

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА**

на тему

**Конструктор создания распределенных
контрактов на основе технологии блокчейн**

Студент М.А. Веселова 03.06.2020 (подпись, дата) Веселова Марина Дмитриевна

Руководитель М.П. Синев 03.06.2020 (подпись, дата) Синев М.П.

Нормоконтролёр А.В. Кучин (подпись, дата) А.В. Кучин

Работа допущена к защите (протокол зас. кафедры от 03.06.2020г. № 12)

Заведующий кафедрой М.А. Митрохин (подпись) М.А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от 11.06.2020 № 2)

Секретарь ГЭК Е.И. Гурин (подпись) Е.И. Гурин
Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«30» апреля 2020 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

1. Студент Веселова Марина Дмитриевна гр.16ВВ1 факультета вычислительной техники.
2. Тема работы: «Конструктор создания распределенных контрактов на основе технологии блокчейн»

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020г.

3. Руководитель бакалаврской работы к.т.н. каф. ВТ ПГУ Синев М.П.

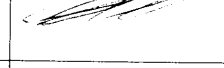
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать структуру базы данных для СУБД PostgreSQL, которая должна хранить данные о пользователе, созданных им спецификациях и экземплярах процессов, данные по выполнению процесса и правли отправки информации в Multichain. Разработать и отладить приложение «Конструктор создания распределенных контрактов на основе технологии блокчейн». Разработать способ транспортировки и хранения информации в блокчейн, а именно, транспортировка данных о процессе, спецификации, пользователе. Система должна взаимодействовать с базой данных и выполнять следующий функционал: создание спецификаций, процессов, выполняемых по спецификациям, обмен документами и их заверение с использованием технологии блокчейн.

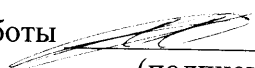
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить особенности технологии блокчейн, изучить платформы для разработки блокчейн приложений, изучить типы СУБД, виды архитектуры для веб-приложений. Необходимо изучить методологии проектирования веб-приложений, а также изучить функционал фреймворка Spring.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	30.04.2020 – 02.05.2020	
2.	Анализ требований	10	03.05.2020 – 05.05.2020	
3.	Выбор технологий, применяемых при проектировании и разработке	15	06.05.2020 – 10.05.2020	
4.	Разработка БД	15	11.05.2020 – 17.05.2020	
5.	Разработка приложения «Конструктор создания распределенных контрактов на основе технологии блокчейн»	30	18.05.2020 – 25.05.2020	
6.	Оформление пояснительной записки	10	26.05.2020 – 01.06.2020	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы  30.04.2020 г. Синев М.П.
(подпись, дата)

Задание к исполнению принял студент  30.04.2020 Веселова М.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 149 листов, 39 рисунков, 21 таблиц, 12 источников, 4 приложения.

БЛОКЧЕЙН, ДЕЦЕНТРАЛИЗОВАННАЯ БД, БД, СУБД, СМАРТ-КОНТРАКТ.

Цель работы – разработать систему, позволяющую конструировать создание распределенных контрактов на основе технологии блокчейн.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: авторизацию пользователя в системе, создание спецификации, создание процесса на базе спецификации, запуск процесса с приложением к нему документа, заверение документов в блокчейн. Хранение информации о пользователях, спецификациях, процессах, операциях должно осуществляться в локальной БД. Кроме того, должна быть реализована транспортировка данных с их хранением в децентрализованной распределенной базе данных.

В результате проведенной работы разработано приложение, удовлетворяющее поставленной задаче, а именно: разработана структура БД, разработанный механизм транспортировки, фиксирования и заверения данных с использованием технологии блокчейн. Приложение успешно протестировано и готово к использованию.

Перечень принятых сокращений

API (Application Programming Interface) – программный интерфейс приложения;

MIM — Man in the middle.

ID (Identifier) – идентификатор;

NoSQL (not only SQL) – не только SQL;

REST (Representational State Transfer) – передача состояния представления;

SQL (Structured Query Language) – язык структурированных запросов;

UML (Unified Modeling Language) – унифицированный язык моделирования;

БД – База данных.

ЕС – Едина система

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств.

ЛВС – Локальная вычислительная сеть.

ОРСУБД – объектно-реляционная система управления базами данных;

РСУБД – система управления реляционными базами данных;

СУБД – система управления базами данных.

ЮЛ – юридическое лицо.

ФЛ – физическое лицо.

Содержание

Перечень принятых сокращений.....	5
Содержание.....	6
Введение.....	6
1.Анализ предметной области.....	8
1.1Аналитический обзор технологии блокчейн.....	8
1.2Анализ платформ для разработки блокчейн приложений и выбор подходящей для разрабатываемой системы.....	14
1.3Анализ и выбор реляционной СУБД.....	20
1.4Анализ и выбор архитектуры	24
1.5Выводы по главе.....	27
2.Проектирование приложения.....	28
2.1Транспортировка данных.....	30
2.2Пользовательское приложение.....	34
2.3Выводы по главе.....	54
3.Инструкция пользователя.....	55
3.1Авторизация в системе.....	55
3.2Рабочий стол пользователя.....	56
3.3Экран «Процессы».....	58
3.4Экран «Новый процесс». Создание процесса.....	59
3.5Экран «Спецификации».....	61
3.6Экран «Новая спецификация». Создание новой спецификации.....	62
3.7Выполнение конкретного процесса между пользователями.....	64
3.8Выводы по главе.....	69
4.Безопасность жизнедеятельности.....	71
5.Заключение.....	72
Список использованных источников.....	73
Приложение А. Листинги программы на Java.....	74
Приложение В. Листинг SQL-файла.....	115
Приложение С. Пример выполнения REST API.....	128
Приложение D. Презентация.....	142

Введение

В настоящее время развитие информационных технологий тесно связано с общественной жизнью, их использование значительно облегчает работу в любой сфере общественной жизни за счет сокращения временных затрат, доступности удаленного взаимодействия между пользователями; кроме того, оцифровывание многих процессов дает более точные вычисления, а также долговременное и безопасное хранение огромного количества данных. Особенно важным на данный момент является применение вычислительных средств в экономической области, так как эта сфера сейчас терпит нестабильное положение на мировой арене, поэтому автоматизация анализа больших объемов данных, их хранение, передача и использование в существующих экономических процессах является основной задачей цифровой экономики. Актуальным решением возникшей задачи является разработка и проектирование систем или приложений, упорядочивающих все протекающие процессы и исключающих возможность бесконтрольной модификации информации.

В данной работе будет рассмотрена реализация веб-приложения «Конструктор создания распределенных контрактов на основе технологии блокчейн». Данное приложение направлено на автоматизацию процесса обмена документов и их заверение участниками системы в рамках формализованных процессов. Также будет произведен сравнительный анализ уже существующих систем документооборота на базе блокчейн, выявлены их преимущества и недостатки.

1. Анализ предметной области

При разработке программных продуктов необходимо провести анализ существующих вариантов для решения задачи. Данный раздел включает в себя обзор технологии блокчейн: её виды и применение. Производится анализ и выбор платформы для разработки приложения, использующего технологию блокчейн, СУБД и архитектуры системы. Кроме того, в каждом рассматриваемом подразделе обосновывается выбор того или иного технологического решения выполнения поставленной задачи.

1.1 Аналитический обзор технологии блокчейн

Внедрение и развитие цифровой экономики в *it* сфере характеризуется применением таких технологий как: искусственный интеллект, большие данные (*Big Data*), блокчейн (*Blockchain*)[1][2]. Особое внимание в последнее время уделяется технологии блокчейн, которая представляет собой децентрализованную распределенную базу данных, в которой хранится информация о всех транзакциях, выполненных пользователями системы. Все данные помещаются в блоки, у каждого из которых есть метка времени, ссылка на предшествующий блок, таким образом формируются цепочки блоков. Важно отметить, что количество блоков в цепочке неограниченно, все записи упорядочены. При этом все цепочки хранятся на разных компьютерах, а доступ пользователя к определенной части цепочки осуществляется при помощи закрытого ключа. Несмотря на используемое условие хранения, благодаря современным методам шифрования, отдельные записи надежны защищены от копирования или модифицирования другими пользователями, тем самым гарантируя конфиденциальность и целостность данных.

Алгоритм функционирования технологии блокчейн представлен на рисунке 1.

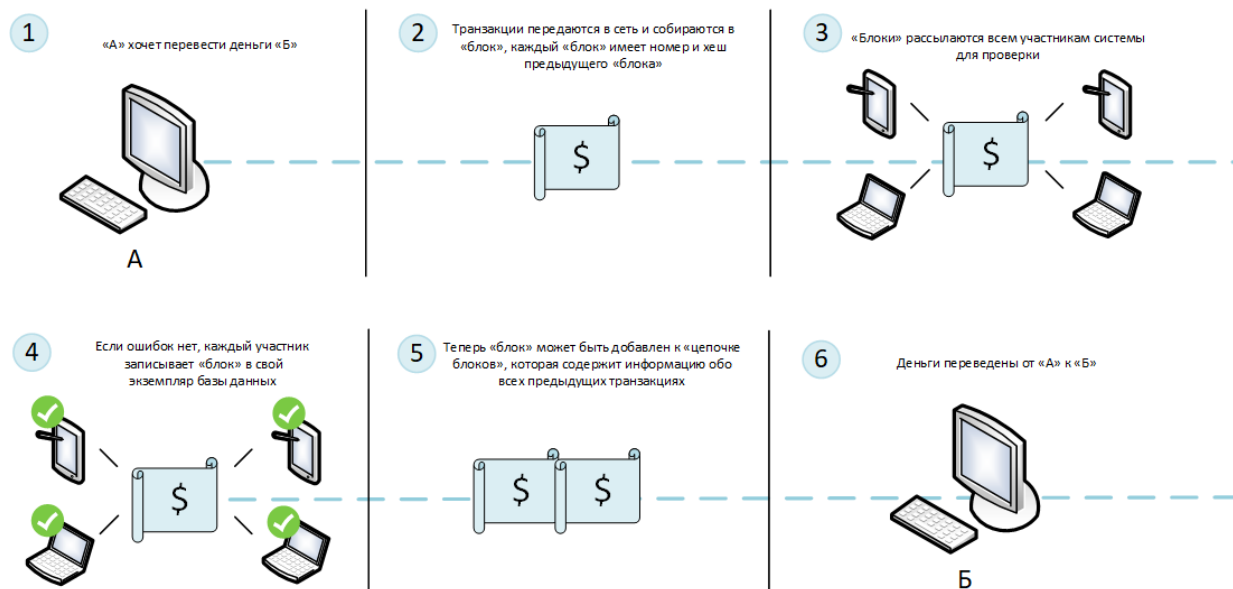


Рисунок 1 – Схема алгоритма совершения транзакции

Первоначально пользователь запрашивает транзакцию, далее выполняется формирование записи данных о транзакции в виде блока. При этом, каждый блок обязательно включает указатель на предыдущий блок, метку времени и заголовок, который будет использоваться в следующем блоке для продолжения цепочки блоков. После того, как блок сформирован, выполняется его рассылка всем участникам системы, это необходимо для верификации блока и его дальнейшей записи в экземпляр базы данных каждого пользователя. Успешным завершением транзакции является добавление блока к цепочке блоков.

Хранение информации в виде децентрализованной базы данных открытой для любого пользователя системы обладает рядом существенных преимуществ [3]:

- данные легко проверяемы;
- невозможность повреждения базы данных, в виду отсутствия централизованного хранилища;
- устойчивость к системным сбоям, поскольку данные в блокчейн технологии находятся на различных устройствах, подключенных к распределенной сети;

— отсутствие единой точки отказа, таким образом никакой один узел не влияет на работу и доступность сети.

На данный момент множество компаний ведут разработку бизнес-приложений, основываясь на блокчейн, так как данная методика позволяет:

- исключить такой тип атаки, как *ММ* («человек посередине»);
- более точно зафиксировать время выполнения процесса, операции для решения споров, например, при единовременном размещении документов;
- хранить цифровые сертификаты, в том числе электронные подписи, обеспечивающие защищенный доступ к важным ресурсам;
- обеспечить безопасное совершение сделки;
- легко подтвердить подлинность товара, работы или услуги, а также права собственности любого типа;
- ускорить процесс заключения сделок.

Создателями данной технологии являются Стюарт Хабер и У. Скотт Шторнетт, которые разработали решение для цифровых документов со штампом времени, это позволило избежать оформления данных «задним числом». Далее в 1992 году к уже имеющимся наработкам было добавлено применение хеш-дерева – дерева Меркала, дав возможность объединять несколько документов для хранения в один блок. Тем не менее, новаторская идея не стала широкого распространения и применения до 2008 года, пока Сатоши Накамото не создал биткоин – финансовую систему одноранговой сети, которая базировалась на модернизированной технологии Стюарта Хабера и У. Скотта Шторнетта.

На сегодняшний день, рассматривая жизнеспособность технологии блокчейн в широком масштабе, шанс получить однозначный и четкий ответ невелик[4]. Однако, несмотря на свое сравнительно недавнее появление в мире информационных технологий, блокчейн занимает одну из лидирующих позиций. Кроме того, многие новые разработанные технологии по хранению,

передаче и защите данных базируются на алгоритмах блокчейна. Это факт несомненно подтверждает, что технология распределенных реестров набирает популярность использования в современном мире.

1.1.1.1 **Использование технологии в России и мире**

Несмотря на наличие огромных преимуществ технология блокчейн всё еще слабо внедрена, поскольку является новейшей разработкой. Для ее дальнейшего использования по мнению представителей различных компаний необходима сформированная экосистема, которая докажет экономическую эффективность данного проекта.

Однако, в России и за рубежом блокчейн всё же нашел свое применение не только больших корпорациях, но и в государственных структурах. Например, пилотный проект «Городская транспортная карта «УмКа» разрабатываемый компанией «Сервионика». Целью данного проекта является создание масштабируемой транспортной системы, которая позволит минимизироваться транзакционные издержки. Также в России существует блокчейн-платформа лояльности «*BioCoin*», которая представляет собой накопительную систему баллов при совершении покупки у компаний, присоединившихся к данной платформе. Баллы могут быть потрачены в сетях компаний-участников программы.

Множество разработок ведется на базе протокола *Ethereum*, например, российская национальная блокчейн-сеть «Мастерчейн», позволяющая выполнять обмен ценными цифровыми данными и информацией о них между участниками сети. Механизм *Ethereum* был модернизирован под требования российской криптографии. «Мастерчейн» является продуктом Ассоциации ФинТех, в которую входят такие крупные корпорации и предприятия как: «Банк России», «Сбербанк», «ВТБ», «Газпромбанк», «КИВИ банк», «Тинкофф банк», «Ростелеком», ПАО «МТС» и др.

Кроме данного проекта на базе *Ethereum* работает блокчейн платформа для «*S7 Airlines*» и «Альфа-банк», которая направлена на автоматизацию

торговых операций по приобретению авиабилетов при помощи смарт-контрактов. В данном проекте клиенты «*S7 Airlines*» совершают покупку билетов, оплачивая их при помощи онлайн сервисов. Банк получает информацию о выполняемых операциях при помощи приватного блокчейна.

На мировой арене перспективность развития блокчейн технологии была отмечена компанией Microsoft, которая разработала блокчейн экосистему «*Airalab*» в облаке *Microsoft Azure Blockchain-as-a-Service (BaaS)*. Данный проект создал биржу карбоновых контрактов, платформа проекта может быть использована по всему миру, где утвержден Киотский протокол. Благодаря «*Airalab*», множество стран сократили и урегулировали выбросы парниковых газов. Помимо данного проекта *Microsoft* и *Bank Hapoalim* создали решение для автоматизации работы с клиентской документацией, таким образом, теперь обновление данных в системе банка можно пройти без личного присутствия. Реализацией многочисленных проектов на блокчейн-основе славится и корпорация IBM, которая в начале 2018 года выпустила порядка 400 блокчейн-проектов в самых различных отраслях: от экономики до здравоохранения. Особенное внимание стоит уделить совместной с *AIG* разработкой «Умного» страхового договора, который должен покрывать страховые события на международном уровне.

1.1.1.2 Применение блокчейна в документообороте

Алгоритм функционирования смарт-контрактов был создан и описан в 1996 году американским криптографом и программистом Ником Сабо ещё до появления технологии блокчейн. По определению Н.Сабо интеллектуальный контракт – это цифровой протокол для передачи информации, использующий математические алгоритмы для автоматического выполнения транзакции после выполнения установленных условий и полного контроля процесса. С появлением *bitcoin* в цифровой индустрии был совершен скачок, подтолкнувший к дальнейшей разработке смарт-контрактов, в частности смарт-контракты использованы в проекте *Ethereum*.

Как уже было описано выше, обязательства участников контракта предоставляются по принципу «если..., то...». В заключении договора могут участвовать два и более участников, которые являются либо организациями, либо отдельными лицами. Ключевым вопросом при заключении и подписании смарт-контрактов является подпись транзакции, которая может быть выполнена посредством использования пары «публичный ключ-приватный ключ», а именно:

- на каждой стороне должна быть сгенерирована пара «публичный ключ-приватный ключ»;
- в договоре или соглашении должна быть установлена возможность обмена сообщениями, подписанными простой электронной подписью;
- в договоре необходимо указать реквизиты сторон блокчейн-сети и установить порядок формирования документа и его подписания.

Как только заданные в договоре условия выполнены, смарт-контракт производит транзакцию и гарантирует соблюдение соглашения.

Алгоритм заключения смарт контракта изображен на рисунке 2.



Рисунок 2 – Пример заключения смарт-контракта

Применение смарт-контрактов обширно, они смогут облегчить процедуры обмена денег, товаров, услуг, недвижимости, ценных бумаг и других активов. Однако для работы со смарт-контрактами важно отметить, что все активы находятся в цифровой экосистеме.

Преимуществом внедрения смарт-контрактов в повседневную жизнь пользователя является:

- исключение третьих лиц при заключении контракта, таким образом, не нужно тратить лишнее время на связь с «посредником», а также денежные средства за его услуги;
- повышенная безопасность хранения смарт-контракта, поскольку он будет находиться в зашифрованном виде в распределенном реестре, шифрование данных также обеспечит анонимность сторон-участников;
- невозможность изменения условий контракта после или в момент исполнения контракта, что позволяет избежать конфликтов различного рода и подтасовки данных;
- автоматизированность процесса, отсутствие лишнего ручного заполнения множества форм, что обеспечивает высокую точность при исполнении контракта.

Использование смарт-контракта наиболее эффективно в отраслях, связанных с финансами, страхованием, электронной коммерцией, аудитом и налогообложением, выборами и пр.

1.2 Анализ платформ для разработки блокчейн приложений и выбор подходящей для разрабатываемой системы

Существует множество различных площадок для разработки приложений, в основе которых лежит технология блокчейн[5]. Как было описано выше блокчейн-технологии не стоят на месте, а постоянно модернизируются, ввиду изменений создаются новые платформы под разработку приложений или обновляется функционал существующих. Часто используемыми являются:

- *Sidechains*;
- *Ethereum*;
- *Hyperledger Fabric*;
- *EOS*;
- *DocChain*;
- *Multichain*.

1.2.1 *Sidechains*

Sidechains – «боковые» или «касательные» цепочки, была предложена в 2015 году компанией *Blockstream*. Её идея заключается в разделении блокчейнов на главный и дочерние. Таким образом, пользователь может вывести из первичного (главного) блокчейна свои активы и переместить их на дополнительный (дочерний) «сайдчейн», где может выполнять операции с различными видами криптовалют. Использование метода разделения блокчейнов делает пользователя свободнее в работе с различными цифровыми активами, позволяя проводить платежи в других системах, все транзакции аналогично блокчейну фиксируются, что позволяет избежать утечки средств или их параллельное использование в разных системах. Однако, данный проект имеет весомый недостаток: он направлен лишь на операции с цифровой валютой, что является узкой сферой, если рассматривать масштабность возможностей цифровых технологий. Схема функционирования данного способа отображена на рисунке 3.

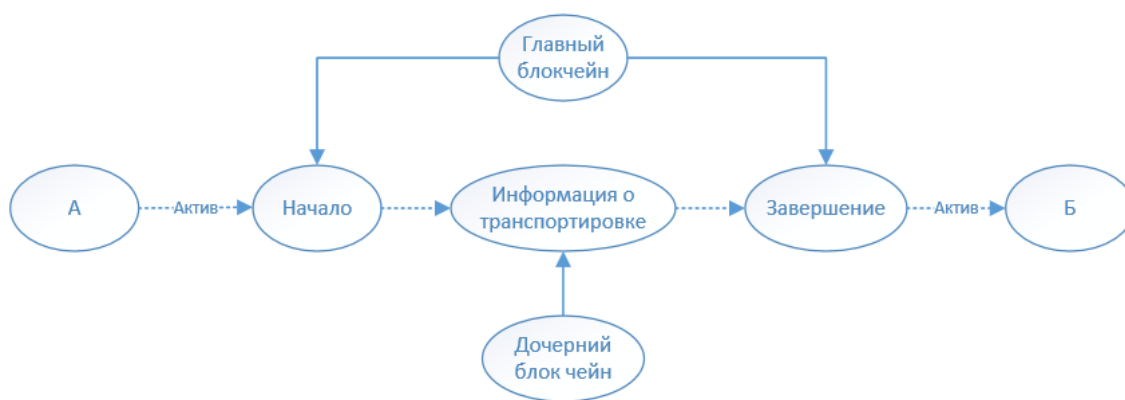


Рисунок 3 – Схема функционирования *Sidechains*

1.2.2 *Ethereum*

Ethereum – распределенный компьютер, является работой нашего соотечественника, программиста Виталика Бутерина. Блокчейн *Ethereum* представляет собой систему состояний транзакций, то есть систему, обрабатывающую вводимую информацию на основании которой происходит преобразование из первоначального в состояния в новое. Состояние в *Ethereum* имеет миллионы транзакций, сгруппированные в блоки, который скреплены друг с другом образуя цепочку блоков. Чтобы транзакция была корректной, вызывается ее переход из одного состояния в другое, этот переход обусловлен процессом проверки – майнингом.

Этот проект изменяет и расширяет определение транзакции, в отличие от биткоина, здесь транзакция представляет собой «умный» контракт – самовыполняющуюся программу. Взяв блокчейн за основу, разработчики *Ethereum* гарантируют валидность не только операций с денежными средствами, но и условий, а также сделок. Сделкой считается любая операция, выполняющаяся в формате «если ..., то...», благодаря такому принципу, заключение контракта упрощается – этап обращения к юридическим и нотариальным органам для заверения исключается, а значит, и время, затрачиваемое на совершение сделки, уменьшается. Каждый смарт контракт, как и традиционный имеет набор условий, соглашений, обязательств сторон и штрафы за их нарушения, однако, «умные» контракты могут выполняться параллельно сразу несколькими пользователями сети, что еще раз подчеркивает преимущества использования данной технологии. *Ethereum* является одной из самых популярных платформ, использующих смарт-контракты, но в данном случае популярность платформы выступает скорее её недостатком, чем преимуществом, поскольку при одновременном нахождении нескольких пользователей в сети, выполняющих какие-либо операции, повышается нагрузка сети, этот аспект может привести к нестабильной работе.

1.2.3 *Hyperledger*

Hyperledger – многомодульный проект по созданию распределенного реестра от *The Linux Foundation*. Его возможности позволяют создавать приватный блокчейн, поддерживают конфиденциальность транзакций, доступных к просмотру, лишь разрешенному кругу пользователей, имеющих необходимые ключи шифрования. В данном проекте нет необходимости в использовании майнинга и выпуска токенов, поскольку узлы, выполняющие проверку, самостоятельно сигнализируют друг другу о произведенных операциях, а также самостоятельно достигают консенсуса и создают новые блоки для хранения данных. *Hyperledger* имеет множество фреймворков и расширений, которые используют крупные компании по всему миру. Аналогично *Ethereum* самым главным недостатком *Hyperledger* является его популярность, которая не дает полной уверенности в безотказности работы системы. Алгоритм работы *Hyperledger* представлен на рисунке 4.

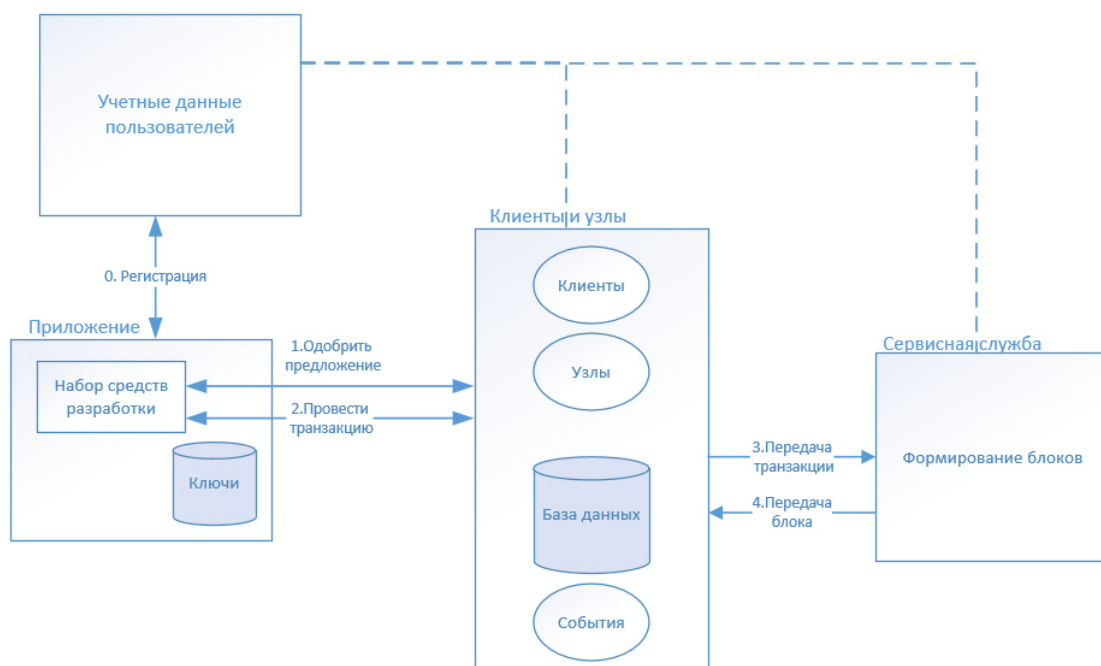


Рисунок 4 - Алгоритм работы *Hyperledger*

1.2.4 *EOS*

EOS менее популярна по сравнению с *Ethereum* и *Hyperledger Fabric*, но не менее эффективнее. Впервые компания *block.one* объявила о своем

продукте летом 2018 года, а также предоставила свободный доступ по лицензии *MIT* (лицензия открытого программного обеспечения). Важным свойством *EOS* является предоставление вычислительных ресурсов для разработчиков и возможность работы на основе *Web Assembly: Binaryen* и *WAVM*. Смарт-контракты *EOS* способны при помощи ставок на токены, токен-голосованием распределять квоты вычислительных ресурсов. Данная платформа помогает при принятии решений в планировании, аутентификации и создании интернет-приложений, поскольку сервисы *EOS* дают возможность разработчикам уделить большее внимание бизнес-процессам. Недостатком *EOS* является то, что проект находится в разработке, а значит финальной версии платформы нет, этот факт может подвергнуть сомнениям инвесторов проекта.

1.2.5 *DocChain*

Самым молодым проектом является *DocChain*, основанный на криптографических алгоритмах. Передача документов между пользователями осуществляется при помощи открытого и закрытого ключа, что позволяет надёжно и безопасно передавать доступ к документам. Кроме того, поддерживается аудит текущего и исторического доступа пользователя в систему. Важным аспектом можно выделить тот факт, что *DocChain* не является криптовалютой и не использует токенов, а это значит полное отсутствие комиссий за транзакцию.

Для более подробного представления алгоритма функционирования платформы *DocChain*, необходимо рассмотреть: алгоритм загрузки данных для хранения и алгоритм передачи данных между пользователями.

При загрузке данных используя *DocChain*, приложение зашифровывает ключом сообщение. Далее зашифрованное сообщение отправляется на узел и фиксируется в блокчейне, после чего вновь создается ключ для документа, который шифруется при помощи загружаемого закрытого ключа

пользователя, это позволяет защищенно делиться ключом документа с собой, создавая первую точку доступа в истории обмена.

При передаче документа первоначально приложение расшифровывает ключ документа, используя закрытый ключ пользователя-отправителя после чего, шифрует с помощью открытого ключа пользователя-получателя и направляет ему. Получатель расшифровывает ключ от данных при помощи своего ключа, после чего дешифрованный ключ можно применить для расшифровки содержания контракта.

Однако, данная платформа является недавней разработкой и несмотря на все свои преимущества еще до конца не изучена и не проверена.

1.2.6 *Multichain*

Завершает список популярных платформ *Multichain* – работа компании *Coin Sciences* [6]. Платформа предусмотрена для корпоративных клиентов, основана на блокчейне и используется для создания контролируемых блокчейнов в среде множества бизнесов. У программного обеспечения открытый код, что дает ей преимущество в бесплатном использовании. Также платформа оснащена свободным и открытым ценообразованием, активно обновляющимся в *GitHub*. *Multichain* популярна за счет создания безопасных, частных сетей. Довольно быстрая разработка может вестись на таких популярных языках программирования, как: *Python*, *C#*, *C++*, *JavaScript*.

Одним из важных преимуществ помимо открытости кода является возможность у конечного пользователя, которая позволяет контролировать приватность и публичность цепи, целевое время блоков, способы взаимодействия сторон, выбор этих сторон, максимальный размер блока и метаданные, которые можно включать в транзакции. Это происходит посредством присвоения специальных метаданных сетевой транзакции, которая необходима при предоставлении или аннулировании клиенту тех или иных прав. Благодаря тому, что все изменения привилегий записаны в

метаданные транзакции, происходит быстрое распространение данной информации на все узлы сети, создавая консенсус текущего состояния.

Вывод по данному разделу: наиболее подходящей платформой для разрабатываемого продукта является платформа *Multichain*, поскольку она является более надежным, универсальным для любого бизнеса и при этом открытым решением.

1.3 Анализ и выбор реляционной СУБД

Реляционная СУБД – это система управления реляционными базами данных. Реляционные базы данных – это набор данных с предопределенным связями между ними.

В настоящее время существует множество популярных и развивающихся баз данных, например, *MySQL*, *PostgreSQL*, *MS SQL Server*, *Oracle Database*.

1.3.1 *MySQL*

MySQL является быстрым многопоточным, многопользовательским *SQL*-сервером базы данных. Данное программное обеспечение постоянно обновляется дополнительным функционалом для усовершенствования и облегчения работы пользователей. *MySQL* предполагает как платную так и бесплатную версию, является довольно гибкой СУБД, поскольку дает возможность использования множества типов таблиц.

Среди достоинств *MySQL* можно выделить:

- бесплатную версию для личного пользования;
- доступную к пониманию документацию;
- безопасность;
- взаимодействие с другими базами данных, например, *DB2* и *Oracle*.

Недостатки:

- угрозы надежности;
- платная версия для коммерческого использования;
- существенные ограничения функционала в бесплатной версии;

— не поддерживает форматы *XML* и *OLAP*.

Исходя из выше описанных преимуществ и недостатков данный продукт используется чаще всего для мелких и средних проектов, это обусловлено ограниченностью функционала и коммерческой основой расширенной версии.

1.3.2 *Oracle Database*

Oracle Database – объектно-реляционная СУБД компании *Oracle*. Такие СУБД реализуют объектно-ориентированный подход: объекты, классы, наследование реализованы внутри базы данных и языке запросов. *Oracle Database* ведет свою работы с конца 1970-х годов. В настоящее время выпущена последняя версия продукции «19с». Суффикс «с» означает «*cloud*», то есть «облако». Впервые «облачная» версия была выпущена в 2013 году, данная версия обеспечивает подключение БД через облачные сервисы, такой подход дает весомые преимущества: возможность хранения огромного количества данных и размещение на множестве серверов.

Из достоинств *Oracle Database* можно выделить:

- надежность;
- мощность;
- постоянное усовершенствование и развитие продукта совместно с современными технологиями.

Недостатками являются:

- высокая стоимость продукта для юридических лиц;
- необходимость в мощных, производительных аппаратных ресурсах.

1.3.3 *MS SQL Server*

Microsoft SQL Server – продукт, разработанный корпорацией *Microsoft*; система управления реляционными базами данных. Основной язык для написания запросов – *Transact-SQL*.

Transact-SQL – это процедурное расширение языка *SQL*. В этой модификации стандартного языка реляционных баз данных реализованы возможности:

- управление операторами;
- локальные и глобальные переменные;
- дополнительные функции для обработки строк, дат, математики;
- поддержка аутентификации *Microsoft Windows*.

