ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

СИСТЕМА ПОДТВЕРЖДЕНИЯ И ВЫДАЧИ СТРАХОВЫХ ПОЛИСОВ ДЛЯ АВТОМОБИЛЬНОГО ТРАНСПОРТА

Выпускная квалификационная работа
обучающегося по направлению подготовки 09.03.02 Информационные системы и технологии
заочной формы обучения, группы 07001353
Левенсон Семена Яковлевича

Научный руководитель
доцент
Гахов Р.П.

БЕЛГОРОД 2018

РЕФЕРАТ

Система подтверждения и выдачи страховых полисов для автомобильного транспорта. – Левенсон Семен Яковлевич, выпуская квалификационная работа бакалавра Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 76, включая приложения 13, количество рисунков 18, количество таблиц 3, количество использованных источников 23.

КЛЮЧЕВЫЕ СЛОВА: блокчейн, ОСАГО, Ethereum, страхование, автострахование, оптимизация бизнес-процессов, Solidity, smart contracts.

ОБЪЕКТ ИССЛЕДОВАНИЯ: рынок страховых услуг.

ПРЕДМЕТ ИССЛЕДОВАНИЯ: социально-экономические отношения, возникающие в процессе формирования и развития рынков страхования.

ЦЕЛЬ РАБОТЫ: оптимизация текущих бизнес-процессов при оформлении и поддержке полиса ОСАГО через автоматизацию валидации входящих запросов от страхователя, для снижения издержек у страховых компаний и привнесения большего удобства для страхователей

ЗАДАЧИ ИССЛЕДОВАНИЯ: изучить понятие «блокчейн»; исследовать принцип оформления страхового полиса ОСАГО; проанализировать текущее состояния автострахования и применение блокчейна в нем; выбрать платформу для разработки; реализовать выдачу страхового полиса средствами блокчейн технологий.

МЕТОДЫ ИССЛЕДОВАНИЯ: идеализация, синтез, сравнение.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: в результате работы была спроектирована и реализована система выдачи и поддержки полисов ОСАГО средствами блокчейн технологий.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автострахование является неотъемлемой частью жизни автовладельцев и водителей в России. С 2002 года появилось обязательное страхование гражданской ответственности владельцев транспортных средств. Сфера развивается, за эти годы появилась возможность оформлять страхование удаленно через Интернет, но все равно есть ряд проблем, которые можно исправить, внедрив в эту сферу децентрализацию средствами блокчейн технологий.

Блокчейн – одна из самых перспективных технологических отраслей (наряду с Big Data, Machine learning, искусственным интеллектом), сравнимая по масштабу, степени влияния и распространению в будущем с тем эффектом, который в 1990-2000-е годы произвел интернет.

Снижение издержек, повышение уровня безопасности и более высокая прозрачность оформления полиса ОСАГО – три главные и сильные стороны, которые может предоставить блокчейн. И в связи с потребностью страховщиков, бизнеса и общества в этих трех аспектах, любая теоретическая работа или разработка в этой области становится достаточно актуальной.

Целью данной работы является оптимизация текущих бизнес-процессов при оформлении и поддержке полиса ОСАГО через автоматизацию валидации входящих запросов от страхователя, для снижения издержек у страховых компаний и привнесения большего удобства для страхователей. Для достижения поставленной цели необходимо решить следующие задачи:

- изучить понятие «блокчейн»;
- исследовать принцип оформления страхового полиса ОСАГО;
- проанализировать текущее состояния автострахования и применение блокчейна в нем;
- выбрать платформу для разработки;

– реализовать выдачу страхового полиса средствами блокчейн технологий;

Объектом исследования является рынок страховых услуг.

Предметом исследования стали социально-экономические отношения, возникающие в процессе формирования и развития рынков страхования.

Данная работа состоит из введения, четырех глав, заключения, списка использованных источников и приложений. Первая – аналитическая. В ней будет исследовано текущее состояние рынка автострахования, теория блокчейна и есть ли сегодня какие-то движения к их объединению. Вторая – обоснование проектных решений. В ней будет проведен анализ того, какие платформы для создания смартконтрактов есть и какие у них преимущества. Третья – проектная, в ней будет реализация смартконтрактов и децентрализованного приложения и экономический анализ. В нем рассмотрим, как разработанный продукт повысит продажи страховых полисов, снизит издержки на их выдачу и минимизирует возможность злоумышленных действий, в отношении оформления полиса.

1 Аналитическая часть

1.1 Что такое ОСАГО

Обязательное страхование гражданской ответственности владельцев транспортных средств — вид страхования ответственности, возникший в США в 20-х годах XX века и получивший очень широкое распространение в 40—50-хх годах в Европе (а позже — и в остальном мире), при котором объектом страхования являются имущественные интересы, связанные с риском гражданской ответственности владельца транспортного средства по обязательствам, возникающим вследствие причинения вреда жизни, здоровью или имуществу потерпевших при использовании транспортного средства. ОСАГО вводилось во всех странах как социальная мера, направленная на создание финансовых гарантий возмещения ущерба, причинённого владельцами транспортных средств и как финансовый инструмент повышения безопасности дорожного движения. Подобное страхование действует во многих странах мира, а также в рамках транснациональных соглашений — например, «зелёная карта».

В СССР идея введения этого закона обсуждалась в 60-хх годах XX века, но тогда от внедрения такого страхования было решено воздержаться. Вернулись к разработке этого закона в 1993—1994 годах, когда в Государственную Думу стали поступать различные версии соответствующего законопроекта. Финальный этап разработки закона пришелся на 2000—2002 годы. В 2000 году законопроект в первом чтении был рассмотрен Государственной думой. Окончательно ОСАГО пришло в Россию 1 июля 2003 года со вступлением в силу Федерального закона № 40-ФЗ от 25 апреля 2002 года «Об обязательном страховании гражданской ответственности владельцев транспортных средств».

С 1 июля 2015 года автовладельцы могут оформить полис ОСАГО через интернет. Данное постановление было подписано Дмитрием Медведевым.

1.2 Электронный ОСАГО

Электронное страхование является аналогом бумажного полиса, который юридически ничем не отличается от обычного бумажного страхования. Сегодня его можно приобрести на сайтах страховых компаний в Интернете. Поскольку весь процесс регистрации является удаленным, электронная версия является полностью виртуальной и не имеет "физического воплощения" в виде бумажного договора и страхового свидетельства, напечатанного на бланке Гознака. По сути, этот формат является единственным существенным отличием электронного страхового ОСАГО от обычного. Для оформления такой политики использовали те же исходные данные: КБМ, регион проживания, тип и мощность автомобиля, стаж и возраст водителя, и т. д.

Основным преимуществом электронного страхования является возможность дистанционного оформления автострахования. Страхователю не нужно лично посещать офис, готовить необходимые документы и терять время в очереди. Особенно это касается занятых граждан, а также жителей отдаленных регионов из-за недоступности привычной политики.

Кроме этого, к плюсам онлайн-страхования относят следующие моменты:

- электронную страховку невозможно потерять (поскольку она не имеет материального воплощения), а сама копия полиса будет храниться сразу в 3 экземплярах (в базе данных РСА, у страховой компании и у самого собственника транспортного средства);
- возможность оформить так называемый «чистый полис» (то есть без дополнительных услуг и ненужных опций, которые порой насильно навязываются клиентам в офисах страховщика);

– при соблюдении самых базовых правил осторожности вероятность мошенничества минимальна (тогда как покупая бумажный вариант ОСАГО, можно стать жертвой обмана со стороны сотрудников самой страховой организации);

– у граждан отдаленных регионов появилась реальная возможность выбрать любую понравившуюся страховую компанию (до появления электронного ОСАГО в таких районах, как правило, действовали 1-2 страховщика и у водителей просто не было другого выхода, кроме как оформлять автогражданку в этих организациях).

Несмотря на большое число положительных сторон, у электронного ОСАГО можно выделить и ряд негативных черт:

– оформление возможно только для тех граждан, информация о которых уже содержится в базе данных РСА (то есть онлайн заказ полиса недоступен для водителей новичков);

– ошибки, допущенные при вводе большого количества информации о транспортном средстве и его владельце, будут дорого стоить пользователям (такие полисы недействительны, но деньги за неудачную регистрацию не возвращаются, так как эти ошибки интерпретируются страховыми компаниями как предоставление ложной информации);

– проверка электронного ОСАГО отнимает у инспекторов ГИБДД куда больше времени, чем бумажный вариант (ведь им нужно узнать данные автомобиля, а затем сверять их с базой);

– большое количество технических сложностей, которые могут возникнуть при оформлении полиса в интернете (со временем систему должны отладить, но на данный момент различные сбои, ошибки базы данных и прочие проблемы на сайте страховщиков – привычная картина). Стоит упомянуть, что из-за блокировки Роскомнадзором IP-адресов Google были проблемы с оформлением электронных полисов ОСАГО, их продажи рухнули в четыре раза. Об этом предупредил Российский союз автостраховщиков (РСА) на своем сайте.

В дополнение, в существующей системе очень велика вероятность потери скидки за безаварийное вождение, ввиду потери страховой истории. Наиболее распространены причины:

- Ошибки операторов при вводе данных. Здесь речь идёт о человеческом факторе. Перепутанные цифры даты рождения или номера паспорта сводят на нет будущие усилия по поиску автолюбителя в базе.
- Смена фамилии, замена паспорта (для страхования «без ограничений») или водительского удостоверения. Часто страхователи не тратят время на внесение соответствующих корректировок в полис.

Как видим, сфера развивается, но все равно имеет ряд проблем. Как альтернативу текущей системе, предлагаю рассмотреть блокчейн технологии, которая внесет прозрачность в оформлении полиса ОСАГО, сохранение безаварийной истории и удобство в закрытии полиса. Но, для начала, надо разобраться, что из себя представляет технология блокчейн.

1.3 Что такое блокчейн

Винченцо Морабито – автор одной из самых известных книг о блокчейне «Business innovation through blockchain» – определяет технологию как распределенную децентрализованную защищенную шифром базу, публичный депозитарий информации, в котором каждая совершенная транзакция записывается и становится известна всем участникам сети. Любая транзакция в реестре признается действительной, только если ее одобряет более чем половина участников сети. Это означает, что никакая внешняя сторона или агент не может проводить транзакции без согласия других пользователей.

Любая сделка по существу является передачей права собственности. Характер практически любой такой сделки подразумевает отсутствие взаимного доверия между сторонами сделки, что требует присутствия в сделке третьей стороны, которая гарантировала бы ее исполнение. Концепция блокчейна позволяет участникам системы договариваться о сделке без участия

и подтверждения от посредника. Таким образом, отпадает необходимость в посреднике, что теоретически позволяет изменить все сферы жизни человека, где так или иначе происходит обмен между людьми, не имеющими взаимного доверия.

