Санкт-Петербургский государственный университет

Кафедра системного программирования

Кантеев Леонид Дмитриевич

Технология блокчейн для аудита IT-проекта

Бакалаврская работа

Научный руководитель:
к.ф.-м.н., ст. преп. Луцив Д. В.

Рецензент:
Инженер ООО "САП ЛАБС" Ражев Н. А.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Software engineering

Leonid Kanteev

Blockchain technology for IT-project audit

Bachelor's Thesis

Scientific supervisor:
PhD Dmitry Luciv

Reviewer:
Engineer SAP LABS LLC Nikita Razhev

Saint-Petersburg
2019

Оглавление

Введение

Менеджмент ИТ-проектов является достаточно трудоемкой работой, которая требует большого количества человеческих ресурсов. Существует много способов контролировать проект. Один из них — это аудит. Аудит призван уменьшить риски, сократить расходы или получить стороннюю оценку проекта. Внутренний аудит проводится службой внутри компании-разработчика с целью анализа функционирования компании для помощи руководству в управлении. Задача внешнего аудита — объективная оценка и контроль действий менеджмента и самого проекта.

Одной из важных функций внешнего аудита является аудит требований. Требования существуют и изменяются на протяжении всего проекта. Каждое изменение необходимо контролировать и подтверждать, чтобы не было разногласий между заказчиком и исполнителем по факту выполненных работ. Для этого приглашается сторонний человек из нейтральной организации-аудитора и заверяет все действия по изменению требований.

Особенно большая проблема возникает в сфере медицины и питания. Обычно, государство очень ответственно относится к контролю разработок в этих сферах, в том числе подвержены пристальному вниманию и связанные с ними ИТ-проекты. В данный момент для того, чтобы внести изменение в любую ИТ-систему, связанную с медициной или питанием, необходимо вызвать человека из государственного аудита, чтобы он согласовал и утвердил требования. Это долгий и дорогой процесс. На всей территории США существует только одна такая организация — FDA [20], она имеет очень строгий регламентированный порядок утверждения требований.

В связи с вышеописанными проблемами появилась идея создать хранилище, которое позволит гарантировать корректность требований в любой момент времени и упростит процедуру утверждения требований. Основная сложность внедрения нового решения — отсутствие доверия к системе у государства. Поэтому одним из основных требований

к такому хранилищу должно быть математически доказанная корректность работы в случае отказа или некорректного поведения некоторых участников сети.

Существует много способов хранения данных. Одним из них является технология блокчейн [9]. Она позволяет создать распределенную базу данных с подтверждением транзакций, гарантией стабильности и, главное, неизменяемой историей действий. Однако, в привычном понимании блокчейн является открытым, а все участники сети равноправны, что не подходит по концепции нашей задачи. Но технология не ограничивается только публичным биткоин-блокчейном, существует много разновидностей, например, таких как приватный блокчейн [8]. Он позволяет задавать права участников в сети и создавать смарт-контракты, которые дают возможность описать взаимодействие пользователей с хранилищем. Одной из важных особенностей приватного блокчейна является возможность выбора консенсуса [1], то есть выбора алгоритма, по которому подтверждаются или отклоняются транзакции, что позволяет гарантировать работу блокчейна даже в экстремальных условиях.

1. Постановка задачи

Целью данной работы является разработка сервиса для внешнего аудита требований IT-проекта на основе технологии блокчейн. Для достижения этой цели были поставлены следующие задачи:

- разработать требования к сервису;
- спроектировать архитектуру сервиса;
- реализовать сервис.

2. Обзор предметной области и существующих решений

2.1. Общий подход аудита требований

Аудит требований — это известная задача, и многие компании-аудиторы решают её. На данный момент общим подходом является заверение всех изменений требований с помощью человеческих ресурсов внешнего аудита.

2.2. Контроль питания и медицины со стороны государства

Что касается государственных органов по надзору, то подход для аудита требований еще строже. Нельзя заверить требования по отдельности, необходимо предоставить все требования к проекту, и они будут полностью пересмотрены, что занимает много времени и исключает возможность быстрой разработки.

2.3. Технология PwC

Один из крупнейших аудиторов в мире — PwC [11] в 2019 году представил сервис для аудита с помощью технологии блокчейн. Однако, это не универсальный сервис для решения конкретных задач, а абстрактное решение, которое будет создаваться конкретно под ваш случай.

2.4. Технология Deloitte

Еще в сентябре 2017 года другим аудитором из "Большой четверки" [19] - Deloitte совместно с компанией DNV GL было запущено первое в мире решение для цифрового хранения сертификатов на блокчейне [2]. С тех пор Deloitte много раз использовал технологию блокчейн для своих проектов, однако общего решения для аудита требований до сих пор нет.