История *Microsoft SQL Server* ведется со второй половины 80-х годов, в настоящее время актуальна версия 2016 года *SQL Server 2016*, поддерживающая гибридную обработку транзакций – это совмещение стандартной и аналитической обработки, и их выполнение в реальном времени, исключая дублирование данных. Также база данных работает с облаком.

Достоинствами *Microsoft SQL Server* являются:

- стабильность работы системы;
- визуализация на мобильных устройствах;
- интеграция с другими продуктами корпорации *Microsoft*;
- легкость в использовании.

Недостатки:

- высокая стоимость для юридических лиц;
- не исключено возникновение проблем при интеграции импорта файлов;
- повышенная ресурсоёмкость.

1.3.4 *PostgreSQL*

PostgreSQL – свободная объектно-реляционная СУБД. Имеет широкий круг реализаций для большого количества *UNIX*-подобных платформ, включая *AIX*, различных *BSD*-системы, *HP-UX*, *IRIX*, *Linux*, *macOS*, *Solaris/OpenSolaris*, *Tru64*, *QNX*, а также для *Microsoft Windows*[7]. Началом разработки данного продукта является 1986 год.

Работа в *PostgreSQL* базируется на использовании языка запросов SQL. Последней версией программного обеспечения является версия 12.2.

В последнее время отмечается большой прирост крупных коммерческих компаний, использующих данную СУБД для реализации своих проектов.

Можно выделить следующие достоинства *PostgreSQL*:

- работа с огромнейшими объемами данных;
- поддержка формата JSON;
- простота и понятность сопутствующей документации;
- большое количество встроенных функций;
- бесплатная версия для личного и коммерческого использования;
- хорошая масштабируемость.

Среди недостатков можно выделить:

- неэффективную репликацию данных;
- трудности с обновлением ПО.

На основании проведенного анализа популярных средств для работы с базами данных можно сделать вывод, что наиболее подходящим решением является *PostgreSQL*.

1.4 Анализ и выбор архитектуры

При проектировании веб-приложения необходимо учесть на базе какой архитектуры оно будет выстроено[8]. Выделяют два типа архитектуры: монолитная и микросервисная[9].

1.4.1 Микросервисная архитектура

Такой тип архитектуры характеризуется делением системы на независимые компоненты, модули, которые взаимодействуют между собой при помощи *API*. Каждый независимый модуль выполняет свою определенную задачу. Схема данной архитектуры представлена на рисунке 5.

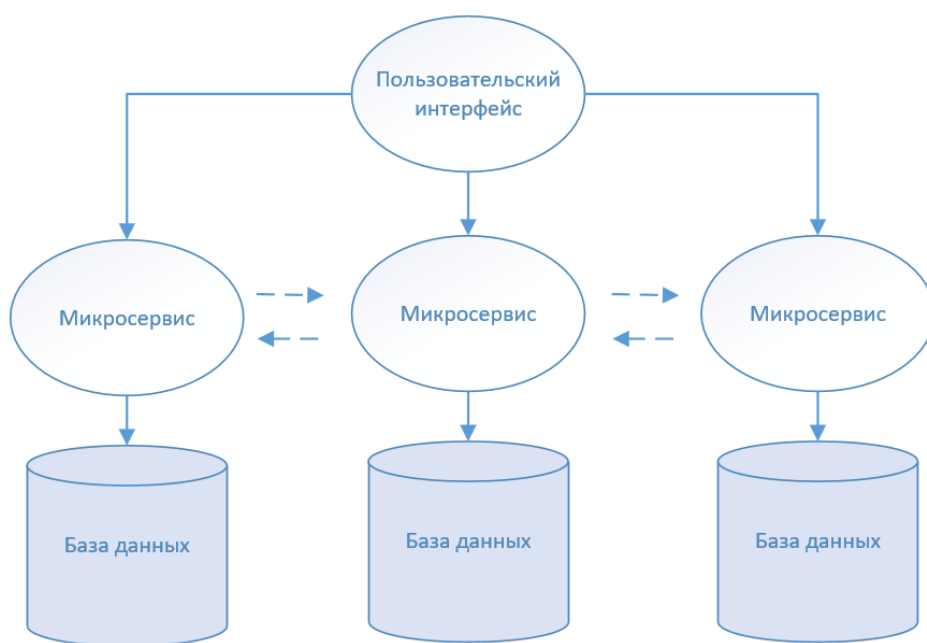


Рисунок 5 – Микросервисная архитектура

Микросервисная архитектура наделена следующими свойствами:

- гетерогенность: дает возможность использовать различные языки программирования и технологии в системе;
- децентрализованное управление данными: обусловлено тем, что каждый микросервис содержит свой набор данных, который доступен другим микросервисам через соответствующий интерфейс;

- независимость инфраструктуры: характеризуется обособленностью каждого микросервиса, то есть, изменять и разворачивать его можно независимо от других;

- масштабируемость: позволяет повысить производительность системы локально, а именно, расширяет только те микросервисы, которые необходимо улучшить.

Несмотря на все вышеуказанные преимущества, данный подход имеет и недостатки, например:

- сложность разработки и создания инфраструктуры;
- необходимо правильно подобрать транспортный протокол данных для наиболее эффективной передачи данных между микросервисами;
- сложность поддержания синхронизации между сервисами для строгой согласованности.

1.4.2 Монолитная архитектура

Монолитная архитектура представляет проектирование системы как большого единого связного модуля. Каждый компонент модуля взаимосвязан друг с другом и использует общую память и общие ресурсы. Схема монолитной архитектура изображена на рисунке 6.



Рисунок 6 – Монолитная архитектура

У данной архитектуры можно выделить следующие достоинства:

- простая реализация, управление и развертывание;
- более простое масштабирование при помощи балансировки нагрузки;
- облегченная эксплуатация;
- легкость при обработке ошибок;
- достаточно высокая производительность для малых и средних по размеру систем.

Однако, как и любая технология или подход монолитная архитектура тоже имеет свои недостатки, например:

- возрастающая сложность при увеличении масштабов системы;

- внесение изменений довольно трудоемко;
- сбой одного компонента может привести к сбою всей системы.

Ввиду того, что разрабатываемая система является небольшой по своему масштабу и функционалу и не требует модульности при разработке было принято решение использовать монолитную архитектуру.

При выборе архитектуры системы были учтены все риски от использования того или иного подхода.

1.5 Выводы по главе

В данной главе был выполнен анализ технологии блокчейн, обоснована актуальность её использования с выявлением преимуществ. Был проведен обзор различных платформ для разработки блокчейн приложений, среди которых происходил выбор наиболее подходящей платформы. Кроме того, были проанализованы и выбраны: средства для разработки реляционной СУБД приложения и архитектура приложения.

Работа, описанная в данной главе, является важным этапом при проектировании продукта.

2. Проектирование приложения

Разрабатываемое веб-приложение «Конструктор создания распределенных контрактов на основе технологии блокчейн» позволяет пользователям взаимодействовать между собой при помощи обмена документами, их подписания в рамках формализованных процессов. Каждый авторизованный пользователь системы имеет возможность публиковать свои процессы, а также формировать процессы по существующим доступным процессам – шаблонам, который были созданы другими пользователями.

Под процессом понимается выполнение заданного спецификацией набора действий между конкретными пользователями системы. Процесс содержит информацию об участниках, прикрепленных документах, статусе выполнения действий. В рамках процесса выполняются следующие действия:

- создание нового процесса;
- публикация документов в рамках процесса с использованием хранилища или локальной загрузки;
- подпись документов в рамках процесса;
- просмотр статуса выполнения процесса;
- остановка процесса.

Под спецификацией понимается набор инструкций к процессу:

- максимальное время длительности процесса;
- наличие принудительного прерывания процесса создателем;
- порядок выполнения действия: последовательный, параллельный;
- ограничения на выполнение тех или иных действий пользователем с определенной ролью;
- условия завершения процесса.

Как было указано ранее документы можно загрузить сразу в процесс – локально или сначала сохранить их в своем хранилище, а затем прикрепить к процессу.

Таким образом система хранит спецификации процессов, экземпляры процессов, экземпляры документов, подписи и финальные статусы процессов в распределенном блокчейне, исключая возможности подмены документов, благодаря автоматически установленным временным меткам. Блокчейн формируется с использованием криптоалгоритмов и протокола консенсуса из *Bitcoin Core*, расширенного для валидации соответствия статуса процесса спецификации.

Электронные подписи, заверяющие экземпляры объектов внутри записей, создаются при помощи криптоалгоритмов ЕС, ГОСТ, которые разрешены для конкретного экземпляра системы.

В самом блокчейне фиксируются следующие типы записей:

- определение права участника;
- создание спецификации процесса;
- маркер удаления спецификации процесса;
- создание экземпляра процесса;
- документ;
- подпись документа;
- маркер завершения экземпляра процесса.

В рамках *MultiChain* все записи, которые фиксируют создание спецификации процесса или удаления спецификации, записываются в поток с идентификатором. Для каждого нового процесса создается отдельный поток, содержащий связанные с ним записи.

Разработку системы можно разделить на две основные части проектирование и реализация пользовательского веб-приложения и решение транспортировки данных на базе открытой платформы *MultiChain*.

2.1 Транспортировка данных

Приложение основано на принципах организации децентрализованной одноранговой сети, где каждый узел синхронизирует весь блокчейн[10]. Схема сети изображена на рисунке 7.

Децентрализованная одноранговая сеть представляет собой сеть, не имеющую единого сервера хранения данных, где каждый узел равноправен, то есть является как клиентом, так и выполняет функции сервера. Применение такой сети позволяет сохранить её работоспособность при любом количестве узлов.

Данный подход может быть обеспечен при использовании открытой платформы *MultiChain*.

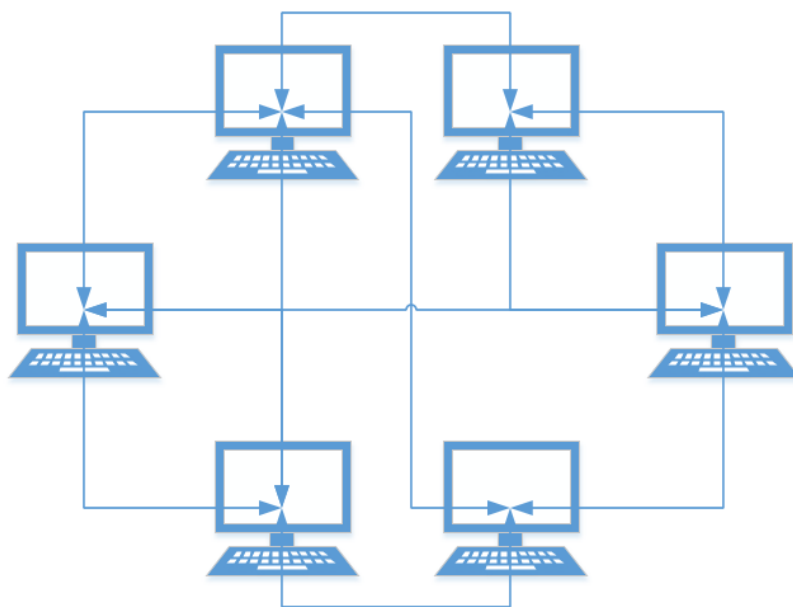


Рисунок 7 – Схема сети

MultiChain в данном случае выступает как ядро транспортной системы, которое позволяет выполнять процессы передачи и хранения данных в распределенной цепочке благодаря модели потоков. В *MultiChain* может содержаться любое количество потоков, каждый из которых есть независимый набор элементов. Элементы потока состоят из:

- издателей;

- ключа;
- данных;
- информации о транзакции и блоке элемента.

Как уже было описано ранее, блок представляет собой группу подтвержденных, захешированных транзакций. Блок состоит из заголовка и списка транзакций, заголовков – хеша, хеша предыдущего блока(при условии, что данный блок не первый), хеша транзакции и дополнительной служебной информации. Блок считается подтвержденным с возможностью включения во все репликации сети, если все узлы подтвердили его подлинность. Далее попадает в поток.

Поток может быть создан исключительно клиентом, у которого есть на это права, соответственно после создания поток имеет два состояния: открыт или закрыт. Открытость потока свидетельствует о том, запись в него доступна любому, у кого есть разрешение на отправку транзакций. Закрытость означает, что над потоком есть администрирование, которое устанавливает разрешение или запрет на запись.

Каждый узел сети может сам установить, на какой поток подписаться, в этом случае блокчейн гарантирует, что узлы, подписанные на один поток, будут получать одни и те же данные. Для хранения данных в приложении созданы данные потоки:

- *cds.certificate* – поток содержит данные о сертификатах пользователей,
- *cds.specification* – поток с данными о спецификациях,
- *cds.process* – поток со всей информацией о процессах.

Любой поток содержит элемент – результат добавления записи об операции в блокчейн, рассмотрим его структуру более подробно.

Результат добавления записи об операции в блокчейн «ChainResult» представлен в виде класса, содержащего:

- переменные «txId» (тип – *string*), «inChain» (тип – *boolean*), «rejectInfo» (тип – *string*), «chainStatus» (тип – *enum*);

- методы *get* и *set* на каждую указанную переменную.

«TxId» представляет собой идентификатор транзакции в блокчейне, «inChain» – флаг нахождения в блокчейне, «rejectInfo» – причину отклонения добавления в блокчейн, а «chainStatus» – статус сертификата пользователя, переменная принимает значения: *UNKNOWN* – неизвестен или отсутствует, *SEND* – отправлен на подтверждение, *FAIL* – не подтвержден, *SUCCESS* – подтвержден, программный листинг представлен в приложении А.

Взаимодействие с платформой *MultiChain* происходит по средством *API JSON-RPC*, который реализует протокол состоящий из набора различных типовых команд, описанных в файле *CommandElt.java* в приложении А. Так, например, для получения информации о подтверждении транзакции необходимо выполнить стандартную команду *GETSTREAMINFO*. Реализована она следующим образом: *GETSTREAMINFO("getstreaminfo", null, new Class<?>[] {Stream.class})*, в файле *CommandElt.java* в приложении А. В параметрах данной команды обязательно должно быть указано имя потока, например, *cds.certificate* или *txId* для идентификации конкретной транзакции. Команда *GETSTREAMINFO*, вызывается в момент вызова метода *ChainResult()*.

Таким образом, чтобы получить требуемую информацию из *MultiChain*, отправляется определённая команда с необходимыми параметрами, всё это в совокупности конкретизирует, что *MultiChain* должен вернуть в качестве

ответа, если необходимо загрузить данные в *MultiChain*, то выполняется команда с параметрами, в которых в формате *JSON* передаются данные, требующие заверения и фиксации в *MultiChain*, по выполненным операциям в системе пользователем. Получив ответ, также в формате *JSON*, система «раскладывает» значения полей в соответствующие поля локальной БД системы.

При подключении нового узла к блокчейну выполняется его валидация, если она происходит успешно, то начинается процесс синхронизации последнего актуального состояния базы с узлом. После того, как данные получены, и связь установлена все изменения, происходящие на данном узле моментально реплицируются по всей сети, таким образом, что все пользователи сети будут считывать обновленные данные в режиме реального времени. Процесс подключения нового узла проиллюстрирован на рисунках 8.1, 8.2.

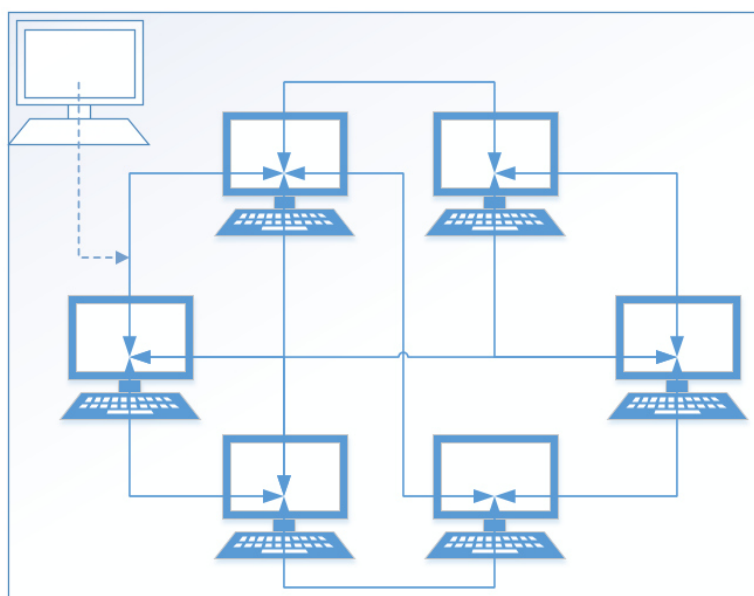


Рисунок 8.1 – Запрос на подключение

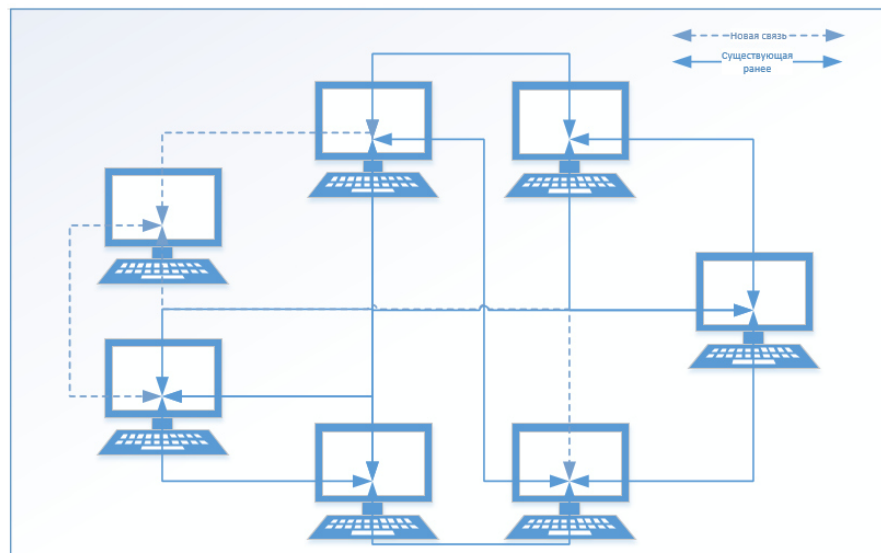


Рисунок 8.2 – Установление подключения и синхронизация данных

2.2 Пользовательское приложение

Пользовательское приложение включает в себя три основных компонента:

- реляционная СУБД *PostgreSQL* необходимая для выполнения стандартных пользовательских операций, заявленных в требованиях к системе, поскольку данные операции невозможно реализовать на базе блокчейна;

- серверный функционал, взаимодействующий с реляционной базой данных, поддерживающий основную бизнес логику, реализующий валидацию получаемых из транспорта блоков;

- пользовательский интерфейс и его взаимодействие с узлом посредством использования протокола *REST JSON* при помощи *JavaScript*-фреймворка *Vue.js*.

Схема взаимодействия компонентов приведена на рисунке 9.

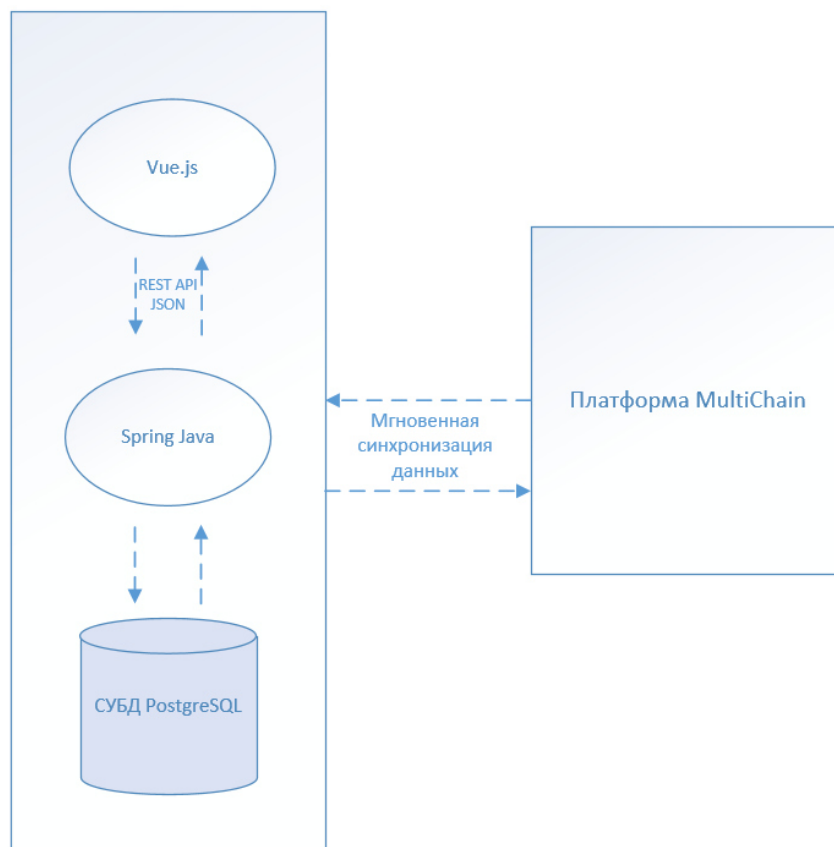


Рисунок 9 - Схема взаимодействия компонентов

2.2.1. Структура базы данных

Для разрабатываемого веб-приложения «Конструктор создания распределенных контрактов на основе технологии блокчейн» в качестве СУБД был использован продукт *PostgreSQL*.

В проектируемой базе данных необходимо реализовать таблицы, описанные в таблицах 1-20, а также отношения сущностей, описанные в таблице 21. Физическая структура базы данных приведена на рисунке 10. Скрипт базы данных представлен в приложении В.

Таблица 1 – Структура таблицы «Данные о рабочем столе клиента» (client).

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор клиента
	mail	VarChar(254)	Электронная почта клиента

Таблица 2 – Структура таблицы «Данные по сертификату участника» (certificate)

Ключ	Название поля	Тип	Комментарий
PK	fingerprint	VarChar(128)	Отпечаток сертификата
	body	Bytea	Тело сертификата
	client_id	UUID	Идентификатор владельца сертификата
	name	VarChar(255)	Наименование
	dn	VarChar(512)	Служебное поле
	not_after	Timestamp	Дата окончания действия сертификата
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения сертификата в блокчейне
	chain_status	Integer	Статус сертификата
	reject_info	VarChar(512)	Причина отклонения добавления сертификата в блокчейн

Таблица 3 – Структура таблицы «Спецификации» (specification)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор спецификации
	name	VarChar(255)	Наименование спецификации
	ttl	Timestamp	Время жизни процессов, созданных по спецификации
	created_at	Timestamp	Время создания спецификации
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	sign_fingerprint	VarChar(64)	Отпечаток сертификата

Продолжение таблицы 3

Ключ	Название поля	Тип	Комментарий
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения сертификата в блокчейне
	chain_status	Integer	Статус сертификата
	reject_info	VarChar(512)	Причина отклонения добавления сертификата в блокчейн

Таблица 4 – Таблица «Участник спецификации» (actor)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор участника в рамках спецификации
	specification_id	UUID	Идентификатор спецификации
	title	VarChar(512)	Текстовое описание
	rights	Integer	Права участника
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	number	Unsigned integer	Номер участника

Таблица 5 – Таблица отношений участник – процесс (actor_constraint)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор отношения
	actor number	Integer	Номер участника
	process id	UUID	Идентификатор процесса

Таблица 6 – Таблица уведомления о процессе (notification)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор уведомления
	client_id	UUID	Идентификатор владельца сертификата
	is seen	Boolean	Видимость
	type	Integer	Тип уведомления
	object id	UUID	Идентификатор объекта
	process id	UUID	Идентификатор процесс
	process name	VarChar(255)	Название процесса
	created_at	Timestamp	Время создания процесса
	updated_at	Timestamp	Время обновления процесса

Таблица 7 – Таблица данных по последовательности операций (operation_quene)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор последовательности операций
	type	Integer	Тип SEQ, ALL, XOR
	order_in	Integer	Порядок последовательности в спецификации

Продолжение таблицы 7

Ключ	Название поля	Тип	Комментарий
FK	Specification id	UUID	Идентификатор спецификации

Таблица 8 – Таблица данных по операции публикации документа (operation_publish)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор операции
	queue_id	UUID	Идентификатор последовательности
	actor number	Integer	Номер участника
	actor title	VarChar(255)	Описание участника
	type_of	VarChar(128)	Уникальная строка в рамках спецификации
	description	VarChar(512)	Описание
	order_in	Integer	Порядок операции в последовательности

Таблица 9 – Таблица данных по операции завершения (operation_finish)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор операции
	queue_id	UUID	Идентификатор последовательности
	actor_number	Integer	Номер участника
	actor_title	VarChar(255)	Описание участника
	result	Integer	Результат
	info	VarChar(255)	Информация
	order_in	Integer	Порядок операции в последовательности

Таблица 10 – Таблица данных о занесении данных в блок (v_chain_block)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор блока
	data	BYTEA	Метаданные

Таблица 11 – Таблица данных о ключе блокчейна (v_chain_block)

Ключ	Название поля	Тип	Комментарий
PK	key	VARCHAR(255)	Ключ
	value	VARCHAR(255)	Вспомогательная переменная

Таблица 12 – Таблица данных о завершении публикации (publish_finish)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор публикации
	process_id	UUID	Идентификатор процесс
	operation_id	UUID	Идентификатор операции
	created_at	Timestamp	Время создания публикации
	result	Integer	Результат
	info	VARCHAR(255)	Дополнительная информация

Продолжение таблицы 12

Ключ	Название поля	Тип	Комментарий
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения публикации в блокчейне
	chain_status	Integer	Статус добавления в блокчейн
	reject_info	VarChar(512)	Причина отклонения публикации в блокчейн

Таблица 13 – Таблица данных по операции подписания нового документа (operation_sign)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор операции
FK	queue_id	UUID	Идентификатор последовательности
	actor_number	Integer	Номер участника
	type_of	VarChar(128)	Уникальная строка в рамках спецификации
	order_in	Integer	Порядок операции в последовательности

Таблица 14 – Таблица данных по операции публикации фиксированного документа (operation_publish_fix)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор операции
FK	queue_id	UUID	Идентификатор последовательности
	type_of	VarChar(128)	Уникальная строка в рамках спецификации
	description	VarChar(512)	Описание
	hash	Bytea	Хэш документа
	hash_algorithm	VarChar(16)	Алгоритм хэширования
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	order_in	Integer	Порядок операции в последовательности

Таблица 15 – Таблицы данных по операции подписания фиксированного документа (operation_sign_fix)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор операции
FK	queue_id	UUID	Идентификатор последовательности
	actor_number	Integer	Номер участника
	actor_title	VarChar(255)	Информация об участнике

Продолжение таблицы 15

Ключ	Название поля	Тип	Комментарий
	type_of	VarChar(128)	Уникальная строка в рамках спецификации
	hash	Bytea	Хэш документа
	hash_algorithm	VarChar(16)	Алгоритм хэширования
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	order_in	Integer	Порядок операции в последовательности

Таблица 16 – Таблица данных по операции фиксирования публикации в блокчейне (publish_sign)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор публикации
FK	process_id	UUID	Идентификатор процесс
	operation_id	UUID	Идентификатор операции
	created_at	Timestamp	Время создания публикации
	hash	Bytea	Хэш документа
	hash_algorithm	VarChar(16)	Алгоритм хэширования
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения публикации в блокчейне
	chain_status	Integer	Статус добавления в блокчейн
	reject_info	VarChar(512)	Причина отклонения публикации в блокчейн

Таблица 17 – Таблица данных по операции публикации документа в блокчейне (publish_document)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор публикации
FK	process_id	UUID	Идентификатор процесс
	operation_id	UUID	Идентификатор операции
	created_at	Timestamp	Время создания публикации
	type	VarChar (128)	Тип документа
	hash	Bytea	Хэш документа
	hash_algorithm	VarChar(16)	Алгоритм хэширования
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения публикации в блокчейне
	chain_status	Integer	Статус добавления в блокчейн

Продолжение таблицы 17

Ключ	Название поля	Тип	Комментарий
	reject_info	VarChar(512)	Причина отклонения публикации в блокчейн

Таблица 18 – Таблица данных по процессу (process)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор процесса
FK	specification id	UUID	Идентификатор спецификации
	name	VarChar(255)	Наименование процесса
	created at	Timestamp	Время создания процесса
Ключ	Название поля	Тип	Комментарий
	sign_fingerprint	VarChar(64)	Отпечаток сертификата
	sign_algorithm	VarChar(16)	Алгоритм подписи
	sign	Bytea	Подпись
	tx_id	VarChar(64)	Идентификатор транзакции в блокчейне
	in_chain	Boolean	Флаг нахождения публикации в блокчейне
	chain_status	Integer	Статус добавления в блокчейн
	reject_info	VarChar(512)	Причина отклонения публикации в блокчейн

Таблица 19 – Таблица данных по ограничениям (selector)

Ключ	Название поля	Тип	Комментарий
PK	id	UUID	Идентификатор процесса
FK	actor_id	UUID	Идентификатор участника
FK	actor_constraint_id	UUID	Идентификатор связи участник-процесс
FK	selector_template_id	UUID	Идентификатор шаблона ограничений
	type	integer	Тип ограничения
	oid	VarChar (255)	Сертификат подписи узла
	attribute	VarChar (255)	Атрибут
	val	VarChar (255)	Дополнительное поле
	action	Char (1)	Действие

Таблица 20 – Реляционные отношения (связи) между сущностями

Наименование отношения	Родительская/дочерняя сущность	Родительское/дочернее поле	Тип связи
fk_operation_publish_queue	operation_queue/ operation_publish	id/queue_id	1:M
fk_process_operation_finish	process/ operation_finish	id/process_id	1:M
fk_operation_queue_specification	specification/ operation_queue	id/specification_id	1:M
fk_process_specification	specification /process	id/specification_id	1:M

Продолжение таблицы 20

Наименование отношения	Родительская/дочерняя сущность	Родительское/дочернее поле	Тип связи
fk_specification_actor_specification	specification/ actor	id/specification_id	1:M
fk_specification_client	client/ specification	id/client_id	1:M
fk_client_notification	client/ notification	id/client_id	1:M
fk_client_certificate	client/ certificate	id/client_id	1:M
fk_certificate_notification	certificate/ notification	fingerprint/client_id	1:M
fk_operation_publish_fix_queue	operation_queue/ operation publish fix	id/queue_id	1:M
fk_operation_sign_queue	operation_queue/ operation sign	id/queue_id	1:M
fk_operation_sign_fix_queue	operation_queue/ operation sign fix	id/queue_id	1:M
fk_operation_finish_queue	operation_queue/ operation finish	id/queue_id	1:M
fk_operation_queue_specification	specification/ operation queue	id/specification_id	1:M
fk_process	specification/ process	id/specification_id	1:M
Примечание - Существует три разновидности связей между сущностями: - 1:M – «один-ко-многим», - 1:1 – «один-к-одному», - M:M – «многие-ко-многим».			

Для предотвращения потери ссылочной целостности при изменении содержания БД используется механизм каскадных воздействий. В таблице 21 приведен перечень каскадных воздействий.

Таблица 21 – Перечень каскадных воздействий

Родительская таблица/ поле родительской таблицы	Дочерняя таблица/поле дочерней таблицы	Типы каскадного воздействия
specification/id	specification_actor/ specification_id	У
actor/selector	specification_actor/ actor_selector	У
specification/id	operation_queue/ specification_id	У
operation_queue/id	operation_finish/queue_id	У
operation_queue/id	operation_sign_fix/queue_id	У
operation_queue/id	operation_sign/queue_id	У
operation_queue/id	operation_publish/queue_id	У
operation_queue/id	operation_publish_fix/ queue_id	У
<p>Примечание – Типы каскадного воздействия имеют следующие обозначения:</p> <ul style="list-style-type: none"> - И – при изменении поля связи в записи родительской таблицы синхронно изменяются значения полей связи в соответствующих записях дочерней таблицы; - У – при удалении записи в родительской таблице удаляются соответствующие записи в дочерней таблице. 		