Блокчейн – информационный массив, имеющий следующие характеристики:

– Функционирующая по принципу peer-to-peer децентрализованная распределенная система

– Определенные участники могут вносить изменения

– Используется цифровая подпись и криптографические алгоритмы для аутентификации, верификации пользователя и предоставлении права вносить изменения и отслеживать факты транзакций

– Структура системы делает практически невозможным внесение изменений в уже состоявшиеся записи (совершенные транзакции)

– Структура системы приводит к тому, что участники системы становятся быстро осведомленными о том, что кто-то пытается внести изменения в совершенные транзакции

– Непосредственные участники и широкая аудитория могут отслеживать транзакции

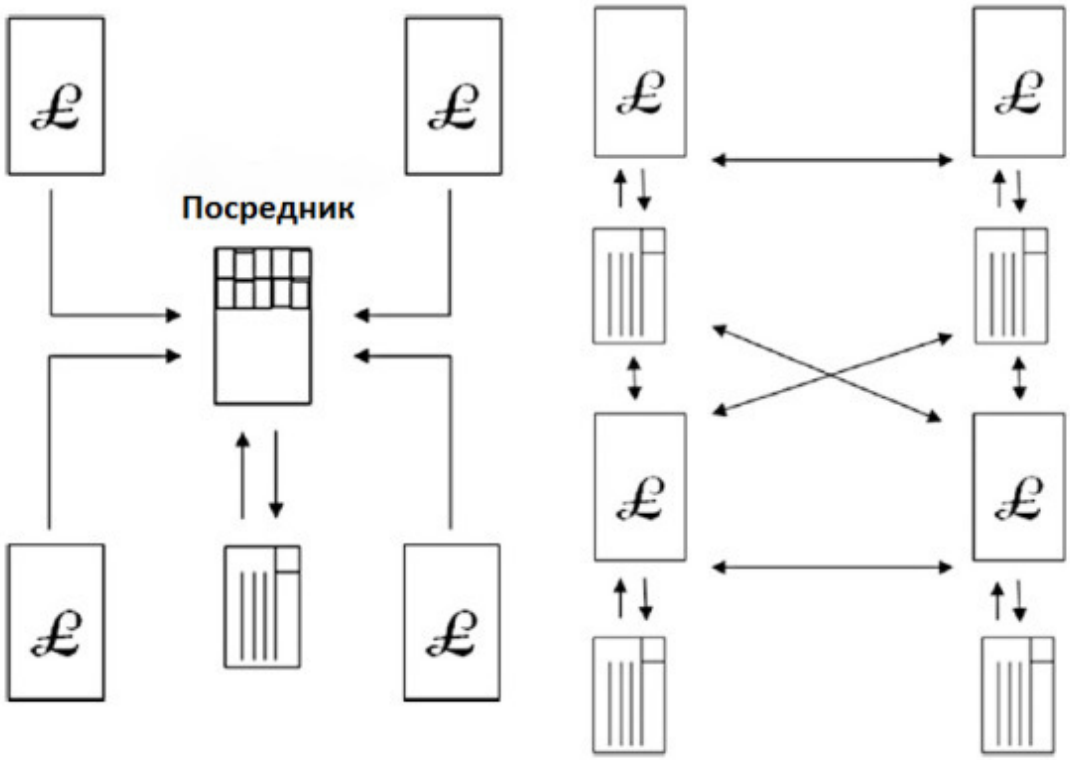

Рисунок 1.1 Централизованная и децентрализованная системы

Блокчейн - цепочка блоков транзакций, реализованных на основе распределенной базы данных. Ключевым элементом является журнал транзакций, и единственным способом изменения состояния реестра является выполнение транзакций. Чтобы транзакция считалась действительной и подтвержденной (требуется согласие более половины участников сети), формат и подписи должны быть валидированы и валидация ее (или группы транзакций) фиксируется в блоке.

Блок включает в себя список транзакций и заголовок (header), который содержит собственный хеш, хеш предыдущего блока, хеш транзакций и дополнительную информацию. Связь между блоками за счет наличия в каждом (за исключением первого) хеша предыдущего означает, что невозможно внести изменения в блок, не изменив всю цепочку с первого блока – нельзя удалить какую-то транзакцию или вставить ее между уже совершенных. Хеш-функции и электронная подпись – два важнейших элемента блокчейна, обеспечивающие связность и авторизацию.

Децентрализация блокчейна обеспечивает его стабильность-даже если некоторые узлы на некоторое время выйдут из строя, система все равно продолжит функционировать. Суть децентрализации и распространения состоит в том, что каждый член сети имеет на своем жестком диске полную копию текущего реестра, что делает невозможным компрометацию.

При проведении транзакции на вход подается информация о ней, а на выходе генерируется хеш, который записывается в хеш-сумму. Таким образом, если в блоке попытаются изменить хотя бы один бит, все участники системы (узлы) будут оповещены об этом.

Несмотря на то, что блокчейн является распределенной системой, и каждый участник может провести транзакцию, не все из них равны. Участники системы делятся на операторов (майнер / валидатор), проводящих транзакции; сетевых регуляторов, ответственных за регистрацию и обычных участников.

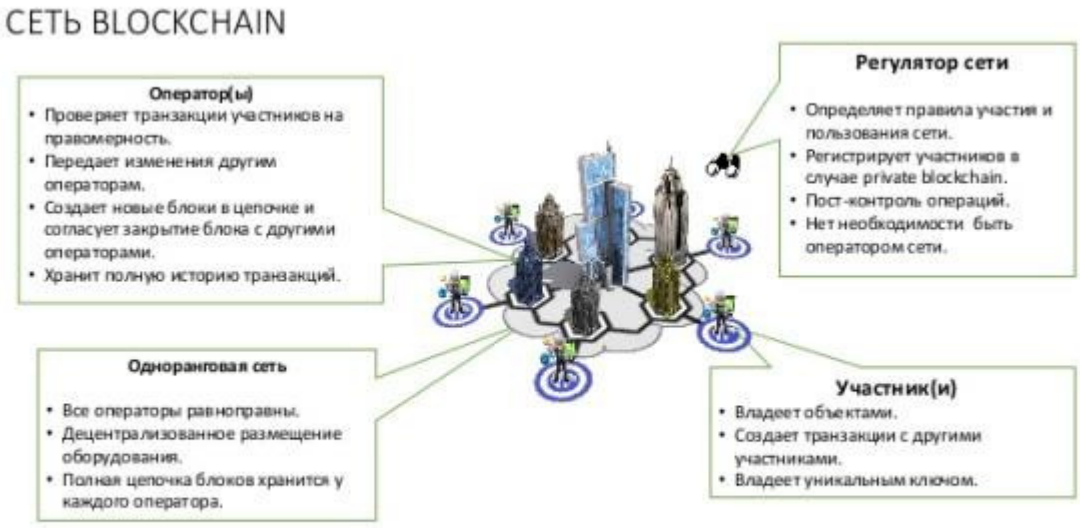

Рисунок 1.2 Структура участников блокчейна

Условия применения (согласно отчету «Сбербанка» на форуме «Блокчейн и открытые платформы»):

a) Наличие в сети определенного минимального количества числа независимых операторов (не менее 50%)

b) Большинство операций проходит напрямую

c) Каждый владеет минимум одной ключевой парой public-private для несимметричного шифрования

d) Открытость истории всех транзакций для всех операторов

e) Конфиденциальна только сами транзакции, а не факт их совершения

1.4 Принцип работы блокчейна

Базовая идея уже была описана выше, здесь будет обозначен весь процесс.

Первым шагом является определение транзакции. Отправитель создает транзакцию, которая содержит информацию об адресе получателя, предметом сделки (сумма средств, товаров и т. д.) и криптографическую цифровую подпись, которая проверяет действительность сделки и ее действительность.

Аутентификация транзакции. Хосты уведомляются о транзакции и проверяют действительность транзакции путем расшифровки электронной подписи. Если транзакция проходит проверку, она переходит в режим ожидания для включения в единицу.

Создание блока. Один из узлов сети один раз в определенный промежуток времени (0,25 минуты в случае Эфириума) собирает транзакции, находящиеся в режиме ожидания, формирует из них блок и отправляет их на подтверждение другим участникам сети для верификации и подключения к цепочке.

Валидация блока. Узлы, ответственные за проверку блоков, получают запрос на проверку созданного блока. Они выполняют повторяющийся процесс, который требует утверждения от других операторов хоста, чтобы сделать блок допустимым.

Присоединения блока к цепочке. Когда все транзакции в блоке одобряются, новый блок становится присоединенным к общей цепочке.

1.5 Преимущества и ограничения блокчейна

Винченцо Марабито обозначил преимущества и возможные ограничения. Сначала список преимуществ.

Уполномоченные пользователи. Блокчейн дает пользователям возможность контролировать информацию, а также транзакции, частью которых они являются.

Прочность, надежность, долговечность. Блокчейн не зависит от централизованной компьютерной архитектуры, что приводит к тому, что потеря отдельных узлов нарушит работу всей системы.

Прозрачный и неизменный процесс. Транзакции в блокчейне можно отследить и невозможно (экономически невыгодно) изменить.

Более быстрые и дешевые транзакции. Блокчейн имеет потенциал значительно снизить время и издержки, путем отказа от посредников и третьих лиц.

Однако внедрение технологии не идет гладко, поскольку влечет за собой некоторые проблемы, вызванные структурой и принципами технологии – среди них процесс верификации сделки и ограничение на количество сделок в определенный промежуток времени. Кроме того, есть несколько, которые могут быть препятствием для внедрения блокчейна.

Статус государства как регулятора. В настоящее время валюты, используемые в финансовых операциях, регулируются государствами. Если блокчейн будет широко распространен в финансовых институтах, то госорганам придется решать, как его регулировать, иначе его статус останется неопределенным. На данный момент нет никаких правил и законов, регулирующих его работу.

Вопросы безопасности и privacy. Несмотря на существование решений в области безопасности с использованием сложных алгоритмов шифрования, проблемы кибербезопасности остаются одним из главных факторов, влияющий на решение общества о передаче персональных данных с помощью блокчейнов.

Уязвимость программного обеспечения. Ошибки в программном коде

всегда существуют, и это особая Уязвимость для злоумышленников. Поскольку программное обеспечение становится все более сложным и привязанным к взаимодействию между пользователями, его надежность падает, а количество уязвимостей увеличивается. Хотя технология быстро совершенствуется, написанный человеком код никогда не будет совершенным-blockchain не исключение. Кроме того, целостность программного обеспечения и сети принципиально важна для трансформации блокчейна в инфраструктурные технологии. Если блокчейн переплетается со всеми основными финансовыми системами мира, то мощные атаки на него могут привести к катастрофическим последствиям.

Вопросы интеграции. Когда организация внедряет новые технологии для модернизации своих бизнес-процессов, она сталкивается с проблемой изменения регулирования для интеграции новой системы в соответствии со старой системой. В этой ситуации внедрение блокчейна ничем не отличается – это приведет к возникновению сложной задачи, как правильно спланировать переход от нынешних систем к блокчейну.

Понимание технологии. Один из самых больших операционных рисков заключается в том, что относительно небольшое количество людей понимают, как это работает. Программисты и хакеры имеют опыт написания программного обеспечения, понимания основных функций и могут работать с ним. Тем не менее, компании должны быть обеспокоены внедрением технологии, которая мало известна.

Затраты на внедрение. Сокращение затрат, которое обещает реализация блокчейна, обнадеживает, но реализация потребует высоких начальных затрат, которые не следует принимать во внимание.

Децентрализация является одним из ключевых элементов блокчейна. С его появлением нет необходимости использовать централизованные системы и электронные платформы, такие как Google Drive. С использованием децентрализованных и зашифрованных протоколов обмена информацией, сообщения могут быть отправлены, сохранены и отправлены без какого-либо

вмешательства со стороны государства.

Децентрализованные базы данных обеспечивают децентрализацию и защиту способов обмена информацией. При необходимости информация может быть опубликована и распространена на большом количестве компьютеров в зашифрованном виде, что исключает возможность редактирования кем угодно. Примером децентрализованной базы данных является анонимная децентрализованная облачная система хранения данных, которая использует блокчейн в сотрудничестве с другой одноранговой системой, что позволяет пользователям использовать дополнительное пространство на жестком диске. Это похоже на централизованную платформу, основанную на облачных вычислениях, но с технологической точки зрения порядок действий на этих платформах отличается.

