стоит отметить, что помимо создания систем с нуля, организация занимается сопровождением и поддержкой уже существующих и функционирующих систем.

Деятельность компании распространяется как на бизнес-клиентов, так и на разработчиков программного обеспечения. Организацией осуществляется реализация систем, ориентированных на:

* удобную работу бизнес-клиентов с существующими сервисами и услугами в едином облаке в любой момент времени из любой точки мира;
* эффективное решение трудовых задач сотрудниками организации в единой информационной среде из любой точки мира;
* контроль и анализ состояния отдельных процессов / бизнеса в целом из любой точки мира;
* реорганизацию и повышение эффективности бизнес-процессов.

Для того, чтобы воспользоваться услугами организации, необходимо перейти на сайт компании и оставить заявку, кратко объясняющую идею приложения или услугу, в которой нуждается клиент.

1. **Краткие теоретические сведения**
   1. **Транзакции блокчейна**

Определим электронную единицу блокчейна как последовательность цифровых подписей, представленную на рисунке 1. Очередной владелец отправляет транзакцию следующему, подписывая хэш предыдущей транзакции и публичный ключ будущего владельца и присоединяя эту информацию к текущей транзакции. Получатель может проверить каждую подпись, чтобы подтвердить корректность всей цепочки владельцев.

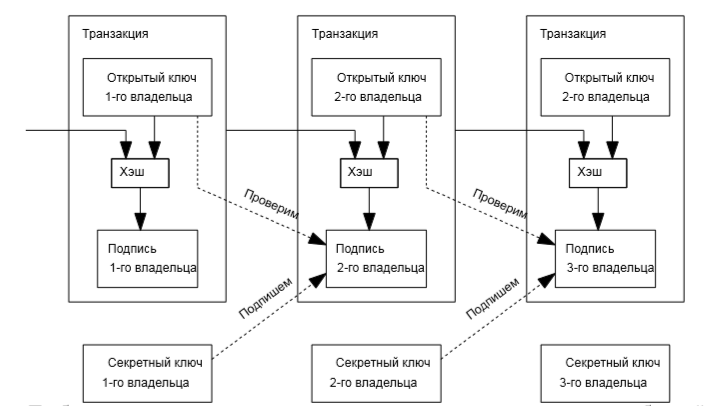


Рисунок 1 – Цепочка транзакций

Адресат должен знать, что никто из предыдущих владельцев не подписал транзакцию, предшествующую по времени той, что находится в цепочке, отправленной ему транзакции. Для наших целей лишь первая транзакция из нескольких является истинной, поэтому мы не должны беспокоиться о поздних попытках двойной траты. В централизованной модели эмитент знал обо всех транзакциях и решал, в каком порядке они идут. Чтобы избавить схему от посредника, участникам необходимо открыто публиковать транзакции, а также уметь приходить к согласию относительно единого порядка их следования [1].

* 1. **Сервер меток времени**

Начнем описание нашего решения с сервера меток времени, принцип работы которого проиллюстрирован на рисунке 2. Его работа заключается в хэшировании блока данных, на который нужно поставить метку, и открытой публикации этого хэша [2]. Метка времени показывает, что в данный момент конкретные данные существовали и потому попали в хэш блока. Каждый хэш включает в себя предыдущую метку: так выстраивается цепь, где очередное звено укрепляет все предыдущие.

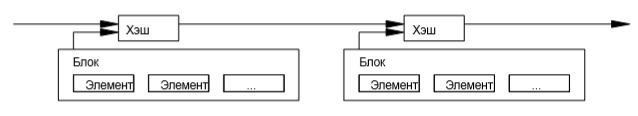


Рисунок 2 – Сервер меток времени

* 1. **Доказательство работы**

Чтобы реализовать распределенный одноранговый сервер меток времени, используется схема «доказательства работы», подобная системе Hashcash Адама Бека, которая представлена на рисунке 3 [3]. Суть заключается в поиске такого значения, чей хэш (например, SHA-256) начинался бы с некоторого числа нулевых битов. Требуется выполнить объем работы, экспоненциально зависящий от числа нулей, но для проверки найденного значения достаточно вычислить лишь один хэш. В нашем сервере меток времени поиск значения с нужным хэшем происходит путем перебора значения итерируемого поля-добавки (nonce) в блоке данных [4]. Как только блок, удовлетворяющий условию, найден, его содержимое нельзя изменить, не выполнив заново всей работы. И если он не является последним в цепочке, эта работа включает в себя и перевычисление всех блоков, следующих за ним.

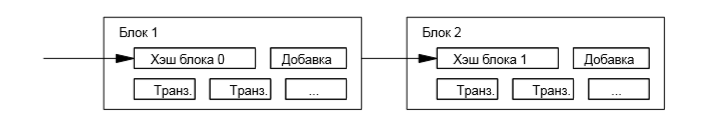


Рисунок 3 – Доказательство работы

Доказательство работы через хэширование также решает вопрос об определении версии, поддерживаемой большинством. Если голосом считается один IP-адрес, то такую схему можно скомпроментировать, если контролировать большой диапазон адресов. Схема основана на принципе «один процессор — один голос» [5]. Самая длинная из хэш-цепочек выражает мнение большинства, которое вложило в нее наибольшее количество ресурсов. Если более половины вычислительной мощи принадлежит честным узлам, то цепочка честных транзакций будет расти быстрее и опередит любую конкурирующую цепь. Чтобы внести изменения в любой из прошлых блоков, атакующему придется выполнить заново работу над этим блоком и всеми последующими, а затем догнать и перегнать честных участников по новым блокам. Ниже будет показано, что вероятность такого успеха у злоумышленника, обладающего меньшими ресурсами, экспоненциально убывает в зависимости от числа блоков. Для компенсации возрастающей вычислительной мощи процессоров и колебания числа работающих узлов в сети, сложность хэширования должна изменяться, чтобы обеспечивать равномерную скорость генерации блоков. Если они появляются слишком часто — сложность возрастает, и наоборот [6].

* 1. **Сеть блокчейн**

Система работает по следующим правилам:

* новые транзакции рассылаются всем узлам;
* каждый узел объединяет пришедшие транзакции в блок;
* каждый узел пытается подобрать хэш блока, удовлетворяющий текущей сложности;
* как только такой хэш найден, этот блок отправляется в сеть;
* узлы принимают блок, только если все транзакции в нем корректны и не используют уже потраченные средства;
* свое согласие с новыми данными узлы выражают, начиная работу над следующим блоком и используя хэш предыдущего в качестве новых исходных данных.

Участники всегда считают истинной самую длинную версию цепочки и работают над ее удлинением. Если два узла одновременно опубликуют разные версии очередного блока, то кто-то из остальных пиров получит раньше одну версию, а кто-то — другую. В таком случае каждый начнет работать над своей версией цепочки, сохранив другую на случай, если она окажется продолжена раньше. Двойственность исчезнет, как только будет получен новый блок, который продолжит любую из ветвей, и те узлы, что работали над конкурирующей версией, переключатся на нее. Новые транзакции не обязательно должны достигать всех узлов. Если о них будет знать достаточно много узлов, вскоре они попадут в один из блоков. Правила рассылки блоков тоже не являются строгими в отношении потерянных сообщений. Как только узел, пропустивший один из блоков, получит уже следующий за ним, он запросит недостающую информацию, чтобы заполнить очевидный пропуск.