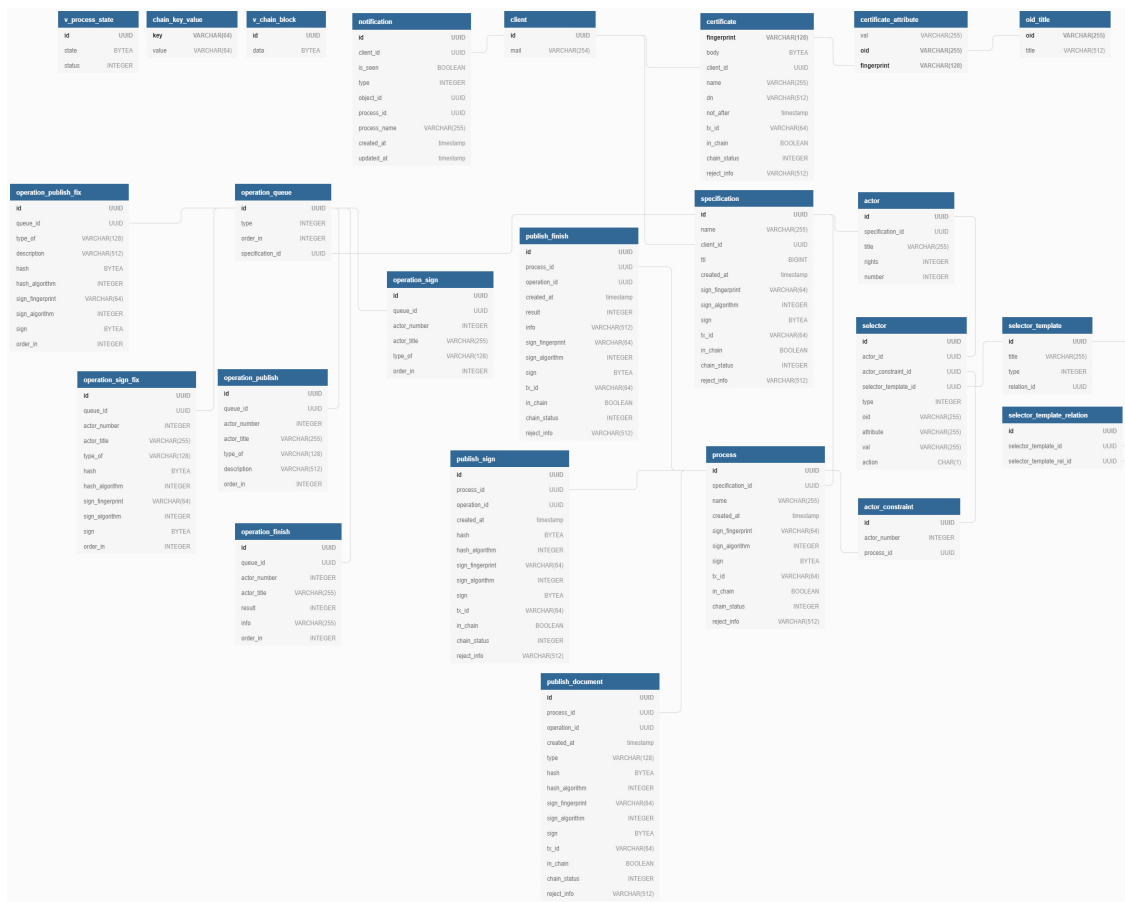


Рисунок 10 - Схема базы данных «Конструктор создания распределенных контрактов на основе технологии блокчейн»

2.2.2. Описание классов

В веб-приложении «Конструктор создания распределенных контрактов на основе технологии блокчейн» для взаимодействия описанной ранее СУБД и пользовательского интерфейса выделены следующие основные функциональные блоки:

- классы-сущности. Такой тип классов необходим для описания сущности базы данных в программе, он облегчает работу с данными, которые являются результатом функции класса-репозитория;

- классы-репозитории. Данный функциональный элемент взаимодействует с СУБД, получает данные из классов-сервисов;

- классы-сервисы. Выполняют взаимодействие между классами-репозиториями и классами-контроллерами;

- классы-контроллеры. Необходимы для реализации запросов *REST API*.

На примере сущности «Участник спецификации» будет рассмотрено взаимодействие всех описанных выше классов.

Класс-сущность «Участник» описывает структуру таблицы «Участник спецификации» (*Actor*). Кроме того, данный класс взаимодействует с другими сущностями (сущности: «Ограничения на участника», «Данные по спецификации», «Права участника», «Данные по операциям»). Класс содержит методы, позволяющие получить или установить такие свойства, как права, ограничения, описание и номер участника спецификации. Данный класс наглядно представляет информацию об участнике спецификации и его возможностях.

UML-диаграмма классов сущности «Участник спецификации» представлена на рисунке 11.

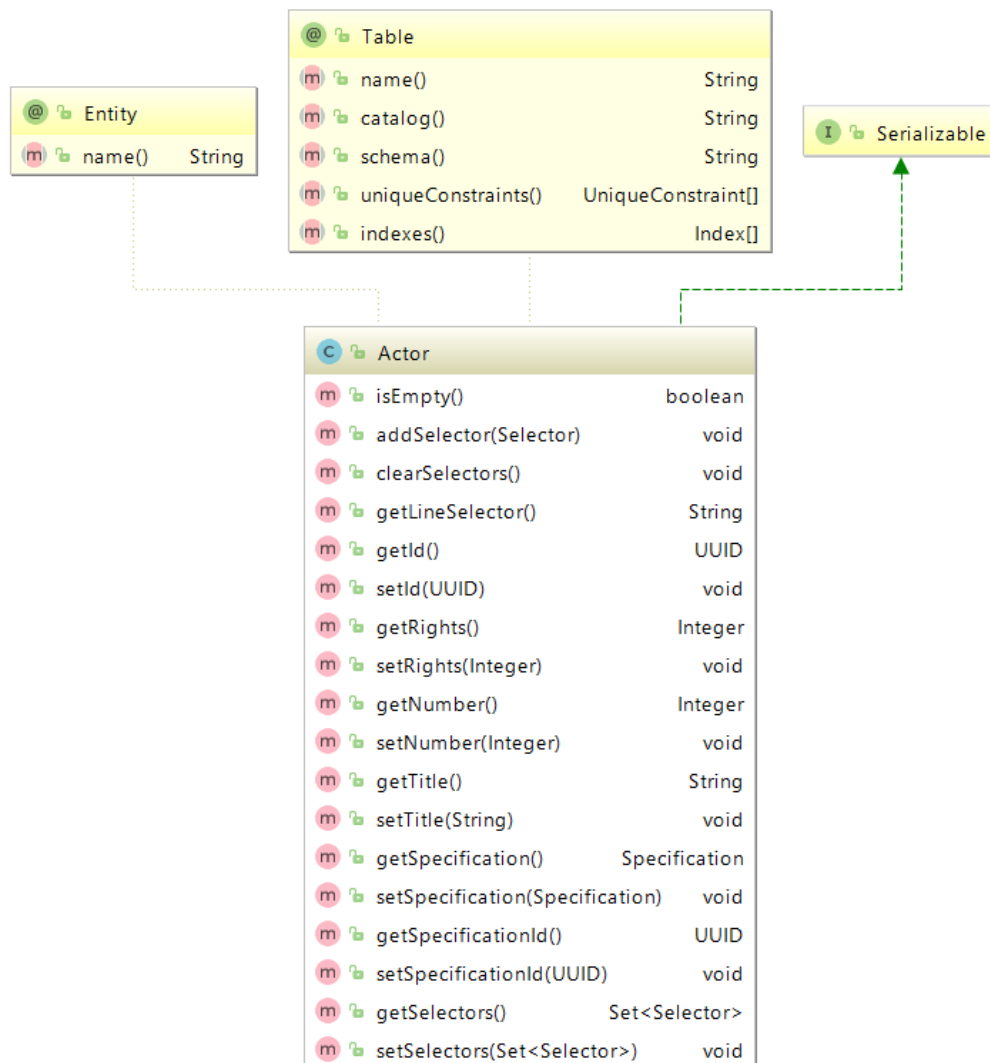


Рисунок 11 – UML-диаграмма классов сущности «Участник спецификации»

Следующий рассмотренный класс – класс-репозиторий «*ActorRepository*», который взаимодействует с базой данных в части обработки данных об участниках и связанными с ними спецификациями. Например, через *UUID* участника выполняется поиск всех его спецификаций, выполнив поиск можно отредактировать связи: удалить связь спецификация-участник, изменить саму спецификацию, сохранить изменения. UML-диаграмма класса-репозитория отображена на рисунке 12.

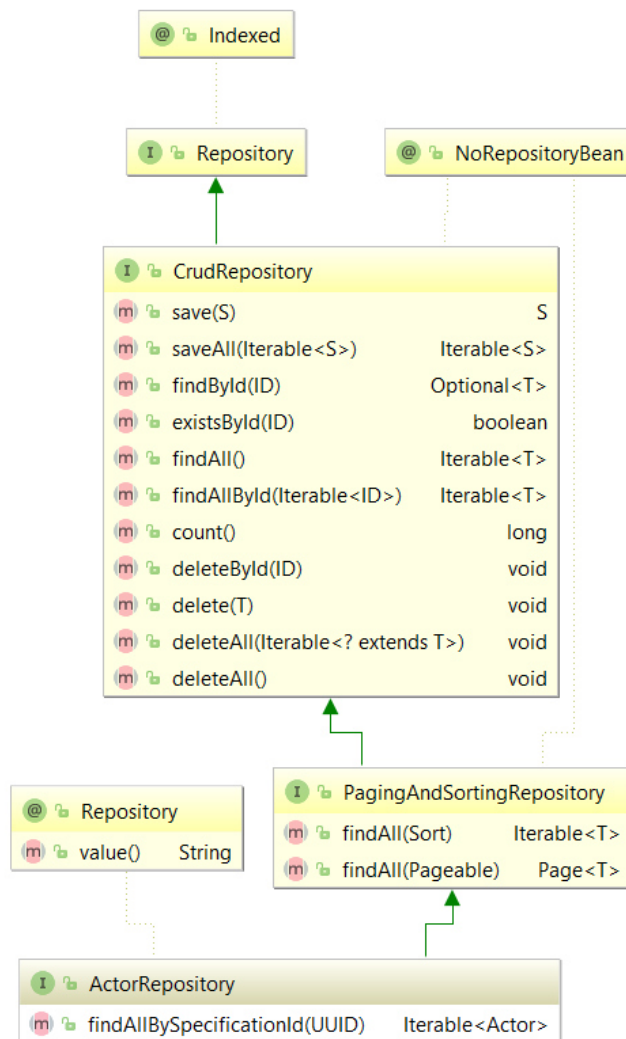


Рисунок 12 – UML-диаграмма класса-репозитория «*ActorRepository*»

Класс-сервис «*ActorService*» необходим для работы с объектами, описывающими участников спецификации, выполняя подготовку объектов к передаче для обработки в класс-репозиторий. UML-диаграмма «*ActorService*» представлена на рисунке 13.

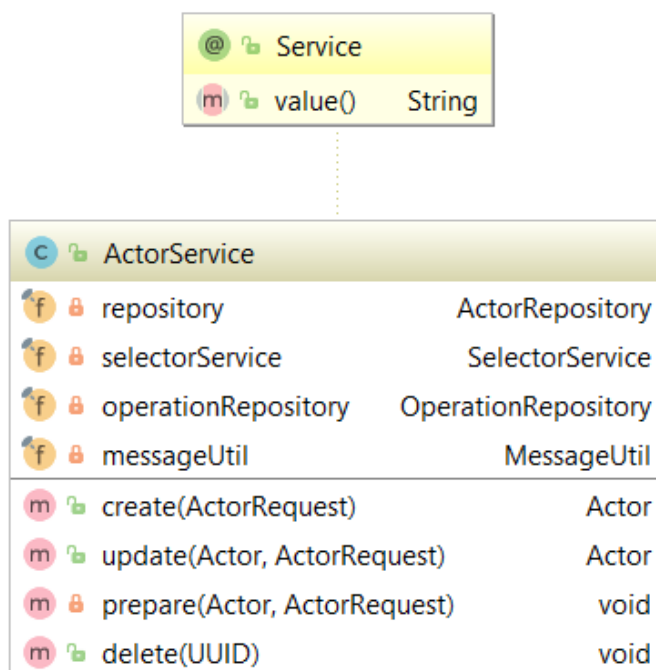


Рисунок 13 – UML-диаграмма класса-сервиса

Класс-контроллер. необходимый для реализации запросов *REST API* в работе с сущностью «Участник спецификации» представлен как «*ActorController*». Данный класс производит обработку запросов со стороны пользователя, валидацию полученных данных и вызывает методы класса-сервиса «*ActorService*», а также возвращает выходные данные-результат. UML-диаграмма описанного класса представлена на рисунке 14.

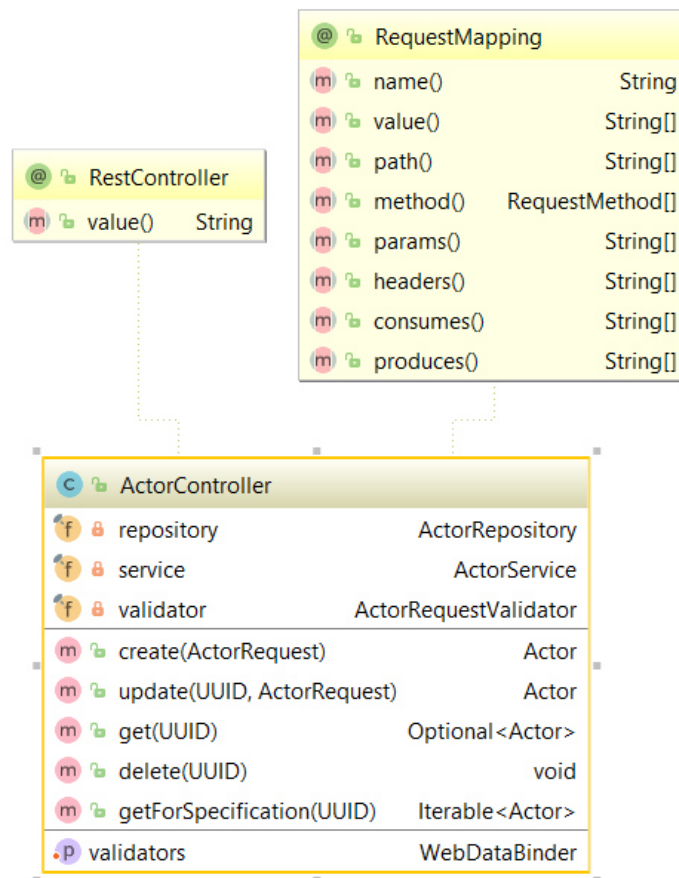


Рисунок 14 – UML-диаграмма класса-контроллера

2.2.3. Программная реализация

Для реализации данного приложения был использован язык программирования *Java*, фреймворки *Spring* и *Vue.js*[11]. Исходный код описанных в предыдущем разделе классов приведен в приложении А.

Пользователь приложения выполняет действия при помощи интерфейса и обращения через *REST API* к реляционной базе данных приложения.

Наиболее наглядным примером *REST API* будет являться *GET*-запрос `https://ucalls.ru:8721/api/specification/{id}`, ответом на данный запрос будет информация о спецификации. Для того, чтобы запрос был выполнен успешно вместо *id* нужно указать *id* спецификации в формате *UUID*. При успешном выполнении запроса будет получен ответ, далее представлен его шаблон:

{

```

"content": [
  {
    "id": "",
    "name": "",
    "clientId": "",
    "ttl": "",
    "createdAt": "",
    "signature": {...},
    "certificate": {...},
    "queues": [...],
    "actors": [...],
    "chainResult": {...},
    "empty": "",
    "emptyForSign": ""
  }
]

```

Пример выполнения *GET*-запроса в программе представлен в приложении С.

Если запрос выполняется с ошибкой то, придет ответ в зависимости от полученного кода ошибки. В данном случае могут выполняться следующие коды ошибок: 502 – ошибка сервера, 404 – отсутствие данных на сервере по

заданному запросу и 400 – некорректный запрос, реализация последнего приведена в приложении С.

Кроме выполнения запроса *REST API*, важно понимать логику работы программы, а именно, взаимодействие цепочки: пользователь – приложение – локальная БД – *Multichain*.

В качестве примера будет рассмотрен процесс регистрации и дальнейшей авторизации пользователя при помощи сертификата.

Первоначально пользователь загружает в приложение сертификат формата *base64*, используя *REST API*. На этом этапе выполняется *POST*-запрос (<https://ucalls.ru:8721/api/registration>), в теле которого 64-разрядный код, полученный из файла сертификата. *Spring Framework* фиксирует загруженный сертификат в базе данных приложения флагом *inChain = false*, далее при помощи *MultiChain RPC API* записывает данную транзакцию в поток, выполнив команду *send* с параметром – *address* (задает адрес кошелька, в данном случае адрес узла) и получает в ответ её номер «*txid*», который фиксируется в локальной БД приложения.

На текущий момент запись имеет статус не подтвержденной транзакции, поэтому далее она валидируется по правилам технологии блокчейн и дополнительно проходит валидацию путем обратной отправки на валидатор *Spring* приложения. С помощью криптографических библиотек валидатор проверяет сертификат на действительность. При успешном прохождении валидации, загруженный сертификат находится через номер транзакции и флаг *inChain = true*, если процесс валидации закончен неуспешно, то флаг *inChain = false*, в поле *reject_info* (причина отклонения добавления сертификата в блокчейн) прописывается, почему запись отклонена.

Подобный механизм валидации действий пользователей применяется и при создании спецификации, процесса и его выполнении пользователями.

Для наглядного отображения создания спецификации, на базе которой будет создан процесс и в дальнейшем выполнен между пользователями предоставлена *UML*-диаграмма, отображенная на рисунке 15[12].

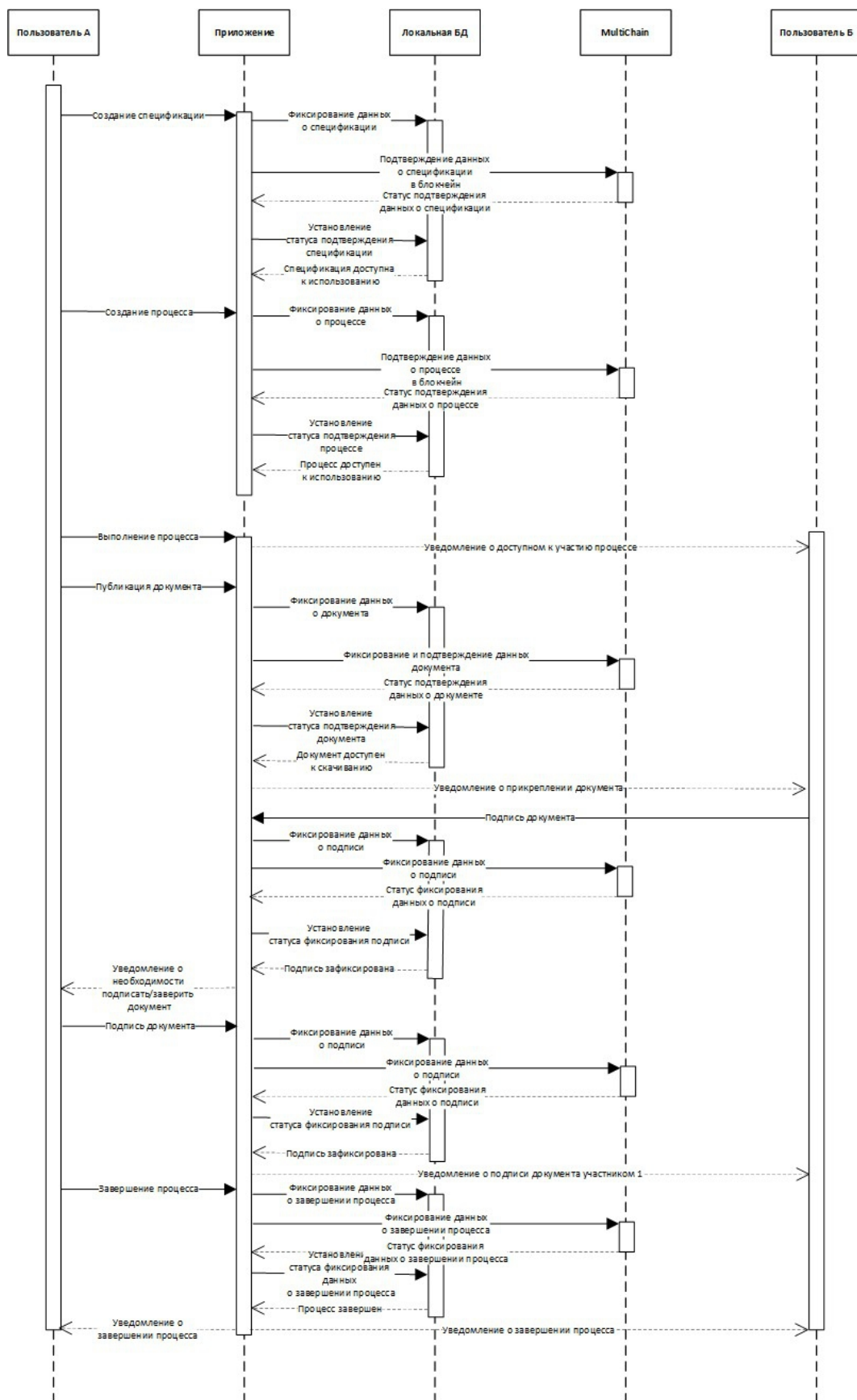


Рисунок 15 – UML-диаграмма алгоритма взаимодействия пользователей с предсозданием спецификации и процесса

2.3 Выводы по главе

Таким образом, рассмотренные в этом разделе аспекты, наглядно описывают внутренний механизм работы приложения, а именно, как происходит процесс транспортировки данных между самим приложением и *Multichain*, как выполнено структурирование локальной базы данных, необходимой для хранения сведений о пользователе. Также в данном разделе была описана программная реализация взаимодействия базы данных с интерфейсом по средством различных функциональных блоков, более подробно при помощи *UML*-диаграммы был представлен алгоритм взаимодействия пользователя с точки зрения программной реализации.

3. Инструкция пользователя

Данный раздел представляет собой руководство пользователя к приложению «Конструктор создания распределенных контрактов на основе технологии блокчейн». Уровень подготовки пользователей системы должен соответствовать следующим требованиям:

- владение операционной системой *Windows*;
- умение пользоваться веб-браузерами;
- наличие навыков работы на персональном компьютере в сети Интернет;
- наличие навыков работы в сфере электронного документооборота.

3.1 Авторизация в системе

Первоначально пользователю необходимо пройти авторизацию путем загрузки собственного сертификата в систему, рисунках 16.1, 16.2.

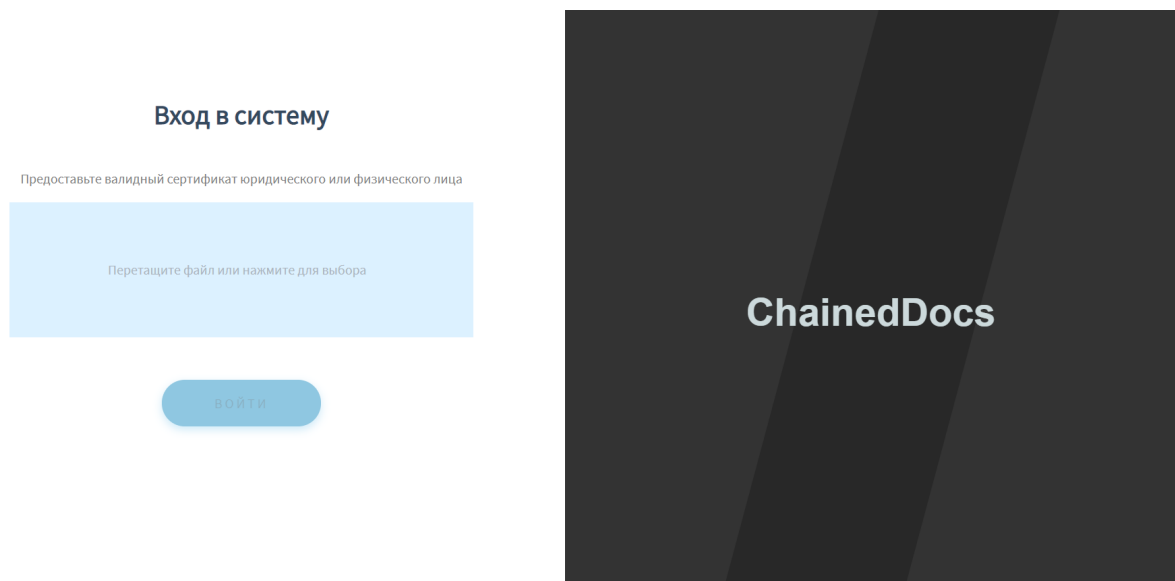
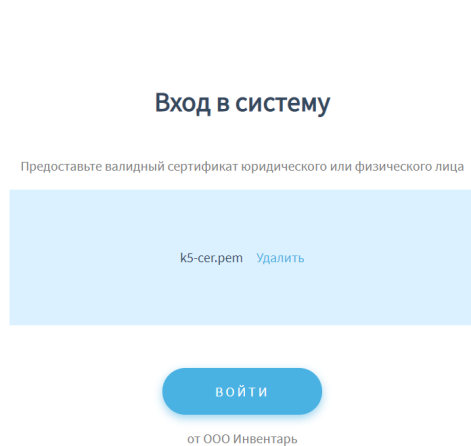
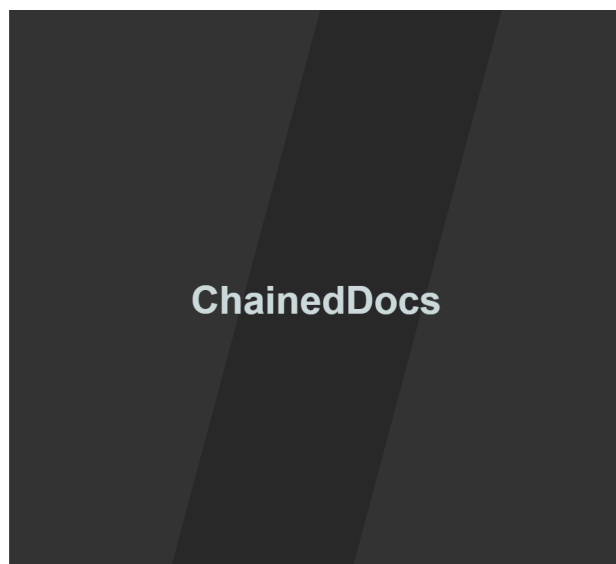


Рисунок 16.1 – Экран авторизации. Первоначальный экран.



Б)

Рисунок 17.2 – Экран после загрузки сертификата в систему.

После того, как сертификат загружен в систему, кнопка «Войти» становится активной – доступной к нажатию. В случае, если пользователь загрузил ошибочный сертификат, сертификат может быть удален кнопкой «Удалить», далее пользователь может загрузить новый сертификат повторно.

3.2 Рабочий стол пользователя

Выполнив авторизацию в системе, нажав кнопку «Войти», перед пользователем открывается экран «Рабочий стол», который можно разделить на несколько блоков: блок горизонтального меню, блок вертикального меню, блок выполняемых событий и процесса, с возможностью создания нового процесса, рисунок 17.

В блоке горизонтального меню представлены следующие функциональные элементы: кнопка скрытия/раскрытия вертикального меню, по умолчанию вертикальное меню раскрыто; наименование организации, при нажатии на данный функциональный элемент появится выпадающее меню, содержащее пункты: профиль пользователя, при нажатии выполняется переход в профиль пользователя, где можно добавить или удалить сертификат; выйти, при выборе данного пункта меню будет выполнен выход пользователя из системы и переход к странице авторизации.

Вертикальное меню включает следующие пункты:

— рабочий стол. Переход к этому экрану выполняется по умолчанию после авторизации пользователя в системе, данный экран считается — главной страницей приложения, рисунок 17;

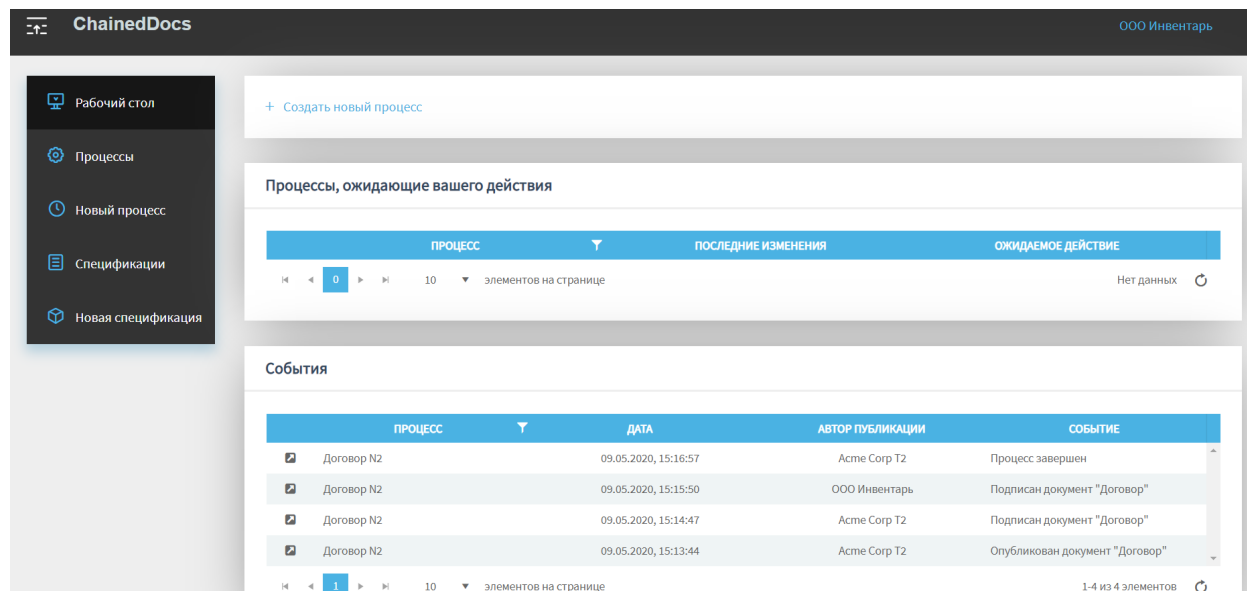


Рисунок 18 — Экран «Рабочий стол»

— процессы. При выборе данного пункта меню будет выполнен переход на страницу с процессами, которые доступны пользователю для участия, выполняются или уже завершены, рисунок 18. Подробный алгоритм возможных действий на экране «Процессы» описан в пункте 3.3;

— новый процесс. Данный пункт открывает страницу создания нового процесса, рисунок 19. Подробный алгоритм возможных действий на экране «Новый процесс» описан в пункте 3.4;

— спецификации. При выборе данного пункта меню происходит переход на страницу, где отображены все созданные пользователем или доступные ему спецификации, рисунок 20. Подробный алгоритм возможных действий на экране «Спецификации» описан в пункте 3.5;

— новая спецификация. Данный пункт открывает страницу создания новой спецификации, рисунок 21. Подробный алгоритм возможных действий на экране «Новая спецификация» описан в пункте 3.6;

Блок выполняемых событий и процесса, с возможностью создания нового процесса содержит кнопку «Создать новый процесс», при нажатии на которую выполняется переход к экрану «Создание процесса».

Подблок «Процессы, ожидающие вашего действия» представлен в виде таблицы с данными о процессах, в которых пользователь принимает участие. Таблица состоит из столбцов: «Процесс» - название процесса, возможна фильтрация по алфавиту; «Последнее изменение» - дата и время последнего внесенного изменения в процесс; «Ожидаемое действие» - действие, которое необходимо выполнить участнику процесса.

Подблок «События» также представлен в виде таблицы, состоящей из столбцов: «Процесс» - название процесса, возможна фильтрация по алфавиту; «Дата» - дата и время события; «Автор публикации» - инициатор события; «Событие» - тип события.

3.3 Экран «Процессы»

Экран «Процессы» включает в себя чек-бокс «Показать процессы доступные для участия» и блок, представленный в виде таблицы, со сведениями о процессах, рисунок 18.

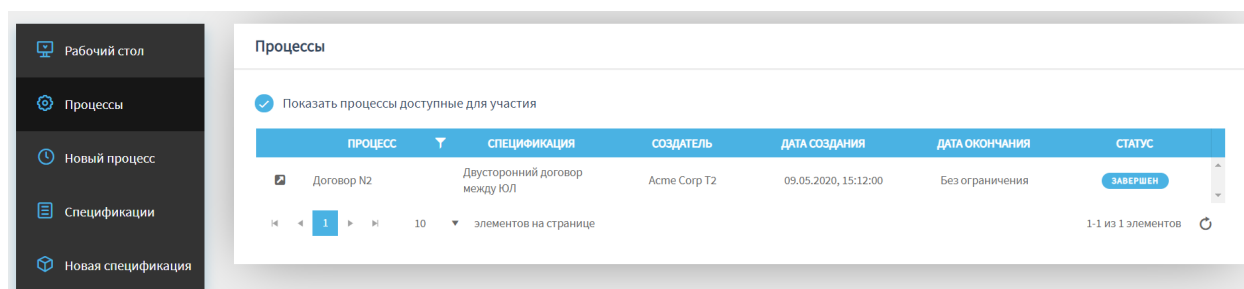


Рисунок 19 – Экран «Процессы»

Чек-бокс «Показать процессы доступные для участия» включён по умолчанию. Если отключить данный чек-бокс, то в таблице появятся все существующие в системе процессы.