1.6 Блокчейн в страховании

Существует как минимум три способа использования блокчейна, которые открывают новые возможности роста для страховых компаний: повышение интереса клиентов, предложение новых экономически эффективных продуктов для развивающихся рынков и разработка страховых продуктов, связанных с "Интернетом вещей". Ключ к успеху заключается в создании надежной распределенной платформы, основанной на новой технологии для работы непосредственно с клиентами с их личными данными, коллективными страховыми полисами и смарт-контрактами.

Повышение интереса клиентов. Важным инструментом влияния на клиентов, позволяющим им повысить интерес к сервисам компании, станет использование преимуществ блокчейна для работы со своей персональной информацией. Каждый раз люди должны заполнять формы, формы с информацией о себе, при этом также необходимо подтвердить эту информацию. И, как написано выше, при таких операциях могут возникать ошибки, которые приводят к нежелательным последствиям.

Такие стартапы, как Tradle работают над блокчейн-решениями для осуществления процедуры проверки данных клиента (англ. Know your client (KYC) на подлинность и благонадежность. Данная процедура требует согласия клиента на все необходимые проверки его данных для закрытия договора. После того, как профиль клиента был проверен, он может отправить подтвержденные идентификационные данные другим компаниям для подписания новых контрактов с тем же инструментом, избегая необходимости проходить весь процесс проверки снова. Это ускоряет и повышает эффективность процесса привлечения клиентов.

Еще одним способом повышения заинтересованности клиентов является повышение прозрачности, а также обеспечение четких схем образования тарифов и механизмов страховых выплат потребителю. Например, InsureETH стартап продемонстрировал схему авиа-страхования пассажиров на blockchain с помощью смарт-контрактов. Смарт-контракты автоматически запускают страховые выплаты в случае отмены или задержки рейса. Полетная информация поступает из проверенных источников и обрабатывается так называемыми оракулами, которые адаптируют данные из внешних источников для их последующего использования в блокчейне. Несмотря на то, что коллективное страхование как бизнес-модель уже давно получило широкое распространение и качественное внедрение с использованием стандартных технологий, блокчейн, в силу своего децентрализованного характера, сделает его еще более прозрачным и понятным для потребителей. Что касается поставщиков страховых услуг, то преимущество блокчейна заключается в автоматизации бизнес-процессов.

Независимо от контекста применения смарт-контрактов, в совокупности с блокчейном, они предлагают ряд преимуществ: позволяют автоматизировать удовлетворение страховых требований, предлагают надежный и прозрачный механизм организации платежей для всех сторон и могут быть использованы для индивидуальной и тонкой настройки условий каждого отдельного контракта. Например, в случае ДТП смарт-контракт может гарантировать, что

оплата будет произведена только в том случае, если автомобиль будет отремонтирован в заранее выбранном страховщиком или более удобном пункте обслуживания. Несмотря на возможность реализации таких программ без использования блокчейна, Платформа смарт-контрактов на основе новой технологии может обеспечить уникальные преимущества. Она не только гарантирует повышение прозрачности и доверия со стороны клиентов за счет децентрализации и автоматизации процесса согласования и проверки данных, но и обеспечивает ощутимый сетевой эффект, которого нет в наличии благодаря использованию централизованных платформ.

По некоторым оценкам, от 5 до 10% претензий на страховые выплаты являются попытками мошенничества. По данным ФБР, такие случаи стоят страховым компаниям (кроме меда. страховщиков) более 40 миллиардов долларов в год. В целях повышения эффективности выявления мошенничества, фальсификации информации о телесных повреждениях или ущербе имуществу рассматривается возможность создания специальной блокчейн-базы данных, которая будет служить единым распределенным реестром для всей отрасли, содержащим как внешние данные, так и информацию о клиентах.

Это позволит:

- Подтверждать личность клиентов, подлинность их документов (медицинских отчетов), право собственности и проверять происхождение средств на их счету.
- Проверять переданные в полицию заявления о краже и другие аналогичные документы, обнаруживать мошеннические наклонности отдельных личностей.
- Подтверждать дату и время выпуска полисов или покупки продуктов и активов.
- Подтверждать данные о смене владельца имущества или изменения его местонахождения.

В любом случае, для того чтобы ощутить полную выгоду от использования блокчейн-приложений вместо традиционных решений и существующих форм сотрудничества, таких как отраслевые ассоциации, потребуется интенсивное сотрудничество между страховщиками, производителями, потребителями и другими заинтересованными сторонами. Примером блокчейн-транспортного средства является ранняя демонстрация уже формирующейся сегодня новой экосистемы, существование которой выходит за рамки традиционной сферы страхования.

Страхование в России остается очень консервативной отраслью. На страховщиков работают сотни тысяч человек, оформляющих страховки, и большую часть из них технология может лишить рабочих мест. Однако изменения уже есть: «Сбербанк страхование» уже работает над единой базой данных всех страховых историй на основе блокчейна, которая была бы доступна всем страховщикам. Другие игроки, уверен, присоединятся к первопроходцу, ведь страховая отрасль больше всех остальных около финансовых отраслей страдает от мошенников.

2 Обоснование проектных решений

2.1 Smart contracts (умные контракты)

Блокчейн может автоматизировать сообщения с помощью специальных фрагментов кода, называемых смарт-контрактами. Эти контракты используют логику «if-this-then-that» – «если это, тогда – то». Процесс функционирования смарт-контрактов не предполагает какого-либо вовлечения людей. Это означает, что эти контракты децентрализованы, что позволяет им работать без посредников и регуляторов в форме третьих сторон.

Вдобавок, умные контракты могут быть запрограммированы так, чтобы отражать бизнес-логику, основанную на данных, что может включать в себя:

- голосование
- передачу прав собственности
- распределение долей
- расчеты стоимости
- сбор средств до определенного значения или даты
- совместная покупка собственности

Рисунок 2.1 Представление бизнес-логики в smart contracts

Таблица 2.1 Описание бизнес-логики в smart contracts

Номер на блок-схеме	Текст на блок-схеме	Описание
1	Условия контракта	Стороны сделки устанавливают обязанности и правила. Активы под доверительным распоряжением smart contracts Условия для использования логики контракта («if…then»)
2	События	События запускают действие контракта События относятся к сделкам или полученной информации
3	Бизнес-логика	Бизнес-логика диктует движение активов исходя из условий
4a	Предмет контракта	Активы передаются получателю, указанному в условиях договора
4b	Регистрация соглашения	Изменения в праве собственности на активы отражаются в реестре

Впервые идея смарт-контракта была описана в 1994 году известным ученым в области информатики и криптографии Ником Сабо, но была применена лишь спустя 14 лет с появлением блокчейна. Уже тогда алгоритм блокчейна Биткоина был основан на принципах исполнения смарт-контрактов, но они не были реализованы в клиентском программном обеспечении по соображениям безопасности.

Смарт-контракты широко использовались с запуском блокчейна Ethereum в 2015 году. Сегодня этот блокчейн считается наиболее удобным для реализации не только смарт-контрактов, но и dapps-децентрализованных приложений. Обязательные условия смарт-контракта:

Децентрализованный распределенный реестр — блокчейн-платформа, на которой будет выполняться смарт-контракт.

Стороны с электронными подписями — участники договора, которые подтверждают свое участие и согласие с условиями контракта электронной подписью.

Предмет договора — объект, который находится внутри среды существования самого смарт-контракта, то есть блокчейна. Такими объектами могут быть криптовалюты, которые обеспечивают прямой доступ смарт-контракта к предмету договора без участия человека.

Условия — алгоритм, описывающий логику исполнения пунктов предмета договора математическим путем.

Среди недостатков смарт-контрактов эксперты отмечают:

Затраты и сложность внедрения новой технологии-для реализации смарт-контрактов требуется понимание программирования, а для того, чтобы создать надежный смарт-контракт, отражающий потребности компании, желательно иметь опытного разработчика в штате, который хорошо разбирается не только в программировании, но и в предметной области. Так как в ситуации, когда человек гибко подстраивается под ситуацию – контракт нуждается в четких и понятных инструкциях. Более того, большинство пользователей до сих пор мало понимают, как работают алгоритмы, на которых строятся смарт-контракты.

Человеческий фактор - так как смарт-контракт-это сложный алгоритм, который должен учитывать множество факторов и условий сделки, то для его подготовки необходимо прописывать различные варианты развития. Чем сложнее процесс, тем сложнее создать смарт-контракт, и тем выше шанс совершить ошибку. По данным газеты материнской платы, на данный момент существует более чем 34 000 смарт-контрактов с ошибками в коде на рынке крипто, который предоставляет предприятию огромные потенциальные риски.

Правовой статус-криптовалюты используются для работы смарт-контрактов, законодательный статус которых определен не во всех странах. Более того, если государственные органы решат создать отдельную правовую базу для смарт-контрактов, предприниматели могут столкнуться с рядом новых проблем.

2.2 Выбор платформы

Основной игрок на этом рынке, это Ethereum. Он позволяет разработчикам программировать смарт-контракты благодаря JavaScript-подобному языку программирования Solidity, который был создан специально для написания самовыполняющихся контрактов. Solidity — это полный кроссплатформенный язык программирования по Тьюрингу, однако на практике он используется преимущественно на платформе Ethereum.

С точки зрения продолжительности блокчейн бума, Ethereum считается весьма зрелым и стабильным. Первоначально он был описан в одной из публикаций канадско-российским программистом, Виталиком Бутериным — в конце 2013 года. Ethereum неформально описывался как платформа «следующего поколения Биткойна» (или «Биткойн 2.0»).

Платформа Ethereum с помощью «умных контрактов» может быть использована в различных областях сферы финансов, но прежде всего Ethereum это криптовалюта. О своём интересе к платформе заявили различные организации, включая Microsoft, IBM и JPMorgan Chase. Bloomberg Businessweek утверждает, что распределённое программное обеспечение Ethereum может быть использовано всеми, кому нужна защита от несанкционированного вмешательства. Согласно тексту издания: «вы можете спокойно делать бизнес с кем-то, кого вы не знаете, потому что условия прописаны в смарт-контракте, встроенном в блокчейн». И это очень интересное предложение, так как, вместо необходимости обращаться к нотариусу или любому другому посреднику, достаточно на основе контракта заключить сделку. Так же Сбербанк присоединился к некоммерческому

альянсу Enterprise Ethereum Alliance, для более тесного контакта с разработчиками.

По данным Нью-Йорк Таймс, к марту 2016 года используются десятки функционирующих приложений, построенных на платформе Ethereum — децентрализованных приложений, децентрализованных автономных организаций и смарт-контрактов.

Но область смартконтрактов достаточно бурная, поэтому, помимо Ethereum реализацию смарт-контрактов предлагают и другие:

NEO — данный некоммерческий блокчейн-проект был запущен в Китае в 2014 году для развития децентрализованной «умной экономики». Многие эксперты отмечают, что NEO превосходит блокчейн Ethereum по многим показателям, в том числе и по принципам работы со смарт-контрактами. В сети NEO для выполнения заданных условий смарт-контракты используют виртуальные машины (VM), которые автоматически оптимизируют код смарт-контракта перед тем, как запустить его, организуя его так, чтобы он работал с наивысшим коэффициентом полезного действия.

Nxt — децентрализованная площадка с открытым исходным кодом для запуска защищенных DApps: электронных платежных систем, мессенджеров и торговых площадкок. Платформа была запущена в ноябре 2013 года с целью генерации собственных токенов на NXT блокчейне с неограниченной областью использования. Также платформа NXT содержит ограниченный набор шаблонов для умных контрактов, однако пользователи не имеют возможности запустить собственные смарт-контракты.