* 1. **Конфиденциальность**

Традиционная банковская модель поддерживает необходимый уровень конфиденциальности, предоставляя доступ к информации лишь сторонам-участницам и доверенному третьему лицу как показано на рисунке 4. Необходимость открытой публикации транзакций исключает такой подход, однако приватность по-прежнему можно сохранить, если публичные ключи будут анонимными. Открытой будет информация о том, что кто-то отправил кому-то некоторую сумму, но без привязки к конкретным личностям. Столько же данных раскрывается и на фондовых биржах, которые публикуют время и объем частных сделок, не указывая, между кем именно они были совершены.

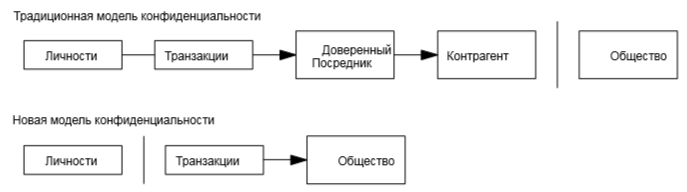


Рисунок 4 – Модели конфиденциальности

Дополнительной защитой будет являться генерация новой пары «открытый/закрытый ключ» для каждой транзакции: это предотвратит связывание различных платежей с их общим отправителем или адресатом. Некоторого публичного связывания все же не избежать: транзакции с несколькими входами доказывают, что эти суммы принадлежат одному лицу. Риск состоит в том, что раскрытие личности владельца ключа может привести к раскрытию и всех принадлежащих ему транзакций.

**2 Техническое задание**

**2.1 Постановка задачи**

В рамках производственной практики поставлена задача реализации блокчейн модуля системы электронного документооборота. Данная система в целом направлена на хранение и обмен документами между зарегистрированными организациями. Реализованная блокчейн платформа является неотъемлемой частью системы, т.к. выполняет функцию подтверждения подлинности загруженных документов, а также получения базовой информации относительно записи блокчейна (кем была добавлена запись, кем был добавлен документ, когда был добавлен документ, а также проверка имеющегося у пользователя документа с помощью системы).

**2.2 Требования к системе**

# **2.2.1 Функциональные требования**

* Загрузка пользовательских документов на сервер;
* добавление записи о документе в блокчейн;
* подпись пользовательских транзакций;
* генерация пары ключей для идентификации пользователя;
* получение даты добавления документа;
* реализация пиринговой сети между узлами блокчейна;
* синхронизация цепи блокчейна между узлами сети;
* проверка подлинности документа по его хэшу;
* проверка подлинности документа по файлу;

# **2.2.2 Нефункциональные требования**

* Программная система должна отвечать основным требованиям сервис ориентированной архитектуры;
* в процессе работы с системой должно присутствовать стабильное подключение к сети интернет;
* система должна поддерживать загрузку файлов формата \*.doc, \*.docx, \*.pdf, \*.txt;
* размер загружаемых файлов не должен превышать 100 Мб;
  + 1. **Варианты использования**

Диаграмма вариантов использования представлена на рисунке 5.



Рисунок 5 – Use-Case диаграмма системы

1. **Архитектура системы**

# **3.1 Структурные диаграммы**

Структурная диаграмма высокого уровня представлена на рисунке 6.

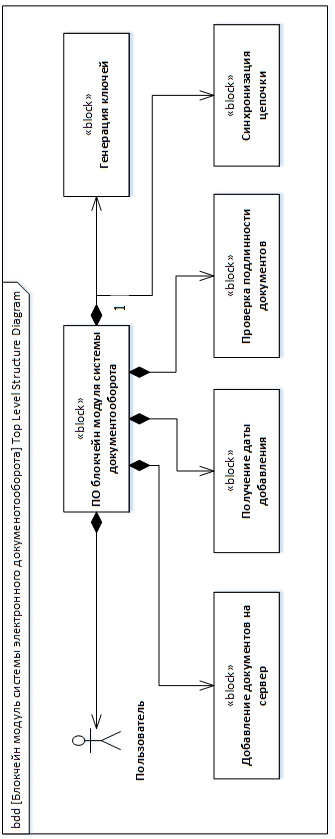


Рисунок 6 – Структурная диаграмма высокого уровня

Общая структурная диаграмма системы представлена на рисунке 7.

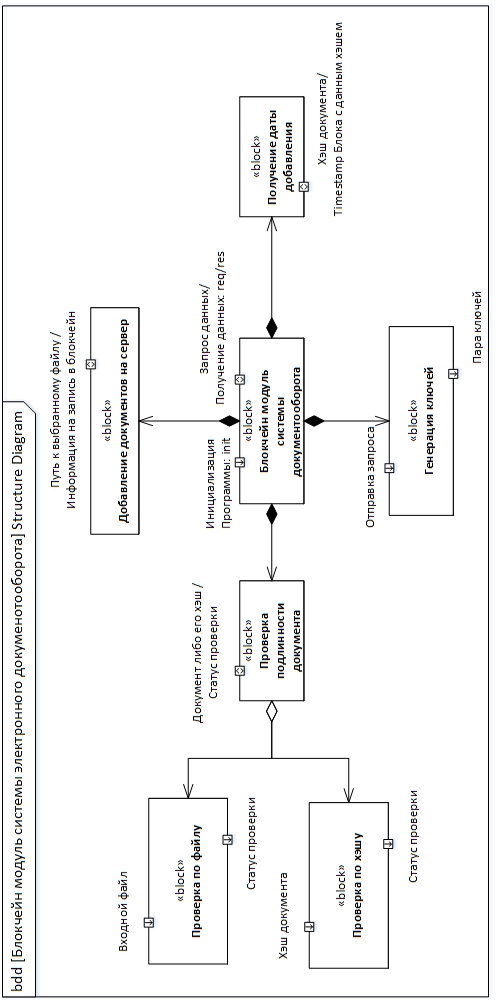


Рисунок 7 – Общая структурная диаграмма системы

# **3.2 Диаграмма деятельности**

Диаграмма деятельности системы представлена на рисунке 8.