Табличный блок «Процессы» содержит сведения о процессах, а именно: название, связанную спецификацию, кто является инициатором процесса,

дату создания, дату завершения процесса, статус. Кроме того, процессы можно посмотреть подробнее, нажав кнопку перехода, рисунок 19.

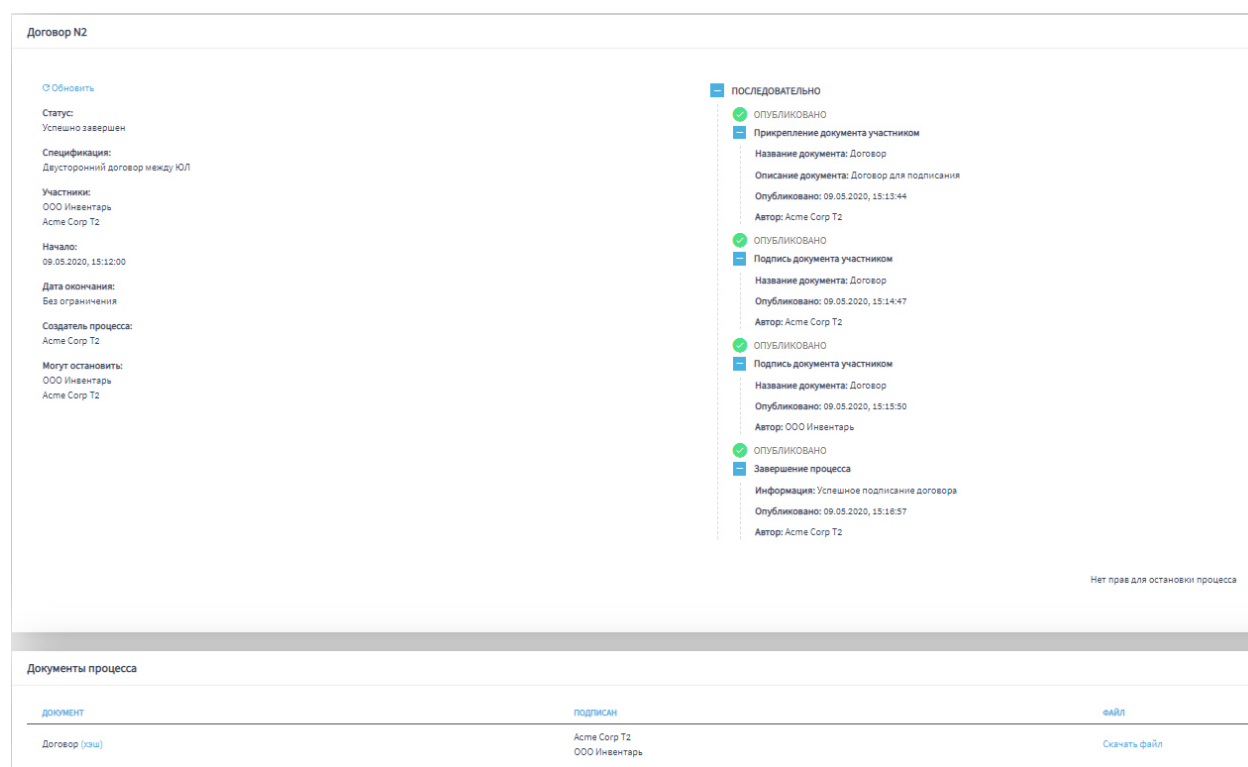


Рисунок 20 — Сведения о процессе

3.4 Экран «Новый процесс». Создание процесса

Создать процесс возможно, нажав кнопку меню «Новый процесс», рисунок 20.

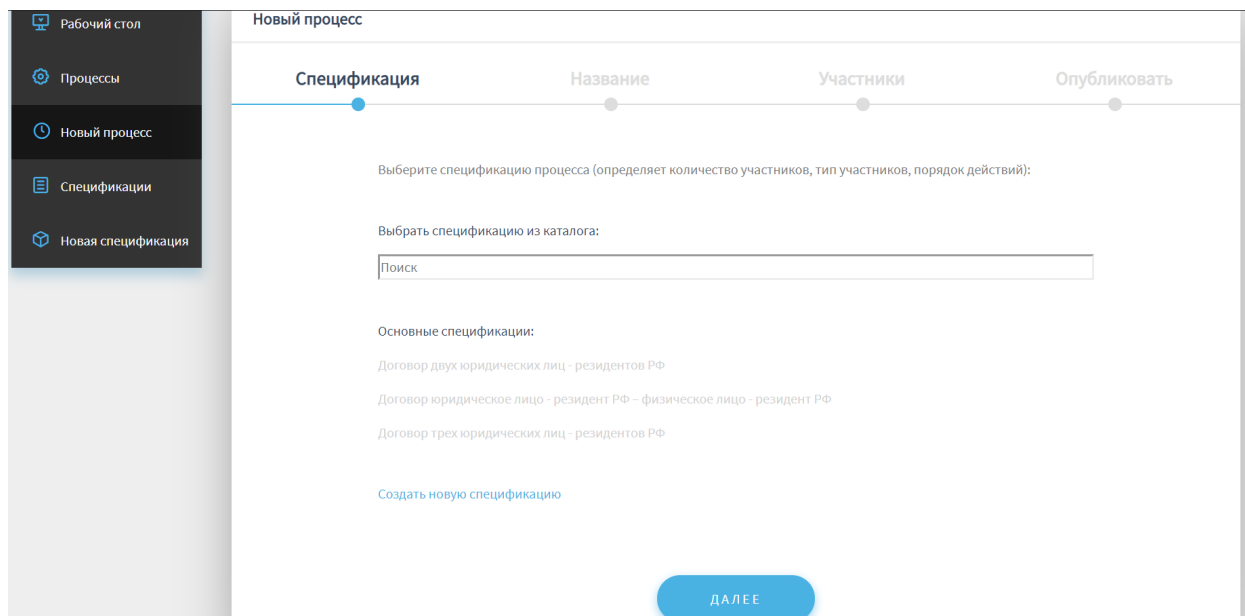


Рисунок 21 – Экран «Новый процесс»

Процесс создается поэтапно, сначала выбирается спецификация, на базе которой будет создана инструкция процесса, затем процессу присваивается название, рисунок 21.

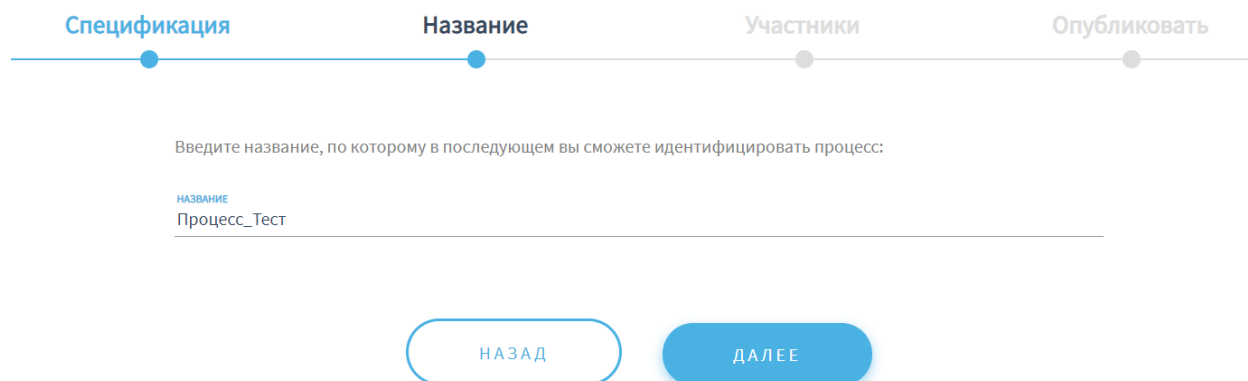
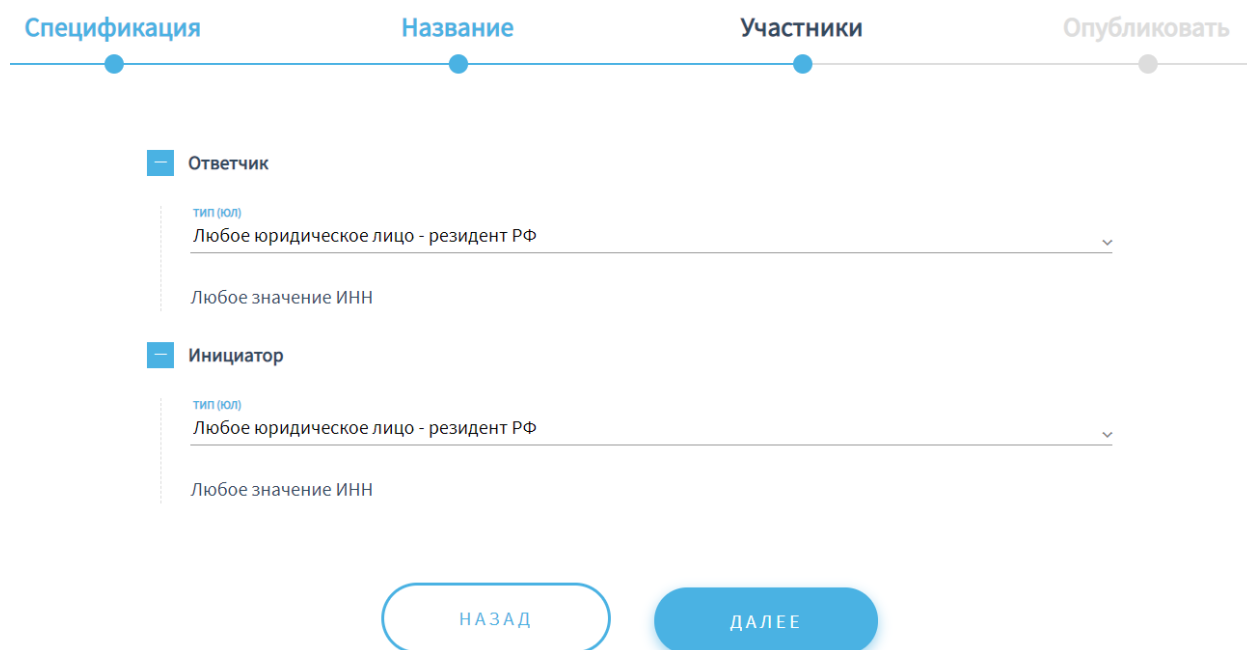


Рисунок 22 – Создание процесса. Шаг 2

Далее выбирается тип участников, допущенных к процессу: любое ЮЛ или ФЛ, в зависимости от типа участника, заданного в спецификации; конкретное ЮЛ или ФЛ, в последнем случае обязательно необходимо указать ИНН участника системы, рисунок 22.



Спецификация Название Участники Опубликовать

Ответчик

тип (юл)
Любое юридическое лицо - резидент РФ

Любое значение ИНН

Инициатор

тип (юл)
Любое юридическое лицо - резидент РФ

Любое значение ИНН

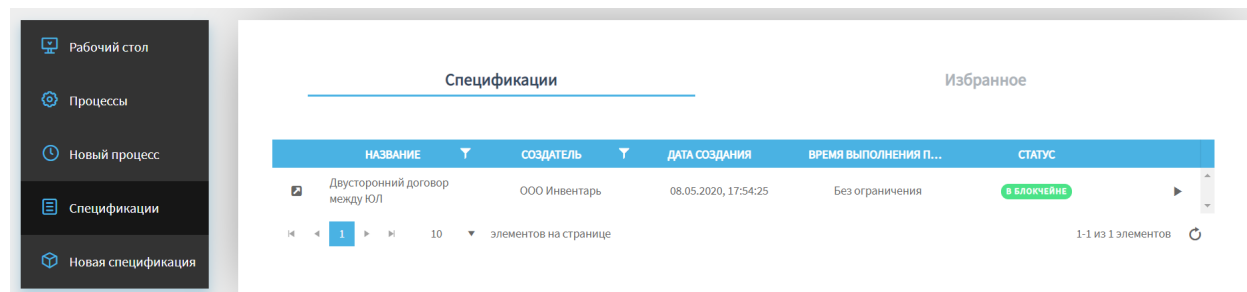
НАЗАД ДАЛЕЕ

Рисунок 23 – Создание процесса. Шаг 3

На заключительном этапе процесс подписывается создателем и публикуется в системе и блокчейн.

3.5 Экран «Спецификации»

При переходе к пункту меню «Спецификации», откроется экран «Спецификации», рисунок 23, на котором размещен табличный блок с данными о спецификации, а именно: названием, создателем, датой создания, временем выполнения и статусов. Кроме того, есть блок «Избранное», в который могут быть помещены спецификации, которые использовались пользователем ранее.



НАЗВАНИЕ	СОЗДАТЕЛЬ	ДАТА СОЗДАНИЯ	ВРЕМЯ ВЫПОЛНЕНИЯ П...	СТАТУС
Двусторонний договор между ЮЛ	ООО Инвентарь	08.05.2020, 17:54:25	Без ограничений	В БЛОКЧЕЙНЕ

1-1 из 1 элементов

Рисунок 24 – Экран «Спецификации»

Также спецификацию можно посмотреть подробнее, нажав кнопку просмотра спецификации, рисунок 24.

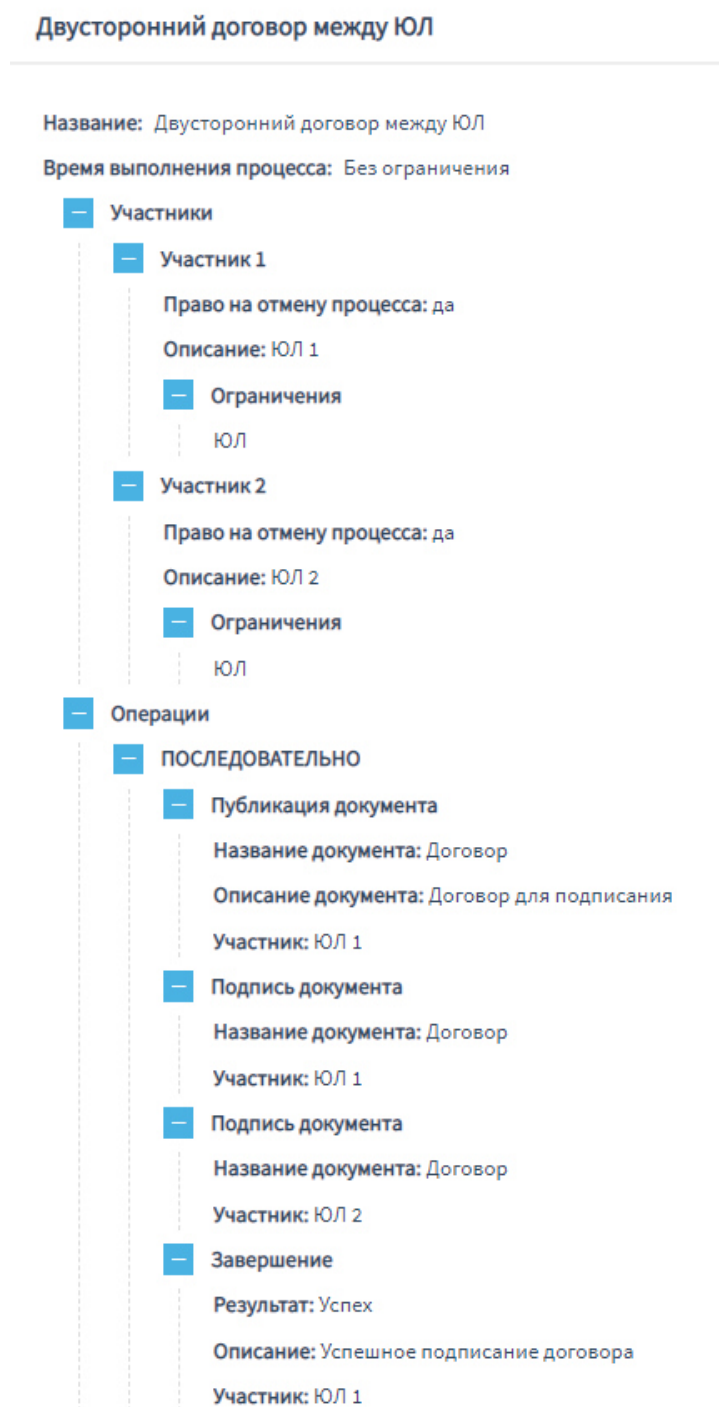


Рисунок 25 — Данные по спецификации

3.6 Экран «Новая спецификация». Создание новой спецификации

Создание спецификации выполняется аналогично созданию процесса — поэтапно, рисунок 25.

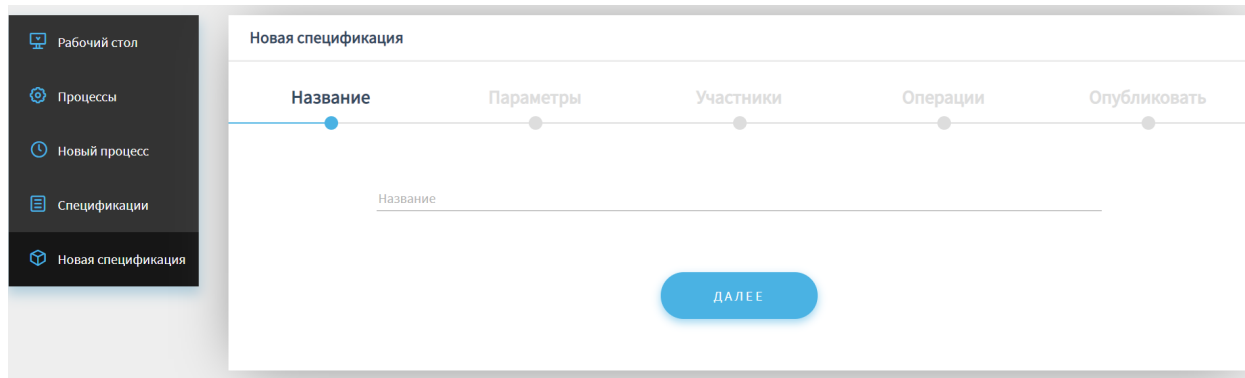


Рисунок 26 – Экран «Новая спецификация»

Первоначально необходимо ввести название спецификации, на следующем шаге выбираются параметры – временные ограничения, рисунок 26.

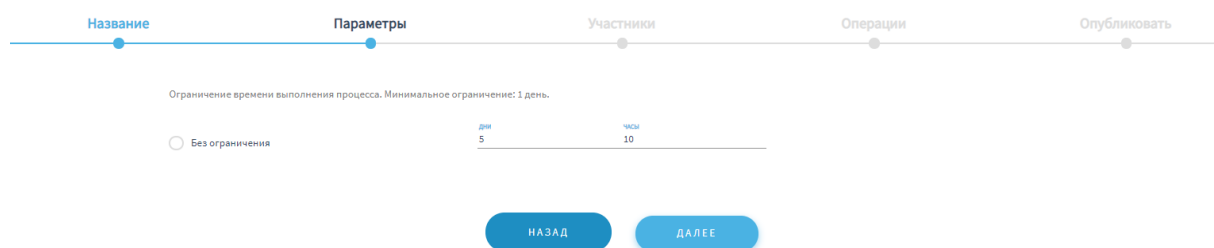


Рисунок 27 – Создание спецификации. Шаг 2

После выбора параметров, на третьем шаге задается количество и тип участников, рисунок 27, далее выбираются последовательность и тип операций – инструкций спецификации, в соответствии с которыми далее будет выполняться связанный со спецификацией процесс, рисунки 28, 29.

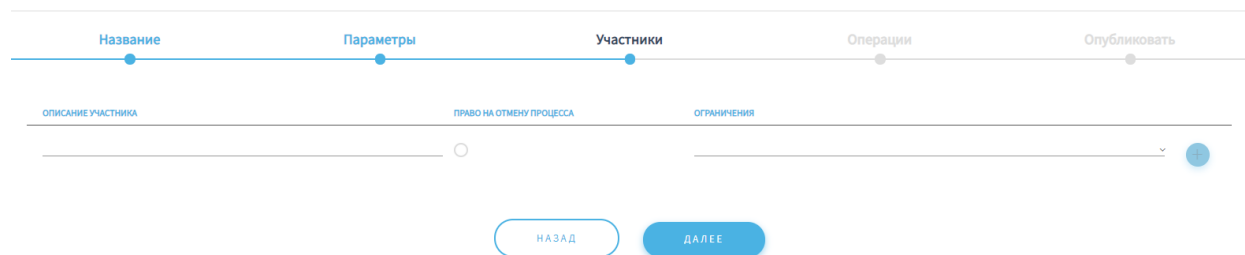


Рисунок 28 – Создание спецификации. Шаг 3

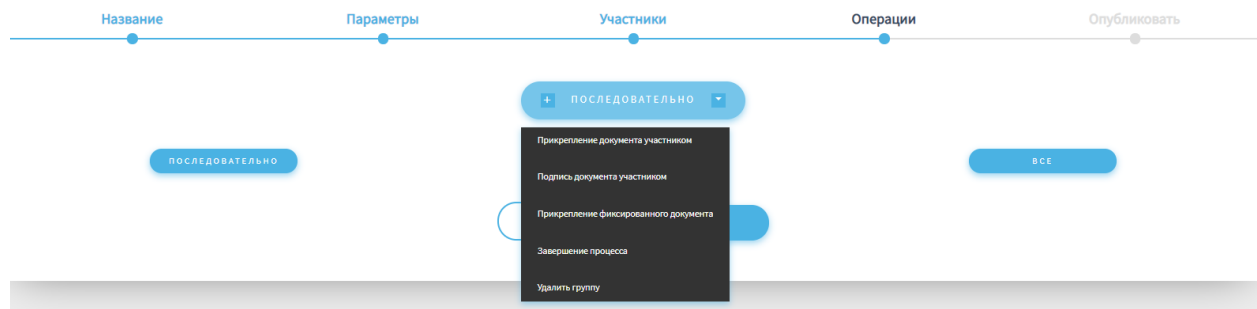


Рисунок 29 – Создание спецификации. Шаг 4

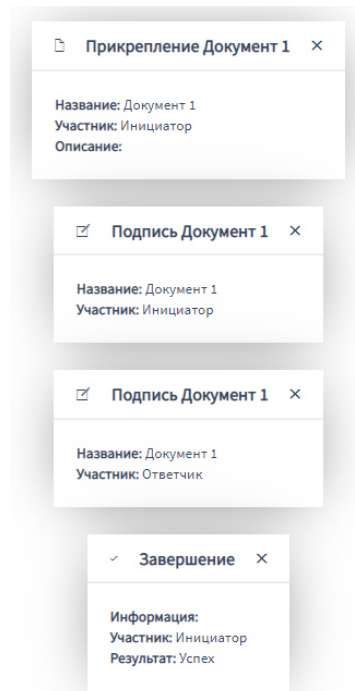


Рисунок 30 – Создание спецификации. Шаг 4, заданные операции

На завершающем этапе, пользователь должен подписать и опубликовать спецификацию, аналогично процессу, спецификация опубликуется в системе и блокчейн.

3.7 Выполнение конкретного процесса между пользователями

Авторизованы два клиента, между которыми планируется заключение договора, рисунок 30.

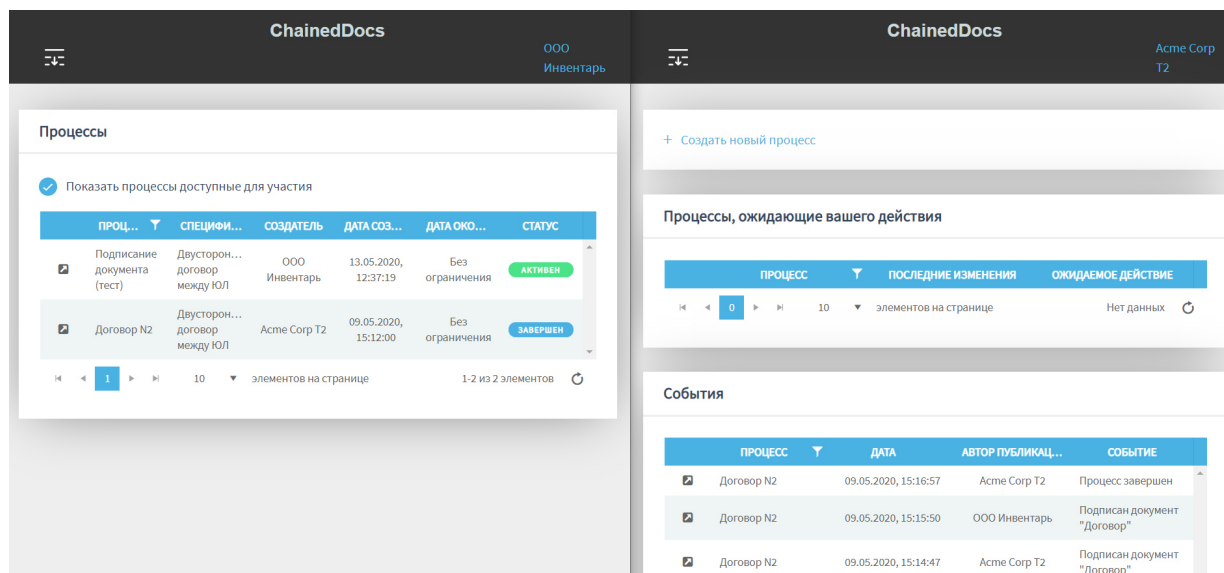


Рисунок 31 — Авторизация двух клиентов

Далее клиент ООО «Инвентарь» запускает активный процесс. Страница выполнения последовательных шагов процесса, согласно инструкциям связанной с ним спецификации изображена рисунке 31.

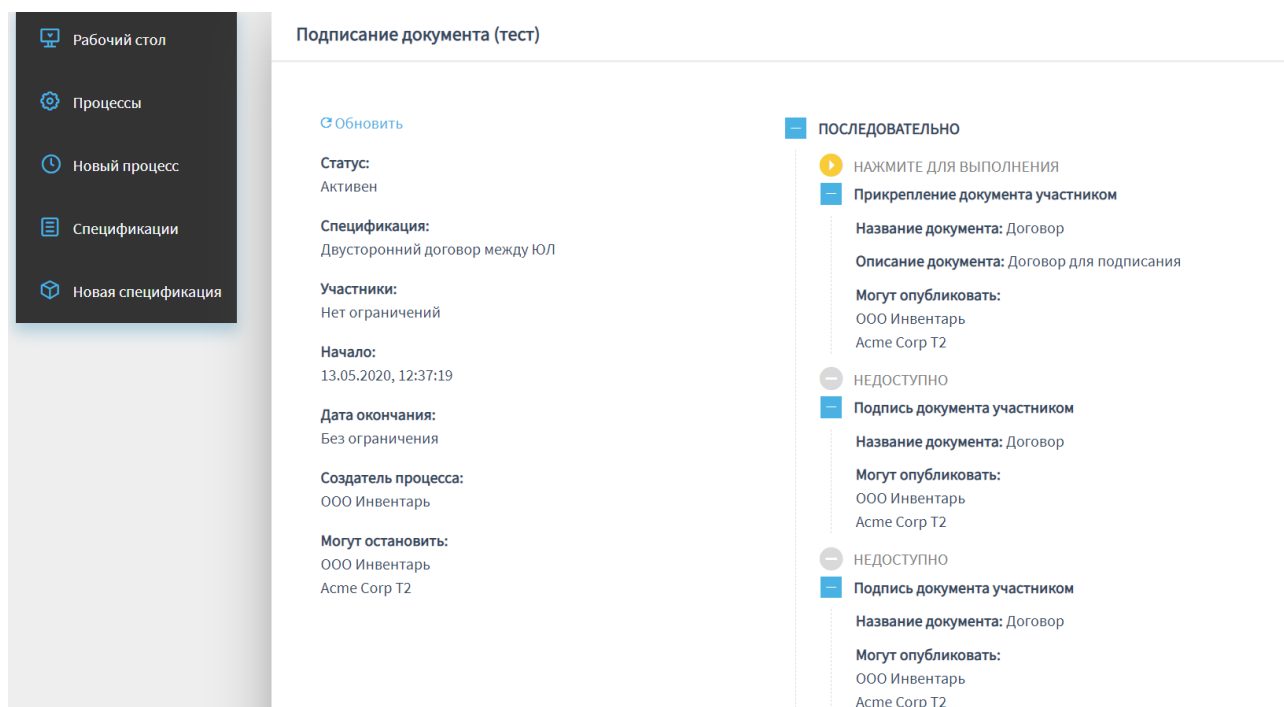


Рисунок 32 — Алгоритм выполнения процесса

Из алгоритма видно, что участниками могут быть пользователи «ООО Инвентарь» и «АсmeCorp T2». Пользователь «ООО Инвентарь» загружает

документ в систему и подписывает своим, ранее сгенерированным ключом. Данный шаг процесса изображен на рисунке 32.

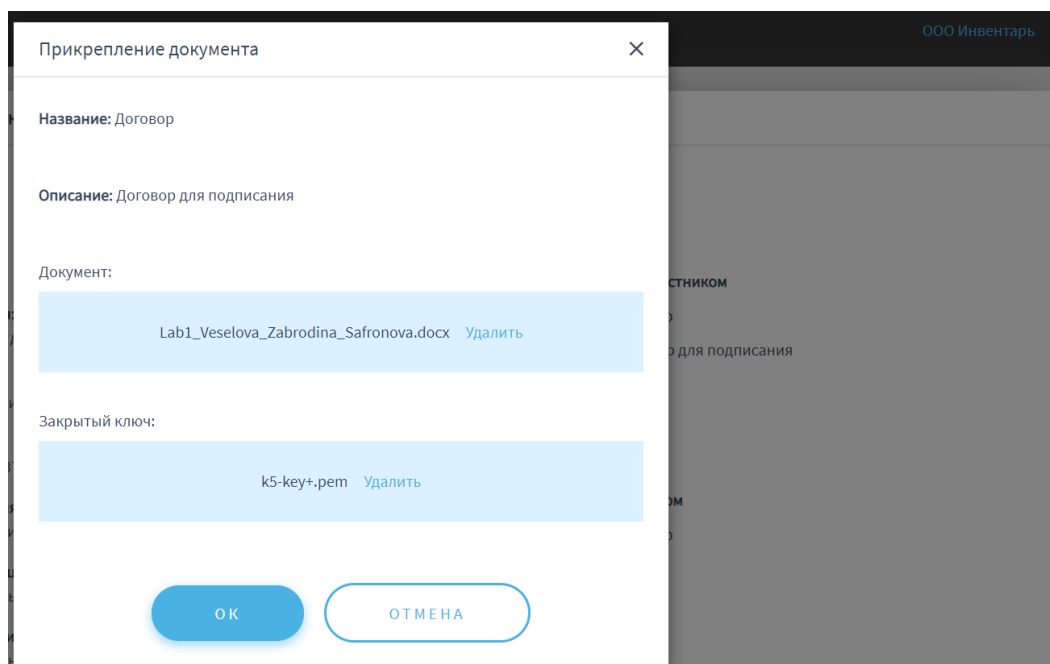


Рисунок 33 – Прикрепление документа к процессу

Данный шаг отправляется на подтверждение в блокчейн, рисунок 33.

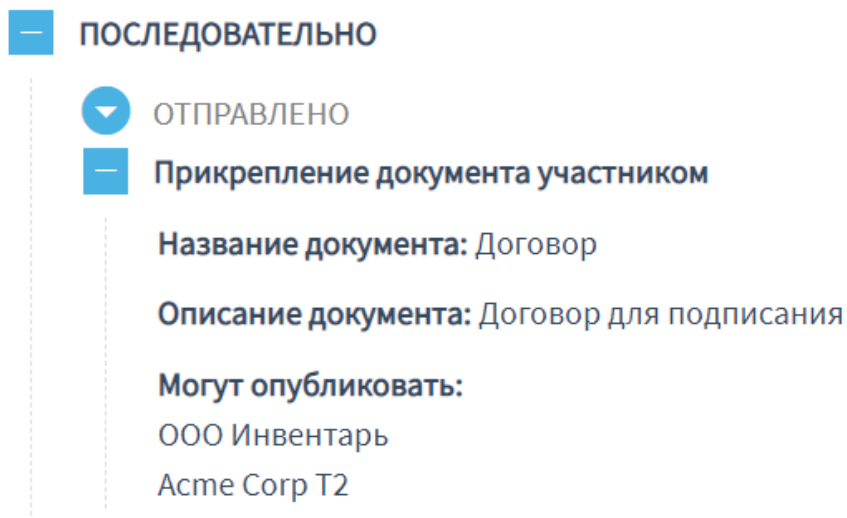


Рисунок 34 – Изменение статуса шага

Как только данная итерация подтверждена в блокчейн, пользователю приходит уведомление и становится доступным следующий шаг, рисунок 34.