Jincor — блокчейн-платформа, позволяющая любому бизнесу работать с умными контрактами и криптовалютными платежами без каких-либо юридических, технических или финансовых затрат. Платформа закрыла ICO в ноябре 2017 года, а запуск альфа-версии конструктора смарт-контрактов запланирован на конец 2018 года. Данный конструктор смарт-контрактов будет применим в различных областях бизнеса и юрисдикций, предоставляя удобные криптовалютные платежи и децентрализованную арбитражную

систему для разрешения спорных случаев, связанных с исполнением смарт-контрактов.

Qtum — китайская гибридная блокчейн-платформа, которая была запущена в мае 2017 года с целью объединить тьюринг-полные смарт-контракты и DApps для удобного использования в бизнесе. Криптовалюта платформы сочетает в себе возможности блокчейна Bitcoin с виртуальной машиной Ethereum и совместима с обеими экосистемами. В рамках платформы функционируют умные контракты новой формации — мастер-контракты, особенность которых заключается в том, что контроль над расторжением контракта или его подписанием полностью находится в руках самих участников. Также разработчики предусмотрели запуск мобильной версии платформы, что открывает возможности блокчейн-технологии для более широкого круга пользователей.

Ubiq — децентрализованная платформа с открытым исходным кодом для запуска и реализации смарт-контрактов и DApps, работающих в автоматическом режиме. Платформа была запущена в сентябре 2014 года, а в январе 2017 года сменила блокчейн Jumbucks на блокчейн UBIQ, который основан на Ethereum. Проект Ubiq ориентирован на предоставление автоматизированных смарт-контрактов с высокой пропускной способностью для бизнеса, в то время как разработчики позиционируют платформу в качестве суперкомпьютера для работы с блокчейном.

Urbit — сеть персональных облачных p2p серверов, запущенная в 2016 году для хранения данных, выполнения программ и подключения к другим пользователям. В сентябре 2017 года проект Urbit добавил поддержку смарт-контрактов Ethereum. Согласно блогу компании, смарт-контракты будут основываться на стандарте ERC20, чтобы позволить владельцам «облачной недвижимости» Urbit криптографически защищать свои активы. Таким образом, смарт-контракты Ethereum будут выполнять функцию защиты пользователей.

В итоге перед нами стоит выбор, какую платформу выбрать? Узнав, что в октябре 2017 года Сбербанк вступил в некоммерческий альянс Enterprise Ethereum Alliance, став первым российским банком в его составе, стало ясно, что Ethereum — проверенный выбор. Используя платформу Ethereum, можно не сомневаться в ее работе. Ethereum — это Золотой Стандарт мира смарт-контрактов, а платформа имеет самую крупную рыночную капитализацию среди конкурентов.

2.3 Разработка смартконтрактов для Ethereum на Solidity

Solidity, как язык и Ethereum, как платформа, на, довольно, раннем этапе развития, по сравнению с Java, например. На данный момент нет такого широкого количества инструментов для разработки, и полностью устоявшихся подходов к проектированию контрактов. Но, тем не менее сообщество растет, а вместе с ним и база знаний и инструментов. Например, разработаны стандарты реализации определенных контактов. ERC20 – для создания валюты. Но как оказалось, этот стандарт не покрывает случаи, когда нужно создавать уникальные сущности. Из этих потребностей появился стандарт ERC-721.

Если посмотреть на инструменты, то тут тоже есть свои лидеры. Если требуются небольшие контракты, то можно использовать IDE в браузере Remix. Он предоставляет редактор, средство для дебагинга кода контрактов, а так их же анализ, что очень важно, так как цена ошибки, как говорилось ранее, может быть очень высокой.

Remix достаточно удобен, для прототипирования и анализа контрактов, но когда необходимо полноценное окружение, то в этом случае его недостаточно.

При создании смарт контрактов на платформе Ethereum разработчик закладывает определенную логику работы, определяющую как методы должны изменять состояние контракта, какие должны эмитироваться события,

когда и кому нужно произвести перевод средств, а когда бросить исключение. Инструменты отладки смарт контрактов еще не очень развиты, поэтому тесты зачастую становятся необходимым инструментом разработки, т.к. запускать контракты после каждого изменения может быть достаточно долгой процедурой. Также, в случае обнаружения ошибок, изменить код развернутого в сети контракта уже невозможно, можно только уничтожить контракт и создать новый, поэтому тестирование стоит проводить максимально подробно, особенно методы, связанные с платежами.

Общий принцип тестирования смартконтрактов схож с тестированием любого другого кода — создается набор эталонных вызовов методов в предопределенном окружении, для результатов которых прописываются утверждения. Для тестирования удобно использовать практики TDD – Test-driven development, которые гарантируют 100% покрытие кода. Суть заключается в том, что появляется три этапа написания кода:

a) написание теста;

b) написание кода, чтобы тест был пройден;

c) рефакторинг кода;

Стоит отметить то, что на все эти пункты должно уходить примерно 15 минут. Это значит, что мелкими шашками мы можем создавать большие и оттестированные контракты. Эта техника применима в любой разработке.

В настоящее время разработаны ряд фреймворков и библиотек тестирования смарт контрактов Ethereum, но я остановился на Truffle. Он предоставляет коробочное решение для тестирования, развертывания и компилирования смартконтрактов.

Для работы с Truffle необходимо иметь NodeJS (платформа для выполнения JavaScript на сервере) и NPM (пакетный менеджер в мире JavaScript). Структура проекта на Truffle весьма логичная, ключевыми моментами являются три папки – contracts, migrations и test. Разберем подробнее. В contracts у нас находится код на Solidity – контракты и библиотеки, в migrations скрипты, который позволяют автоматически

выкладывать контракты в блокчейн, и в test, непосредственно тесты наших контрактов. Хотелось бы отметить, что для тестирования контрактов Truffle не требуется выкладывать их сеть, а он сам разворачивает тестовую сеть блокчейна и взаимодействует. Это очень удобно и выгодно, так как в реальной сети, за каждую выкладку контракта необходимо платить. Тестовая сеть еще хороша тем, что операции в ней происходят мгновенно, в отличии от реальной сети. В дополнение, Truffle имеет набор готовых пакетов, для различного рода желаемый разработок – Boxes. Есть несколько от разработчиков Truffle – react, react-auth, react-uport, webpack. Но есть возможность использовать боксы сторонних разработчиков. Это позволяет сократить время на разворачивание и конфигурирование инфраструктуры и сконцентрировать на разработке смартконтрактов и Dapp.

Помимо инструментов, существует набор библиотек и смартконтрактов, на основе которых можно разрабатывать свои решения. Лидером в Etherium является openzeppelin. Он предоставляет уже готовые контракты различных стандартов. Так как контракты перенимают поведение классов в других языка, значит мы можем наследоваться от уже готовых контрактов и дорабатывать их функциональность от потребностей задачи. Это очень хороший подход, так как при написании с нуля, велика вероятность допустить ошибку.

Openzeppelin является open source проектом – это значит, что большое количество разработчиков учувствуют в контроле качества и корректности этой библиотеки.

Окружение есть, теперь можно углубится в особенности языка Solidity. Для людей, знающих JavaScript, Solidity покажется весьма знакомым, за исключением того, что Solidity типизируемый язык. Типов тут достаточно – целые числа (int, uint), булев тип, массивы, строки, структуры. Но стоит отметить, что нет float типа, что ведет за собой небольшое неудобство при работе с коэффициентами. Есть еще специфичный для Solidity тип, называемый address – это 20-байтное значение (размер адреса в Ethereum).

В начале каждого кода на Solidity необходимо указывать Ключевое слово pragma с версией компилятора, для которого писался этот контракт. Так как контракты могут выполнятся на разных видах станций, необходимо гарантировать, что код контракта интерпретируется так, как этого ожидал разработчик.

В отличии от обычного программного обеспечения, где цена операции рассчитывается из потребляемого электричества и оборудования, в сети Ethereum каждая операция имеет цену, и оплачивается в Gas. Gas – это внутренняя валюта сети Ethereum. Имеются конкретные цифры, сколько стоит вычисление, хранение.

Таблица 2.2 Стоимость операций в сети Ethereum

Операция	**Цена в GAS**
Сложение или вычитание двух чисел	3
Умножение или деление двух чисел	5
Сравнение двух чисел	3
Создание нового контракта	32000
Сохранение слова в 256 бит	20000
Отправка 1 транзакции	21000

Выкладка смартконтракта в основную сеть тоже является платной. Что хорошо – можно заранее просчитать стоимость выполнения программы. Но нюанс состоит в том, что мы будем знать цену в Gas, которая является очень динамичной в отношении к обычным деньгам. Из этого следует, что если в сегодня ваш smart contract требует 100 рублей для выкладывания в сеть, то завтра может потребоваться 110, а может и 90. Надо понимать эти риски, при взаимодействии с реальной сетью.

2.4 Dapp

Когда разработана вся логика контрактов, необходимо реализовать Dapp. Dapp – это decentralized application. Отличие от обычного приложения является в том, что в Dapp нет единого сервера, на котором выполняется код, а сервера распределены. Для реализации Dapp на выбор дается множество решений и платформ, так как это уже веб-технологии. Но все эти решения будет объединять библиотека Web3. Основная библиотека написана на языке JavaScript, но есть имплементации и для других языков. Web3 позволяет взаимодействовать с сетью Ethereum, вызывать методы смартконтрактов, запрашивать журнал выполнения контрактов. Название дано было не случайно, здесь идет ставка на то, что это треть поколение веб приложений, которое пришло на смену Web 2.0.

Для взаимодействия с контрактами web3 необходимо предварительно передать его в скомпилированном виде. Truffle предоставляет инструмент для этого.

Существует два подхода для реализации Dapp:

a) работа с блокчейн сетью напрямую, но в этом случае пользователю необходимо иметь специальное программное обеспечение. Как правило, это еще требует хранения всего блокчейна, а это 73 гигабайта на момент написания этой работы

b) работа через посредника, у которого уже развернуто программное обеспечение для работы с блокчейном.

В этой работе будет реализован первый вариант, так как при разработке Truffle уже предоставляет тестовую сеть, что значит, нам не требуется дополнительное программное обеспечение.

Для разработки Веб интерфейса есть стандартная библиотека для взаимодействия со смартконтрактами - web3. Название не случайно, здесь идет отсылка к тому, то на смену Web 2.0 приходит Web 3.0, который состоит из децентрализованных приложений. Для взаимодейтвия с Ethereum

необходимо задать адрес доступа, либо можно установить расширение для браузера – MetaMask. Помимо взаимодействия с приложением, этот плагин позволяет организовывать работу с Ethereum и агригировать несколькими учетными записями.

Но одного web3 недостаточно. В современном мире уже мало кто пишет на чистом JavaScript. Все большее предпочтение отдается различным библиотекам и фреймворкам, которые упрощают взаимодействие с DOM.

На данный момент существует большое колличество средств, но основными считаются React, Angular, Vue и Ember. У всех есть свои плюсы и минусы. Например Angular и Ember являются коробочным решением, где многие модули уже предоставлены разработчиками. Но Ember развивается достаточно медленно, а для Angular требуется знание Typescript, что может усложнить вход в проект для новых разработчиков. Vue представляет реактивный подход, но на достаточно больших проектах код на Vue становится сложно поддерживать. React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций. Он может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В классическом web приложении, разработчику необходимо следить за актуальностью данных в JavaScript и отображением в HTML, а React берет это на себя. Плюсом React является сформировавшееся экосистема и множество готовых решений. А React Native, позволяет на JavaScript писать нативные мобильные приложения.

В итоге сформировался следующий стек технологий. Ethereum для реализации смартконтрактов, web3 для взаимодействия со смартконтрактами и React, как библиотека для отображения информации в браузере и на мобильном приложении.