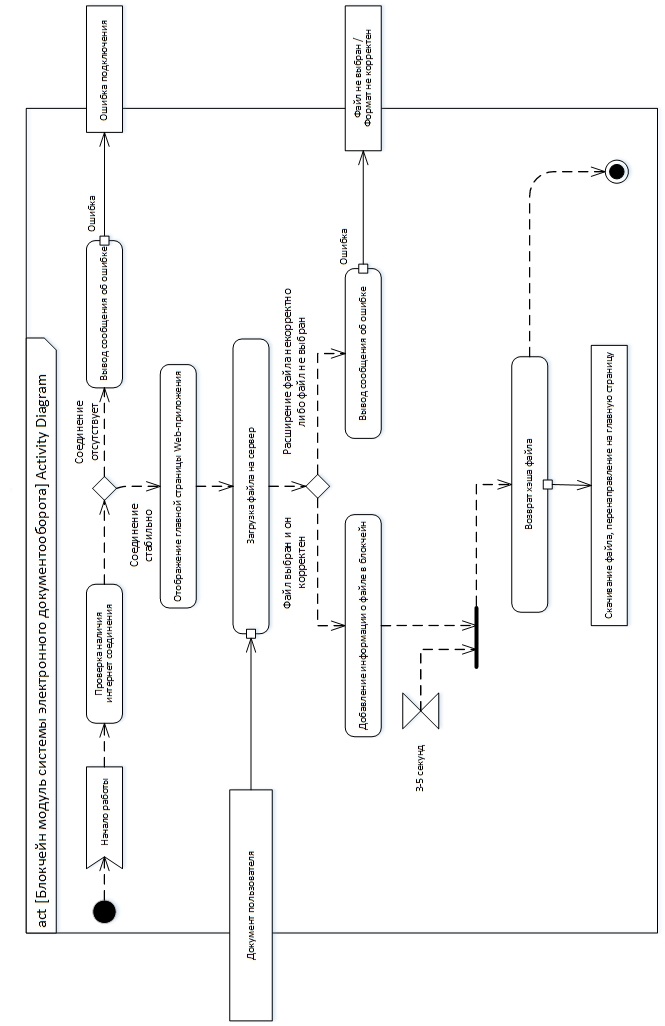


Рисунок 8 – Диаграмма деятельности системы

# **3.3 Диаграмма развертывания**

Диаграмма развертывания представлена на рисунке 9.

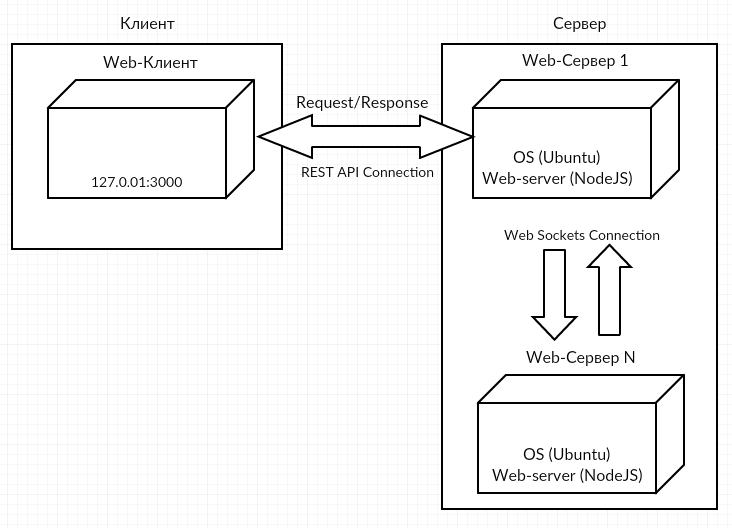


Рисунок 9 – Диаграмма развертывания

1. **Программные средства**

Back-end часть проекта реализована на основе платформы NodeJS с использованием следующих библиотек:

* Express (Express представляет собой популярный веб-фреймворк, написанный на JavaScript и работающий внутри среды исполнения node.js. Этот модуль освещает некоторые ключевые преимущества этого фреймворка, установку среды разработки и выполнение основных задач веб-разработки и развертывания);
* Body-Parser (библиотека, которая служит для парсинга данных, поступающих с клиента на сервер; данная библиотека служит как для обработки json данных, так и для текстовых и xml документов);
* Md5-file (библиотека, которая позволяет взять md5 хэш от файла произвольного формата и размера, на выходе имеем строку длинной 32 символа);
* Multer (библиотека, позволяющая осуществлять загрузку различных файлов на сервер NodeJS в определенную директорию; работает как с одиночными файлами, так и с массивом файлов для загрузки);
* FS (модуль NodeJS, открывающий доступ к взаимодействию с файловой системой сервера приложения, позволяет создавать, модифицировать, удалять необходимые файлы, а также поддерживает ряд других операций);
* Crypto (библиотека, поддерживающая огромное количество методов шифрования данных, электронную подпись данных, проверку подписи на подлинность, а также множество других криптографических операций);
* URSA (библиотека, позволяющая генерировать пару «публичный/личный ключ», а также впоследствии использовать данные ключи для подписи и проверки подписей документов);
* WS (модуль NodeJS, позволяющий организовывать пиринговую связь между несколькими экземплярами серверов; используется в приложении для обмена вновь принятой цепочки блокчейна).

Front-end часть проекта выполнена с помощью фреймворка Vue.JS. Vue.js – JavaScript фреймворк с открытым исходным кодом для создания пользовательских интерфейсов в парадигме реактивного программирования. Легко интегрируется в проекты с использованием других JavaScript библиотек, благодаря постепенно расширяемой экосистеме. Может функционировать как веб-фреймворк, помогающий разрабатывать продвинутые одностраничные приложения.

1. **Результаты работы**

**6.1 Back-end часть системы**

Серверная часть системы представляет собой полноценную блокчейн платформу, ориентированную на хранение хэшей документов, загружаемых пользователем.

Данная платформа имеет функционал, присущий большинству блокчейн систем, а именно:

* ген**е**рация хэшей для блоков цепи;
* реализация пиринговой сети между узлами, на основе технологии Web-sockets;
* реализация алгоритма подтверждения работы (Proof of work);
* подпись отправляемых в блокчейн транзакций электронными подписями, основанными на приватных ключах пользователей;
* реализация пула транзакций, собирающего все текущие транзакции со всех узлов сети в реальном времени;
* внедрение транзакций в блоки с последующим майнингом последних.

Проиллюстрируем работу блокчейн платформы с помощью настольной программы Postman, посредством вызовов различных API методов. Для начала работы запустим 2 экземпляра сервера, как показано на рисунке 10.

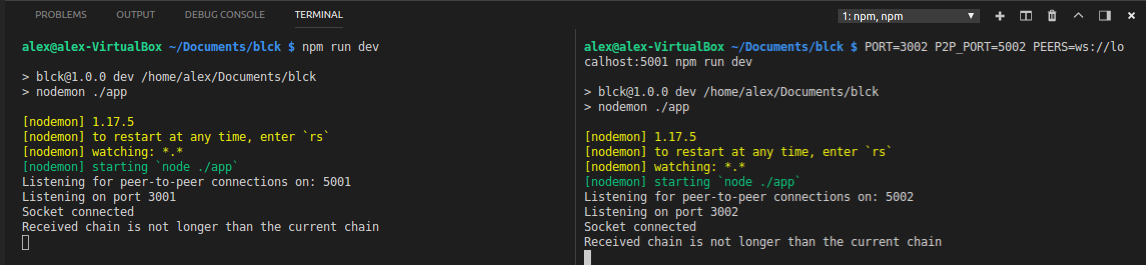


Рисунок 10 – Запуск двух экземпляров сервера

После того, как мы запустили 2 экземпляра сервера, изменения в цепочке одного из них повлечет немедленное изменение цепочки второго. Для начала, вызовем метод «/blocks» и посмотрим на исходное содержимое блокчейна. Вывод содержимого блокчейна приведен на рисунке 11.