2.5. Технология MediLedger

MediLedger [7] — это современная блокчейн-платформа для поддержки медицинских компаний. Она предоставляет сервис для хранения каталога медицинских продуктов и их идентификации. Сервис написан в условиях стандарта DSCSA [4] и официально может работать на территории США. Пример этой платформы показывает, что интеграция современных решений с государственными органами возможна и даже нужна.

3. Требования к сервису

3.1. Мотивация

Желание разработать сервис, который будет соответствовать современным стандартам разработки, а также изучить работу сторонних блокчейн-решений, сподвигло меня провести анализ нескольких похожих сервисов. Также немаловажным фактором, который способствовал написанию подробных требований к проекту, было более подробное изучение предметной области.

3.2. Анализ похожих сервисов

В ходе работы были проанализированы архитектуры и пользовательский опыт системы Storj [15] и нескольких проектов на основе Blockchain-as-a-Service от компании SAP [14].

Хранилище Storj архитектурно представляет собой многослойную систему, в которой каждый слой управляет своей частью задач. Из интересных особенностей - весь сервис является облачным за счет пользователей - каждый участник может сдать в аренду часть своего диска для нужд сети. Также можно подчеркнуть хорошую документацию этого проекта [16].

Проекты на основе основе Blockchain-as-a-Service от SAP по большей части реализуют одну архитектуру - облачное хранение данных, смарт-контракты для работы с блокчейном и облачное приложение для работы со смарт-контратами. Пользовательский опыт проектов говорит о необходимости создания максимально простого интерфейса для пользователей и скрытии деталей блокчейна в пользу упрощения взаимодействия с сервисом.

В качестве результатов проведенного анализа были сформулированы следующие требования.

3.3. Функциональные требования

3.3.1. Работа с требованиями

На основе общения с потенциальными заказчиками я пришел к мнению, что сервису необходимо уметь распознавать и фиксировать изменения требований в документах Excel.

3.3.2. Хранение данных в блокчейне

Действия записи и чтения из блокчейна должны быть доступны ограниченному кругу пользователей с возможностью добавлять новых членов и изменять уже существующие роли.

Для каждого документа в блокчейне должны храниться:

- название файла;
- id-номер документа;
- время последнего изменения документа;
- количество версий;
- история требований;
- история подтверждений требований.

3.3.3. REST-интерфейс

Для большей масштабируемости и возможного расширения, сервис должен иметь программный интерфейс, соответствующий архитектуре REST, со следующими конечными точками:

- /list
  - GET: список всех id документов;
- /requirements
  - POST: создать новый документ с требованиями;

- /requirements/id
  - GET: список всех версий требований по id документа;
  - PUT: обновить требования по id документа;
- /requirements/id/version
  - GET: список требований по id документа и версии;
- /requirements/id/last
  - GET: последняя версия требований по id документа;
- /requirements/id/version/confirm
  - POST: подтвердить список требований по id документа и версии;

3.3.4. Пользовательский интерфейс

Пользовательский интерфейс должен соответствовать изученному пользовательскому опыту и реализовывать следующие функции:

- просмотр истории требований и подтверждений;
- подтверждение версии требований.

3.4. Нефункциональные требования

3.4.1. Модульность

Для сервиса, который соответствует современным тенденциям разработки и проектирования, важна модульность. Она упростит доработку и тестирование сервиса.

3.4.2. Внедрение

Для наиболее быстрого внедрения сервиса потенциальному заказчику необходимо придерживаться следующих принципов:

- максимальная независимость от данных заказчика;
- автономность развертывания сервиса;
- максимальное количество модулей сервиса должны быть облачными.

3.4.3. Масштабируемость

Для сервиса, который претендует на универсальность и быстрое внедрение к новым пользователям, архитектуру необходимо создавать так, чтобы можно было в кратчайшие сроки расширить функциональность введением новых плагинов для работы с другими типами документов.

3.4.4. Отказоустойчивость

В связи с высокой ответственностью сервиса, необходимо чтобы он работал корректно даже при отказе или недобросовестном поведении менее трети всех членов сети.

3.4.5. Дружелюбность пользовательского интерфейса

Пользовательский интерфейс необходимо реализовать с учетом лучших практик и учетом пользовательского опыта, чтобы использование системы было наиболее простым.

4. Архитектура сервиса

4.1. Выбор платформы

Сформулированные требования говорят о необходимости поддерживать быстрое внедрение и облачность нашего сервиса, поэтому в качестве платформы для интеграции была выбрана SCP (SAP Cloud Platform) [13]. Она дает возможность развертывать приложения на облачных серверах, имеет встроенную авторизацию для клиентов SAP, а также предоставляет много интегрированных блокчейн-решений [14]. Самые интересные из этих решений для нас — это Multichain и Hyperledger Fabric.