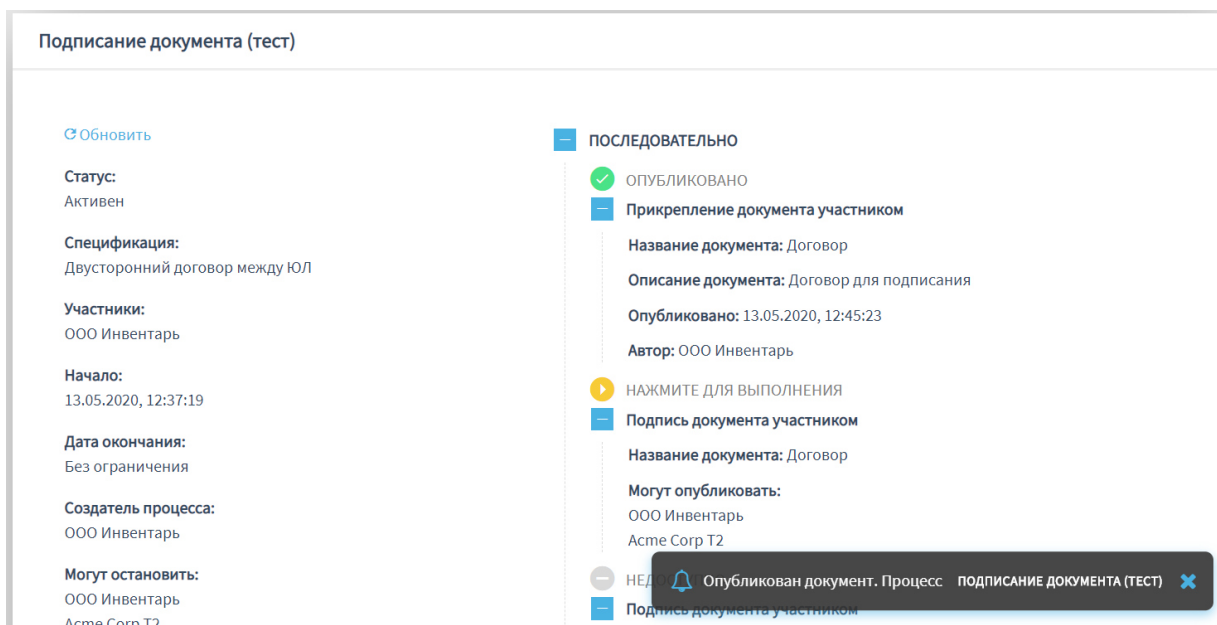


Рисунок 35 – Изменения статуса шага и уведомление о доступности следующего шага

Далее пользователю «AcmeCorp T2» необходимо подписать приложенный документ. Данная операция, аналогично предыдущей отправляется в блокчейн для фиксирования и подтверждения. Как только следующий шаг становится доступен, его необходимо выполнить первому пользователю – инициатору процесса. На данном этапе инициатор должен еще раз подписать документ и после подтверждения данной итерации в блокчейне завершить процесс, рисунки 35 и 36. Функция завершения процесса доступна каждому участнику, если все шаги выполнены согласно инструкции связанной с процессом спецификации и зафиксированы.

Также есть кнопка немедленного завершения процесса, в этом случае процесс прерывается на последнем, выполненном шаге и становится не доступным к дальнейшим действиям.

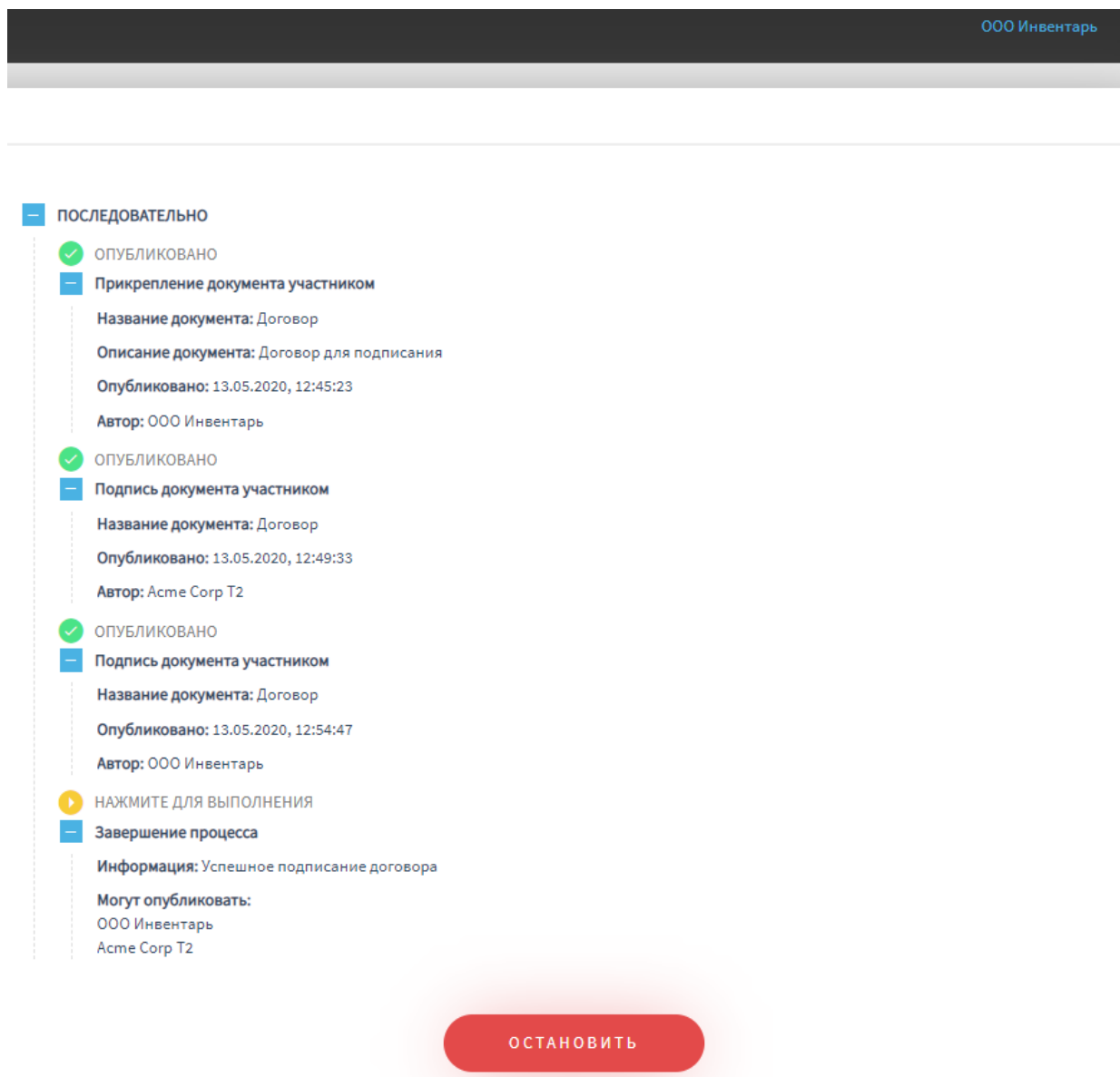


Рисунок 36 — Состояние процесса после подписания документа инициатором

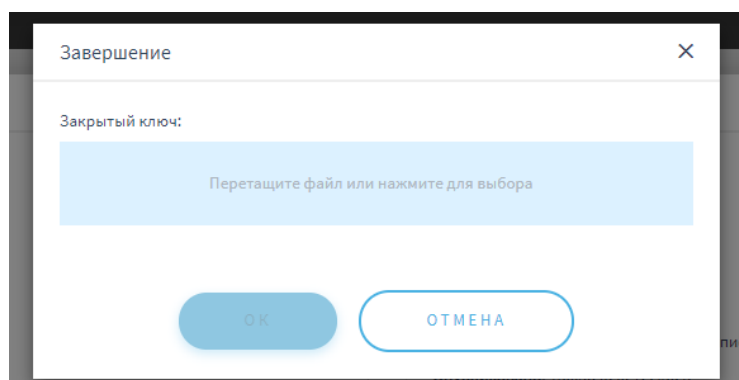


Рисунок 37 — Завершение процесса

После того, как процесс завершен, обоим пользователям приходит уведомление о завершении процесса. Процесс переходит в статус завершен на странице «Процессы», рисунок 37.

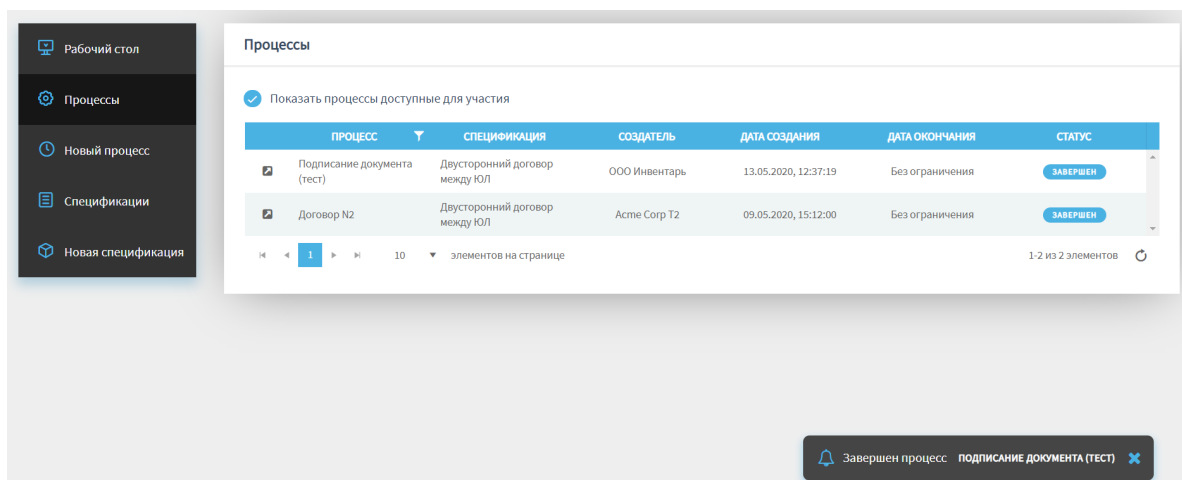


Рисунок 38 – Финальный статус процесса на экране «Процессы»

В конце страницы каждого процесса, есть блок для документов, которые были приложены во время его выполнения. Документы доступны к скачиванию и дальнейшему просмотру/редактированию, но уже локально на компьютере пользователя, скачавшего файл, рисунок 38. Для публикации измененного документа и фиксирование его в блокчейн необходимо запустить новый процесс, построенный на соответствующей спецификации.

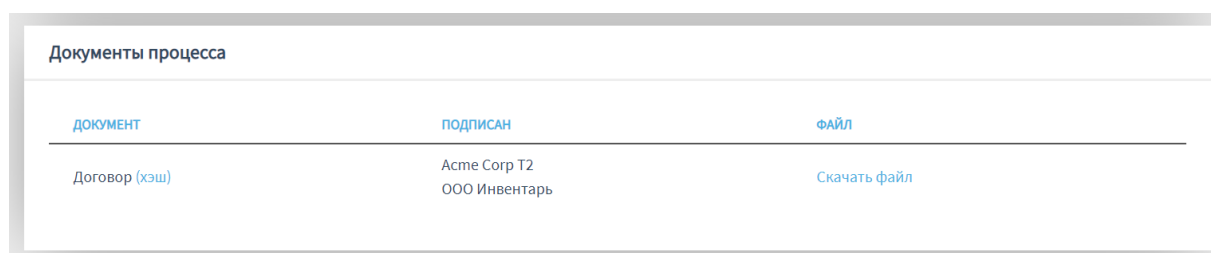


Рисунок 39 – Вложенные файлы процесса

3.8 Выводы по главе

Таким образом, ознакомившись с инструкцией по работе с приложением, пользователь, обладающий требуемыми навыками с легкостью может создавать спецификации и процессы, начинать и выполнять процесс

совместно с другими пользователями системы, прикреплять и подписывать документы, а также выполнять обзор своего рабочего стола в системе.

4. Безопасность жизнедеятельности

Для нормальной работы пользователя с программным обеспечением должны быть соблюдены рекомендации по обеспечению общих санитарно-технических норм к помещению и рабочим местам.

А именно, помещение, в котром планируется взаимодействие с разработанным продуктом, его размер (площадь, объем) должно соответствовать количеству полностью оснащенных рабочих мест сотрудников. Кроме того, должны быть учтены соответствующие нормы температуры, освещения, чистоты воздуха, обеспечена изоляция, от производственных шумов и т.п. На одного работающего сотрудника, согласно санитарным нормам труда СН 245-71, должно быть помещение объемом не менее 15 м³, площадью не менее 4,5 м².

Основные требования к организации рабочего места и его оборудованию прописаны в ГОСТ 12.2.032 78, а именно, высота рабочей поверхности стола для сотрудника должна регулироваться в пределах 680-800 мм; при отсутствии таковой возможности высота рабочей поверхности стола должна составлять 725 мм.

Рабочий стол должен иметь следующие параметры: ширина соответствует 800, 1200 или 1400 мм, глубина – 800 или 1000 мм при нерегулируемой высоте, которая равна 725 мм. Кроме того, пространство для ног высотой в рабочей части должно быть не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также по расстоянию спинки до переднего края сиденья.

5. Заключение

В ходе выполнения данной выпускной работы была изучена технология блокчейн и получены навыки по работе с фреймворками *Spring* и *Vue.js*. Результатами решения поставленной задачи являются:

- разработанная структура базы данных для СУБД *PostgreSQL*, которая позволяет хранить данные о пользователе, созданных им спецификаций и экземпляров процессов, данные по выполнению операций процесса, а также правила для отправки данных в *Multichain*;

- разработанный механизм транспортировки данных в *Multichain*. Транспортируются следующие данные: данные о пользователе (сертификат, ключи), данные о процессе (операции, выполненные в рамках процесса; хэш приложенного документа), данные о спецификации (хэш записи спецификации процесса);

- разработанный механизм фиксирования и заверения данных с использованием технологии блокчейн, реализованной в рамках одноранговой децентрализованной сети;

- разработанное и протестированное приложение «Конструктор создания распределенных контрактов на основе технологии блокчейн».

Благодаря используемой технологии блокчейн, разработанное приложение обеспечивает высокую степень защищенности хранимой информации, ее общедоступность, а также временную привязку, заверение и невозможность изменения электронных документов.

Список использованных источников

1. Генкин А. С., Михеев А. А. Блокчейн. Как это работает и что ждет нас завтра. — М.: Альпина Паблишер, 2018. — С. 562.
2. Свон М. Блокчейн: схема новой экономики — М.: Издательство «Олимп-Бизнес», 2017. — С. 17 — 19.
3. Roger Wattenhofer The Science of the Blockchain / Roger Wattenhofer — К. : Information technologies, 2016 — С. 94 – 120.
4. Andreas M. Antonopoulos Mastering Bitcoin: Unlocking Digital Cryptocurrencies / Andreas M. Antonopoulos — К. : NGITS, 2014. — С. 150 – 290
5. Имран Башир. Блокчейн: архитектура, криптовалюты, инструменты разработки, смарт-контракты — ДМК-Пресс, 2019. — С. 538.
6. Dr. Gideon Greenspan MultiChain Private Blockchain — White Paper / Dr. Gideon Greenspan — Founder and CEO, Coin Sciences Ltd, 2015. — С.17.
7. Дж. Уорсли, Дж. Дрейк. PostgreSQL. Для профессионалов — СПб.: «Питер», 2003. — С. 23-320.
8. Карл Вигерс, Джой Битти. Разработка требований к программному обеспечению — «Русская Редакция», 2016. — С. 796.
9. Леон Шкляр, Рич Розен. Архитектура веб-приложений. — М.: «Эксмо», 2010. — С.35-150.
10. Jesse Russell. Одноранговая сеть. — «Книга по требованию», 2012. — С. 23-99.
11. Шилдт Г. Java 8: руководство для начинающих. — М.: «Вильямс», 2018. — С. 149- 348.
12. Гради Буч, Джеймс Рамбо, Ивар Якобсон. Язык UML. Руководство пользователя, 2-е издание — ДМК «Пресс», 2007. — С. 221-367.

Приложение А. Листинги программы на Java

Файл «Actor.java»

```
package org.cds.core.actor;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.apache.commons.lang3.StringUtils;
import org.cds.core.selector.Selector;
import org.cds.core.selector.SelectorType;
import org.cds.core.specification.Specification;

import javax.persistence.*;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import java.util.UUID;

import static
org.cds.core.selector.SelectorService.toLineString;

@Entity
@Table(name = "actor")
public class Actor implements Serializable {
    @Id
    @Column(name="id")
    private UUID id;

    @Column(name = "rights")
    private Integer rights;

    @Column(name = "number")
    private Integer number;

    @Column(name = "title")
    private String title;

    @OneToMany(mappedBy = "actor", cascade =
CascadeType.ALL, orphanRemoval = true)
    private Set<Selector> selectors = new HashSet<>();

    @JsonIgnore
```

```

    @ManyToOne
    @JoinColumn(name = "specification_id", insertable =
false, updatable = false)
    private Specification specification;

    @Column(name = "specification_id", nullable =
false)
    private UUID specificationId;

    public boolean isEmpty() {
        return id == null || rights == null || number
== null || StringUtils.isEmpty(title) ||
        selectors.isEmpty() ||
        selectors.stream().anyMatch(s -> s.getNeedValue() && !
s.getHasValue());
    }

    public void addSelector(Selector selector) {
        this.selectors.add(selector);
        selector.setActor(this);
        selector.setActorId(this.id);
        selector.setType(SelectorType.ACTOR);
    }

    public void clearSelectors() {
        this.selectors.clear();
    }

    @JsonProperty(value = "selectorLine")
    public String getLineSelector() {
        return toLineString(selectors);
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public Integer getRights() {
        return rights;
    }

```

```

    }

    public void setRights(Integer rights) {
        this.rights = rights;
    }

    public Integer getNumber() {
        return number;
    }

    public void setNumber(Integer number) {
        this.number = number;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public Specification getSpecification() {
        return specification;
    }

    public void setSpecification(Specification
specification) {
        this.specification = specification;
    }

    public UUID getSpecificationId() {
        return specificationId;
    }

    public void setSpecificationId(UUID
specificationId) {
        this.specificationId = specificationId;
    }

    public Set<Selector> getSelectors() {
        return selectors;
    }

```

```

        public void setSelectors(Set<Selector> selectors) {
            this.selectors = selectors;
        }
    }
}

```

Файл «ActorRepository.java»

```

package org.cds.core.actor;

import
org.springframework.data.repository.PagingAndSortingRep
ository;
import org.springframework.stereotype.Repository;

import java.util.UUID;

@Repository
public interface ActorRepository extends
PagingAndSortingRepository<Actor, UUID> {
    Iterable<Actor> findAllBySpecificationId(UUID
specificationId);
}

```

Файл «ActorService.java»

```

package org.cds.core.actor;

import org.cds.core.actor.web.ActorRequest;
import org.cds.core.operation.*;
import org.cds.core.selector.Selector;
import org.cds.core.selector.SelectorService;
import org.cds.core.selector.web.SelectorRequest;
import org.cds.error.ApplicationException;
import org.cds.error.EntityNotFoundException;
import org.cds.util.MessageUtil;
import
org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactiona
l;

import java.util.Set;
import java.util.UUID;

import static org.cds.util.CommonUtil.newUuid;

@Service
public class ActorService {

    private final ActorRepository repository;
}

```

```

        private final SelectorService selectorService;
        private final OperationRepository
operationRepository;
        private final MessageUtil messageUtil;

        public ActorService(final ActorRepository
repository,
                                final SelectorService
selectorService,
                                final OperationRepository
operationRepository,
                                final MessageUtil messageUtil)
        {
            this.repository = repository;
            this.selectorService = selectorService;
            this.operationRepository = operationRepository;
            this.messageUtil = messageUtil;
        }

        public Actor create(ActorRequest request) {
            Actor actor = new Actor();
            actor.setId(new UUID());
            prepare(actor, request);
            return repository.save(actor);
        }

        public Actor update(Actor actor, ActorRequest
request) {
            prepare(actor, request);
            return repository.save(actor);
        }

        private void prepare(Actor actor, ActorRequest
request) {
            actor.setTitle(request.getTitle());
            actor.setNumber(request.getNumber());
            actor.setRights(request.getRights());

            actor.setSpecificationId(request.getSpecification());
            actor.clearSelectors();
            Set<SelectorRequest> selectorsRequest =
request.getSelectors();
            for (SelectorRequest selectorRequest :
selectorsRequest) {
                Selector selector =
selectorService.prepare(selectorRequest);
                actor.addSelector(selector);
            }
        }

        @Transactional
        public void delete(UUID id) {

```

```

        Actor actor;
        try {
            actor = repository.findById(id).get();
        } catch (EmptyResultDataAccessException e) {
            throw new EntityNotFoundException("Not
found actor");
        }
        UUID specificationId =
actor.getSpecificationId();
        Iterable<Operation> operations =
operationRepository.findAllByQueueSpecificationId(speci
ficationId);
        for (Operation oper : operations) {
            if (oper instanceof OperationPublish) {
                if (((OperationPublish)
oper).getActorNumber().equals(actor.getNumber()))
                    throw new ApplicationException(

messageUtil.getValidationMessage("actor.operationExists
"));
            } else if (oper instanceof OperationSign) {
                if (((OperationSign)
oper).getActorNumber().equals(actor.getNumber()))
                    throw new ApplicationException(

messageUtil.getValidationMessage("actor.operationExists
"));
            } else if (oper instanceof OperationFinish)
            {
                if (((OperationFinish)
oper).getActorNumber().equals(actor.getNumber()))
                    throw new ApplicationException(

messageUtil.getValidationMessage("actor.operationExists
"));
            }
        }
        for (Operation oper : operations) {
            if (oper instanceof OperationPublish) {
                if (((OperationPublish)
oper).getActorNumber() > actor.getNumber()) {
                    ((OperationPublish)
oper).setActorNumber(
((OperationPublish)
oper).getActorNumber() - 1);
                    operationRepository.save(oper);
                }
            } else if (oper instanceof OperationSign) {
                if (((OperationSign)
oper).getActorNumber() > actor.getNumber()) {
                    ((OperationSign)
oper).setActorNumber(

```

```

                                ((OperationSign)
oper).getActorNumber() - 1);
                                operationRepository.save(oper);
                                }
                                } else if (oper instanceof OperationFinish)
{
                                if (((OperationFinish)
oper).getActorNumber() > actor.getNumber()) {
                                ((OperationFinish)
oper).setActorNumber(
                                ((OperationFinish)
oper).getActorNumber() - 1);
                                operationRepository.save(oper);
                                }
                                }
                                }

repository.findAllBySpecificationId(specificationId).fo
rEach(act -> {
                                if (act.getNumber() > actor.getNumber()) {
                                act.setNumber(act.getNumber() - 1);
                                repository.save(act);
                                }
                                });
                                repository.deleteById(id);
                                }
}

```

Файл «ActorController.java»

```

package org.cds.core.actor.web;

import org.cds.core.actor.Actor;
import org.cds.core.actor.ActorRepository;
import org.cds.core.actor.ActorService;
import org.cds.error.EntityNotFoundException;
import
org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.Optional;
import java.util.UUID;

@RestController
@RequestMapping(value = "/actor")
public class ActorController {

    private final ActorRepository repository;
    private final ActorService service;

```

```

        private final ActorRequestValidator validator;

        public ActorController(final ActorRepository
repository,
                                final ActorService service,
                                final ActorRequestValidator
validator) {
            this.repository = repository;
            this.service = service;
            this.validator = validator;
        }

        @InitBinder
        public void setValidators(WebDataBinder binder) {
            binder.addValidators(validator);
        }

        @PostMapping(consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
        @ResponseBody
        @ResponseStatus(HttpStatus.CREATED)
        public Actor create(@Valid @RequestBody
ActorRequest request) {
            return service.create(request);
        }

        @PutMapping(value =("/{id}", consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
        @ResponseBody
        public Actor update(@PathVariable UUID id, @Valid
@RequestBody ActorRequest request) {
            Actor actor = repository.findById(id)
                .orElseThrow(() -> new
EntityNotFoundException("Not found actor"));
            return service.update(actor, request);
        }

        @GetMapping(value =("/{id}")
        @ResponseBody
        public Optional<Actor> get(@PathVariable UUID id) {
            return repository.findById(id);
        }

        @DeleteMapping(value =("/{id}")
        @ResponseStatus(HttpStatus.NO_CONTENT)
        public void delete(@PathVariable UUID id) {
            service.delete(id);
        }

        @GetMapping(value = "/actor/specification/
{specificationId}")
        @ResponseBody

```

```

        public Iterable<Actor>
getForSpecification(@PathVariable UUID specificationId)
{
    return
repository.findAllBySpecificationId(specificationId);
}

}

```

Файл «Process.java»

```

package org.cds.core.process;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.apache.commons.lang3.StringUtils;
import org.cds.common.ChainResult;
import org.cds.core.certificate.Certificate;
import org.cds.core.specification.Specification;
import
org.cds.core.specification.web.SpecificationSimpleResponse;
import org.cds.crypto.Signature;

import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
import java.util.UUID;

@Entity
@Table(name = "process")
@NamedEntityGraphs({
    @NamedEntityGraph(name = "graph.Process.constraints",
        attributeNodes = {
            @NamedAttributeNode(value =
"actorConstraints", subgraph = "selectors")
        },
        subgraphs = @NamedSubgraph(name =
"selectors", attributeNodes =
@NamedAttributeNode("selectors")))
})
public class Process implements Serializable {

    @Id
    @Column(name="id")
    private UUID id;

    @JsonIgnore
    @ManyToOne
    @JoinColumn(name = "specification_id", insertable =
false, updatable = false)
    private Specification specification;

```

```

@Column(name = "specification_id", nullable = false)
private UUID specificationId;

@Column(name="name")
private String name;

@Column(name="created_at")
private Date createdAt;

@Embedded
private Signature creatorSign;

@OneToMany(mappedBy = "process", cascade =
CascadeType.ALL)
private Set<ActorConstraint> actorConstraints = new
HashSet<>();

@Embedded
private ChainResult chainResult = new ChainResult();

/* relation from Signature.fingerprint */
@ManyToOne
@JoinColumn(name = "sign_fingerprint",
referencedColumnName = "fingerprint", insertable = false,
updatable = false)
private Certificate certificate;

@JsonProperty(value = "specification")
public SpecificationSimpleResponse getSpecificationName()
{
    return this.specification == null ? null : new
SpecificationSimpleResponse(this.specification);
}

@JsonProperty(value = "specificationTtl")

public void addActorConstraint(ActorConstraint
constraint) {
    this.actorConstraints.add(constraint);
    constraint.setProcess(this);
    constraint.setProcessId(this.id);
}

public boolean isEmpty() {
    return id == null || StringUtils.isEmpty(name) ||
createdAt == null ||
        specification == null || specificationId ==
null ||
        actorConstraints.isEmpty() ||
actorConstraints.stream().anyMatch(ActorConstraint::isEmpty)
||

```

```

        creatorSign.isEmpty();
    }

    public boolean isEmptyForSign() {
        return id == null || StringUtils.isEmpty(name) ||
        createdAt == null ||
            specification == null || specificationId ==
        null ||
            actorConstraints.isEmpty() ||
        actorConstraints.stream().anyMatch(ActorConstraint::isEmpty);
    }

    @PrePersist
    public void onPrePersist() {
        if (createdAt == null)
            createdAt = new Date();
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public Specification getSpecification() {
        return specification;
    }

    public void setSpecification(Specification specification)
    {
        this.specification = specification;
    }

    public UUID getSpecificationId() {
        return specificationId;
    }

    public void setSpecificationId(UUID specificationId) {
        this.specificationId = specificationId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Date getCreatedAt() {

```

```

        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    public Signature getCreatorSign() {
        return creatorSign;
    }

    public void setCreatorSign(Signature creatorSign) {
        this.creatorSign = creatorSign;
    }

    public Set<ActorConstraint> getActorConstraints() {
        return actorConstraints;
    }

    public void setActorConstraints(Set<ActorConstraint>
actorConstraints) {
        this.actorConstraints = actorConstraints;
    }

    public ChainResult getChainResult() {
        return chainResult;
    }

    public void setChainResult(ChainResult chainResult) {
        this.chainResult = chainResult;
    }

    public Certificate getCertificate() {
        return certificate;
    }

    public void setCertificate(Certificate certificate) {
        this.certificate = certificate;
    }
}

```

Файл «ProcessRepository.java»

```

package org.cds.core.process;

import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
import java.util.UUID;

@Repository

```

```

public interface ProcessRepository extends
CrudRepository<Process, UUID>, ProcessRepositoryCustom {
    @EntityGraph(value = "graph.Process.constraints", type =
EntityGraph.EntityGraphType.LOAD)
    Optional<Process> findOneByChainResultTxId(String txId);
    Optional<Process> findOneByName(String Name);
    boolean existsByName(String name);
}

```

Файл «ProcessService.java»

```

package org.cds.core.process;

import multichain.command.MultichainException;
import org.cds.client.notice.CdsNoticeService;
import org.cds.client.notice.event.Event;
import org.cds.client.notice.event.EventType;
import org.cds.common.CdsService;
import org.cds.common.CdsValidateService;
import org.cds.common.ChainStatus;
import org.cds.common.SignResponse;
import org.cds.core.certificate.Certificate;
import org.cds.core.certificate.CertificateRepository;
import org.cds.core.certificate.CertificateService;
import org.cds.core.certificate.converter.CertificateToProto;
import org.cds.core.client.Client;
import org.cds.core.client.ClientRepository;
import org.cds.core.newchain.ChainStreamWriterService;
import org.cds.core.process.converter.ProcessToProto;
import org.cds.core.process.converter.ProtoToProcess;
import org.cds.core.process.state.ProcessStateService;
import org.cds.core.process.state.ProcessStatus;
import org.cds.core.process.state.persistence.VProcessState;
import
org.cds.core.process.state.persistence.VProcessStateRepository;
import
org.cds.core.process.web.ProcessRequest;
import org.cds.core.process.web.ProcessUpdateRequest;
import org.cds.core.publish.PublishRepository;
import org.cds.core.specification.SpecificationRepository;
import org.cds.crypto.CryptographyService;
import org.cds.crypto.Signature;
import org.cds.crypto.SignatureAlgorithm;
import org.cds.error.InternalException;
import org.cds.error.ValidateException;
import org.cds.util.MessageUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Lazy;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

```

```

import
org.springframework.transaction.annotation.Transactional;

import java.security.cert.X509Certificate;
import java.util.*;

import static org.cds.util.CommonUtil.*;

@Service
public class ProcessService implements CdsService,
CdsValidateService {
    private static final Logger logger =
LoggerFactory.getLogger(ProcessService.class);

    private final MessageUtil messageUtil;
    private final ProcessRepository repository;
    private final ChainStreamWriterService
StreamWriterService;
    private final ProcessToProto toProto;
    private final ProtoToProcess toProcess;
    private final CryptographyService cryptoService;
    private final CertificateRepository
certificateRepository;
    private final SpecificationRepository
specificationRepository;
    private final CertificateToProto certToProto;
    private final VProcessStateRepository
processStateRepository;
    private final ProcessStateService processStateService;
    private final CdsNoticeService cdsNoticeService;
    private final ClientRepository clientRepository;

    public ProcessService(final MessageUtil messageUtil,
                          final ProcessRepository repository,
                          final @Lazy
ChainStreamWriterService StreamWriterService,
                          final ProcessToProto toProto,
                          final ProtoToProcess toProcess,
                          final CryptographyService
cryptoService,
                          final CertificateRepository
certificateRepository,
                          final SpecificationRepository
specificationRepository,
                          final CertificateToProto
certToProto,
                          final VProcessStateRepository
processStateRepository,
                          final ProcessStateService
processStateService,
                          final CdsNoticeService
cdsNoticeService,

```

```

                                final ClientRepository
clientRepository) {
    this.messageUtil = messageUtil;
    this.repository = repository;
    this.streamWriterService = streamWriterService;
    this.toProto = toProto;
    this.toProcess = toProcess;
    this.cryptoService = cryptoService;
    this.certificateRepository = certificateRepository;
    this.specificationRepository =
specificationRepository;
    this.certToProto = certToProto;
    this.processStateRepository = processStateRepository;
    this.processStateService = processStateService;
    this.cdsNoticeService = cdsNoticeService;
    this.clientRepository = clientRepository;
}

@Transactional
public Process create(ProcessRequest request) {
    Process process = new Process();
    process.setId(newUuid());
    prepare(process, request);
    return repository.save(process);
}

    public Process update(Process process,
ProcessUpdateRequest request) {
        Signature signature = process.getCreatorSign() ==
null ?
            new Signature() : process.getCreatorSign();

signature.setFingerPrint(request.getSignFingerPrint());
        SignatureAlgorithm signAlgorithm =
request.getSignAlgorithm();
        if (signAlgorithm == null)
            signAlgorithm = SignatureAlgorithm.sign_dummy;
        signature.setAlgorithm(signAlgorithm);
        String signBase64 = request.getSignBase64();
        byte[] bytes = decodeBase64(signBase64);
        signature.setSignBytes(bytes);
        process.setCreatorSign(signature);
        return repository.save(process);
}

    private void prepare(Process process, ProcessRequest
request) {
        Signature signature = process.getCreatorSign() ==
null ?
            new Signature() : process.getCreatorSign();