3 Проектная часть

3.1 Разработка программного продукта

Сегодня оформления полиса ОСАГО достаточно понятный процесс для обычного пользователя. Но это на первый взгляд. На сайте РСА есть страница (http://www.autoins.ru/ru/e-Polis/rsa-members_e-Polis_resistration/index.wbp), где указаны 55 страховщиков и ссылки на их сайты.

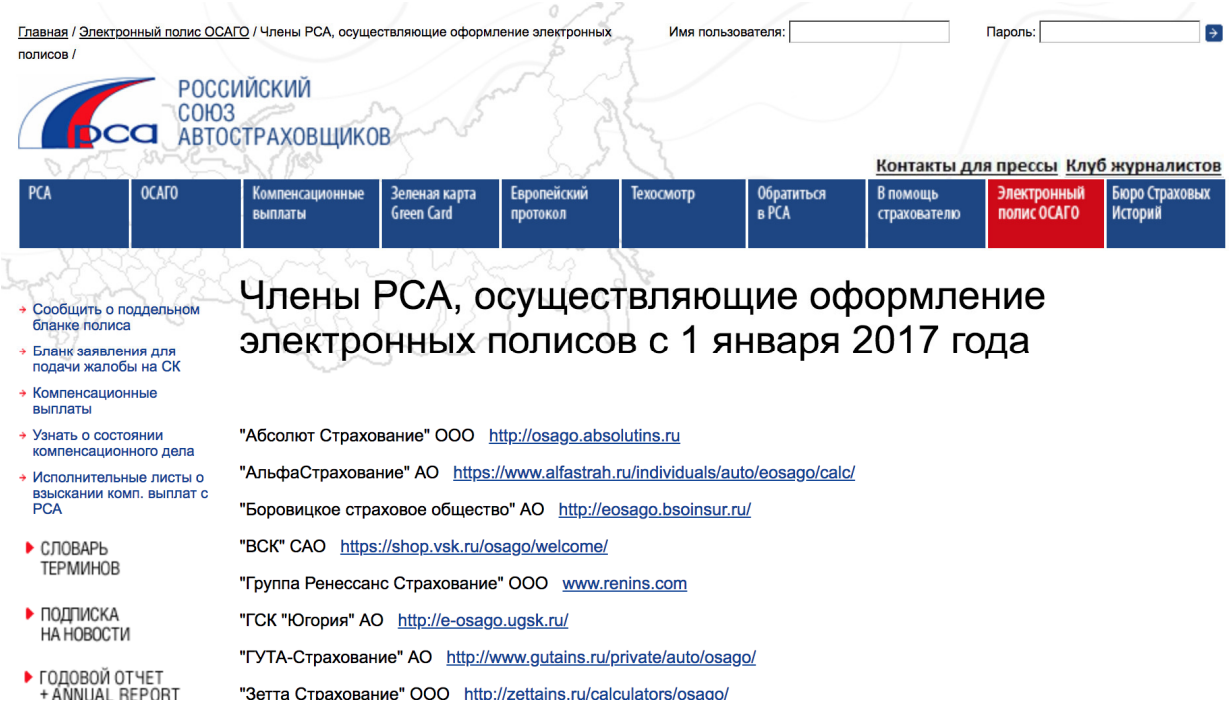

Рисунок 3.1 – страница со списком членов РСА

После выбора страховщика, на его сайте надо заполнить данные по автомобилю и паспортные данные страхователя. Если страховка оформляется на определенных водителей, то необходимо указать данные их водительских удостоверений.

Удобство на этом заканчивается. Данные приходится вводить вручную каждый раз, при оформлении нового полиса. При этом велика вероятность сделать ошибку в данных. А полис, оформленный с ошибкой, является не действительным, что потенциально опасно, так как выяснится это может после совершения ДТП. Ввиду этих ошибок не корректно рассчитается КБМ. КБМ

– коэффициент бонус-малус, зависящий от наличия или отсутствия страховых возмещений при наступлении страховых случаев, в период прошлых договоров ОСАГО. Иными словами, скидка за безаварийную езду. За каждый год безаварийного вождения страхователя, класс ОСАГО повышается. При наличии страховых случаев класс понижается. Проблема КБМ в том, что не многие про него знают, и страховая компания может его просто не учесть, либо указать меньше, чем есть на самом деле. Другая проблема в том, что на паспорт и на водительское удостоверение прикрепляется разный КБМ. Если пользователь оформлял два года неограниченную страховку на паспорт и после захочет оформить ограниченную на водительское удостоверение, то два года безаварийной истории не будут учтены.

В 2018 году появилось еще одно нововведение - если автовладелец или водитель ранее не страховался, то он не может через Интернет оформить полис.

При желании прекратить действие страхового полиса, необходимо оформить заявление и прийти в страховую компанию, либо отправлять его по почте.

При желании добавить нового водителя в страховой полис, необходимо так же писать заявление и доставить его до страховой компании.

Исходя из вышеперечисленных проблем, было решено реализовать оформление страхового полиса прозрачным и удобным с помощью технологии блокчейн на смартконтрактах Ethereum. Для разработки системы были учтены следующие требования:

– минимизировать возможность ошибки ввода пользовательских данных

– простота в использовании

– прозрачность процессов и возможность отследить все этапы

– возможность добавлять новых водителей «в один клик»

– возможность оперативной остановки полиса

Сначала стоит рассмотреть принцип оформления полиса. Для оформления необходимы документы: паспорт, диагностическая карта, свидетельство о регистрации авто или паспорт транспортного средства, водительское удостоверение, если требуется оформление ограниченной страховки. Ограниченная страховка – это страховка, которая распространяется на определенных водителей, которые заведомо известны до оформления. Но в период действия страхового полиса, можно добавлять водителей, в этот список.

Данные документы разделим на две группы – личные и те, которые относятся к автомобилю. Это нам нужно будет в дальнейшем, для разделения ответственности смартконтрактов.

По этим документам подсчитывается общая стоимость полиса. Стоимость рассчитывается перемножением девяти коэффициентов на базовую ставку.

Расчёт коэффициента по мощности происходит в зависимости от количества лошадиных сил транспортного средства.

Таблица 3.1 – расчеты коэффициента по мощности ТС

Количество лошадиных сил	Коэффициент
до 50 включительно	0.6
свыше 50 до 70 включительно	1.0
свыше 70 до 100 включительно	1.1
свыше 100 до 120 включительно	1.2
свыше 120 до 150 включительно	1.4
свыше 150	1.6

КБМ повышается или понижается в зависимости от аварийности в предыдущие периоды использования полиса ОСАГО. На сегодняшний день установлено 15 классов страхования водителей, предусматривающих применение соответствующих коэффициентов. Если полис оформляется

впервые, то КБМ равен 3. Более подробно система расчёта и перехода от одного уровня к другому представлены в рисунке 3.2.

Класс на начало годового срока страхования	Коэффициент	Класс по окончании годового срока страхования с учетом наличия страховых случаев, произошедших в период действия предыдущих договоров обязательного страхования				
		0 страховых выплат	1 страховая выплата	2 страховые выплаты	3 страховые выплаты	4 и более страховых выплат
М	2,45	0	М	М	М	М
0	2,3	1	М	М	М	М
1	1,55	2	М	М	М	М
2	1,4	3	1	М	М	М
3	1	4	1	М	М	М
4	0,95	5	2	1	М	М
5	0,9	6	3	1	М	М
6	0,85	7	4	2	М	М
7	0,8	8	4	2	М	М
8	0,75	9	5	2	М	М
9	0,7	10	5	2	1	М
10	0,65	11	6	3	1	М
11	0,6	12	6	3	1	М
12	0,55	13	6	3	1	М
13	0,5	13	7	3	1	М

Рисунок 3.2 – таблица расчета КБМ

Есть еще семь коэффициентов - по периоду использования, по сроку страхования, по наличию прицепа ТС, по территории использования, по возрасту и стажу (при оформлении ограниченного полиса), по допуску лиц к управлению (1.8 если неограниченный полис и 1 если ограниченный), по грубым нарушениям.

Базовая ставка варьируется в зависимости от страхующего лица (физическое или юридическое лицо) и от типа транспортного средства (легковой, грузовой и т.д.). Задает ставку Центральный банк России. Ставка является не точным числом, а страховая компания может выбрать точную сумму. Для физических лиц, которые владеют легковым автомобилем диапазон базовой ставки зачинается от 3432 руб. и заканчивается 4118 руб. включительно.

Коэффициенты, принцип расчёта и базовые ставки взяты с калькулятора стоимости ОСАГО РСА (http://www.autoins.ru/ru/osago/calculator/). Для наглядности была сформирована контекстная диаграмма выписывания полиса, приведенная на рисунке 3.1, декомпозиция изображена ниже.

Создав диаграмму IDEF0, которая будет описывать схему оформления полиса ОСАГО, получаем все зависимости, которые учувствуют в процессе.

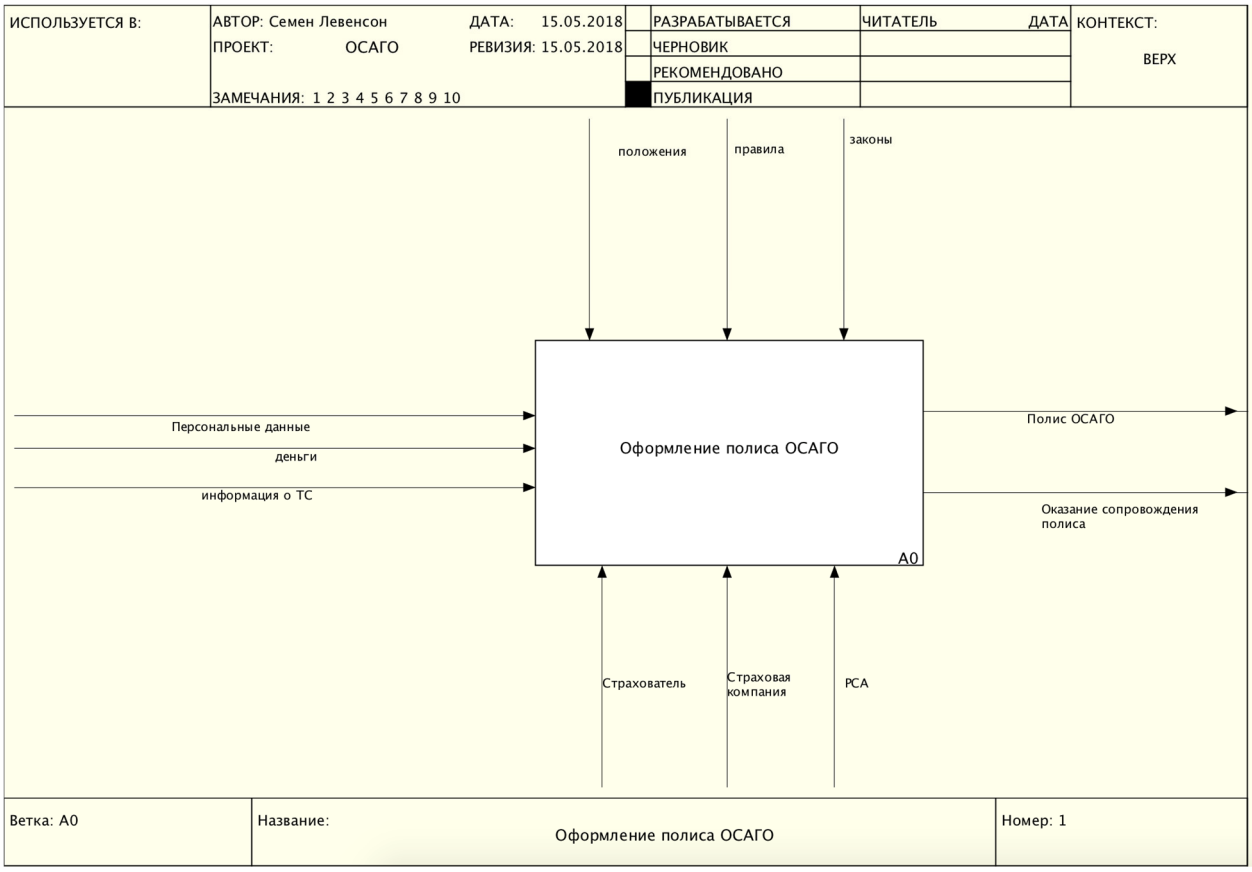

Рисунок 3.3 – контекстная диаграмма оформления полиса ОСАГО

Но контекстной диаграммы недостаточно для понимания цепочки бизнес-процессов. Схема, от того, как страхователь оформляет заявку и до получения полиса приведена на рисунке 3.4.