Multichain чаще используется для хранения и использования электронных валют. Это приватный блокчейн, который позволяет хранить очень много данных при достаточно высокой скорости записи. Многие банки начали интегрировать Multichain для своих проектов [17]. Однако, для нашей задачи этот фреймворк подходит плохо из-за отсутствия смарт-контрактов.

Hyperledger Fabric [5] позволяет хранить в своем блокчейне любые данные. Смарт-контракты дают возможность контролировать хранилище и реализовать алгоритм консенсуса. Узлы могут иметь разные роли, обеспечивая приватность блокчейна. Поэтому в качестве блокчейн-фреймворка был выбран Hyperledger Fabric.

4.2. Общее взаимодействие модулей сервиса

С учетом требований была разработана архитектура сервиса, представленная на рис 1.

Ниже представлено краткое описание модулей сервиса:

- блокчейн, основанный на платформе Hyperledger Fabric;
- смарт-контракты Chaincode, реализующие доступ к блокчейну;
- приложение-прослойка для связывания пользовательского интерфейса и плагинов с блокчейном;

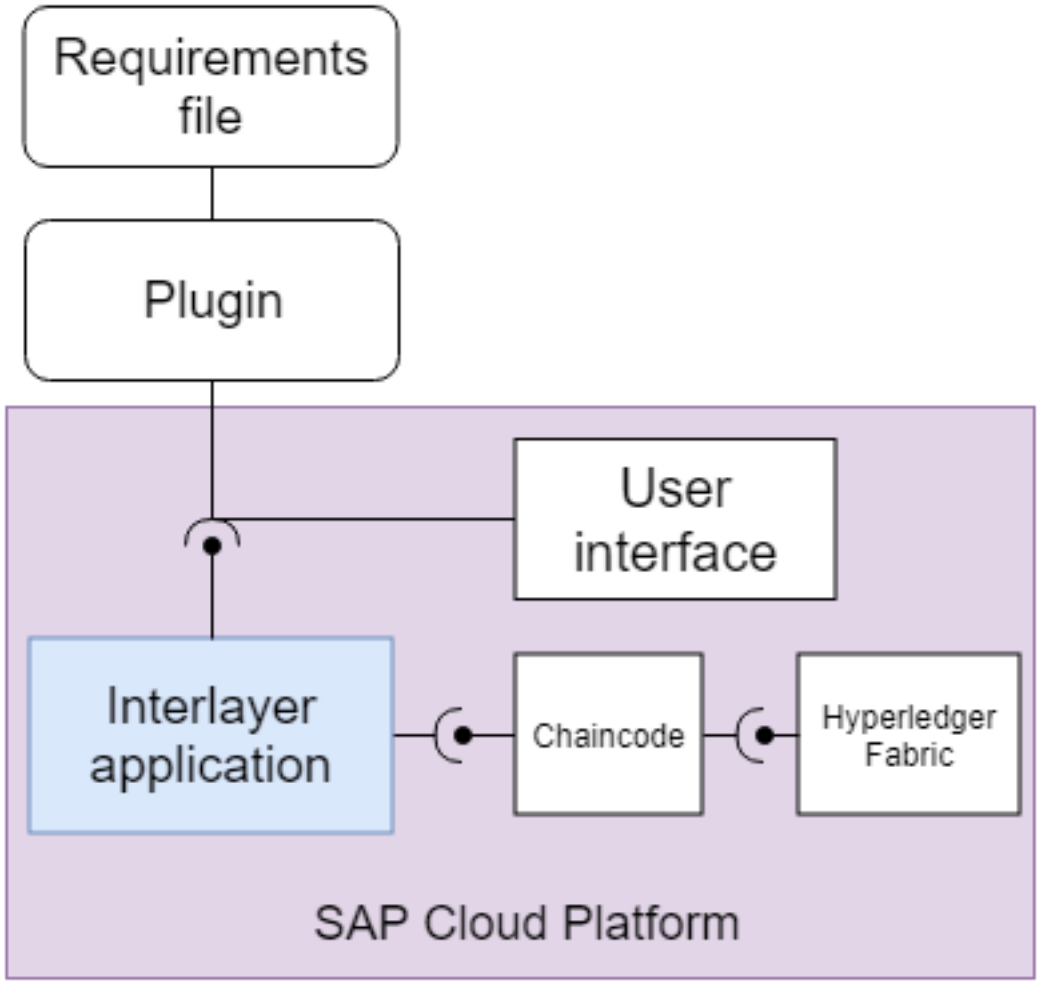

Рис. 1: Архитектура сервиса

- плагины для обработки файлов разных форматов и общению с приложением-прослойкой;
- пользовательский интерфейс для просмотра истории изменения и одобрения требований.