```

```

signature.setFingerPrint(request.getSignFingerPrint());
    SignatureAlgorithm signAlgorithm =
request.getSignAlgorithm();
    if (signAlgorithm == null)
        signAlgorithm = SignatureAlgorithm.sign_dummy;
    signature.setAlgorithm(signAlgorithm);
    String signBase64 = request.getSignBase64();
    byte[] bytes = decodeBase64(signBase64);
    signature.setSignBytes(bytes);
    process.setCreatorSign(signature);
    process.setName(request.getName());

process.setSpecificationId(request.getSpecification());
    process.getChainResult().setInChain(false);

process.getChainResult().setChainStatus(ChainStatus.UNKNOWN);
    }

    public void publish(Process p) {
        proto.msg.Process process = toProto.convert(p);
        String txId;
        try {
            txId = streamWriterService.writeProcess(process);
        } catch (MultichainException e) {
            throw new InternalException("Can't write to
stream " + e.toString());
        }
        p.getChainResult().setTxId(txId);
        p.getChainResult().setChainStatus(ChainStatus.SEND);
        p.setCreatedAt(new Date());
        repository.save(p);
        processStateService.create(p.getId());
        Client client =
clientRepository.findOneByCertificatesFingerPrint(p.getCreato
rSign().getFingerPrint()).get();
        cdsNoticeService.sendEvent(client, new
Event(EventType.SEND_PROCESS));
    }

    public Process createFromChain(proto.msg.Process p,
String txId, Date createdAt) {
        Process process = toProcess.convert(p);
        process.setCreatedAt(createdAt);
        process.getChainResult().setTxId(txId);
        process.getChainResult().setInChain(true);

process.getChainResult().setChainStatus(ChainStatus.SUCCESS);
        return repository.save(process);
    }

    public void approve(Process p, Date createdAt) {
        p.setCreatedAt(createdAt);

```

```

        p.getChainResult().setInChain(true);

p.getChainResult().setChainStatus(ChainStatus.SUCCESS);
        p.getChainResult().setRejectInfo(null);
        repository.save(p);
    }

    @Override
    public void validate(Object o) throws ValidateException {
        if (o == null)
            throw new ValidateException("Validated object is
null!");
        if (!(o instanceof proto.msg.Process))
            throw new InternalException("Not implemented
operation type");
        proto.msg.Process p = (proto.msg.Process) o;
        /*Process proc =
repository.findById(parseUUID(p.getGuid()))
                .orElseThrow(() -> new
NotFoundException("Process not found: " + p.getGuid()));
        if (proc.getChainResult().isInChain()) {
            throw new ValidateException(

messageUtil.getValidationMessage("process.AlreadyExists",
"guid" + p.getGuid()));
        }*/
        Process proc =
repository.findOneByName(p.getName()).orElse(null);
        if (proc != null &&
proc.getChainResult().isInChain()) {
            throw new ValidateException(

messageUtil.getValidationMessage("process.AlreadyExists",
"name" + p.getName()));
        }
        if (!
specificationRepository.findById(parseUUID(p.getSpecGuid()))
.isPresent()) {
            throw new ValidateException(
                "Specification with guid not found: " +
p.getSpecGuid() + " for process with guid: " + p.getGuid());
        }
        String fingerprint =
p.getCreatorSign().getFingerprint();
        proto.msg.Certificate certificate =
certToProto.convert(

certificateRepository.findById(fingerprint).orElseThrow(() ->
new ValidateException(

messageUtil.getValidationMessage("certificate.NotExists",
fingerprint)))));

```

```

        try {
            if (!verifySignature(p, certificate)) {
                throw new
ValidateException(messageUtil.getValidationMessage("signature
.NotValid"));
            }
        } catch (Exception e) {
            throw new
ValidateException(messageUtil.getValidationMessage("signature
.FailedCheck", e.getMessage()));
        }
    }

    @Override
    public void reject(String txId, Throwable t) {

repository.findOneByChainResultTxId(txId).ifPresent(process
-> {
            boolean inChain =
process.getChainResult().isInChain();
            if (!inChain) {

process.getChainResult().setChainStatus(ChainStatus.FAIL);
                repository.save(process);
                Client client =
clientRepository.findOneByCertificatesFingerPrint(process.get
CreatorSign().getFingerPrint()).get();
                cdsNoticeService.sendEvent(client, new
Event(EventType.REJECT_PROCESS));
            }
        });

        public Optional<Process> getByTxId(String createTxId32) {
            return
repository.findOneByChainResultTxId(createTxId32);
        }

        /* validate method for front service */
        public void validate(Process process) throws
ValidateException {
            Certificate cert =
certificateRepository.findById(process.getCreatorSign().getFi
ngerPrint())
                .orElseThrow(() -> new ValidateException(

messageUtil.getValidationMessage("certificate.NotExists",
process.getCreatorSign().getFingerPrint())));
            if (!cert.getChainResult().isInChain())
                throw new ValidateException(

```

```

messageUtil.getValidationMessage("certificate.UnChain",
process.getCreatorSign().getFingerPrint()));
    if (process.isEmpty()) {
        if (process.getCreatorSign() == null ||
process.getCreatorSign().isEmpty()) {
            throw new ValidateException(

messageUtil.getValidationMessage("signature.Missing"));
        }
        throw new
ValidateException(messageUtil.getValidationMessage("process.I
sEmpty", process.toString()));
    }
}

    public boolean verifySignature(proto.msg.Process p,
proto.msg.Certificate c) throws Exception {
        byte[] certificateBytes = c.getBody().toByteArray();
        byte[] signatureBytes =
p.getCreatorSign().getSignBytes().toByteArray();
        X509Certificate x509Certificate =
cryptoService.loadX509CertFromBytes(certificateBytes);
        return cryptoService.verifySignature(x509Certificate,
getSignedBytes(p), signatureBytes);
    }

    public byte[] getSignedBytes(proto.msg.Process p) {
        proto.msg.Process.Builder b =
proto.msg.Process.newBuilder();
        b.mergeFrom(p);
        b.clearCreatorSign();
        b.clearCreatedAt();
        return b.build().toByteArray();
    }

    public SignResponse getSignResponse(Process process) {
        proto.msg.Process proc =
toProto.convertForSign(process);
        byte[] body = proc.toByteArray();
        String bodyBase64 = encodeBase64(body);
        return new SignResponse(bodyBase64,
bodyBase64.length(), body.length);
    }

    public ProcessStatus getStatusForProcess(Process process)
{
        VProcessState state =
processStateRepository.findById(process.getId()).orElse(null)
;
        if (state != null)
            return state.getStatus();
    }

```

```

        return null;
    }

    public Page<ProcessWithStatus> findWithStatus(UUID
clientId, String filter, Pageable pageable, boolean actual) {
        Page<Process> processes = repository.find(clientId,
filter, pageable, actual);
        List<ProcessWithStatus> processesWithStatus = new
ArrayList<>();
        processes.forEach(process -> {
            ProcessWithStatus processWithStatus = new
ProcessWithStatus();
            processWithStatus.setProcess(process);

processWithStatus.setStatus(getStatusForProcess(process));
            if (process.getSpecification().getTtl() > 0)
                processWithStatus.setTimeover(
                    new
Date(process.getCreatedAt().getTime() +
process.getSpecification().getTtl()));
            processesWithStatus.add(processWithStatus);
        });
        return new PageImpl<>(processesWithStatus, pageable,
processes.getTotalElements());
    }
}

```

Файл «Specification.java»

```

package org.cds.core.specification;

import com.fasterxml.jackson.annotation.JsonIgnore;
import org.apache.commons.lang3.StringUtils;
import org.cds.common.ChainResult;
import org.cds.core.actor.Actor;
import org.cds.core.certificate.Certificate;
import org.cds.core.client.Client;
import org.cds.core.operation.OperationQueue;
import org.cds.crypto.Signature;

import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
import java.util.UUID;

@Entity
@Table(name = "specification")
public class Specification implements Serializable {

    @Id
    @Column(name="id")

```

```

private UUID id;

@Column(name = "name")
private String name;

@Column(name = "client_id", nullable = false)
private UUID clientId;

@JsonIgnore
@ManyToOne
@JoinColumn(name = "client_id", insertable = false,
updatable = false)
private Client client;

@Column(name = "ttl")
private long ttl;

@Column(name = "created_at")
private Date createdAt;

@Embedded
private Signature signature;

/* relation from Signature.fingerprint */
@ManyToOne
@JoinColumn(name = "sign_fingerprint",
referencedColumnName = "fingerprint", insertable = false,
updatable = false)
private Certificate certificate;

@OneToMany(mappedBy = "specification", cascade =
CascadeType.ALL)
private Set<OperationQueue> queues = new HashSet<>();

@OneToMany(mappedBy = "specification", cascade =
CascadeType.ALL)
private Set<Actor> actors = new HashSet<>();

@Embedded
private ChainResult chainResult = new ChainResult();

public boolean isEmpty() {
    return id == null || StringUtils.isEmpty(name) ||
clientId == null || createdAt == null ||
signature.isEmpty() || queues.size() == 0 ||
queues.stream().anyMatch(OperationQueue::isEmpty) ||
actors.isEmpty() ||
actors.stream().anyMatch(Actor::isEmpty);
}

public boolean isEmptyForSign() {
    return id == null || StringUtils.isEmpty(name) ||

```

```

clientId == null || createdAt == null ||
        queues.size() == 0 ||
queues.stream().anyMatch(OperationQueue::isEmpty) ||
        actors.isEmpty() ||
actors.stream().anyMatch(Actor::isEmpty);
    }

    @Override
    public String toString() {
        return "Specification{" +
            "name='" + name + '\'' +
            ", ttl=" + ttl +
            ", createdAt=" + createdAt +
            ", signature=" + signature.getFingerPrint() +
            ", queues=" + queues.size() +
            ", actors=" + actors.size() +
            '}';
    }

    @PrePersist
    public void onPrePersist() {
        createdAt = new Date();
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getTtl() {
        return ttl;
    }

    public void setTtl(long ttl) {
        this.ttl = ttl;
    }

    public Date getCreatedAt() {
        return createdAt;
    }
}

```

```
public void setCreatedAt(Date createdAt) {
    this.createdAt = createdAt;
}

public Signature getSignature() {
    return signature;
}

public void setSignature(Signature signature) {
    this.signature = signature;
}

public Set<OperationQueue> getQueues() {
    return queues;
}

public void setQueues(Set<OperationQueue> queues) {
    this.queues = queues;
}

public void addQueue(OperationQueue queue) {
    queue.setSpecificationId(this.id);
    queue.setSpecification(this);
    this.queues.add(queue);
}

public Set<Actor> getActors() {
    return actors;
}

public void setActors(Set<Actor> actors) {
    this.actors = actors;
}

public void addActor(Actor actor) {
    actor.setSpecification(this);
    actor.setSpecificationId(this.id);
    this.actors.add(actor);
}

public void deleteActor(Actor actor) {
    this.actors.remove(actor);
}

public void clearActors() {
    this.actors = new HashSet<>();
}

public ChainResult getChainResult() {
    return chainResult;
}
```

```

    public void setChainResult(ChainResult chainResult) {
        this.chainResult = chainResult;
    }

    public UUID getClientId() {
        return clientId;
    }

    public void setClientId(UUID clientId) {
        this.clientId = clientId;
    }

    public Client getClient() {
        return client;
    }

    public void setClient(Client client) {
        this.client = client;
    }

    public Certificate getCertificate() {
        return certificate;
    }

    public void setCertificate(Certificate certificate) {
        this.certificate = certificate;
    }
}

```

Файл «SpecificationRepository.java»

```

package org.cds.core.specification;

import
org.springframework.data.jpa.repository.JpaSpecificationExecu
tor;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
import java.util.UUID;

@Repository
public interface SpecificationRepository extends
CrudRepository<Specification, UUID>,
    JpaSpecificationExecutor<Specification> {
    Optional<Specification> findOneByChainResultTxId(String
txId);
    Optional<Specification> findOneByName(String Name);
    boolean existsByName(String name);
}

```

Файл «SpecificationService.java»

```

package org.cds.core.specification;

import multichain.command.MultichainException;
import org.cds.client.notice.CdsNoticeService;
import org.cds.client.notice.event.Event;
import org.cds.client.notice.event.EventType;
import org.cds.common.CdsService;
import org.cds.common.CdsValidateService;
import org.cds.common.ChainStatus;
import org.cds.common.SignResponse;
import org.cds.core.certificate.Certificate;
import org.cds.core.certificate.CertificateRepository;
import org.cds.core.certificate.converter.CertificateToProto;
import org.cds.core.client.Client;
import org.cds.core.client.ClientService;
import org.cds.core.newchain.ChainStreamWriterService;
import org.cds.core.operation.Operation;
import org.cds.core.operation.OperationQueue;
import
org.cds.core.specification.converter.ProtoToSpecification;
import
org.cds.core.specification.converter.SpecificationToProto;
import org.cds.core.specification.web.SpecificationRequest;
import
org.cds.core.specification.web.SpecificationUpdateRequest;
import org.cds.crypto.CryptographyService;
import org.cds.crypto.Signature;
import org.cds.crypto.SignatureAlgorithm;
import org.cds.error.InternalException;
import org.cds.error.ValidateException;
import org.cds.util.MessageUtil;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.security.cert.X509Certificate;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import static org.cds.util.CommonUtil.*;

@Service
public class SpecificationService implements CdsService,
CdsValidateService {

    private final SpecificationRepository repository;
    private final CryptographyService cryptoService;
    private final ChainStreamWriterService
streamWriterService;

```

```

        private final ClientService clientService;
        private final ProtoToSpecification toSpecification;
        private final SpecificationToProto toProto;
        private final MessageUtil messageUtil;
        private final CertificateRepository
certificateRepository;
        private final CertificateToProto certToProto;
        private final CdsNoticeService cdsNoticeService;

        public SpecificationService(final SpecificationRepository
repository,
                                   final CryptographyService
cryptoService,
                                   final @Lazy
ChainStreamWriterService streamWriterService,
                                   final ClientService
clientService,
                                   final ProtoToSpecification
toSpecification,
                                   final SpecificationToProto
toProto,
                                   final MessageUtil
messageUtil,
                                   final CertificateRepository
certificateRepository,
                                   final CertificateToProto
certToProto,
                                   final CdsNoticeService
cdsNoticeService) {
            this.repository = repository;
            this.cryptoService = cryptoService;
            this.streamWriterService = streamWriterService;
            this.clientService = clientService;
            this.toSpecification = toSpecification;
            this.toProto = toProto;
            this.messageUtil = messageUtil;
            this.certificateRepository = certificateRepository;
            this.certToProto = certToProto;
            this.cdsNoticeService = cdsNoticeService;
        }

        @Transactional
        public Specification create(SpecificationRequest request)
        {
            Client client =
clientService.getOrCreate(request.getClientId());
            Specification specification = new Specification();
            specification.setId(newUuid());
            specification.setName(request.getName());
            specification.setClientId(client.getId());
            specification.setTtl(request.getTtl());
            Signature signature = new Signature();

```

```

signature.setFingerPrint(request.getSignFingerPrint());
    SignatureAlgorithm signAlgorithm =
request.getSignAlgorithm();
    if (signAlgorithm == null)
        signAlgorithm = SignatureAlgorithm.sign_dummy;
    signature.setAlgorithm(signAlgorithm);
    String signBase64 = request.getSignBase64();
    byte[] bytes = decodeBase64(signBase64);
    signature.setSignBytes(bytes);
    specification.setSignature(signature);
    specification.getChainResult().setInChain(false);

specification.getChainResult().setChainStatus(ChainStatus.UN
KNOWN);
    cdsNoticeService.sendEvent(client, new
Event(EventType.CREATE_SPECIFICATION));
    return repository.save(specification);
}

    public Specification update(Specification specification,
SpecificationUpdateRequest request) {
        // not update name and clientId
        specification.setTtl(request.getTtl());
        Signature signature = specification.getSignature() ==
null ?
            new Signature() :
specification.getSignature();

signature.setFingerPrint(request.getSignFingerPrint());
    SignatureAlgorithm signAlgorithm =
request.getSignAlgorithm();
    if (signAlgorithm == null)
        signAlgorithm = SignatureAlgorithm.sign_dummy;
    signature.setAlgorithm(signAlgorithm);
    String signBase64 = request.getSignBase64();
    byte[] bytes = decodeBase64(signBase64);
    signature.setSignBytes(bytes);
    specification.setSignature(signature);
    specification.getChainResult().setInChain(false);

specification.getChainResult().setChainStatus(ChainStatus.UN
KNOWN);
    Client client =
clientService.getOrCreate(specification.getClientId());
    cdsNoticeService.sendEvent(client, new
Event(EventType.UPDATE_SPECIFICATION));
    return repository.save(specification);
}

    public void delete(Specification specification) {
        Client client =

```

```

        clientService.getOrCreate(specification.getClientId());
        cdsNoticeService.sendEvent(client, new
Event(EventType.DELETE_SPECIFICATION));
        repository.deleteById(specification.getId());
    }

    @Transactional
    public Specification
createFromChain(proto.msg.Specification s, String txId, Date
createdAt) {
        Specification specification =
toSpecification.convert(s);
        Client client =
clientService.getOrCreate(specification.getClientId());
        specification.setCreatedAt(createdAt);
        specification.getChainResult().setTxId(txId);
        specification.getChainResult().setInChain(true);

specification.getChainResult().setChainStatus(ChainStatus.SU
CCESS);
        cdsNoticeService.sendEvent(client, new
Event(EventType.PUBLISH_SPECIFICATION));
        return repository.save(specification);
    }

    public void approve(Specification s, Date createdAt) {
        s.setCreatedAt(createdAt);
        s.getChainResult().setInChain(true);

s.getChainResult().setChainStatus(ChainStatus.SUCCESS);
        s.getChainResult().setRejectInfo(null);
        Client client =
clientService.getOrCreate(s.getClientId());
        cdsNoticeService.sendEvent(client, new
Event(EventType.PUBLISH_SPECIFICATION));
        repository.save(s);
    }

    @Override
    public void validate(Object o) throws ValidateException {
        if (o == null)
            throw new ValidateException("Validated object is
null!");
        if (!(o instanceof proto.msg.Specification))
            throw new InternalException("Not implemented
operation type");
        proto.msg.Specification s = (proto.msg.Specification)
o;
        Specification spec =
repository.findById(parseUUID(s.getGuid())).orElse(null);
        if (spec != null &&
spec.getChainResult().isInChain()) {

```

```

        throw new ValidateException(

messageUtil.getValidationMessage("specification.AlreadyExists
", "guid" + s.getName()));
    }
    spec =
repository.findOneByName(s.getName()).orElse(null);
    if (spec != null &&
spec.getChainResult().isInChain()) {
        throw new ValidateException(

messageUtil.getValidationMessage("specification.AlreadyExists
", "name" + s.getName()));
    }
    String fingerPrint =
s.getCreatorSign().getFingerPrint();
    proto.msg.Certificate certificate =
certToProto.convert(

certificateRepository.findById(fingerPrint).orElseThrow(() ->
    new ValidateException(

messageUtil.getValidationMessage("certificate.NotExists",
fingerPrint))));
    try {
        if (!verifySignature(s, certificate)) {
            throw new
ValidateException(messageUtil.getValidationMessage("signature
.NotValid"));
        }
    } catch (Exception e) {
        throw new
ValidateException(messageUtil.getValidationMessage("signature
.FailedCheck", e.getMessage()));
    }
}

@Override
public void reject(String txId, Throwable t) {

repository.findOneByChainResultTxId(txId).ifPresent(specifica
tion -> {
    if (!specification.getChainResult().isInChain())
{

specification.getChainResult().setChainStatus(ChainStatus.FA
IL);

specification.getChainResult().setRejectInfo(t.getMessage());
        Client client =
clientService.getOrCreate(specification.getClientId());
        cdsNoticeService.sendEvent(client, new

```

```

        Event(EventType.REJECT_SPECIFICATION));
        repository.save(specification);
    }
    });
}

    public Optional<Specification> getByTxId(String
createTxId32) {
        return
repository.findOneByChainResultTxId(createTxId32);
    }

    public void sign(Specification s, String privateKeyPath,
String x509certPath) throws Exception {
        proto.msg.Specification spec =
toProto.convertForSign(s);
        byte[] signedBytes = spec.toByteArray();
        Signature sign =
cryptoService.makeCdsSignature(signedBytes, privateKeyPath,
x509certPath);
        s.setSignature(sign);
    }

    public void publish(Specification s) {
        proto.msg.Specification specification =
toProto.convert(s);
        String txId;
        try {
            txId =
streamWriterService.writeSpecification(specification);
        } catch (MultichainException e) {
            throw new InternalException("Can't write to
stream " + e.toString());
        }
        s.getChainResult().setTxId(txId);
        s.getChainResult().setChainStatus(ChainStatus.SEND);
        Client client =
clientService.getOrCreate(s.getClientId());
        cdsNoticeService.sendEvent(client, new
Event(EventType.SEND_SPECIFICATION));
        repository.save(s);
    }

    /* validate method for front service */
    public void validate(Specification spec) throws
ValidateException {
        Certificate cert = spec.getCertificate();
        if (!cert.getChainResult().isInChain())
            throw new ValidateException(

messageUtil.getValidationMessage("certificate.UnChain",

```

```

spec.getCertificate().getFingerPrint()));
    if (spec.isEmpty()) {
        if (spec.getSignature() == null ||
spec.getSignature().isEmpty()) {
            throw new ValidateException(

messageUtil.getValidationMessage("signature.Missing"));
        }
        throw new
ValidateException(messageUtil.getValidationMessage("specifica
tion.IsEmpty", spec.toString()));
    }
    List<Integer> queueOrders = spec.getQueues()
        .stream()
        .map(OperationQueue::getOrder)
        .collect(Collectors.toList());
    if (hasDuplicate(queueOrders)) {
        throw new
ValidateException(messageUtil.getValidationMessage("queue.Dup
licateOrderIn"));
    }
    for (OperationQueue queue : spec.getQueues()) {
        if (queue.isEmpty()) {
            throw new
ValidateException(messageUtil.getValidationMessage("queue.IsE
mpty", queue.toString()));
        }
        List<Integer> operationOrders =
queue.getOperations()
            .stream()
            .map(Operation::getOrder)
            .collect(Collectors.toList());
        if (hasDuplicate(operationOrders)) {
            throw new
ValidateException(messageUtil.getValidationMessage("operation
.DuplicateOrderIn"));
        }
        for (Operation operation : queue.getOperations())
        {
            if (operation.isEmpty()) {
                throw new
ValidateException(messageUtil.getValidationMessage("operation
.IsEmpty", operation.toString()));
            }
        }
    }
}

    public boolean verifySignature(proto.msg.Specification s,
proto.msg.Certificate c) throws Exception {
        byte[] certificateBytes = c.getBody().toByteArray();
        byte[] signatureBytes =

```

```

s.getCreatorSign().getSignBytes().toByteArray();
    X509Certificate x509Certificate =
cryptoService.loadX509CertFromBytes(certificateBytes);
    return cryptoService.verifySignature(x509Certificate,
getSignedBytes(s), signatureBytes);
}

    public byte[] getSignedBytes(proto.msg.Specification s) {
        proto.msg.Specification.Builder b =
proto.msg.Specification.newBuilder();
        b.mergeFrom(s);
        b.clearCreatorSign();
        b.clearCreatedAt();
        return b.build().toByteArray();
    }

    public SignResponse getSignResponse(Specification s) {
        proto.msg.Specification spec =
toProto.convertForSign(s);
        byte[] body = spec.toByteArray();
        String bodyBase64 = encodeBase64(body);
        return new SignResponse(bodyBase64,
bodyBase64.length(), body.length);
    }
}

```

Файл «ChainResult.java»

```

package org.cds.common;

import javax.persistence.Column;
import javax.persistence.Embeddable;
import java.io.Serializable;

@Embeddable
public class ChainResult implements Serializable {
    @Column(name = "tx_id")
    private String txId;

    @Column(name = "in_chain")
    private boolean inChain;

    @Column(name = "reject_info")
    private String rejectInfo;

    @Column(name = "chain_status")
    private ChainStatus chainStatus;

    public String getTxId() {
        return txId;
    }

    public void setTxId(String txId) {

```

```

        this.txId = txId;
    }

    public boolean isInChain() {
        return inChain;
    }

    public void setInChain(boolean inChain) {
        this.inChain = inChain;
    }

    public String getRejectInfo() {
        return rejectInfo;
    }

    public void setRejectInfo(String rejectInfo) {
        this.rejectInfo = rejectInfo;
    }

    public ChainStatus getChainStatus() {
        return chainStatus;
    }

    public void setChainStatus(ChainStatus chainStatus) {
        this.chainStatus = chainStatus;
    }
}

```

Файл «ChainStatus.java»

```

package org.cds.common;

public enum ChainStatus {
    UNKNOWN,
    SEND,
    FAIL,
    SUCCESS
}

```

Файл «CommandElt.java»

```

package multichain.command;

import java.util.HashMap;
import multichain.object.AddedNodeInfo;
import multichain.object.Address;
import multichain.object.AddressReceived;
import multichain.object.AssetInfo;
import multichain.object.AssetTransaction;
import multichain.object.BalanceAsset;
import multichain.object.BalanceAssetGeneral;
import multichain.object.Block;
import multichain.object.BlockTemplate;
import multichain.object.BlockchainInfo;

```

```

import multichain.object.ChainTip;
import multichain.object.ChunkqueueInfo;
import multichain.object.ChunkqueueTotal;
import multichain.object.KeyPairs;
import multichain.object.LastBlockInfo;
import multichain.object.MemPoolInfo;
import multichain.object.MiningInfo;
import multichain.object.MultiBalance;
import multichain.object.MultisigAddress;
import multichain.object.NetTotals;
import multichain.object.NetworkInfo;
import multichain.object.PeerInfo;
import multichain.object.Permission;
import multichain.object.Publisher;
import multichain.object.RawMemPoolTransaction;
import multichain.object.RuntimeParams;
import multichain.object.Script;
import multichain.object.SignedTransactionRAW;
import multichain.object.SinceBlock;
import multichain.object.Stream;
import multichain.object.StreamKey;
import multichain.object.StreamKeyItem;
import multichain.object.Transaction;
import multichain.object.TransactionCurrency;
import multichain.object.TransactionRAW;
import multichain.object.TransactionWallet;
import multichain.object.TxIdVout;
import multichain.object.TxOut;
import multichain.object.TxOutInfo;
import multichain.object.Unspent;
import multichain.object.WalletInfo;

public enum CommandElt {
    ADDMULTISIGADDRESS("addmultisigaddress", null, new Class<?>
>[] {String.class}),
    ADDNODE("addnode", null, null),
    APPENDBINARYCACHE("appendbinarycache", null, new Class<?>[]
{Long.class}),
    APPENDRAWCHANGE("appendrawchange", null, new Class<?>[]
{String.class}),
    APPENDRAWDATA("appendrawdata", null, new Class<?>[]
{String.class}),
    APPENDRAWEXCHANGE("appendrawexchange", null, new Class<?>[]
{SignedTransactionRAW.class}),
    APPENDRAWTRANSACTION("appendrawtransaction", null,
    new Class<?>[] {String.class,
SignedTransactionRAW.class}),
    APPROVEFROM("approvefrom", null, new Class<?>[]
{String.class}),
    BACKUPWALLET("backupwallet", null, null),
    CLEARMEMPOOL("clearmempool", null, null),

```

```

    COMBINEUNSPENT("combineunspent", null, new Class<?>[]
{String.class}, true),
    COMPLETERAWEXCHANGE("completerawexchange", null, new
Class<?>[] {String.class}),
    CREATE("create", null, new Class<?>[] {String.class}),
    CREATEBINARYCACHE("createbinarycache", null, new Class<?>[]
{String.class}),
    CREATEFROM("createfrom", null, new Class<?>[]
{String.class}),
    CREATEKEYPAIRS("createkeypairs", null, new Class<?>[]
{KeyPairs.class}, true),
    CREATMULTISIG("createmultisig", null, new Class<?>[]
{MultisigAddress.class}),
    CREATERAWEXCHANGE("createrawexchange", null, new Class<?>[]
{String.class}),
    CREATERAWSENDFROM("createrawsendfrom", null,
    new Class<?>[] {String.class,
SignedTransactionRAW.class}),
    CREATERAWTRANSACTION("createrawtransaction", null,
    new Class<?>[] {String.class,
SignedTransactionRAW.class}),
    DECODERAWEXCHANGE("decoderawexchange"),
    DECODERAWTRANSACTION("decoderawtransaction", null, new
Class<?>[] {TransactionRAW.class}),
    DECODESCRIPT("decodescript", null, new Class<?>[]
{Script.class}),
    DELETEBINARYCACHE("deletebinarycache", null, null),
    DISABLERAWTRANSACTION("disablerawtransaction", null, new
Class<?>[] {String.class}),
    DUMPPrivKEY("dumpprivkey", null, new Class<?>[]
{String.class}),
    DUMPWALLET("dumpwallet", null, null),
    ENCRYPTWALLET("encryptwallet", null, null),
    ESTIMATEFEE("estimatefee", null, new Class<?>[]
{Long.class}),
    ESTIMATEPRIORITY("estimatepriority", null, new Class<?>[]
{Long.class}),
    GETACCOUNT("getaccount", null, new Class<?>[]
{String.class}),
    GETACCOUNTADDRESS("getaccountaddress", null, new Class<?>[]
{String.class}),
    GETADDEDNODEINFO("getaddednodeinfo", null, new Class<?>[]
{AddedNodeInfo.class}, true),
    GETADDRESSBALANCES("getaddressbalances", null, new Class<?
>[] {BalanceAssetGeneral.class}, true),
    GETADDRESSES("getaddresses", null, new Class<?>[]
{String.class, Address.class}, true),
    GETADDRESSESBYACCOUNT("getaddressesbyaccount", null, new
Class<?>[] {String.class}, true),
    GETADDRESSTRANSACTION("getaddresstransaction", null, new
Class<?>[] {TransactionWallet.class}),

```

```

    GETASSETBALANCES("getassetbalances", null, new Class<?>[]
{BalanceAsset.class}, true),
    GETASSETINFO("getassetinfo", null, new Class<?>[]
{AssetInfo.class}),
    GETASSETTRANSACTION("getassettransaction", null, new
Class<?>[] {TransactionWallet.class}),
    GETBALANCE("getbalance", null, new Class<?>[]
{Double.class}),
    GETBESTBLOCKHASH("getbestblockhash", null, new Class<?>[]
{String.class}),
    GETBLOCK("getblock", null, new Class<?>[] {Block.class}),
    GETBLOCKCHAININFO("getblockchaininfo", null, new Class<?>[]
{BlockchainInfo.class}),
    GETBLOCKCHAINPARAMS("getblockchainparams", null,
    new Class<?>[] {(new HashMap<String,
String>()).getClass()}),
    GETBLOCKCOUNT("getblockcount", null, new Class<?>[]
{Long.class}),
    GETBLOCKHASH("getblockhash", null, new Class<?>[]
{String.class}),
    GETBLOCKTEMPLATE("getblocktemplate", null, new Class<?>[]
{BlockTemplate.class}),
    GETCHaintIPS("getchaintips", null, new Class<?>[]
{ChainTip.class}, true),
    GETCHUNKQUEUEINFO("getchunkqueueinfo", null, new Class<?>[]
{ChunkqueueInfo.class}),
    GETCHUNKQUEUETOTALS("getchunkqueuetotals", null, new
Class<?>[] {ChunkqueueTotal.class}),
    GETCONNECTIONCOUNT("getconnectioncount", null, new Class<?
>[] {Long.class}),
    GETDIFFICULTY("getdifficulty", null, new Class<?>[]
{Double.class}),
    GETFILTERCODE("getfiltercode", null, new Class<?>[]
{String.class}),
    GETGENERATE("getgenerate", null, new Class<?>[]
{Boolean.class}),
    GETHASHESPERSEC("gethashespersec", null, new Class<?>[]
{Long.class}),
    GETINFO("getinfo", null, new Class<?>[] {(new
HashMap<String, Object>()).getClass()}),
    GETLASTBLOCKINFO("getlastblockinfo", null, new Class<?>[]
{LastBlockInfo.class}),
    GETMEMPOOLINFO("getmempoolinfo", null, new Class<?>[]
{MemPoolInfo.class}),
    GETMININGINFO("getmininginfo", null, new Class<?>[]
{MiningInfo.class}),
    GETMULTIBALANCES("getmultibalances", null, new Class<?>[]
{MultiBalance.class}),
    GETNETTOTALS("getnettotals", null, new Class<?>[]
{NetTotals.class}),
    GETNETWORKHASHPS("getnetworkhashps", null, new Class<?>[]
{Long.class}),