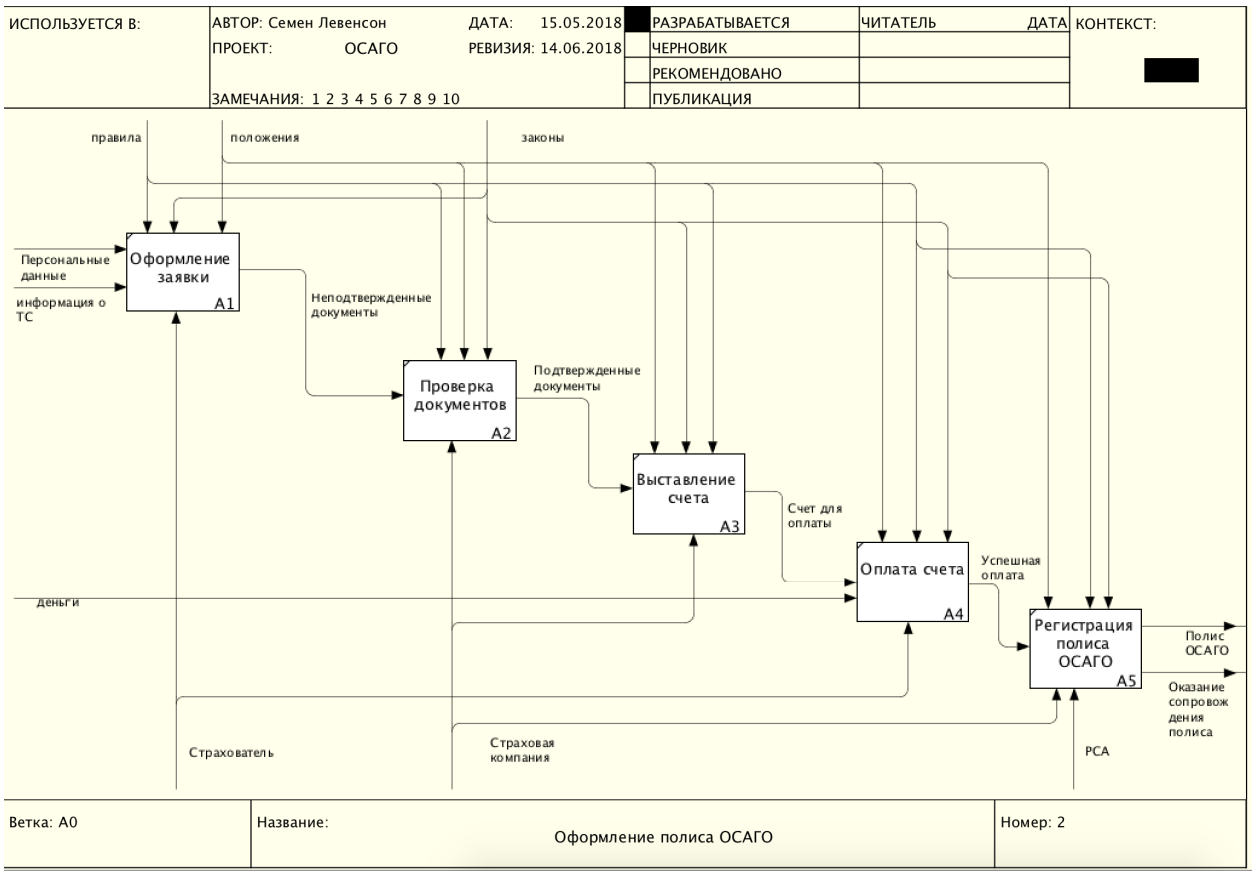

Рисунок 3.4 - Декомпозиция процесса оформления ОСАГО

Первое, что бросается в глаза, это этап проверки документов. На этом моменте возможно большое количество ошибок. Страховым компаниям нужно держать в штате сотрудников, которые будут проверять эти документы на истинность. Далее попробуем убрать этот этап, благодаря блокчейн технологиям. Так же можно автоматизировать все действия со стороны страховой компании – автоматическое выставление счета, исходя из заранее заполненных базовых ставок. Возможна автоматическая регистрация полиса с подписанием соответствующих документов и передачи данных о заключение полиса в РСА.

Рассмотрим еще процесс добавления нового водителя. Это происходит во время сопровождения полиса. Чтобы вписать в страховку нового водителя, страхователь (или его представитель по нотариально заверенной доверенности) должен приехать в офис компании с полисом и водительским удостоверением (либо его копией) человека, которого он собирается вписать.

Что уже не является удобным, так как оформление полиса производится удаленно, то ближайший офис страховой компании может быть в сотнях километров от страхователя. В новой системе решено сделать так, чтобы визит в офис был не обязательным.

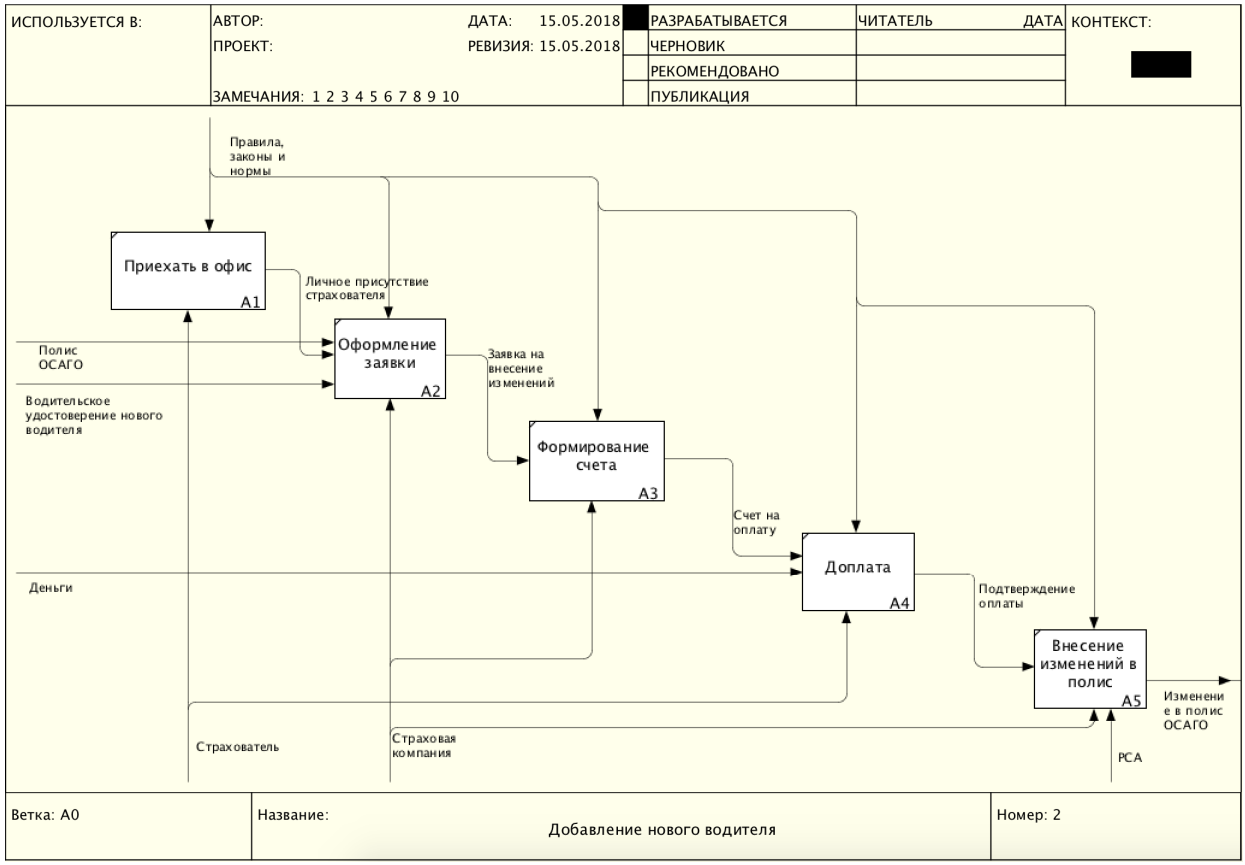

Рисунок 3.5 – декомпозиция процесса добавления нового водителя

Как видим, есть этапы, которые можно оптимизировать и сократить операционные издержки страховой компании и время с нервами страхователя. Но прежде, чем начать решать проблемы текущей системы страхования, необходимо создать базовые сущности новой системы. Ключевые участники остаются те же – страховщики, страхователи, выгодоприобретатели, Российский союз автостраховщиков. Все эти участники и их составляющие будут представлены в виде смартконтрактов.

Смартконтракт РСА - этот контракт является сертификационным центром и местом, где будет хранится история страхования. Эта история нужна для дальнейшего корректного расчёта КБМ. Контракт должен

выпускать смартконтракты для страховой компании (лицензии), владелец лицензии и название указывается при создании. Адреса этих лицензий он должен хранить, для проверки лицензий на подлинность. Еще он будет аккумулировать резерв гарантий (2 % от страховой премии) и резерв компенсационных выплат (1 % от страховой премии). Код смартконтрактра приведен ниже:

```
pragma solidity ^0.4.23;
import "./Company.sol";
contract RUMI {
  address public owner;
  uint public warrantyReserve;
  uint public reserveForCompensationPayments;
  mapping(address => bool) public companies;

  event AddCompany(address newCompany);
  event AddInsurance(address indexed _passport, address indexed
_insurance);

  modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this
function.");
    _;
  }
  modifier onlyRegistredCompany (address companyAddress) {
    require(companies[companyAddress]);
    Company company = Company(companyAddress);
    require(company.owner() == msg.sender);
    _;
  }
  constructor () public {
    owner = msg.sender;
  }
  function addCompany(address companyOwner, string nameCompany)
onlyOwner public {
    Company company = new Company(companyOwner, nameCompany,
this);
    companies[company] = true;
    emit AddCompany(company);
  }
  function addInsurance(
    address passport,
    address insurance,
    address companyAddress,
    uint _warrantyReserve,
    uint _reserveForCompensationPayments
  ) onlyRegistredCompany(companyAddress) public {
    warrantyReserve += _warrantyReserve;
    reserveForCompensationPayments                                  +=
_reserveForCompensationPayments;
    emit AddInsurance(passport, insurance);
  }
}
```

Смартконтракт страховой компании - должен создавать смартконтракты страховых полисов. Во время выпуска лицензий будет применятся шаблон проектирования «Фабрика». Фрагмент кода приведен ниже.

```
function createInsurance (
    address _vehicle,
    address _passport,
    string _className,
    uint _basePrice
  ) onlyOwner public {
    Insurance insurance = new Insurance(
      _vehicle,
      _passport,
      _className,
      _basePrice
    );
    insurances[insurance] = true;
    emit CreateInsurance(insurance);
  }
```

Это значит, на основе входящих значений у нас будет создавать новая сущность. После создания полиса, смартконтракт будет сообщать об этом смартконтракту РСА и оставлять у себя 77% (нетто-премия) от стоимости полиса и 20% (расходы на осуществление страхования).