4.3. Приложение-прослойка

Основное предназначение приложения-прослойки (компонент «Interlayer application» на рис.1) - это связь всех модулей сервиса с блокчейном. Оно предоставляет конечные точки, обращаясь к которым можно работать с блокчейном. На рис. 2 приведена диаграмма классов приложения-прослойки.

Класс Endpoints предоставляет веб-сервер с конечными точками, ко-

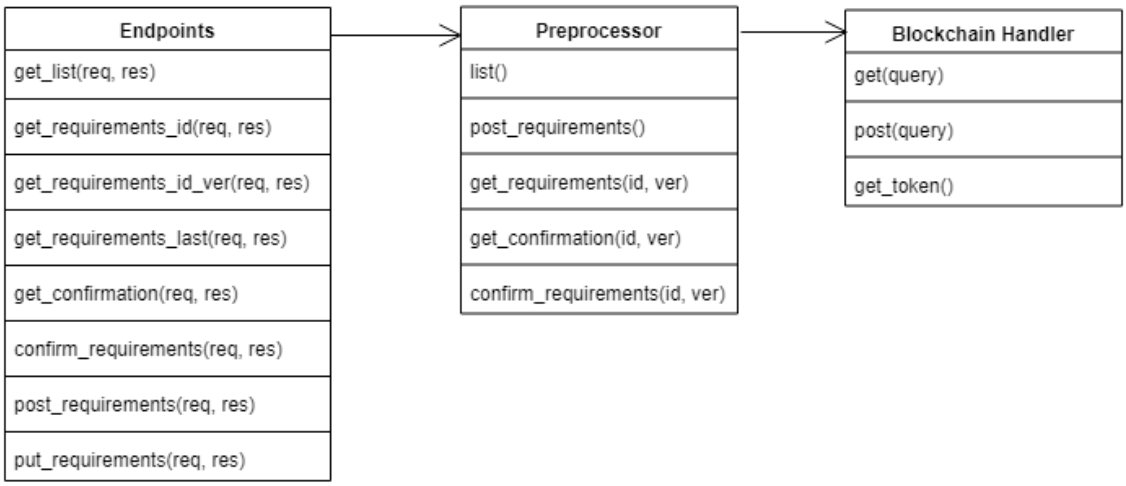

Рис. 2: Архитектура приложения-прослойки

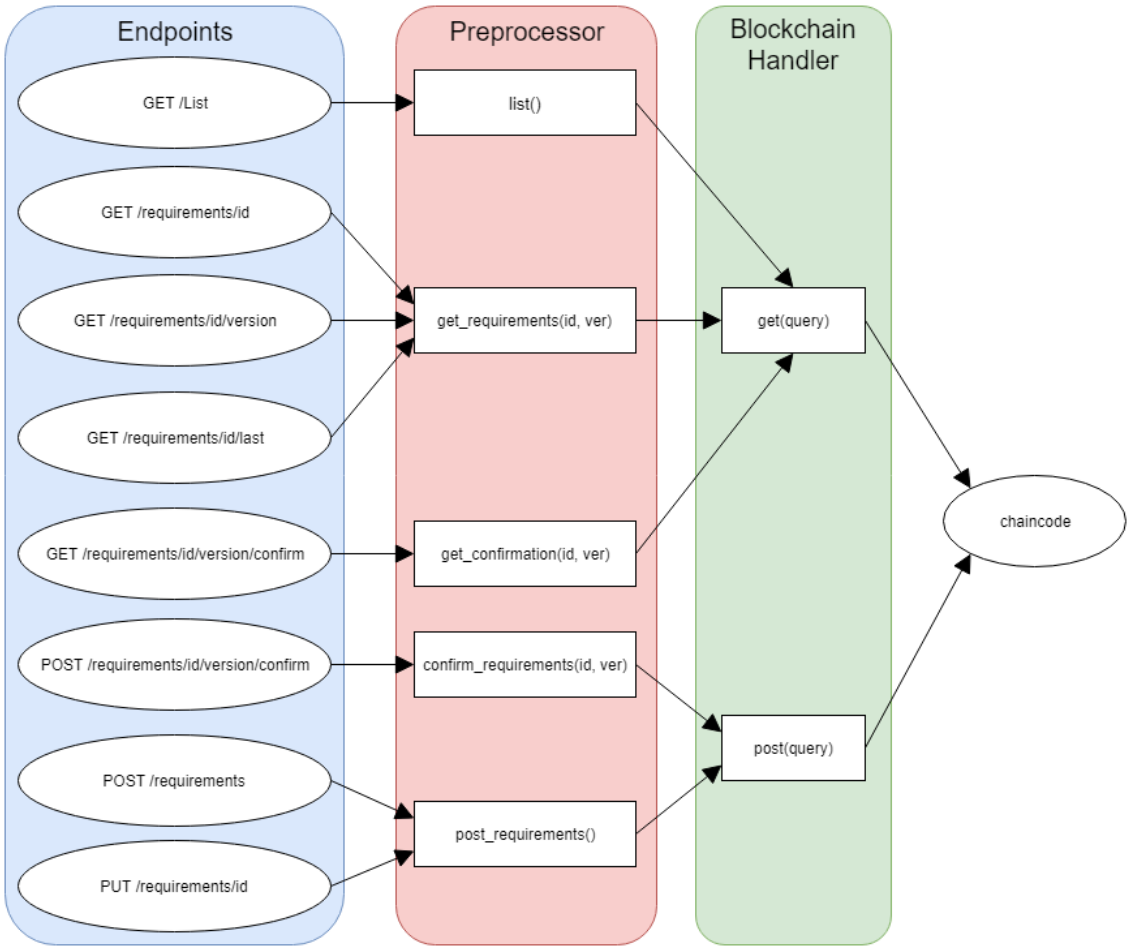

Рис. 3: Схема обработки запросов приложенем-прослойкой

торые получают данные и отправляют их в класс Preprocessor. Он, в свою очередь, обрабатывает запросы, стараясь отсеять некорректные и уменьшить нагрузку на блокчейн для ускорения работы сервиса. В ко-

нечном итоге предобработанный запрос попадает в Blockchain Handler, который отправляет его в смарт-контракт. На рис. 3 можно увидеть схему обработки запросов приложением-прослойкой.

4.4. Смарт-контракты