```

```

    GETNETWORKINFO("getnetworkinfo", null, new Class<?>[]
{NetworkInfo.class}),
    GETNEWADDRESS("getnewaddress", null, new Class<?>[]
{String.class}),
    GETPEERINFO("getpeerinfo", null, new Class<?>[]
{PeerInfo.class}),
    GETRAWCHANGEADDRESS("getrawchangeaddress", null, new
Class<?>[] {String.class}),
    GETRAWMEMPOOL("getrawmempool", null,
        new Class<?>[] {String.class, (new HashMap<String,
RawMemPoolTransaction>()).getClass()},
        true),
    GETRAWTRANSACTION("getrawtransaction", null, new Class<?>[]
{String.class, TransactionRAW.class}),
    GETRECEIVEDBYACCOUNT("getreceivedbyaccount", null, new
Class<?>[] {Double.class}),
    GETRECEIVEDBYADDRESS("getreceivedbyaddress", null, new
Class<?>[] {Double.class}),
    GETRUNTIMEPARAMS("getruntimeparams", null, new Class<?>[]
{RuntimeParams.class}),
    GETSTREAMINFO("getstreaminfo", null, new Class<?>[]
{Stream.class}),
    GETSTREAMITEM("getstreamitem", null, new Class<?>[]
{StreamKeyItem.class}),
    GETSTREAMKEYSUMMARY("getstreamkeysummary"),
    GETSTREAMPUBLISHERSUMMARY("getstreampublishersummary"),
    GETTOTALBALANCES("gettotalbalances", null, new Class<?>[]
{BalanceAssetGeneral.class}, true),
    GETTRANSACTION("gettransaction", null, new Class<?>[]
{Transaction.class}),
    GETTXOUT("gettxout", null, new Class<?>[] {TxOut.class}),
    GETTXOUTDATA("gettxoutdata", null, new Class<?>[]
{String.class}),
    GETTXOUTSETINFO("gettxoutsetinfo", null, new Class<?>[]
{TxOutInfo.class}),
    GETUNCONFIRMEDBALANCE("getunconfirmedbalance", null, new
Class<?>[] {Double.class}),
    GETWALLETINFO("getwalletinfo", null, new Class<?>[]
{WalletInfo.class}),
    GETWALLETTRANSACTION("getwallettransaction", null, new
Class<?>[] {TransactionWallet.class}),
    GRANT("grant", null, new Class<?>[] {String.class}),
    GRANTFROM("grantfrom", null, new Class<?>[]
{String.class}),
    GRANTWITHDATA("grantwithdata", null, new Class<?>[]
{String.class}),
    GRANTWITHDATAFROM("grantwithdatafrom", null, new Class<?>[]
{String.class}),
    HELP("help", null, new Class<?>[] {String.class}, false),
    IMPORTADDRESS("importaddress", null, null),
    IMPORTPRIVKEY("importprivkey", null, null),
    IMPORTWALLET("importwallet", null, null),

```

```

    ISSUE("issue", null, new Class<?>[] {String.class}),
    ISSUEFROM("issuefrom", null, new Class<?>[]
{String.class}),
    ISSUEMORE("issuemore", null, new Class<?>[]
{String.class}),
    ISSUEMOREFROM("issuemorefrom", null, new Class<?>[]
{String.class}),
    KEYPOOLREFILL("keypoolrefill", null, null),
    LISTACCOUNTS("listaccounts", null, new Class<?>[]
{Double.class}, true),
    LISTADDRESSES("listaddresses", null, new Class<?>[]
{Address.class}, true),
    LISTADDRESSGROUPINGS("listaddressgroupings"),
    LISTADDRESSTRANSACTIONS("listadresstransactions", null,
new Class<?>[] {TransactionWallet.class},
    true),
    LISTASSETS("listassets", null, new Class<?>[]
{BalanceAsset.class}, true),
    LISTASSETTRANSACTIONS("listassettransactions", null, new
Class<?>[] {AssetTransaction.class},
    true),
    LISTBLOCKS("listblocks", null, new Class<?>[]
{Block.class}, true),
    LISTLOCKUNSPENT("listlockunspent", null, new Class<?>[]
{TxIdVout.class}, true),
    LISTPERMISSIONS("listpermissions", null, new Class<?>[]
{Permission.class}, true),
    LISTRECEIVEDBYACCOUNT("listreceivedbyaccount", null, new
Class<?>[] {AddressReceived.class},
    true),
    LISTRECEIVEDBYADDRESS("listreceivedbyaddress", null, new
Class<?>[] {AddressReceived.class},
    true),
    LISTSINCEBLOCK("listsinceblock", null, new Class<?>[]
{SinceBlock.class}),
    LISTSTREAMBLOCKITEMS("liststreamblockitems", null, new
Class<?>[] {StreamKeyItem.class}, true),
    LISTSTREAMFILTERS("liststreamfilters"),
    LISTSTREAMITEMS("liststreamitems", null, new Class<?>[]
{StreamKeyItem.class}, true),
    LISTSTREAMKEYITEMS("liststreamkeyitems", null, new Class<?
>[] {StreamKeyItem.class}, true),
    LISTSTREAMKEYS("liststreamkeys", null, new Class<?>[]
{StreamKey.class}, true),
    LISTSTREAMPUBLISHERITEMS("liststreampublisheritems", null,
new Class<?>[] {StreamKeyItem.class},
    true),
    LISTSTREAMPUBLISHERS("liststreampublishers", null, new
Class<?>[] {Publisher.class}, true),
    LISTSTREAMQUERYITEMS("liststreamqueryitems"),
    LISTSTREAMS("liststreams", null, new Class<?>[]
{Stream.class}, true),

```

```

    LISTSTREAMTXITEMS("liststreamtxitems", null, new Class<?>[]
{StreamKeyItem.class}, true),
    LISTTRANSACTIONS("listtransactions", null, new Class<?>[]
{TransactionCurrency.class}, true),
    LISTTXFILTERS("listtxfilters"),
    LISTUNSPENT("listunspent", null, new Class<?>[]
{Unspent.class}, true),
    LISTUPGRADES("listupgrades", null, null, true),
    LISTWALLETTRANSACTIONS("listwallettransactions", null, new
Class<?>[] {TransactionWallet.class},
    true),
    LOCKUNSPENT("lockunspent", null, new Class<?>[]
{Boolean.class}),
    MOVE("move", null, new Class<?>[] {Boolean.class}),
    PAUSE("pause", null, null),
    PING("ping", null, null),
    PREPARELOCKUNSPENT("preparelockunspent", null, new Class<?
>[] {TxIdVout.class}),
    PREPARELOCKUNSPENTFROM("preparelockunspentfrom", null, new
Class<?>[] {TxIdVout.class}),
    PRIORITISETRANSACTION("prioritisetransaction", null, new
Class<?>[] {Boolean.class}),
    PUBLISH("publish", null, new Class<?>[] {String.class}),
    PUBLISHFROM("publishfrom", null, new Class<?>[]
{String.class}),
    PUBLISHMULTI("publishmulti", null, new Class<?>[]
{String.class}),
    PUBLISHMULTIFROM("publishmultifrom", null, new Class<?>[]
{String.class}),
    RESENDWALLETTRANSACTIONS("resendwallettransactions"),
    RESUME("resume", null, null),
    REVOKE("revoke", null, new Class<?>[] {String.class}),
    REVOKEFROM("revokefrom", null, new Class<?>[]
{String.class}),
    RUNSTREAMFILTER("runstreamfilter"),
    RUNTXFILTER("runtxfilter"),
    SEND("send", null, new Class<?>[] {String.class}),
    SENDASSET("sendasset", null, new Class<?>[]
{String.class}),
    SENDASSETFROM("sendassetfrom", null, new Class<?>[]
{String.class}),
    SENDFROM("sendfrom", null, new Class<?>[] {String.class}),
    SENDFROMACCOUNT("sendfromaccount", null, new Class<?>[]
{String.class}),
    SENDMANY("sendmany", null, new Class<?>[] {String.class}),
    SENDRAWTRANSACTION("sendrawtransaction", null, new Class<?
>[] {String.class}),
    SENDWITHDATA("sendwithdata", null, new Class<?>[]
{String.class}),
    SENDWITHDATAFROM("sendwithdatafrom", null, new Class<?>[]
{String.class}),
    SETACCOUNT("setaccount"),

```

```

        SETGENERATE("setgenerate"),
        SETLASTBLOCK("setlastblock", null, new Class<?>[]
{String.class}),
        SETRUNTIMEPARAM("setruntimeparam"),
        SETTXFEE("settxfee", null, new Class<?>[] {Boolean.class}),
        SIGNMESSAGE("signmessage", null, new Class<?>[]
{String.class}),
        SIGNRAWTRANSACTION("signrawtransaction", null, new Class<?
>[] {SignedTransactionRAW.class}),
        STOP("stop"),
        SUBMITBLOCK("submitblock"),
        SUBSCRIBE("subscribe"),
        TESTSTREAMFILTER("teststreamfilter"),
        TESTTXFILTER("testtxfilter"),
        UNSUBSCRIBE("unsubscribe"),
        VALIDATEADDRESS("validateaddress", null, new Class<?>[]
{Address.class}),
        VERIFYCHAIN("verifychain", null, new Class<?>[]
{Boolean.class}),
        VERIFYMESSAGE("verifymessage", null, new Class<?>[]
{Boolean.class}),
        VERIFYPERMISSION("verifypermission", null, new Class<?>[]
{Boolean.class}),
        WALLETLOCK("walletlock"),
        WALLETPASSPHRASE("walletpassphrase"),
        WALLETPASSPHRASECHANGE("walletpassphrasechange");

String commandline;
Object commandParameters; // Will be defined in 6.02
Class<?>[] returnedClass;
boolean resultByList = false;

private CommandElt(String commandline) {
    this.commandline = commandline;
}

private CommandElt(String commandline, Object
commandParameters) {
    this.commandline = commandline;
    this.commandParameters = commandParameters;
}

private CommandElt(String commandline, Object
commandParameters, Class<?>[] returnedClass) {
    this.commandline = commandline;
    this.commandParameters = commandParameters;
    this.returnedClass = returnedClass;
}

private CommandElt(String commandline, Object
commandParameters, Class<?>[] returnedClass,
    boolean resultByList) {

```

```

        this.commandline = commandline;
        this.commandParameters = commandParameters;
        this.returnedClass = returnedClass;
        this.resultByList = resultByList;
    }

    /**
     * @return the commandline
     */
    public String getCommandline() {
        return commandline;
    }

    /**
     * @return the returnClass
     */
    public Class<?>[] getReturnedClass() {
        return returnedClass;
    }

    /**
     * @return the commandArguments
     */
    public Object getCommandParameters() {
        return commandParameters;
    }

    /**
     * @return the resultByList
     */
    public boolean isResultByList() {
        return resultByList;
    }
}

```

Приложение В. Листинг SQL-файла

```
CREATE EXTENSION pgcrypto;

/* клиент (рабочий стол) */
CREATE TABLE client (
    id UUID,
    mail VARCHAR (254),
    CONSTRAINT pk_client PRIMARY KEY (id)
);

CREATE TABLE oid_title (
    oid VARCHAR(255),
    title VARCHAR(512),

    CONSTRAINT pk_oid_title PRIMARY KEY (oid)
);

CREATE TABLE notification (
    id UUID,
    client_id UUID NOT NULL,
    is_seen BOOLEAN NOT NULL,
    type INTEGER,
    object_id UUID,
    process_id UUID,
    process_name VARCHAR(255),
    created_at TIMESTAMP NOT NULL,
    updated_at TIMESTAMP,

    CONSTRAINT pk_notification PRIMARY KEY (id),

    CONSTRAINT fk_client_notification FOREIGN KEY (client_id)
        REFERENCES client(id)
        ON UPDATE NO ACTION ON DELETE NO ACTION
);

/* спецификация */
CREATE TABLE specification (
    id UUID,
    name VARCHAR(255) NOT NULL,
    client_id UUID NOT NULL,
    ttl BIGINT,
    created_at TIMESTAMP NOT NULL,
    sign_fingerprint VARCHAR(64),
    sign_algorithm INTEGER NOT NULL,
    sign BYTEA,
    tx_id VARCHAR(64),
    in_chain BOOLEAN NOT NULL,
    chain_status INTEGER NOT NULL DEFAULT 0,
    reject_info VARCHAR(512),
```

```

CONSTRAINT pk_specification PRIMARY KEY (id),

CONSTRAINT fk_specification_client FOREIGN KEY (client_id)
REFERENCES client (id)
ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE specification IS 'Спецификация';
COMMENT ON COLUMN specification.id IS 'Идентификатор';
COMMENT ON COLUMN specification.name IS 'Наименование
спецификации';
COMMENT ON COLUMN specification.client_id IS 'Идентификатор
клиента';
COMMENT ON COLUMN specification.ttl IS 'Время жизни
процессов';
COMMENT ON COLUMN specification.sign_fingerprint IS
'Отпечаток сертификата';
COMMENT ON COLUMN specification.sign_algorithm IS 'Алгоритм
подписи';
COMMENT ON COLUMN specification.sign IS 'Подпись';

/* Участник спецификации */
CREATE TABLE actor (
    id UUID,
    specification_id UUID NOT NULL,
    title VARCHAR(255),
    rights INTEGER NOT NULL,
    number INTEGER NOT NULL,

    CONSTRAINT pk_actor PRIMARY KEY (id),

    CONSTRAINT fk_actor_specification FOREIGN KEY
(specification_id)
REFERENCES specification (id)
ON UPDATE NO ACTION ON DELETE NO ACTION
);

/* Процесс */
CREATE TABLE process(
    id UUID,
    specification_id UUID NOT NULL,
    name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP NOT NULL,
    sign_fingerprint VARCHAR(64),
    sign_algorithm INTEGER NOT NULL,
    sign BYTEA,
    tx_id VARCHAR(64),
    in_chain BOOLEAN NOT NULL,
    chain_status INTEGER NOT NULL DEFAULT 0,
    reject_info VARCHAR(512),

    CONSTRAINT pk_process PRIMARY KEY (id),

```

```

    CONSTRAINT fk_process_specification FOREIGN KEY
(specification_id)
    REFERENCES specification (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE process IS 'Процесс';
COMMENT ON COLUMN process.specification_id IS 'Идентификатор
спецификации';
COMMENT ON COLUMN process.id IS 'Идентификатор процесса';
COMMENT ON COLUMN process.name IS 'Наименование процесса';
COMMENT ON COLUMN process.created_at IS 'Время создания';
COMMENT ON COLUMN process.sign_fingerprint IS 'Отпечаток
сертификата';
COMMENT ON COLUMN process.sign_algorithm IS 'Алгоритм
подписи';
COMMENT ON COLUMN process.sign IS 'Подпись создателя';

CREATE TABLE v_process_state (
    id UUID,
    state BYTEA,
    status INTEGER NOT NULL,

    CONSTRAINT pk_v_process_state_id PRIMARY KEY (id)
);
COMMENT ON TABLE v_process_state IS 'Состояние процесса';
COMMENT ON COLUMN v_process_state.id IS 'Идентификатор
процесса';
COMMENT ON COLUMN v_process_state.state IS 'Состояние (kryo
сериализация)';
COMMENT ON COLUMN v_process_state.status IS 'Статус
процесса';

CREATE TABLE v_chain_block (
    id UUID,
    data BYTEA,

    CONSTRAINT pk_v_chain_block_id PRIMARY KEY (id)
);
COMMENT ON TABLE v_chain_block IS 'Блок чейна';
COMMENT ON COLUMN v_chain_block.id IS 'Идентификатор блока';
COMMENT ON COLUMN v_chain_block.data IS 'Данные (kryo
сериализация)';

/* Публикация операции публикации документа */
CREATE TABLE publish_document(
    id UUID,
    process_id UUID NOT NULL,
    operation_id UUID NOT NULL,
    created_at TIMESTAMP NOT NULL,
    type VARCHAR(128),
    hash BYTEA,
    hash_algorithm INTEGER NOT NULL,

```

```

sign_fingerprint VARCHAR(64),
sign_algorithm INTEGER NOT NULL,
sign BYTEA,
tx_id VARCHAR(64),
in_chain BOOLEAN NOT NULL,
chain_status INTEGER NOT NULL DEFAULT 0,
reject_info VARCHAR(512),

CONSTRAINT pk_publish_document PRIMARY KEY (id),

CONSTRAINT fk_publish_document_process FOREIGN KEY
(process_id)
REFERENCES process (id)
ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE publish_document IS 'Публикация операции
публикации документа';
COMMENT ON COLUMN publish_document.id IS 'Идентификатор
публикации';
COMMENT ON COLUMN publish_document.process_id IS
'Идентификатор процесса';
COMMENT ON COLUMN publish_document.type IS 'Имя документа';
COMMENT ON COLUMN publish_document.hash IS 'Хэш документа';
COMMENT ON COLUMN publish_document.hash_algorithm IS
'Алгоритм хэширования';
COMMENT ON COLUMN publish_document.sign_fingerprint IS
'Отпечаток сертификата';
COMMENT ON COLUMN publish_document.sign_algorithm IS
'Алгоритм подписи';
COMMENT ON COLUMN publish_document.sign IS 'Подпись
создателя';

/* Публикация операции подписи */
CREATE TABLE publish_sign(
id UUID,
process_id UUID NOT NULL,
operation_id UUID NOT NULL,
created_at TIMESTAMP NOT NULL,
hash BYTEA,
hash_algorithm INTEGER NOT NULL,
sign_fingerprint VARCHAR(64),
sign_algorithm INTEGER NOT NULL,
sign BYTEA,
tx_id VARCHAR(64),
in_chain BOOLEAN NOT NULL,
chain_status INTEGER NOT NULL DEFAULT 0,
reject_info VARCHAR(512),

CONSTRAINT pk_publish_sign PRIMARY KEY (id),

CONSTRAINT fk_publish_sign_process FOREIGN KEY (process_id)
REFERENCES process (id)

```

```

        ON UPDATE NO ACTION ON DELETE NO ACTION
    );
    COMMENT ON TABLE publish_sign IS 'Публикация операции подписи';
    COMMENT ON COLUMN publish_sign.id IS 'Идентификатор публикации';
    COMMENT ON COLUMN publish_sign.process_id IS 'Идентификатор процесса';
    COMMENT ON COLUMN publish_sign.hash IS 'Хэш документа';
    COMMENT ON COLUMN publish_sign.hash_algorithm IS 'Алгоритм хэширования';
    COMMENT ON COLUMN publish_sign.sign_fingerprint IS 'Отпечаток сертификата';
    COMMENT ON COLUMN publish_sign.sign_algorithm IS 'Алгоритм подписи';
    COMMENT ON COLUMN publish_sign.sign IS 'Подпись создателя';

    /* Публикация операции завершения */
    CREATE TABLE publish_finish(
        id UUID,
        process_id UUID NOT NULL,
        operation_id UUID,
        created_at TIMESTAMP NOT NULL,
        result INTEGER NOT NULL,
        info VARCHAR(512),
        sign_fingerprint VARCHAR(64),
        sign_algorithm INTEGER NOT NULL,
        sign BYTEA,
        tx_id VARCHAR(64),
        in_chain BOOLEAN NOT NULL,
        chain_status INTEGER NOT NULL DEFAULT 0,
        reject_info VARCHAR(512),

        CONSTRAINT pk_publish_finish PRIMARY KEY (id),

        CONSTRAINT fk_publish_finish_process FOREIGN KEY
        (process_id)
        REFERENCES process (id)
        ON UPDATE NO ACTION ON DELETE NO ACTION
    );
    COMMENT ON TABLE publish_finish IS 'Публикация операции завершения';
    COMMENT ON COLUMN publish_finish.id IS 'Идентификатор публикации';
    COMMENT ON COLUMN publish_finish.process_id IS 'Идентификатор процесса';
    COMMENT ON COLUMN publish_finish.result IS 'Результат завершения процесса';
    COMMENT ON COLUMN publish_finish.info IS 'Информация о завершении процесса';
    COMMENT ON COLUMN publish_finish.sign_fingerprint IS 'Отпечаток сертификата';

```

```

COMMENT ON COLUMN publish_finish.sign_algorithm IS 'Алгоритм
подписи';
COMMENT ON COLUMN publish_finish.sign IS 'Подпись создателя';

CREATE TABLE actor_constraint(
    id UUID,
    actor_number INTEGER NOT NULL,
    process_id UUID NOT NULL,

    CONSTRAINT pk_actor_constraint PRIMARY KEY (id),

    CONSTRAINT fk_process_actor_constraint FOREIGN KEY
(process_id)
    REFERENCES process (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION
);

/* Шаблон селектора */
CREATE TABLE selector_template (
    id UUID,
    title VARCHAR(255),
    type INTEGER NOT NULL,
    relation_id UUID,

    CONSTRAINT pk_selector_template PRIMARY KEY (id)
);
COMMENT ON TABLE selector_template IS 'Шаблон селектора';

/* Таблица отношений шаблонов селекторов.
Шаблоны селекторов спецификаций <-> Шаблоны селекторов
процессов */
CREATE TABLE selector_template_relation (
    id UUID,
    selector_template_id UUID NOT NULL,
    selector_template_rel_id UUID NOT NULL,

    CONSTRAINT pk_selector_template_relation PRIMARY KEY (id),

    CONSTRAINT fk_selector_template_relation_template_id
FOREIGN KEY (selector_template_id)
    REFERENCES selector_template (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION,

    CONSTRAINT fk_selector_template_relation_template_rel_id
FOREIGN KEY (selector_template_rel_id)
    REFERENCES selector_template (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE selector_template_relation IS 'Таблица
отношений шаблонов селекторов';

/* Селектор */

```

```

CREATE TABLE selector (
    id UUID,
    actor_id UUID,
    actor_constraint_id UUID,
    selector_template_id UUID,
    type INTEGER NOT NULL,
    oid VARCHAR(255),
    attribute VARCHAR(255) NOT NULL,
    val VARCHAR(255),
    action CHAR(1),

    CONSTRAINT pk_selector PRIMARY KEY (id),

    CONSTRAINT fk_selector_actor FOREIGN KEY (actor_id)
    REFERENCES actor (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION,

    CONSTRAINT fk_selector_actor_constraint FOREIGN KEY
(actor_constraint_id)
    REFERENCES actor_constraint (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION,

    CONSTRAINT fk_selector_selector_template FOREIGN KEY
(selector_template_id)
    REFERENCES selector_template (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE certificate(
    fingerprint VARCHAR(128),
    body BYTEA,
    client_id UUID NOT NULL,
    name VARCHAR(255),
    dn VARCHAR(512) NOT NULL,
    not_after TIMESTAMP,
    tx_id VARCHAR(64),
    in_chain BOOLEAN NOT NULL,
    chain_status INTEGER NOT NULL DEFAULT 0,
    reject_info VARCHAR(512),

    CONSTRAINT pk_certificate PRIMARY KEY (fingerprint),

    CONSTRAINT fk_certificate_client FOREIGN KEY (client_id)
    REFERENCES client (id)
    ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE certificate IS 'Сертификат';
COMMENT ON COLUMN certificate.fingerprint IS 'sha256
отпечаток';
COMMENT ON COLUMN certificate.body IS 'Тело сертификата';
COMMENT ON COLUMN certificate.client_id IS 'Идентификатор
клиента';

```

```
COMMENT ON COLUMN certificate.name IS 'Имя';
COMMENT ON COLUMN certificate.not_after IS 'Не действителен
после';
```

```
CREATE TABLE certificate_attribute (
    val VARCHAR(255),
    oid VARCHAR(255) NOT NULL,
    fingerprint VARCHAR(128) NOT NULL,

    CONSTRAINT pk_certificate_attribute PRIMARY KEY
(fingerprint, oid),

    CONSTRAINT fk_certificate_certificate_attribute FOREIGN KEY
(fingerprint)
REFERENCES certificate(fingerprint)
ON UPDATE NO ACTION ON DELETE NO ACTION,

    CONSTRAINT fk_oid_title_certificate_attribute FOREIGN KEY
(oid)
REFERENCES oid_title(oid)
ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

```
/* Последовательность операций */
```

```
CREATE TABLE operation_queue(
    id UUID,
    type INTEGER NOT NULL,
    order_in INTEGER NOT NULL,
    specification_id UUID NOT NULL,

    CONSTRAINT pk_queue PRIMARY KEY (id),

    CONSTRAINT fk_operation_queue_specification FOREIGN KEY
(specification_id)
REFERENCES specification (id)
ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE operation_queue IS 'Последовательность
операций';
COMMENT ON COLUMN operation_queue.id IS 'Идентификатор';
COMMENT ON COLUMN operation_queue.type IS 'Тип SEQ, ALL,
XOR';
COMMENT ON COLUMN operation_queue.order_in IS 'Порядок
последовательности в спецификации';
```

```
/* Операция завершения */
```

```
CREATE TABLE operation_finish (
    id UUID,
    queue_id UUID NOT NULL,
    actor_number INTEGER,
    actor_title VARCHAR(255),
```

```

result INTEGER,
info VARCHAR(255),
order_in INTEGER NOT NULL,

CONSTRAINT pk_operation_finish PRIMARY KEY (id),

CONSTRAINT fk_operation_finish_queue FOREIGN KEY (queue_id)
REFERENCES operation_queue (id)
ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE operation_finish IS 'Операция завершения';
COMMENT ON COLUMN operation_finish.id IS 'Идентификатор';
COMMENT ON COLUMN operation_finish.queue_id IS 'Идентификатор
последовательности';
COMMENT ON COLUMN operation_finish.actor_number IS 'Номер
участника';
COMMENT ON COLUMN operation_finish.result IS 'Результат';
COMMENT ON COLUMN operation_finish.info IS 'Информация';
COMMENT ON COLUMN operation_finish.order_in IS 'Порядок
операции в последовательности';

/* Операция подписания фиксированного документа */
CREATE TABLE operation_sign_fix (
id UUID,
queue_id UUID NOT NULL,
actor_number INTEGER,
actor_title VARCHAR(255),
type_of VARCHAR(128),
hash BYTEA,
hash_algorithm INTEGER NOT NULL,
sign_fingerprint VARCHAR(64),
sign_algorithm INTEGER NOT NULL,
sign BYTEA,
order_in INTEGER NOT NULL,

CONSTRAINT pk_operation_sign_fix PRIMARY KEY (id),

CONSTRAINT fk_operation_sign_fix_queue FOREIGN KEY
(queue_id)
REFERENCES operation_queue (id)
ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE operation_sign_fix IS 'Операция подписания
фиксированного документа';
COMMENT ON COLUMN operation_sign_fix.id IS 'Идентификатор';
COMMENT ON COLUMN operation_sign_fix.queue_id IS
'Идентификатор последовательности';
COMMENT ON COLUMN operation_sign_fix.actor_number IS 'Номер
участника';
COMMENT ON COLUMN operation_sign_fix.type_of IS 'Уникальная
строка в рамках спецификации (подписываем contract№1)';
COMMENT ON COLUMN operation_sign_fix.hash IS 'Хэш';

```

```

COMMENT ON COLUMN operation_sign_fix.hash_algorithm IS
'Алгоритм хэширования';
COMMENT ON COLUMN operation_sign_fix.sign_fingerprint IS
'Отпечаток сертификата';
COMMENT ON COLUMN operation_sign_fix.sign_algorithm IS
'Алгоритм подписи';
COMMENT ON COLUMN operation_sign_fix.sign IS 'Подпись';
COMMENT ON COLUMN operation_finish.order_in IS 'Порядок
операции в последовательности';

/* Операция подписания документа */
CREATE TABLE operation_sign (
    id UUID,
    queue_id UUID NOT NULL,
    actor_number INTEGER,
    actor_title VARCHAR(255),
    type_of VARCHAR(128),
    order_in INTEGER NOT NULL,

    CONSTRAINT pk_operation_sign PRIMARY KEY (id),

    CONSTRAINT fk_operation_sign_queue FOREIGN KEY (queue_id)
        REFERENCES operation_queue (id)
        ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE operation_sign IS 'Операция подписания
документа';
COMMENT ON COLUMN operation_sign.id IS 'Идентификатор';
COMMENT ON COLUMN operation_sign.queue_id IS 'Идентификатор
последовательности';
COMMENT ON COLUMN operation_sign.actor_number IS 'Номер
участника';
COMMENT ON COLUMN operation_sign.type_of IS 'Уникальная
строка в рамках спецификации (подписываем contract№1)';
COMMENT ON COLUMN operation_sign.order_in IS 'Порядок
операции в последовательности';

/* Операция публикации документа */
CREATE TABLE operation_publish (
    id UUID,
    queue_id UUID NOT NULL,
    actor_number INTEGER,
    actor_title VARCHAR(255),
    type_of VARCHAR(128),
    description VARCHAR(512),
    order_in INTEGER NOT NULL,

    CONSTRAINT pk_operation_publish PRIMARY KEY (id),

    CONSTRAINT fk_operation_publish_queue FOREIGN KEY
(queue_id)
        REFERENCES operation_queue (id)

```

```

        ON UPDATE NO ACTION ON DELETE CASCADE
    );
    COMMENT ON TABLE operation_publish IS 'Операция завершения';
    COMMENT ON COLUMN operation_publish.id IS 'Идентификатор';
    COMMENT ON COLUMN operation_publish.queue_id IS
        'Идентификатор последовательности';
    COMMENT ON COLUMN operation_publish.actor_number IS 'Номер
    участника';
    COMMENT ON COLUMN operation_publish.type_of IS 'Уникальная
    строка в рамках спецификации (подписываем contract№1)';
    COMMENT ON COLUMN operation_publish.description IS
        'Описание';
    COMMENT ON COLUMN operation_publish.order_in IS 'Порядок
    операции в последовательности';

    /* Операция публикации фиксированного документа */
    CREATE TABLE operation_publish_fix (
        id UUID,
        queue_id UUID NOT NULL,
        type_of VARCHAR(128),
        description VARCHAR(512),
        hash BYTEA,
        hash_algorithm INTEGER NOT NULL,
        sign_fingerprint VARCHAR(64),
        sign_algorithm INTEGER NOT NULL,
        sign BYTEA,
        order_in INTEGER NOT NULL,

        CONSTRAINT pk_operation_publish_fix PRIMARY KEY (id),

        CONSTRAINT fk_operation_publish_fix_queue FOREIGN KEY
        (queue_id)
            REFERENCES operation_queue (id)
            ON UPDATE NO ACTION ON DELETE CASCADE
    );
    COMMENT ON TABLE operation_publish_fix IS 'Операция
    завершения';
    COMMENT ON COLUMN operation_publish_fix.id IS
        'Идентификатор';
    COMMENT ON COLUMN operation_publish_fix.queue_id IS
        'Идентификатор последовательности';
    COMMENT ON COLUMN operation_publish_fix.type_of IS
        'Уникальная строка в рамках спецификации (подписываем
        contract№1)';
    COMMENT ON COLUMN operation_publish_fix.description IS
        'Описание';
    COMMENT ON COLUMN operation_publish_fix.hash IS 'Хэш';
    COMMENT ON COLUMN operation_publish_fix.hash_algorithm IS
        'Алгоритм хэширования';
    COMMENT ON COLUMN operation_publish_fix.sign_fingerprint IS
        'Отпечаток сертификата';
    COMMENT ON COLUMN operation_publish_fix.sign_algorithm IS

```

```

'Алгоритм подписи';
COMMENT ON COLUMN operation_publish_fix.sign IS 'Подпись';
COMMENT ON COLUMN operation_publish_fix.order_in IS 'Порядок
операции в последовательности';

create table chain_key_value (
    key VARCHAR(64),
    value VARCHAR(64),

    CONSTRAINT pk_key_value_key PRIMARY KEY (key)
);
COMMENT ON TABLE chain_key_value IS 'Ключ-значение';
COMMENT ON COLUMN chain_key_value.key IS 'Ключ';
COMMENT ON COLUMN chain_key_value.value IS 'Значение';