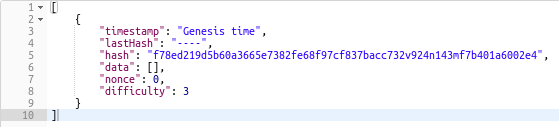


Рисунок 11 – Начальный блок (Genesis Block)

Изначально блокчейн начинается с блока, который называется зачастую «инициирующим» блоком. Информация, содержащаяся в нем, не относится к данным блокчейна, а представляет собой произвольные данные. Чтобы начать работу с блокчейном, необходимо получить пару «публичный/личный ключ». Для этого воспользуемся методом «/register». Пример вызова данного метода представлен на рисунке 12.

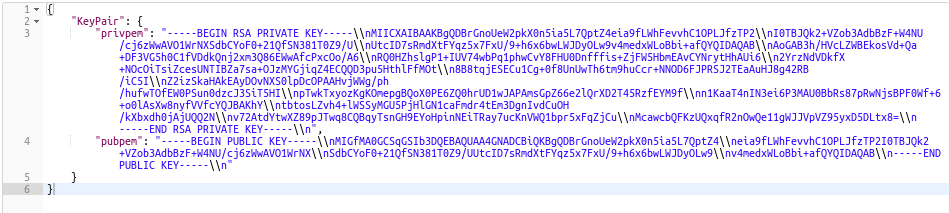


Рисунок 12 – Результат вызова метода «/register»

После получения пары ключей, мы можем воспользоваться методом «/postfile», который принимает три аргумента: приватный ключ, публичный ключ, документ. Проиллюстрируем работу данного метода на рисунке 13.

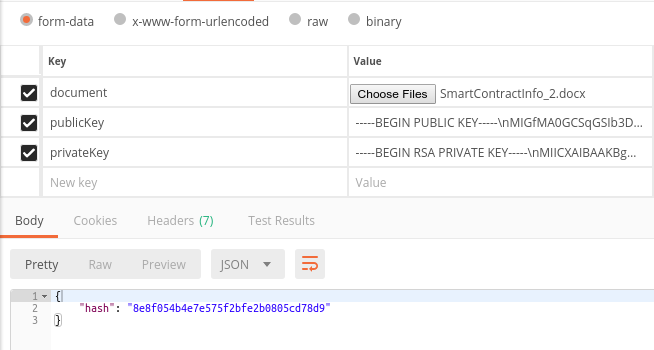


Рисунок 13 – Результат выполнения метода отправки файла

Как видно на рисунке 13, в ответ на добавленный файл система посылает его хэш, который добавляется в новую транзакцию, после чего транзакция добавляется в блок, а блок добавляется в блокчейн. Стоит особо отметить, что обновленная цепочка блокчейна немедленно принимается втором экземпляром запущенного сервера. Посмотрим, как выглядит цепочка блокчейна после добавления нового файла. Рисунок 14 иллюстрирует данный случай.

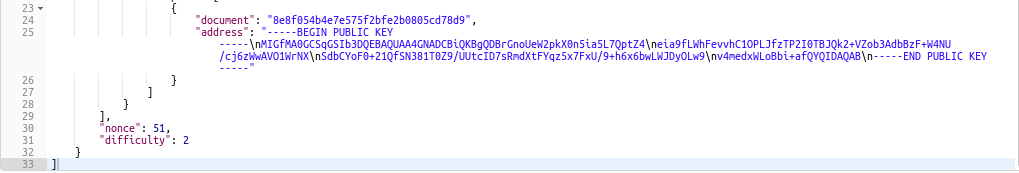


Рисунок 14 – Обновленная цепочка блоков

Теперь воспользуемся методами, направленными на проверку подлинности пользовательских файлов. Существует два способа данной проверки – по файлу и по хэшу файла. На рисунке 15 изображен случай проверки подлинности по файлу.

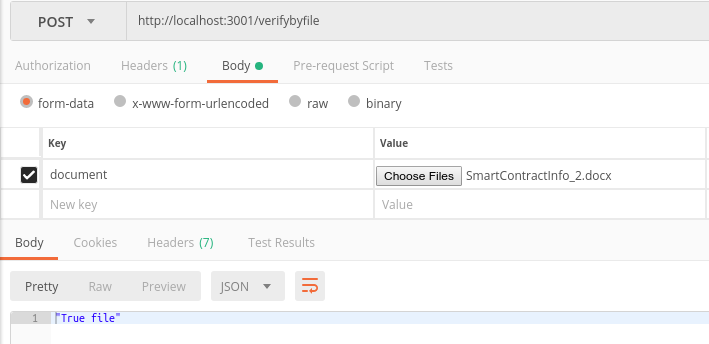


Рисунок 15 – Проверка подлинности по файлу

Как видно на рисунке 15, загруженный файл является подлинным. Теперь воспользуемся методом проверки подлинности документа по хэшу документа. Данную проверку иллюстрирует рисунок 16.

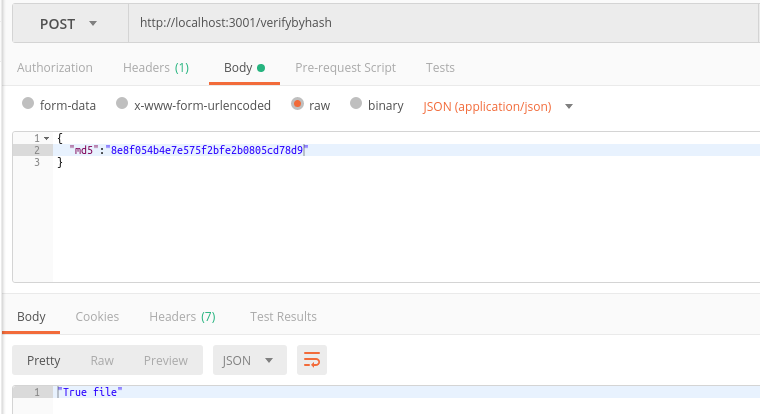


Рисунок 16 – Проверка подлинности документа по его хэшу

Также реализованное API имеет метод получения даты добавления блока в блокчейн. Проиллюстрируем использование данного метода на рисунке 17.

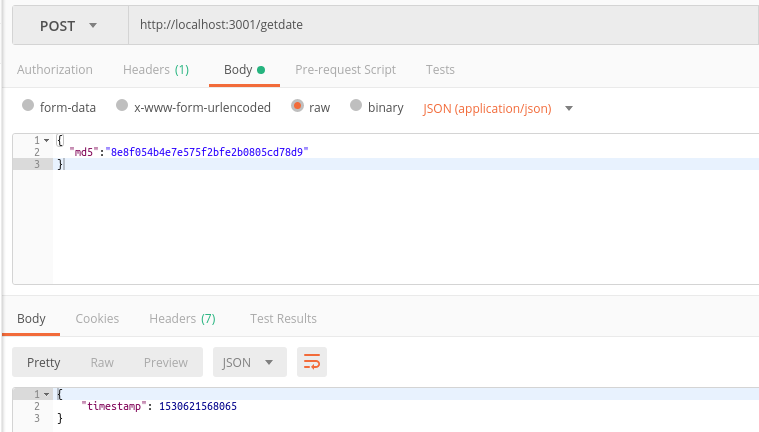


Рисунок 17 – Получение даты добавления блока

Как видно на рисунке 17, сервер возвращает timestamp (метку момента времени добавления блока). Имея данную метку, с помощью стандартных методов можно получить дату и время добавления блока с точностью до секунд.