Задача смарт-контрактов (компонент «Chaincode» на рис.1) - обработать запрос и сделать транзакцию в блокчейн. Hyperledger Fabric предоставляет 2 основный вида транзакций - чтение и запись из key-value базы данных. Таким образом, смарт-контракт разбить каждый запрос на серию чтений и записей в базу данных. Схема работа обработки запросов смарт-контрактом представлена на рис. 4.

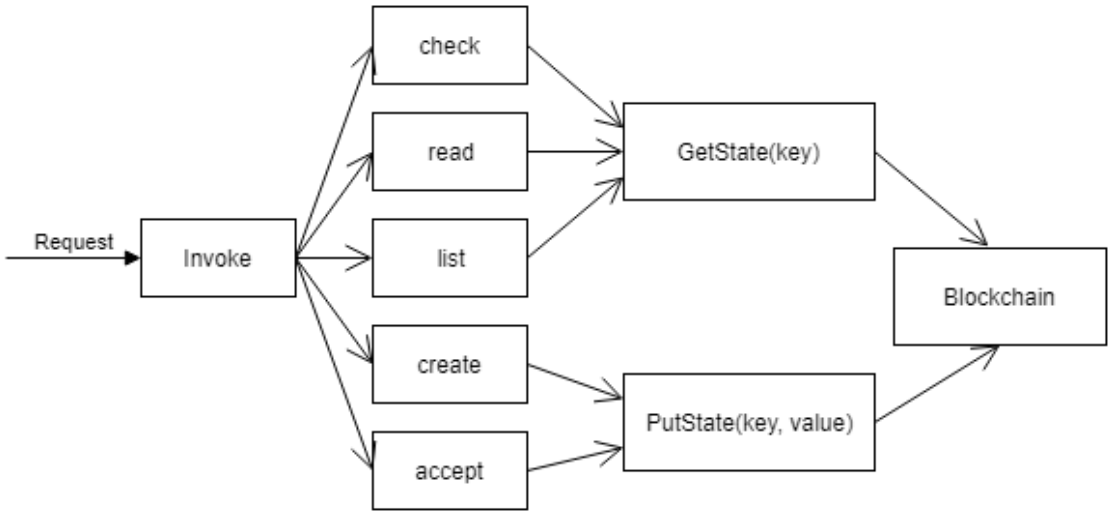

Рис. 4: Схема обработки запросов смарт-контрактом

5. Особенности реализации

5.1. Плагин для Excel

Плагин для Excel реализован на языке Visual Basic и добавляет в документ 2 кнопки - «Upload» и «Download». Кнопка «Upload» собирает метаданные из файла и отправляет его на сервер. Если этот файл еще никогда не загружался в блокчейн, то для него генерируется уникальный id-номер и метаданные нужные для работы сервиса. Для генерации id-номера используется стандарт UUID. Кнопка «Download» собирает метаданные из файла и, если такой файл в блокчейне есть, скачивает последнюю версию требований.