Смартконтракт страховки является ключевым в нашей системе. Он имеет цену страховки, которая рассчитывается в момент создания страховки. Адрес компании, которая создала (выдала) полис хранится в этом контракте, чтобы при работе с полисом можно было сразу понять, кто его выдал. Так же хранится адрес смартконтракта ПТС, на который выдана страховка. Имеет адрес смартконтракта паспорта, на который выдана страховка. Имеет динамическое поле статуса активации, при запросе этого поля, смартконтракт должен смотреть текущее время и сравнивать его с датой начала активации и конца, если текущее время за рамками временного отрезка начала и конца, то должен возвращать false, иначе true. Расчёт цены происходит исходя из коэффициентов выше. Мощность берется из смартконтракта ПТС. Класс страховщика берется исходя из истории оформлений в смартконтракте РСА. Расчеты коэффициентов вынесены в библиотеки. Пример библиотеки для расчета коэффициента, исходя из мощности приведен ниже.

```
pragma solidity ^0.4.23;
```

```
library KM {
  function getKM(uint _power) internal pure returns (uint) {
    if (_power <= 50 * 100) {
      return 0.6 * 100;
    } else if (_power <= 70 * 100) {
      return 1 * 100;
    } else if (_power <= 100 * 100) {
      return 1.1 * 100;
    } else if (_power <= 120 * 100) {
      return 1.2 * 100;
    } else if (_power <= 150 * 100) {
      return 1.4 * 100;
    }
    return 1.6 * 100;
  }
}
```

Так же имеются смартконтракты идентифицирующие – паспорт транспортного средства, паспорт гражданина Российской Федерации и водительское удостоверение. Ожидается, что у пользователя они уже есть. Их выдача лежит за рамками данной работы, так как затрагивает паспортный стол, ГИБДД и т.д. Но тестовые контракты все же необходимо создать, чтобы система могла функционировать. Под тестовыми имеется ввиду то, что функционал у них будет ограничен.

Паспорт имеет поля для имени, фамилии, отчества. Для коэффициента по территории использования транспортного средства еще необходимо адрес прописки владельца. Для коэффициента по допуску лиц необходимо хранить дату рождения, так как для лиц, младше 22 лет применяется повышенный коэффициент.

Водительское удостоверение имеет категории допуска к управлению транспортными средствами, дату выдачу и адрес предыдущих прав, если такие имеются. Это так же необходимо для расчёта коэффициента по допуску лиц, так как, если водительский стаж менее 3 лет применяется повышенный коэффициент.

В ПТС будем хранить мощность транспортного средства, для вычисления коэффициента. Добавим еще дополнительную информацию как идентификационный номер автомобиля и его название. По условию

страхования, у транспортного средства должна быть диагностическая карта, которая свидетельствует о исправности автомобиля. Для данной работы нам будет достаточна временная метка, которая свидетельствует о крайнем моменте, когда автомобиль проходил технический осмотр. Так как автомобиль является движущим имуществом, то должна быть возможность передачи собственности. Реализуем это добавив функцию, которую может вызывать только текущий владелец, а сама сделка будет хранится в истории смартконтракта. Код контракта приведен ниже.

```
pragma solidity ^0.4.23;
contract Vehicle {
  string public name;
  string public vin;
  uint public power;
  uint public lastTI;
  address public owner;

  modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
  }
  event TransferOwnership(address from, address to);

  constructor (
    string _vin,
    uint _power,
    string _name
  ) public {
    name = _name;
    vin = _vin;
    power = _power;
    owner = msg.sender;
  }

  function updateLastTI(uint _time) public {
    lastTI = _time;
  }

  // Передача авто
  function transferOwnership(address newOwner) onlyOwner public {
    address oldOwner = owner;
    owner = newOwner;
    emit TransferOwnership(oldOwner, newOwner);
  }
}
```

Мы получаем систему страхования, которая полностью находится в рамках блокчейна и не подвержена человеческому фактору, так как правила в смартконтрактах четко запрограммированы.

Но во время разработки смартконтрактов могут быть допущены ошибки. Для решения этой проблемы, разработка велась по методологии TDD. Это когда сначала пишутся тесты, а после код, чтобы он удовлетворял требования этих тестов. Тесты можно писать на Solidity, но они ограничены в функционале. Поэтому тесты писались на языке JavaScript. Тестировать необходимо весь функционал смартконтракта, так как ошибка может обойтись очень дорого. Пример тестов для смартконтракта ПТС выглядит следующим образом.

```
const Vehicle = artifacts.require('./Vehicle.sol');

contract('Vehicle', accounts => {
  const _power = Math.ceil(162.2 * 100);
  const _vin = '4USBT53544LT26841';
  const _name = 'Mondeo';
  let vehicle;
  const mainOwner = accounts[0];
  beforeEach(async function() {
    vehicle = await Vehicle.new(_vin, _power, _name, {
      from: mainOwner
    });
  });
  it('should has a name', async () => {
    const name = await vehicle.name.call();
    assert.equal(name, _name);
  });
  it('should has a vin', async () => {
    const vin = await vehicle.vin.call();
    assert.equal(vin, _vin);
  });
  it('should has a power', async () => {
    const power = await vehicle.power.call();
    assert.equal(power, _power);
  });
  it('should has a owner', async () => {
    const owner = await vehicle.owner.call();
    assert.equal(owner, mainOwner);
  });
  it('should has a lastTI', async () => {
    const lastTI = (await vehicle.lastTI.call()).toNumber();
    assert.equal(lastTI, 0);
  });
  it('should update lastTI', async () => {
    const time = Math.floor(new Date().getTime() / 1000);
    await vehicle.updateLastTI(time);
```

```
    const lastTI = (await vehicle.lastTI.call()).toNumber();
    assert.equal(lastTI, time);
  });
  it('should transfer vehicle', async () => {
    const futureOwner = accounts[1];
    const { logs } = await vehicle.transferOwnership(futureOwner);
    const owner = await vehicle.owner.call();
    const eventArgs = logs[0].args;

    assert.equal(eventArgs.from, mainOwner);
    assert.equal(eventArgs.to, futureOwner);
    assert.equal(owner, futureOwner);
  });
});
```

Другие фрагменты листинга программы вынесены в приложении А.

Результат тестов мы можем увидеть в консоли разработчика. Когда все тесты имеют зеленную галочку сбоку, то это означает, что все тесты пройдены успешно. Однако, если тест не будет пройден, то вместо зеленной галочки, будет красный крестик. Вывод в консоли успешно пройденный тестов приведен ниже.

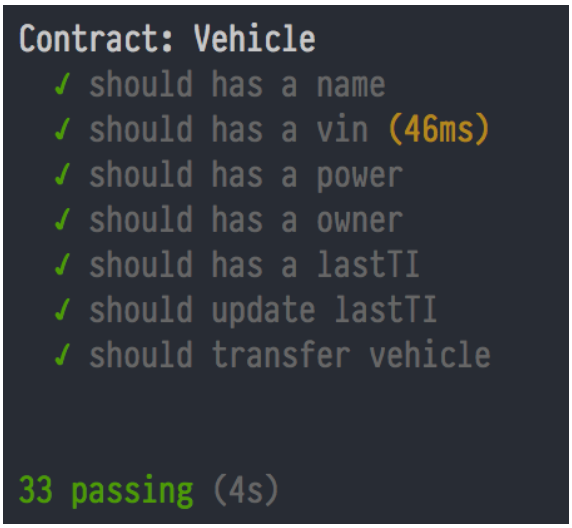

Рисунок 3.6 – пройденные тесты

Стоит отметить, что некоторые ошибки отслеживаются на этапе компиляции, так как Solididy является компилируемым языком. Ошибки типов или вызов необъявленных функций он берет на себя. Ниже показан вывод процесса компиляции.

Рисунок 3.7 – компиляция смартконтрактов

После сборки всей системы мы можем посмотреть интерфейсную часть с роли страхователя. В начале просим страхователя указать адрес своего паспорта. Далее, эту информацию мы можем не спрашивать из года в год. После ему необходимо оформить заявку на оформление полиса.

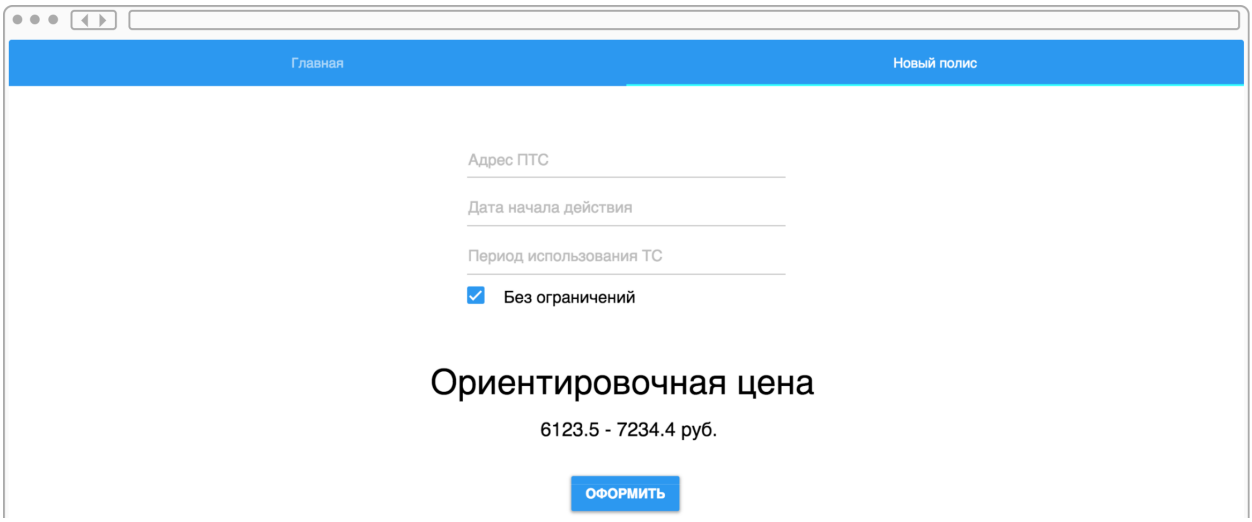

Рисунок 3.8 – экран оформления нового полиса

Пользователю больше не нужно вносить все характеристики автомобиля, так как вся эта информация заложена в смартконтракте ПТС. Дата начала по умолчанию проставляется текущей датой, а период использования 10 и более месяц, что является коэффициентом 1. Цена выводится в диапазоне, на основе введённых данных, так как базовую ставку в конечном итоге

выставляет страховая компания. Территориальный коэффициент берется из смартконтракта паспорта владельца ТС. КБМ рассчитывается автоматически, основываясь на истории из смартконтракта РСА. Если страхователь уберет галочку с «Без ограничений», то ему будут доступны поля ввода адресов водительских удостоверений. Опять же, КБМ будет вычисляется на основании всех водителей и браться наименьший. Если у транспортного средства не был пройден технический осмотр или какой-то из водителей не имеет права вождения данной категорией автомобиля, то страхователю будет показа ошибка и заявка не будет оформлена. Ограничение для страхователей, которые впервые оформляют полис ОСАГО также снято ввиду того, что достоверность документов уже подтверждены в блокчейне, а КБМ по умолчанию задается 3, согласно правилам оформления.