Последним рассмотренным API методом будет метод «/getdocs», позволяющий получить список файлов, загруженных на сервер. Вызов данного метода приведен на рисунке 18.

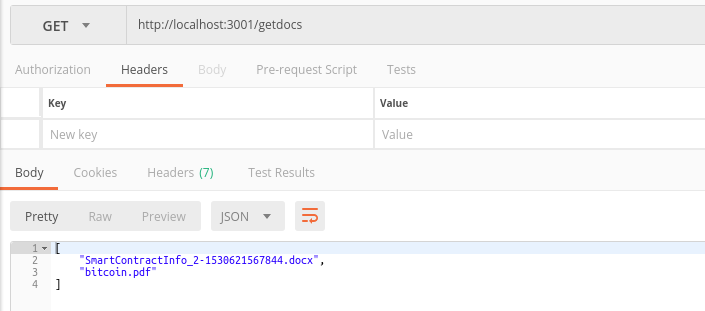


Рисунок 18 – Получение списка загруженных документов

# **6.2 Front-end часть системы**

Front-end часть системы выполнена с помощью фреймворка Vue.JS. Далее будет приведено описание работы front-end части системы с подробным разбором каждого отображаемого компонента.

Стартовым компонентом системы является компонент отображения имеющихся на сервере файлов (вызов API метода «/getdocs»). Внешний вид данного метода представлен на рисунке 19.

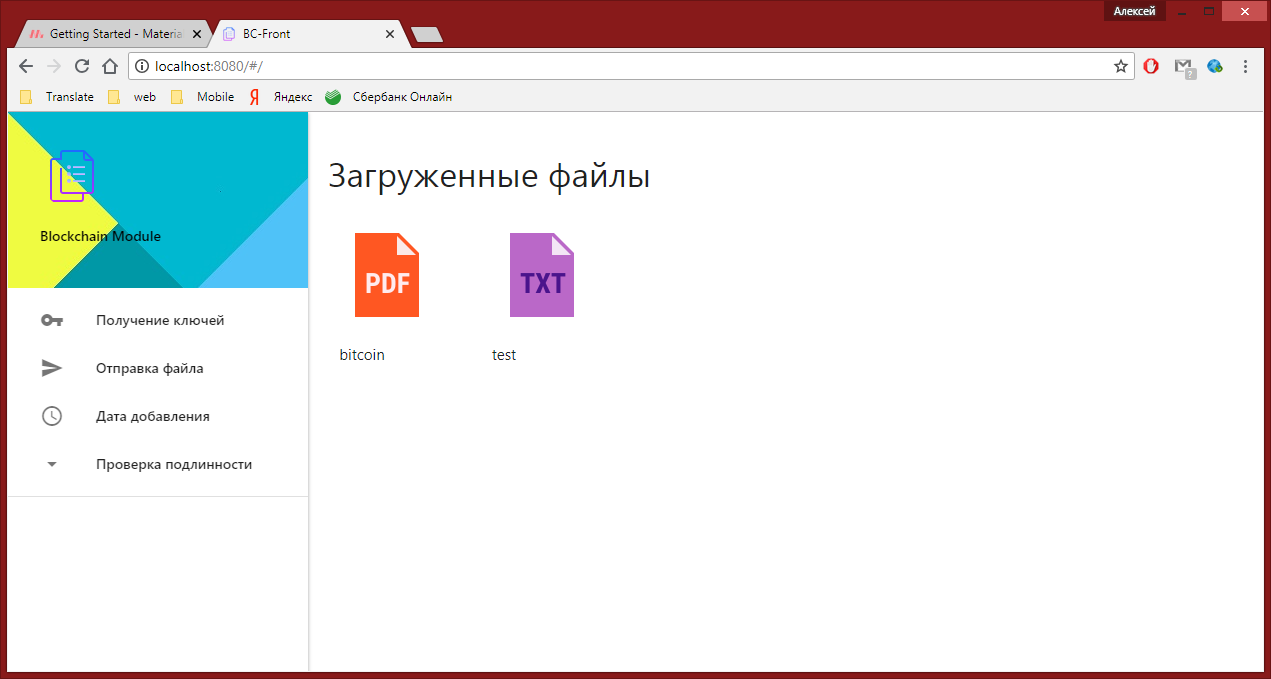


Рисунок 19 – Компонент «Загруженные файлы»

На рисунке 20 представлен компонент «Получение ключей». Данный компонент соответствует вызову API метода «/register». При загрузке компонента происходит запрос к серверу по данному методу. После получения результатов запроса (а именно получения публичного и личного ключа) в разметке появляется кнопка для скачивания файла с данными ключами. На рисунке 21 представлен вид содержимого данного файла.