Таким образом, для плагина реализованы 4 основных функции: сгенерировать метаданные, прочитать метаданные, отправить файл на сервер и скачать файл с сервера.

5.2. Пользовательский интерфейс

Пользовательский интерфейс реализован на JavaScript с использованием фреймворков Vue.js [18] и Quasar [12], размещен в SCP и дает возможность подтверждать конкретную версию требований, просматривать историю изменения требований, историю подтверждения требований и скачивать файлы с требованиями. Пользовательский интерфейс постоянно общается с приложением-прослойкой по средствам HTTP-запросов и обновляет историю в реальном времени, без перезагрузки сайта.

5.3. Приложение-прослойка

Приложение-прослойка разработано на языке JavaScript с использованием платформы Node.js [10] и библиотеки Express [3], размещено в SCP и имеет REST-API, реализующее конечные точки, которые удовлетворяют выдвинутым требованиям. Все входящие файлы преобразуются в строки BASE64 и отправляются в блокчейн. При запросе

на скачивание файла из блокчейна достается строка BASE64 и преобразуется обратно в файл. Также приложение-прослойка фильтрует некорректные запросы, стараясь уменьшить нагрузку на блокчейн и увеличить скорость работы сервиса. Большинство фильтров связано с проверкой необходимых для записи или чтения полей.

5.4. Смарт-контракты

Смарт-контракты реализованы на языке Go. Основная сложность с написанием смарт-контрактов была из-за невозможности выбора структуры хранимых в блокчейне данных. Hyperledger Fabric предоставляет несколько запросов во внутреннюю базу данных, основными из которых являются GetState(key) и PutState(key, value). Таким образом невозможно реализовать хранение, например, списка id-номеров без отдельного ключа. Из-за этой особенности пришлось придумывать структуру хранения файлов внутри базы данных. Благодаря тому, что все id-номера документов генерируются, соответственно стандарту UUID, вероятность того, что ключи совпадут крайне мала, а также ключи имеют фиксированную длину. Это позволяет создать отдельный ключ для хранения списка id-номеров, который обновляется при создании нового документа.

Еще одна неприятная особенность возникает из-за использования SCP - там нет возможности настраивать Hyperledger Fabric, поэтому размер максимальной транзакции 100 Кб, но размер файлов с требованиями могут достигать нескольких Мб. Пришлось разбивать каждую запись в базу на несколько ключей и соответственно на несколько транзакций.

Также для хранения каждой версии, а еще для подтверждения необходимо создавать новый ключ. Структура ключа выглядит следующим образом: id/version/part, где part — это номер части. При этом в id/version/info лежит метаинформация о файле, в том числе количество частей, на которые разбит файл, а в id/last лежит последняя версия требований. Процесс добавления новой версии требований изображен

на рис. 5. Копирование из id/last при каждом обновлении требований обусловлено тем, что чтений, как правило, гораздо больше, чем записей, а доставать номер последней версии при каждом чтении достаточно долгая операция. Подтверждение требований и проверка осуществляются по ключу id/version/confirm. На рис. 6 кратко собрана структура ключа. Таким образом, операции check и list выполняют только 1 транзакцию чтения в блокчейн, операция подтверждения confirm выполняет 1 операцию записи в блокчейн, операция read выполняет столько транзакций чтения, на сколько частей был разбит файл, а операция create выполняет 2 транзакции чтения и от 2 до 10 транзакций записи.

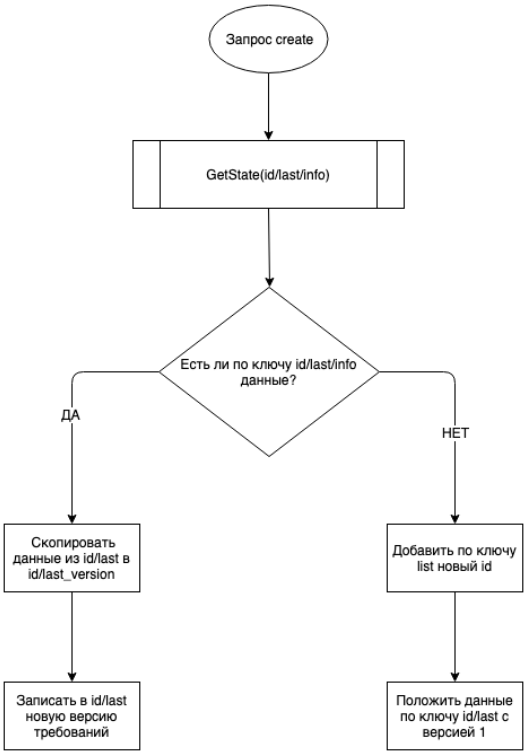

Рис. 5: Схема обработки запроса create смарт-контрактом

5.5. Алгоритм консенсуса

В качестве алгоритма консенсуса было решено использовать византийский отказоустойчивый алгоритм [6]. Он позволяет гарантировать корректную работу блокчейна в случае отказа менее трети узлов. Так как одной из основных задач сервиса является доверие контролиру-