/*Тестовые данные*/
INSERT INTO public.oid_title(oid, title) VALUES
('1.2.643.100.1', 'ОГРН');
INSERT INTO public.oid_title(oid, title) VALUES
('1.2.643.3.131.1.1', 'ИНН');
INSERT INTO public.oid_title(oid, title) VALUES
('1.2.643.100.3', 'СНИЛС');
INSERT INTO public.oid_title(oid, title) VALUES
('1.2.643.6.3.1.2.1', 'ЮЛ');
INSERT INTO public.oid_title(oid, title) VALUES
('1.2.643.6.3.1.2.2', 'ФЛ');
/* шаблоны */
INSERT INTO public.selector_template(
    id, title, type)
VALUES ('fcd17ad0-e17f-4bb1-9f7e-304ddbde8675',
'Юридическое лицо', 0);
INSERT INTO public.selector_template(
    id, title, type)
VALUES ('d55dc431-7619-47c3-a599-5d8b8ec67674', 'Физическое
лицо', 0);
INSERT INTO public.selector_template(
    id, title, type)
VALUES ('a88b41b0-23a1-484d-803d-dbd60d2337d2', 'Любое
юридическое лицо - резидент РФ', 1);
INSERT INTO public.selector_template(
    id, title, type)
VALUES ('49d123ce-58c4-4167-b6f8-d804cdf9a8f4',
'Определенное юридическое лицо - резидент РФ', 1);
INSERT INTO public.selector_template(
    id, title, type)
VALUES ('76f24ee1-7ee2-43ad-8d97-f2e43f26e2f5',
'Определенное физическое лицо - резидент РФ', 1);
--
INSERT INTO public.selector(
    id, attribute, action, val, type, selector_template_id,
oid)
VALUES ('887dc5cd-d38d-4075-af1d-229e112f21b1',

```

```

'1.2.643.6.3.1.2.1', null, null, 2, 'fcd17ad0-e17f-4bb1-9f7e-
304ddbde8675', '1.2.643.6.3.1.2.1');
INSERT INTO public.selector(
    id, attribute, action, val, type, selector_template_id,
oid)
    VALUES ('a97ed2df-b463-473a-9523-c0f5aa98fd7b',
'1.2.643.6.3.1.2.2', null, null, 2, 'd55dc431-7619-47c3-a599-
5d8b8ec67674', '1.2.643.6.3.1.2.2');
INSERT INTO public.selector(
    id, attribute, action, val, type, selector_template_id,
oid)
    VALUES ('ca9d5519-53ef-4ba0-9b08-4c5966a44e61',
'1.2.643.3.131.1.1', null, null, 2, 'a88b41b0-23a1-484d-803d-
dbd60d2337d2', '1.2.643.3.131.1.1');
INSERT INTO public.selector(
    id, attribute, action, val, type, selector_template_id,
oid)
    VALUES ('c1eed8ca-b197-4cfe-8196-b2b1f427aa35',
'1.2.643.3.131.1.1', '=', null, 2, '49d123ce-58c4-4167-b6f8-
d804cdf9a8f4', '1.2.643.3.131.1.1');
INSERT INTO public.selector(
    id, attribute, action, val, type, selector_template_id,
oid)
    VALUES ('52c44d1b-f277-462b-8651-cd038be06635',
'1.2.643.100.3', '=', null, 2, '76f24ee1-7ee2-43ad-8d97-
f2e43f26e2f5', '1.2.643.100.3');
/* создание отношений шаблонов селекторов */
/* Юридическое лицо -> Любое юридическое лицо - резидент РФ,
Определенное юридическое лицо - резидент РФ*/
INSERT INTO public.selector_template_relation(
    id, selector_template_id, selector_template_rel_id)
    VALUES ('02a10f27-4068-417a-bc79-6c038a8b8ea0', '49d123ce-
58c4-4167-b6f8-d804cdf9a8f4', 'fcd17ad0-e17f-4bb1-9f7e-
304ddbde8675');
INSERT INTO public.selector_template_relation(
    id, selector_template_id, selector_template_rel_id)
    VALUES ('ddad3057-f2de-426d-849e-1add60c46c4d', 'a88b41b0-
23a1-484d-803d-dbd60d2337d2', 'fcd17ad0-e17f-4bb1-9f7e-
304ddbde8675');
/* Физическое лицо -> Определенное физическое лицо -
резидент РФ*/
INSERT INTO public.selector_template_relation(
    id, selector_template_id, selector_template_rel_id)
    VALUES ('de3076ba-9af4-4355-a9bf-f18149988ff2', '76f24ee1-
7ee2-43ad-8d97-f2e43f26e2f5', 'd55dc431-7619-47c3-a599-
5d8b8ec67674');

```

Приложение С. Пример выполнения *REST API*

Get-запрос: <https://ucalls.ru:8721/cds/specification?id=57460583-7167-4b3a-9e14-3bc9df7115a9>

Ответ:

```
{
  "content": [
    {
      "id": "13ff8494-b6c7-485d-85d7-2895af274dee",
      "name": "Двусторонний договор между ЮЛ",
      "clientId": "1fab186f-71a8-4152-95d7-617220cf1ff4",
      "ttl": 0,
      "createdAt": 1588949665000,
      "signature": {
        "algorithm": "secp256r1_sha256",
        "signBytes":
"MEQCIBwh0VuT65A8VoXwHMMFwIz1zUZwk4yuO58rG18EXwS+AiAV2buV3Venfd6
btn1hdb8X3vI99HMB1+iukqM9CZ86dQ==",
        "fingerPrint":
"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0",
        "empty": false
      },
      "certificate": {
        "fingerPrint":
"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0",
        "body":
"MIIDUDCCAvYCCQDCCw0z4nqfYzAKBggqhkhjOPQQDAjCCAS4xIjAgBgNVBAMMGdC
e0J7QniDQmNC90LLQtdC90YlQsNGA0YwxIjAgBgNVBAoMGdCe0J7QniDQmNC90LL
QtdC90YlQsNGA0YwxHzAdBgNVBAkMFtGI0L7RgdGB0LUINC00L7QvCAxMzgxFtA
TBgNVBACMDNCY0LbQtdCy0YHQujE1MDMGA1UECAwsNDcg0JvQtdC90LjQvdCz0YD
QsNC00YHQutCw0Y8g0L7QsdC70LDRgdGC0YwxCzAJBgNVBAYTA1JVMRgwFgYIKoU
DA4EDAQEEMCjc3Mjk1MjE5MjAxGDAWBgUqhQNkAQwNMTEyNDcwMzAwMDMzMTEWMBQ"
```

GBSqFA2QDDAswNjUwOTk1NzI5MTEcMBoGCSqGSib3DQEJARYNdWRjMkB0ZXN0LmNvbTAeFw0xODEwMjIyMDI2NDFaFw0yMDEwMjEyMDI2NDFaMIIBLjEiMCAGA1UEAwwZ0J7QntCeINCY0L3QstC10L3RgtCw0YDRjDEiMCAGA1UECgwZ0J7QntCeINCY0L3QstC10L3RgtCw0YDRjDEfMBOGA1UECQwW0YjQvtGB0YHQtsWg0LTQvtC8IDEzODEVMBMGA1UEBwwM0JjQttC10LLRgdC6MTUwMwYDVQQIDCw0NyDQm9C10L3QuNC90LP RgNCw0LTRgdC60LDRjyDQvtCx0LvQsNGB0YLRjDELMakGA1UEBhMCU1UxGDAWBggqhQMDgQMBAQwKNzcyOTUyMTIyMDEYMBYGBSqFA2QBDA0xMTI0NzAzMDAwMzMxMRYwFAYFKoUDZAMMCzA2NTA5OTU3MjkxMRwwGgYJKoZIhvcNAQkBFg11ZGM5QHRlc3QuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEN/DJGjgLe/bTza5aNe+/rwW sSdk3IC81zLhJdya8ix9HsXD+dofqjAjdS8kGnMx7pyYaw8A2pCeZ3Vz3nmW/mjA KBggqghkjOPQQDAgNIADBFAiB6Oz8+xF7LXGmloi4relpE86zfpSVtKpPyDDXItCZ CdAIhAK3VhCtKMdlKxe7MX7elvm360YoEpBz/uaIQ4CCsY1Ln",

"clientId": "1fab186f-71a8-4152-95d7-617220cf1ff4",

"name": "ООО Инвентарь",

"dn": "CN=ООО Инвентарь,O=ООО

Инвентарь,STREET=шоссе\\, дом 138,L=Ижевск,ST=47 Ленинградская область,C=RU,1.2.643.3.131.1.1=7729521220,1.2.643.100.1=1124703000331,1.2.643.100.3=06509957291,E=udc2@test.com",

"notAfter": 1603312001000,

"chainResult": {

"txId":

"ae47540cfd22b5e44ea8354cb05c4f0dced92a1cb6dcc7a2dd54adf754306260",

"inChain": true,

"rejectInfo": null,

"chainStatus": "SUCCESS"

},

"certificateAttributes": [

{

"value": "1124703000331",

"oidTitle": {

"oid": "1.2.643.100.1",

"title": "ОГРН"

},

"oid": "1.2.643.100.1",

```
        "certificateFingerPrint":  
        "2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0  
0"
```

```
    },  
    {  
        "value": "06509957291",  
        "oidTitle": {  
            "oid": "1.2.643.100.3",  
            "title": "СНИЛС"  
        },  
        "oid": "1.2.643.100.3",
```

```
        "certificateFingerPrint":  
        "2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0  
0"
```

```
    },  
    {  
        "value": "7729521220",  
        "oidTitle": {  
            "oid": "1.2.643.3.131.1.1",  
            "title": "ИНН"  
        },  
        "oid": "1.2.643.3.131.1.1",
```

```
        "certificateFingerPrint":  
        "2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0  
0"
```

```
    }  
]  
},  
"queues": [  
    {  
        "id": "b4b28d5f-5634-41a0-b4b2-221e41695a33",  
        "type": "SEQ",  
        "operations": [  
            {  
                "id": "034e38bd-c964-4eea-9991-ac4395ab65c3",
```

```
    "order": 1,
    "actorNumber": 0,
    "actorTitle": "ЮЛ 1",
    "typeOf": "Договор",
    "empty": false,
    "queue": "b4b28d5f-5634-41a0-b4b2-
221e41695a33",
    "type": "sign"
  },
  {
    "id": "306228cb-dafa-4c9d-a3a8-4b94460fb713",
    "order": 3,
    "actorNumber": 0,
    "actorTitle": "ЮЛ 1",
    "result": 0,
    "info": "Успешное подписание договора",
    "empty": false,
    "queue": "b4b28d5f-5634-41a0-b4b2-
221e41695a33",
    "type": "finish"
  },
  {
    "id": "594213da-701b-4630-b0be-a62946ffa7cc",
    "order": 0,
    "actorNumber": 0,
    "actorTitle": "ЮЛ 1",
    "description": "Договор для подписания",
    "typeOf": "Договор",
    "empty": false,
    "queue": "b4b28d5f-5634-41a0-b4b2-
221e41695a33",
    "type": "publish"
  },
  {
    "id": "ed1f3ff4-0fd2-482d-9632-f2643d72071a",
```

```
        "order": 2,
        "actorNumber": 1,
        "actorTitle": "ЮЛ 2",
        "typeOf": "Договор",
        "empty": false,
        "queue": "b4b28d5f-5634-41a0-b4b2-
221e41695a33",
        "type": "sign"
    }
],
    "order": 0,
    "empty": false,
    "specification": "13ff8494-b6c7-485d-85d7-
2895af274dee"
}
],
"actors": [
    {
        "id": "13bcf05c-b65d-4967-8814-7e37a0ce0c1c",
        "rights": 1,
        "number": 0,
        "title": "ЮЛ 1",
        "selectors": [
            {
                "id": "151b474b-3ed5-41f4-ba47-3dcd4b5bbfeb",
                "attribute": "1.2.643.6.3.1.2.1",
                "value": null,
                "action": null,
                "type": "ACTOR",
                "oidTitle": {
                    "oid": "1.2.643.6.3.1.2.1",
                    "title": "ЮЛ"
                },
            },
            "oid": "1.2.643.6.3.1.2.1",
```

```
        "actorId": "13bcf05c-b65d-4967-8814-7e37a0ce0c1c",
        "needValue": false,
        "hasValue": false
    },
    ],
    "specificationId": "13ff8494-b6c7-485d-85d7-2895af274dee",
    "empty": false,
    "selectorLine": "1.2.643.6.3.1.2.1"
},
{
    "id": "4e39db67-5d06-40a4-b5f3-0eb41d54e744",
    "rights": 1,
    "number": 1,
    "title": "ЮЛ 2",
    "selectors": [
        {
            "id": "1a21a0b7-a32a-4f35-8b8e-51811148fde6",
            "attribute": "1.2.643.6.3.1.2.1",
            "value": null,
            "action": null,
            "type": "ACTOR",
            "oidTitle": {
                "oid": "1.2.643.6.3.1.2.1",
                "title": "ЮЛ"
            },
        },
        {
            "oid": "1.2.643.6.3.1.2.1",
            "actorId": "4e39db67-5d06-40a4-b5f3-0eb41d54e744",
            "needValue": false,
            "hasValue": false
        }
    ],
}
```

```
        "specificationId": "13ff8494-b6c7-485d-85d7-
2895af274dee",
        "empty": false,
        "selectorLine": "1.2.643.6.3.1.2.1"
    }
],
    "chainResult": {
        "txId":
        "b7cc004ad43efee68cd0d0fc1db3455127cb7c78145118e6fe3590ef4e60787
3",
        "inChain": true,
        "rejectInfo": null,
        "chainStatus": "SUCCESS"
    },
    "empty": false,
    "emptyForSign": false
},
{
    "id": "57460583-7167-4b3a-9e14-3bc9df7115a9",
    "name": "Спецификация_Тест",
    "clientId": "1fab186f-71a8-4152-95d7-617220cf1ff4",
    "ttl": 468000000,
    "createdAt": 1589651677000,
    "signature": {
        "algorithm": "secp256r1_sha256",
        "signBytes":
        "MEQCIE3/7nSCJwSjBImx21JRG5Al/X0e1PxbY4sAi5Q5pdiPAiBS4u7aJiX5rLT
joMjc3vlll4dKyJPJYxGYjUjoIGeDoQ==",
        "fingerPrint":
        "2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0",
        "empty": false
    },
    "certificate": {
```

"fingerPrint":

"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df00",

"body":

"MIIDUDCCAvYCCQDCCw0z4nqfYzAKBggghkjOPQQDAjCCAS4xIjAgBgNVBAMMGdCe0J7QniDQmNC90LLQtCdC90YLQsNGA0YwxIjAgBgNVBAoMGdCe0J7QniDQmNC90LLQtCdC90YLQsNGA0YwxHzAdBgNVBAkMFtGI0L7RgdGB0LU\$INC00L7QvCAxMzgxFtATBgNVBACMDNCY0LbQtCdCy0YHQujE1MDMGA1UECAwsNDcg0JvQtCdC90LjQvdCz0YDQsNC00YHQutCw0Y8g0L7QsdC70LDRgdGC0YwxCzAJBgNVBAYTA1JVMRgwFgYIKoU\$DA4EDAQEMCjc3Mjk1MjEyMjAxGDAWBgUqhQNkAQwNMTEyNDcwMzAwMDMzMTEWMBQGBS\$FA2QDDAswNjUwOTk1NzI5MTEcMBoGCSqGSib3DQEJARYNdWRjMkBOZXN0LmNvbTAeFw0xODEwMjIyMDI2NDFaFw0yMDEwMjEyMDI2NDFaMIIBLjEiMCAGA1UEAwwZ0J7QntCeINCY0L3QstC10L3RgtCw0YDRjDEiMCAGA1UECgwZ0J7QntCeINCY0L3QstC10L3RgtCw0YDRjDEfMB0GA1UECQwW0YjQvtGB0YHQ\$Swg0LTQvtC8IDEzODEVMBMGA1UEBwwM0JjQtC10LLRgdC6MTUwMwYDVQQIDCw0NyDQm9C10L3QuNC90LPRgNCw0LTRgdC60LDRjyDQvtCx0LvQsNGB0YLRjDELMakGA1UEBhMCU1UxGDAWBggqhQMDgQMBAQwKNzcyOTUyMTIyMDEYMBYGBS\$FA2QBDA0xMTI0NzAzMDAwMzMxMRYwFAYFKoUDZAMMCzA2NTA5OTU3Mj\$KxMRwwGgYJKoZIhvcNAQkBFg11ZGM\$YQHRlc3QuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE\$N/DJGjgLe/bTza5aNe+/rwW\$S\$dk3IC81zLhJdya8ix9HsXD+dofqjAjdS8kGnMx7pyYaw8A2pCeZ3Vz3nmW/mjAKBggghkjOPQQDAgNIADBFAiB6Oz8+xF7LXGmloi4relpE86zfpSVtKpPyDDXItCZCdAIhAK3VhCtKMdlKxe7MX7elvm360YoEpBz/uaIQ4CCsY1Ln",

"clientId": "1fab186f-71a8-4152-95d7-617220cf1ff4",

"name": "ООО Инвентарь",

"dn": "CN=ООО Инвентарь,O=ООО

Инвентарь,STREET=шоссе\\, дом 138,L=Ижевск,ST=47 Ленинградская область,C=RU,1.2.643.3.131.1.1=7729521220,1.2.643.100.1=1124703000331,1.2.643.100.3=06509957291,E=udc2@test.com",

"notAfter": 1603312001000,

"chainResult": {

"txId":

"ae47540cfd22b5e44ea8354cb05c4f0dced92a1cb6dcc7a2dd54adf754306260",

"inChain": true,

"rejectInfo": null,

```
        "chainStatus": "SUCCESS"
    },
    "certificateAttributes": [
        {
            "value": "1124703000331",
            "oidTitle": {
                "oid": "1.2.643.100.1",
                "title": "ОГРН"
            },
            "oid": "1.2.643.100.1",
            "certificateFingerprint":
"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0"
        },
        {
            "value": "06509957291",
            "oidTitle": {
                "oid": "1.2.643.100.3",
                "title": "СНИЛС"
            },
            "oid": "1.2.643.100.3",
            "certificateFingerprint":
"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0"
        },
        {
            "value": "7729521220",
            "oidTitle": {
                "oid": "1.2.643.3.131.1.1",
                "title": "ИНН"
            },
            "oid": "1.2.643.3.131.1.1",
            "certificateFingerprint":
"2dec05353fb9593c22bf2c4dd3e054943b085df21fd5310265f78fa7fa19df0
0"
```

```

    }
  ]
},
"queues": [
  {
    "id": "2fbf1f83-55e3-4d87-96ca-c172cf1379cb",
    "type": "SEQ",
    "operations": [
      {
        "id": "b92c91ec-76f0-469d-b9ca-6cb038e4d749",
        "order": 2,
        "actorNumber": 1,
        "actorTitle": "Ответчик",
        "typeOf": "Документ 1",
        "empty": false,
        "queue": "2fbf1f83-55e3-4d87-96ca-
c172cf1379cb",
        "type": "sign"
      },
      {
        "id": "a6535abc-854b-4df1-b918-c5113758c2cd",
        "order": 1,
        "actorNumber": 0,
        "actorTitle": "Инициатор",
        "typeOf": "Документ 1",
        "empty": false,
        "queue": "2fbf1f83-55e3-4d87-96ca-
c172cf1379cb",
        "type": "sign"
      },
      {
        "id": "45053233-a3e7-4503-a001-8507cffba5a5",
        "order": 0,
        "actorNumber": 0,
        "actorTitle": "Инициатор",

```

```
        "description": "",
        "typeOf": "Документ 1",
        "empty": false,
        "queue": "2fbf1f83-55e3-4d87-96ca-
c172cf1379cb",
        "type": "publish"
    },
    {
        "id": "51924f3b-6b40-4f71-83be-03c6b397d143",
        "order": 3,
        "actorNumber": 0,
        "actorTitle": "Инициатор",
        "result": 0,
        "info": "",
        "empty": false,
        "queue": "2fbf1f83-55e3-4d87-96ca-
c172cf1379cb",
        "type": "finish"
    }
],
"order": 0,
"empty": false,
"specification": "57460583-7167-4b3a-9e14-
3bc9df7115a9"
}
],
"actors": [
    {
        "id": "517ff784-063c-4d30-856c-577317eeca0",
        "rights": 1,
        "number": 0,
        "title": "Инициатор",
        "selectors": [
            {
                "id": "fe151d26-5471-457d-92f7-f59d35dcdca5",
```

```
        "attribute": "1.2.643.6.3.1.2.1",
        "value": null,
        "action": null,
        "type": "ACTOR",
        "oidTitle": {
            "oid": "1.2.643.6.3.1.2.1",
            "title": "ЮЛ"
        },
        "oid": "1.2.643.6.3.1.2.1",
        "actorId": "517ff784-063c-4d30-856c-
577317eeca0",
        "needValue": false,
        "hasValue": false
    },
    ],
    "specificationId": "57460583-7167-4b3a-9e14-
3bc9df7115a9",
    "empty": false,
    "selectorLine": "1.2.643.6.3.1.2.1"
},
{
    "id": "3128cff0-ee25-4d6a-be35-cd963e93f433",
    "rights": 1,
    "number": 1,
    "title": "Ответчик",
    "selectors": [
        {
            "id": "853235d0-0a21-4850-9337-bd0993c4dc4c",
            "attribute": "1.2.643.6.3.1.2.1",
            "value": null,
            "action": null,
            "type": "ACTOR",
            "oidTitle": {
                "oid": "1.2.643.6.3.1.2.1",
                "title": "ЮЛ"
            }
        }
    ]
}
```

```
    },
    "oid": "1.2.643.6.3.1.2.1",
    "actorId": "3128cff0-ee25-4d6a-be35-
cd963e93f433",
    "needValue": false,
    "hasValue": false
  }
],
    "specificationId": "57460583-7167-4b3a-9e14-
3bc9df7115a9",
    "empty": false,
    "selectorLine": "1.2.643.6.3.1.2.1"
  }
],
  "chainResult": {
    "txId":
"9bad212e17030e03421bc9518f9790b0d8f7a58e12a544d7aaae9397dab8abb
f",
    "inChain": true,
    "rejectInfo": null,
    "chainStatus": "SUCCESS"
  },
  "empty": false,
  "emptyForSign": false
}
],
"pageable": {
  "sort": {
    "sorted": false,
    "unsorted": true
  },
  "pageSize": 10,
  "pageNumber": 0,
  "offset": 0,
  "paged": true,
```

```
    "unpaged": false
  },
  "last": true,
  "totalElements": 2,
  "totalPages": 1,
  "first": true,
  "sort": {
    "sorted": false,
    "unsorted": true
  },
  "size": 10,
  "number": 0,
  "numberOfElements": 2
}
```

Ошибка 400:

```
{
  "status": 400,
  "message": "Not found specification",
  "time": 1586893542
}
```

Приложение D. Презентация

Конструктор создания распределенных контрактов на основе технологии блокчейн

Автор работы: студентка гр. 16BB1 Веселова Марина Дмитриевна
Научный руководитель: к.т.н. Синев Михаил Петрович

Пенза, 2020

Цель и объект работы

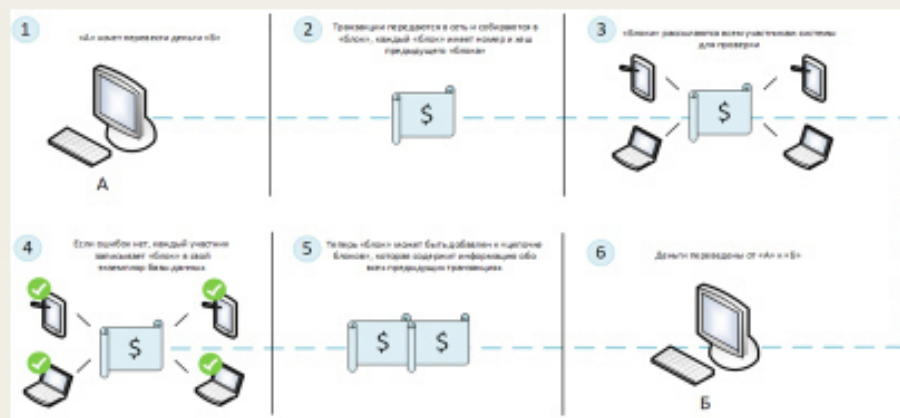
Цель работы: разработать систему, позволяющую конструировать создание распределенных контрактов на основе технологии блокчейн.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций:

- авторизацию пользователя в системе,
- создание спецификации,
- создание процесса на базе спецификации,
- запуск процесса с приложением к нему документа,
- заверение документов в блокчейн.

Хранение информации о пользователях, спецификациях, процессах, операциях должно осуществляться в локальной БД. Кроме того, должна быть реализована транспортировка данных с их хранением в децентрализованной распределенной базе данных.

Схема выполнения транзакции в блокчейн



3

Заключение смарт-контракта



4

Платформы, использующие блокчейн

- Sidechains;
- Ethereum;
- Hyperledger Fabric;
- EOS;
- DocChain;
- Multichain



Схема функционирования Sidechains



Алгоритм работы Hyperledger

5

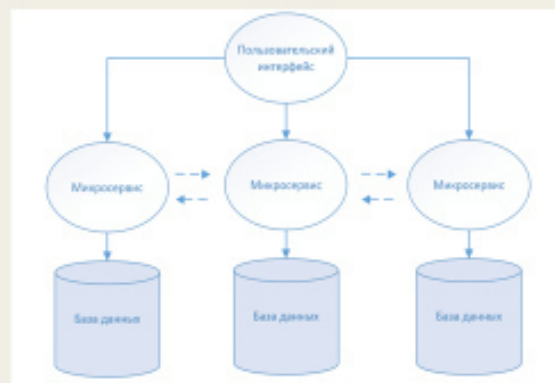
Стек технологий



MultiChain
Private Blockchain Platform

6

Микросервисная и монолитная архитектура приложения



Микросервисная архитектура

Монолитная архитектура



5

Подключение нового узла



Запрос на подключение



Установление подключения

6

Схема взаимодействия компонентов

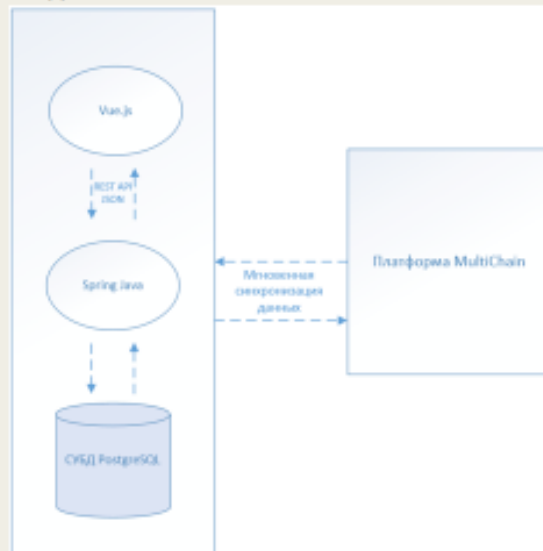
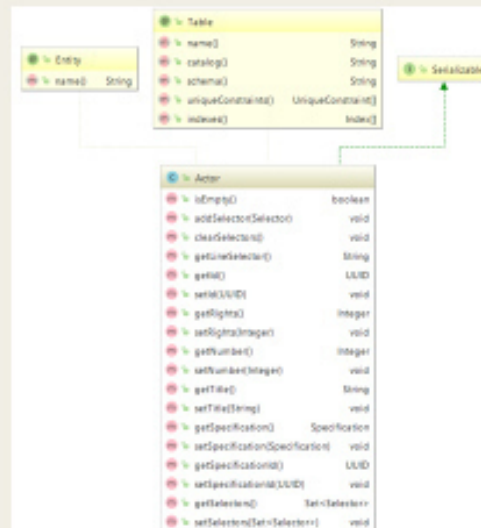


Схема базы данных

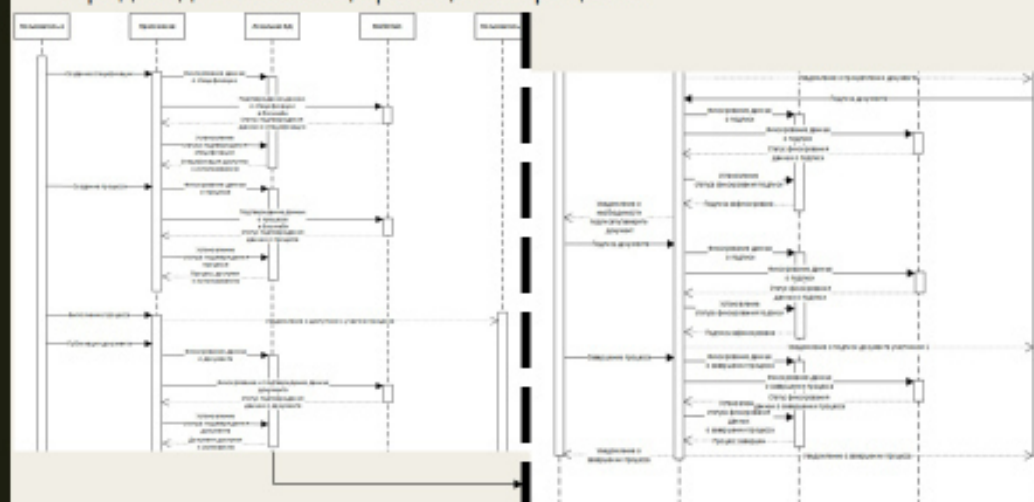


8

UML – диаграмма классов сущности «Участник спецификации»



UML – диаграмма алгоритма взаимодействия пользователей с предсозданием спецификации и процесса



10

Рабочий стол пользователя

ChainedDocs 00000000000000000000

Рабочий стол

- Процессы
- Новый процесс
- Спецификации
- Новая спецификация

[Создать новый процесс](#)

Процессы, ожидающие вашего действия

Процесс	Последнее изменение	Ожидаемое действие
Детектор ПО	09.05.2025, 11:14:37	Процесс завершён
Детектор ПО	09.05.2025, 11:14:38	Получены документы "Детектор"
Детектор ПО	09.05.2025, 11:14:41	Получены документы "Детектор"
Детектор ПО	09.05.2025, 11:14:44	Опубликованы документы "Детектор"

События

Процесс	Дата	ИМПУЛС/Получение	Событие
Детектор ПО	09.05.2025, 11:14:37	Актин Group T1	Процесс завершён
Детектор ПО	09.05.2025, 11:14:38	ООО "Внедрение"	Получены документы "Детектор"
Детектор ПО	09.05.2025, 11:14:41	Актин Group T1	Получены документы "Детектор"
Детектор ПО	09.05.2025, 11:14:44	Актин Group T1	Опубликованы документы "Детектор"

11

Экраны

Процессы

Процессы, ожидающие вашего действия

Процесс	Последнее изменение	Ожидаемое действие	Дата	Последнее изменение	Ожидаемое действие
Детектор ПО	Детектор ПО	Актин Group T1	09.05.2025, 11:14:37	09.05.2025, 11:14:38	09.05.2025, 11:14:41

Новый процесс

Выбор спецификации процесса (определяет возможности участников, структуру, порядок действий)

Выбор спецификации инициатора:

Тип:

Включить спецификации:

Детектор ПО

Детектор ПО

Детектор ПО

Детектор ПО

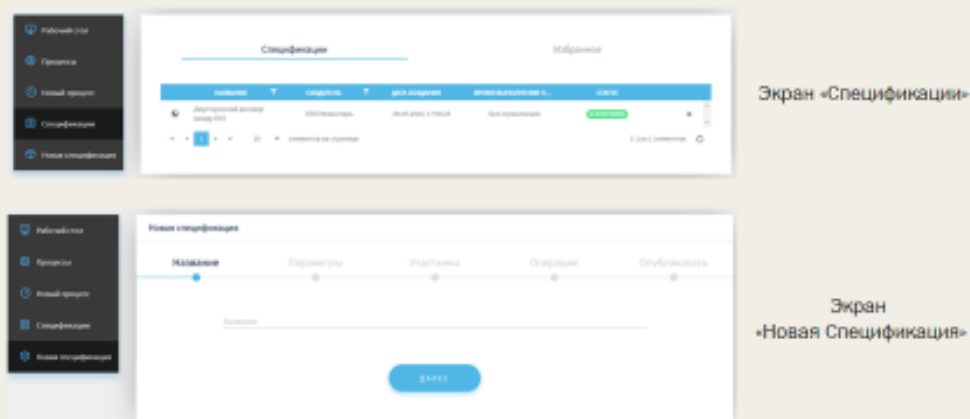
Создать новый процесс

Экран «Процессы»

Экран «Новый процесс»

12

Экраны



13

Результаты работы

Результатами решения поставленной задачи являются:

- разработанная структура базы данных для СУБД PostgreSQL, которая позволяет хранить данные о пользователе, созданных им спецификациях и экземплярах процессов, данные по выполнению операций процесса, а также правила для отправки данных в Multichain;
- разработанный механизм транспортировки данных в Multichain. Транспортируются следующие данные: данные о пользователе (сертификат, ключи), данные о процессе (операции, выполненные в рамках процесса; хэш приложенного документа), данные о спецификации (хэш записи спецификации процесса);
- разработанный механизм фиксирования и заверения данных с использованием технологии блокчейн, реализованной в рамках одноранговой децентрализованной сети;
- разработанное и протестированное приложение «Конструктор создания распределенных контрактов на основе технологии блокчейн».

14

Спасибо за внимание!