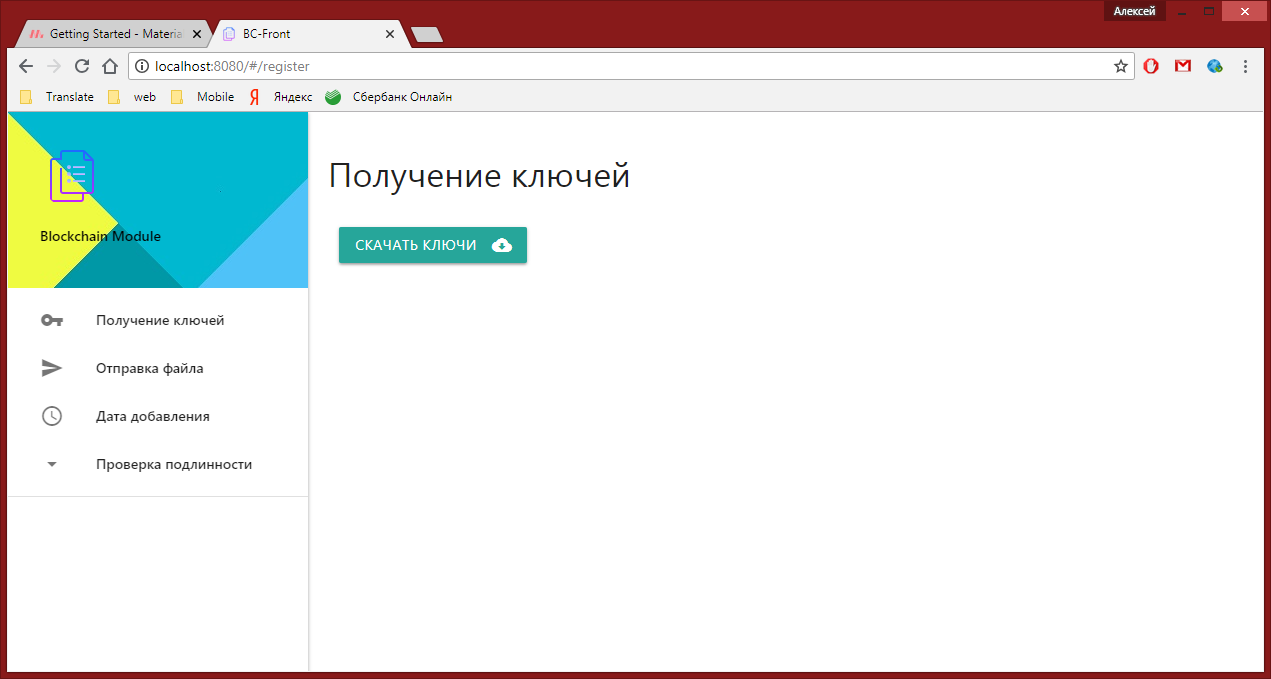


Рисунок 20 – Вид компонента «Получение ключей»

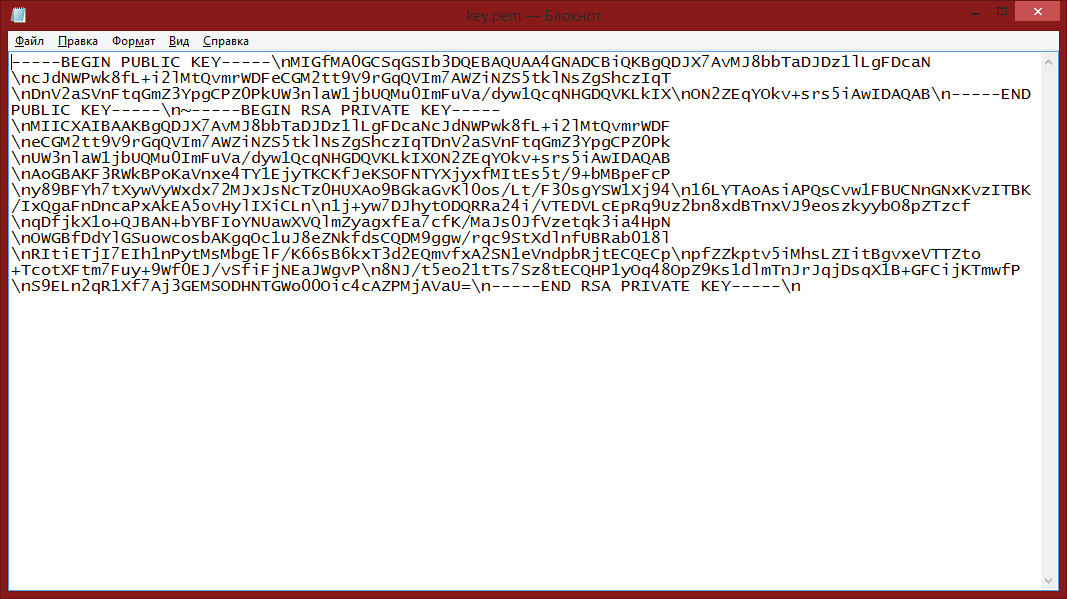


Рисунок 21 – Содержимое файла key.pem

После скачивания файла пользователь может отправить свой документ в систему. Для этого необходимо перейти к компоненту «Отправка файла» (вызов API метода «/postfile») и загрузить файл key.pem, а также файл документа. В случае успешной загрузки, пользователь будет перенаправлен на компонент «Загруженные файлы», где в списке загруженных появится новый файл. Также стоит отметить, что после загрузки файла пользователь увидит всплывающее уведомление с хэшем данного файла. На рисунке 22 представлен вид компонента «Отправка файла», тогда как на рисунке 23 представлен показ сообщения с хэшем.

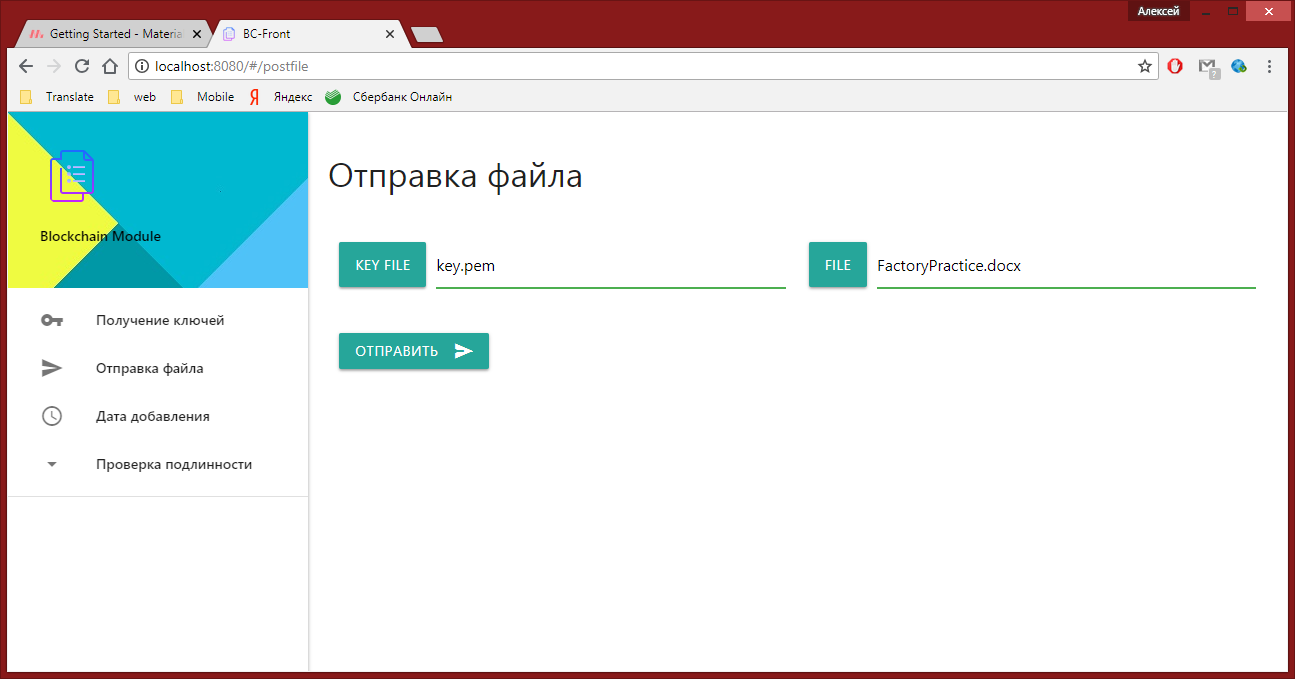


Рисунок 22 – Внешний вид компонента «Отправка файла»