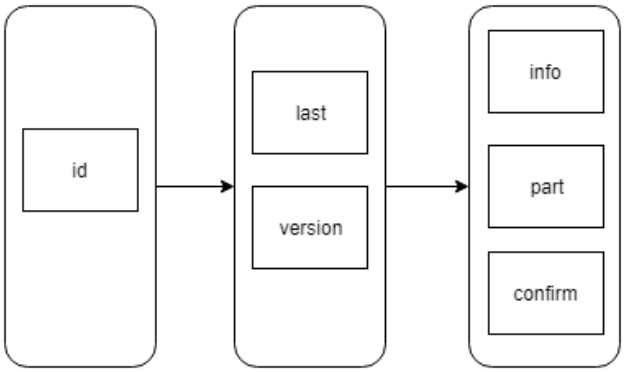

Рис. 6: структура ключа в базе данных

ющего органа, то была разработана архитектура расположения узлов так, чтобы участники сети не могли нарушить работу блокчейна. Таким образом, предлагается разместить 3 * N + 1 узел, N из которых будут стоять в SCP, N других у заказчика, а N + 1 последних у контролирующего органа. Схему можно посмотреть на рис. 7. За счет такой архитектуры и алгоритма консенсуса ни SAP, ни заказчик не смогут даже используя все свои имеющиеся узлы нарушить работу блокчейна.

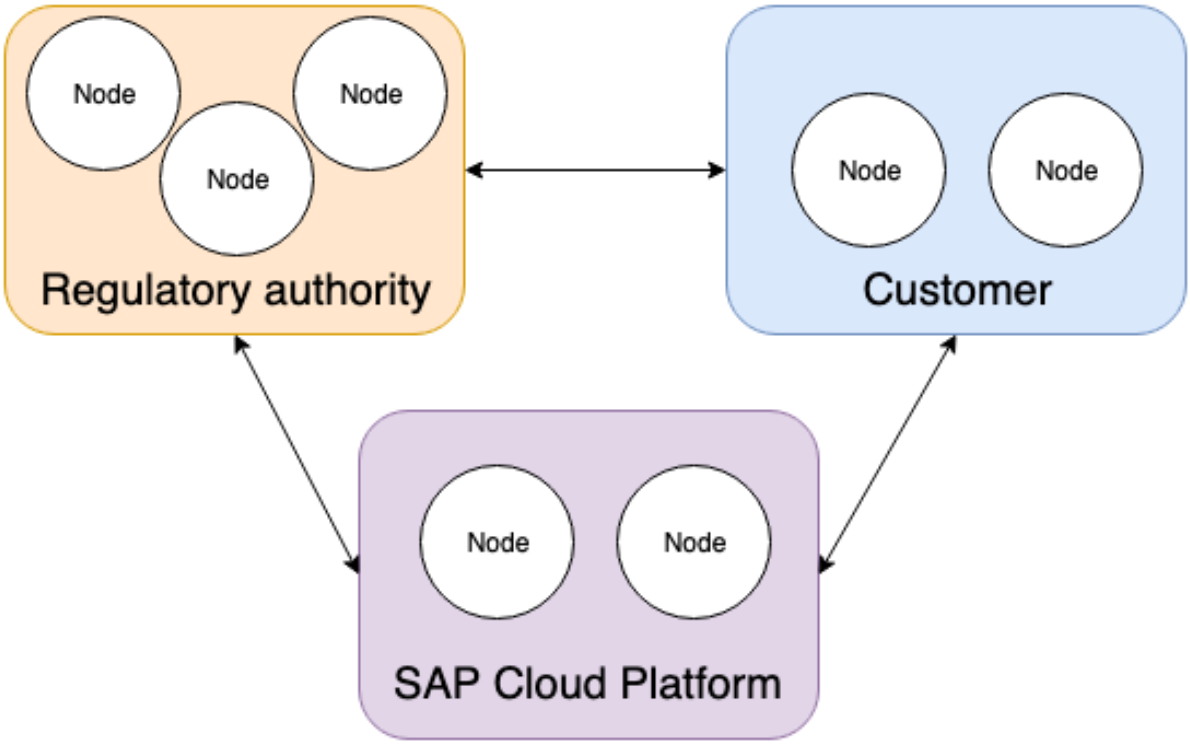

Рис. 7: Расположение узлов сети блокчейн у участников

Заключение

Результаты

В ходе выполнения выпускной квалификационной работы были достигнуты следующие результаты:

- проведен анализ похожих сервисов, сделан обзор функциональности таких решений, и на его основе составлены требования к сервису;
- разработана архитектура сервиса, выбран блокчейн-фреймворк Hyperledger Fabric и платформа для интеграции SCP;
- реализован сервис, включающий следующие модули:
  - плагин для Excel для скачивания и отправки требований в блокчейн, реализованный на языке VBA;
  - пользовательский интерфейс для просмотра истории изменения и одобрения требований, реализованный на языке JavaScript с использованием фреймворков Vue.js и Quasar;
  - приложение-прослойка для связи модулей сервиса с блокчейном, реализованное на языке JavaScript с использованием фреймворков Node.js;
  - смарт-контракты для создания транзакций в блокчейн, реализованные на языке Go.

Дальнейшие планы

На момент написания данной работы проводится общение с европейскими клиентами SAP на тему апробации реализованного сервиса в реальных условиях. Сервис находится в состоянии «пилота» - готов к представлению потенциальным заказчикам и требует незначительных доработок для внедрения. Одной из текущих задач является реализация авторизации пользователей в системе с помощью SAP Identity

Manager. Также в данный момент внедряется возможность шифрования файлов.

Список литературы

[1] Bairathi Rahul. Consensus & Endorsement in Hyperledger Fabric. — 2018. — URL: https://medium.com/coinmonks/consensus-endorsement-in-hyperledger-fabric-5dbf233b452c (online; accessed: 21.02.2019).

[2] Deloitte. Deloitte first to leverage blockchain technology to advance the certification industry. — 2017. — URL: http://tiny.cc/nvy65y (online; accessed: 21.02.2019).

[3] Express Framework. — 2019. — URL: https://expressjs.com (online; accessed: 21.02.2019).

[4] FDA. Drug Supply Chain Security Act. — 2019. — URL: https://www.fda.gov/drugs/drugsafety/drugintegrityandsupplychainsecurity/drugsupplychainsecurityact/ (online; accessed: 21.02.2019).

[5] IBM. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. — 2018. — URL: https://arxiv.org/pdf/1801.10228.pdf (online; accessed: 21.02.2019).

[6] Leslie Lamport Robert Shostak, Pease Marshall. The Byzantine Generals Problem. — 2019. — URL: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf (online; accessed: 21.02.2019).

[7] MediLedger. — 2019. — URL: https://www.mediledger.com/ (online; accessed: 21.02.2019).

[8] Multichain. Private blockchain white Paper. — 2015. — URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf (online; accessed: 21.02.2019).

[9] Nakamoto Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. —

2008. — URL: https://www.bitcoin.com/bitcoin.pdf (online; accessed: 21.02.2019).

[10] Node.js. Node.js Framework. — 2019. — URL: https://nodejs.org/ (online; accessed: 21.02.2019).

[11] PwC. PwC. — 2019. — URL: https://www.pwc.ru/ru.html (online; accessed: 21.02.2019).

[12] Quasar. Quasar Framework. — 2019. — URL: https://quasar-framework.org/ (online; accessed: 21.02.2019).

[13] SAP. SAP Cloud Platform. — 2019. — URL: https://cloudplatform.sap.com/index.html (online; accessed: 21.02.2019).

[14] SAP. SAP blockchain solutions. — 2019. — URL: https://www.sap.com/products/leonardo/blockchain.html (online; accessed: 21.02.2019).

[15] Storj. Storj.io. — 2019. — URL: https://storj.io/ (online; accessed: 21.02.2019).

[16] Storj wiki. — 2019. — URL: https://github.com/storj/storj/wiki (online; accessed: 21.02.2019).

[17] Technologies Primechain. BankChain. — 2018. — URL: http://www.bankchaintech.com/docs/brochures/bankchain.pdf (online; accessed: 21.02.2019).

[18] Vue.js. — 2019. — URL: https://vuejs.org/ (online; accessed: 21.02.2019).

[19] Wikipedia. Big Four accounting firms. — 2019. — URL: https://en.wikipedia.org/wiki/Big_Four_accounting_firms (online; accessed: 21.02.2019).

[20] Wikipedia. FDA. — 2019. — URL: https://ru.wikipedia.org/wiki/Food_and_Drug_Administration (дата обращения: 21.02.2019).