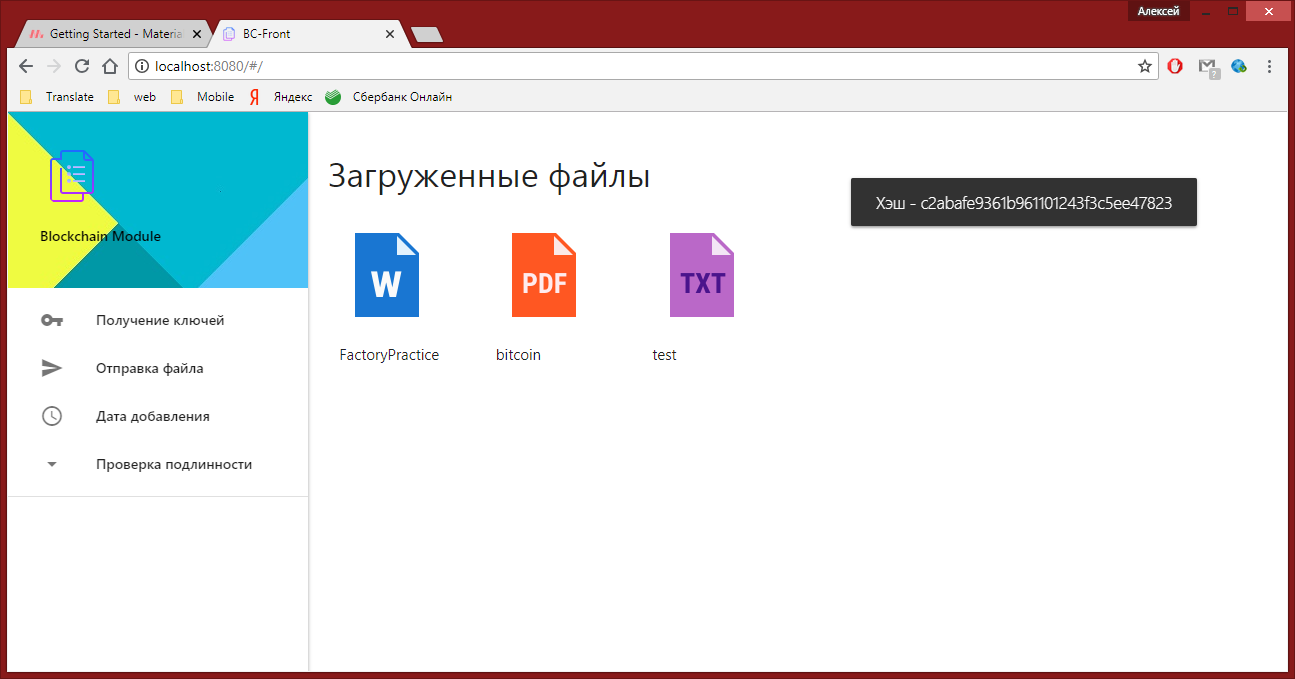


Рисунок 23 – Перенаправление на главный компонент с хэшем

После того, как мы добавили свой документ и получили его хэш, мы можем пользоваться остальным функционалом приложения – проверять подлинность файла, а также узнавать дату его добавления. Для начала узнаем дату добавления файла, перейдя к компоненту «Дата добавления» и введя полученный хэш в форму. Данная процедура представлена на рисунке 24.

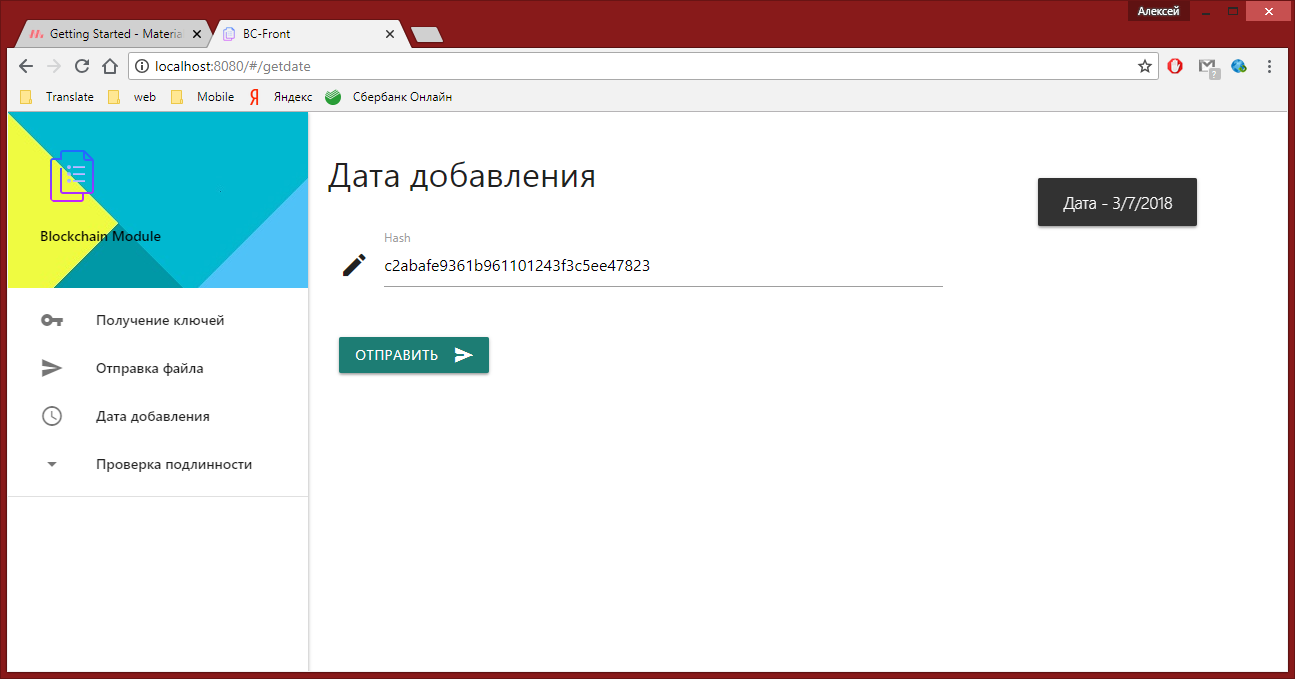


Рисунок 24 – Вывод даты добавления файла

Теперь проверим подлинность загруженного документа, перейдя к компоненту «Проверка подлинности». Для начала выберем проверку подлинности по файлу и загрузим наш файл. Проверка подлинности представлена на рисунке 25.

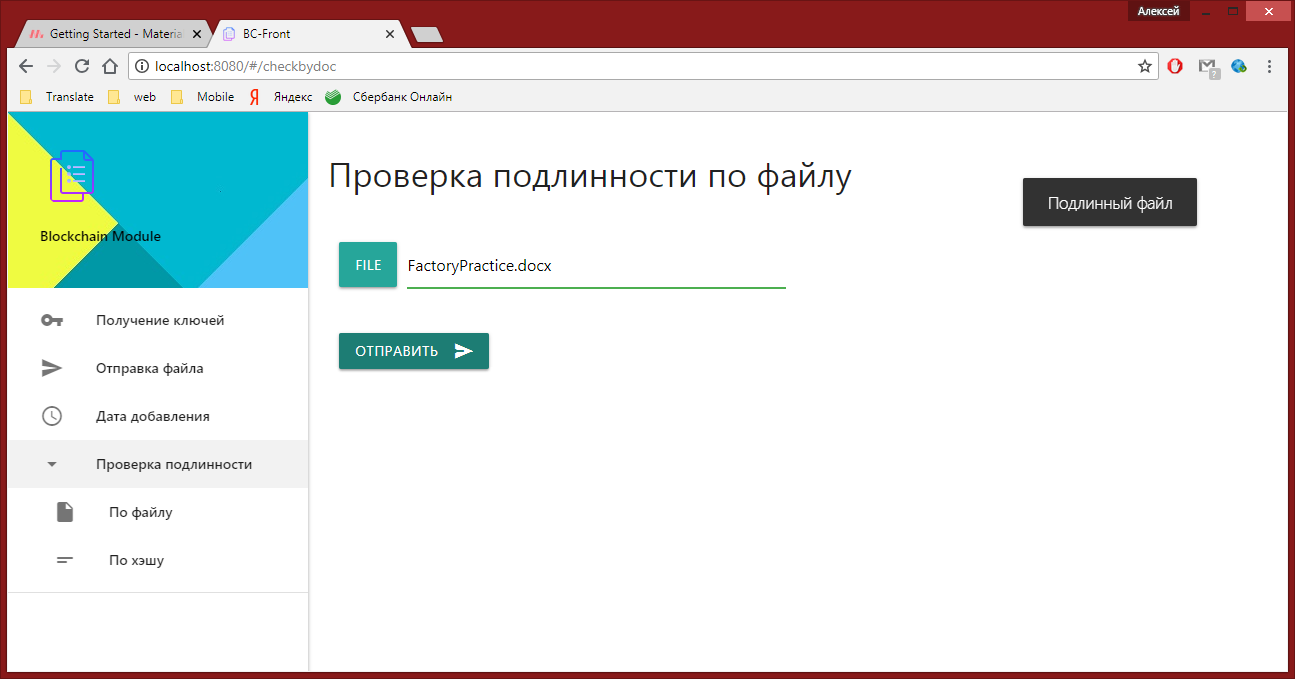


Рисунок 25 – Проверка подлинности по файлу

Проверка подлинности документа по его хэшу представлена на рисунке 26. Для проверки необходимо ввести хэш документа в отведенное для этого поле в компоненте «Проверка подлинности по хэшу».

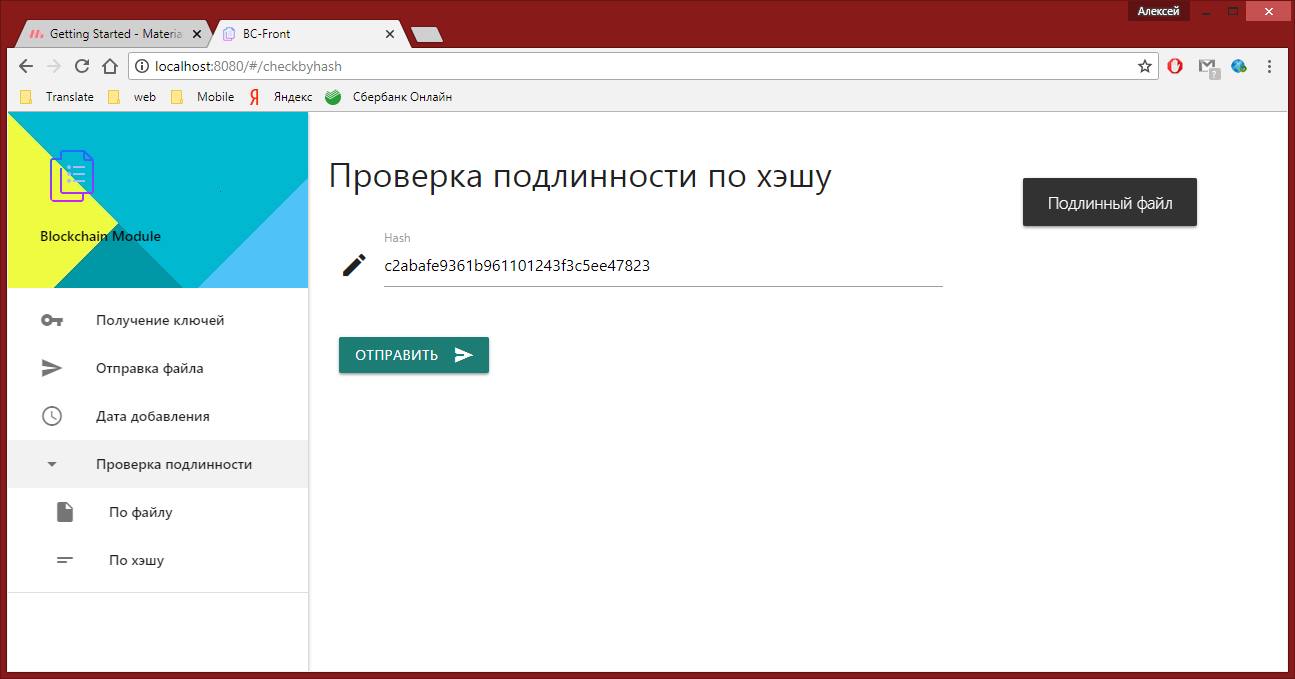


Рисунок 26 – Проверка документа по хэшу

# **Вывод**

В результате работы была реализована блокчейн платформа, позволяющая пользователям загружать документы в систему, подтверждать подлинность загруженных документов несколькими способами: с помощью исходного файла либо с помощью хэша документа. Также узлы блокчейн платформы соединены пиринговой сетью на основе Web сокетов.

Помимо этого, в дополнение к проекту реализована front-end часть, выполненная с помощью фреймворка Vue.JS. Front-end приложение позволяет удобно загружать и узнавать необходимую информацию об интересующих пользователя файлах, проверять и доказывать подлинность файлов.

# **Список источников**

1. W. Dai, "b-money" [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://www.weidai.com/bmoney.txt;
2. H. Massias, X.S. Avila, J.J. Quisquater, Design of a secure timestamping service with minimal trust requirements, [Электронный ресурс]. – Электрон. дан. – Режим доступа: <https://nakamotoinstitute.org/literature/secure-timestamping-service;>
3. S. Haber, W.S. Stornetta, How to timestamp a digital document [Текст] – In Journal of Cryptology, vol 3, pages 99-111, 1991;
4. D. Bayer, S. Haber, W.S. Stornetta, Improving the efficiency and reliability of digital timestamping [Текст] – Security and Computer Science, pages 329-334, 1993;
5. S. Haber, W.S. Stornetta, Secure names for bit-strings [Текст] – Security and Computer Science, pages 28-35, April 1997;
6. R.C. Merkle, Protocols for public key cryptosystems [Текст] – IEEE Computer Society, pages 122-133, April 1980.