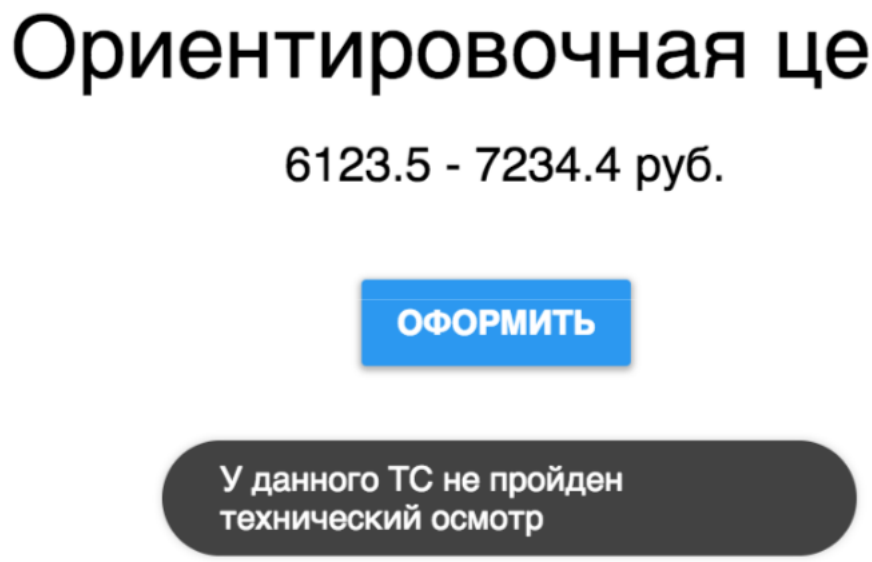

Рисунок 3.9 – пример ошибки, при оформлении заявки

После успешной отправки заявки, система выдает конечную стоимость и кнопку для оплаты страхового полиса. Сразу, после зачисления денег в страховую компанию полис становится действительным с указанной ранее желаемой даты начала действия полиса. Посмотрим на диаграмму процесса оформления полиса, после внедрения блокчейн технологий.

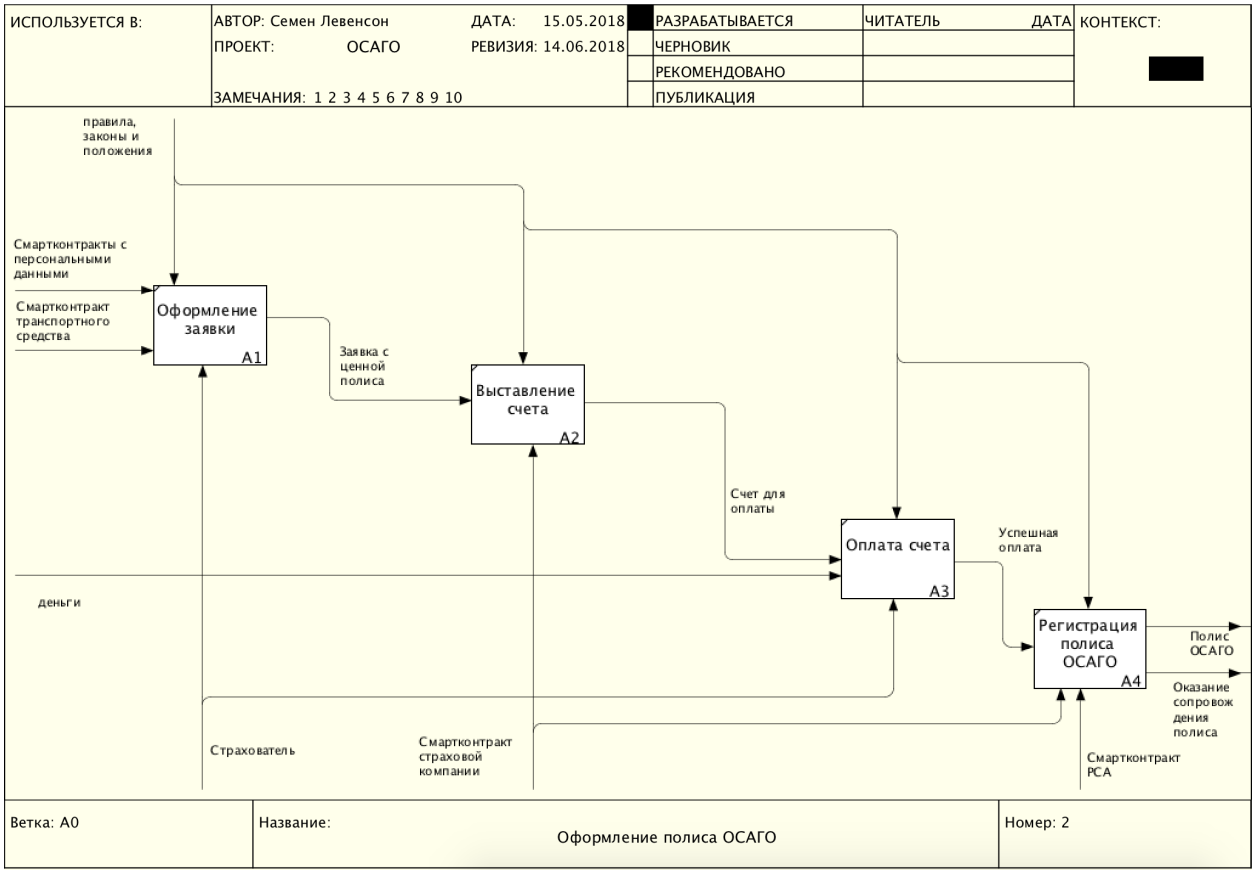

Рисунок 3.10 – декомпозиция процесса оформления полиса в новой системе

Ввиду того, что мы работаем не с сырыми данными, а с смартконтрактами, то их существование уже гарантирует их истинность, так как в блокчейне невозможно и даже не целесообразно что-либо изменять. Как можно увидеть, мы минимизировали действия в цепочке, а страховая компания во всем этом процессе учувствует в автоматическом режиме.

После того, как полис оплачен, он отобразится в общем списке полисов страхователя. В нем отображаются все полиса ранее оформленные страхователем. Список полисов показан на рисунке 3.11.

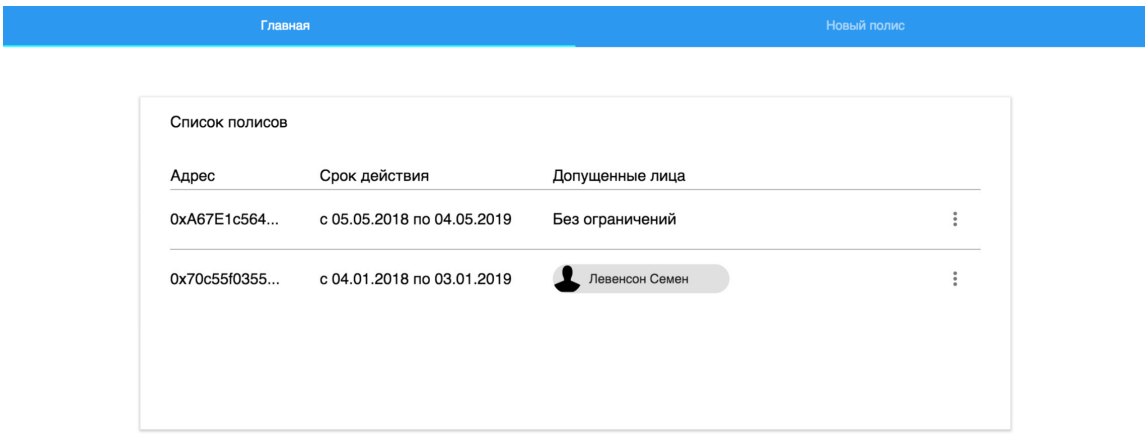

Рисунок 3.11 – список полисов

В списке страхователь может ознакомится с оформленными полисами ОСАГО. Но кроме оформления полиса необходимо обеспечить дальнейшее взаимодействие с ним. В текущей системе реализованы функции продления действия полиса, добавления водителя и прекращение действия полиса.

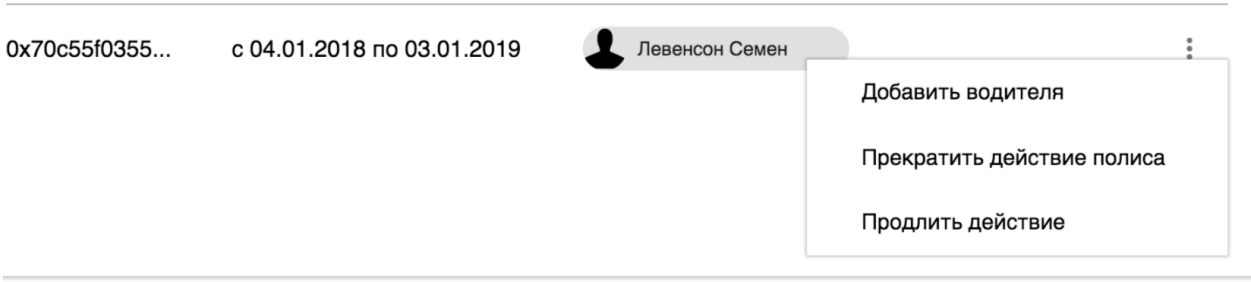

Рисунок 3.12 – дополнительные функции

Продление полиса возможно в том случае, если действие полиса оканчивается не ранее, чем через две недели от текущей даты. В это случае создается автоматически создается заявка и выставляется счет на оплату. После оплаты полис считается продленным.

Прекращение действия полиса теперь возможно удаленно. Нет необходимости ехать в офис и писать заявление или отправлять его по почте. При обработке прекращения действия полиса страхователю сразу же производится возврат денежных средств с учетом использованного времени действия полиса. В случае, если тот платеж, которым была произведена оплата

уже не действителен, например страхователь уже не обслуживается в банке, с счета которого была произведена оплата, деньги остаются на счету системы, и он сможет расплатится ими в счет нового полиса в будущем либо же вывести удобным методом.

Если к страхователю приехали родственники, но они не включены в список допущенных к вождению лиц, то в этом случае ему предоставляется возможность оперативно, в «один клик» добавить нужного водителя в список. Прежде чем добавить нового водителя, необходимо проверить, имеет ли он право на вождение данного транспортного средства? Если имеет, то необходимо узнать, не является ли его КБМ меньше тех, которые уже есть в страховом полисе. В случае, если КБМ нового водителя меньше, то мы сразу предлагаем страхователю доплатить разницу в цене, так как мы уже знаем все коэффициенты и базовую ставку. А если КБМ не ниже имеющихся, то просто добавляем нового водителя в полис. Данный процесс отразим в схеме ниже. Можно заметить, что данный процесс не подразумевает присутствия в офисе страховой компании, а также нет необходимости в этом процессе учувствовать сотрудникам компании, так как весь процесс выполняется автоматически. Это несет удобство для страхователя, так как исключаются транспортные издержки, а страховым компаниям можно сократить штат сотрудников. Тем самым получается ситуация, когда выигрывают обе стороны. Была составлена схема процесса добавления нового водителя в действующий полис обязательного страхования автомобильного транспорта, она представлена на рисунке 3.13. По ней видно, что есть проверка водителя на допуск к данному транспорту, путем сравнения категории допуска в водительском удостоверении и типом транспорта. Далее, возможна ситуация, что добавление нового водителя не повлечет за собой доплаты. Это зависит от класса нового водителя, если он ниже имеющихся, то необходима будет доплата.

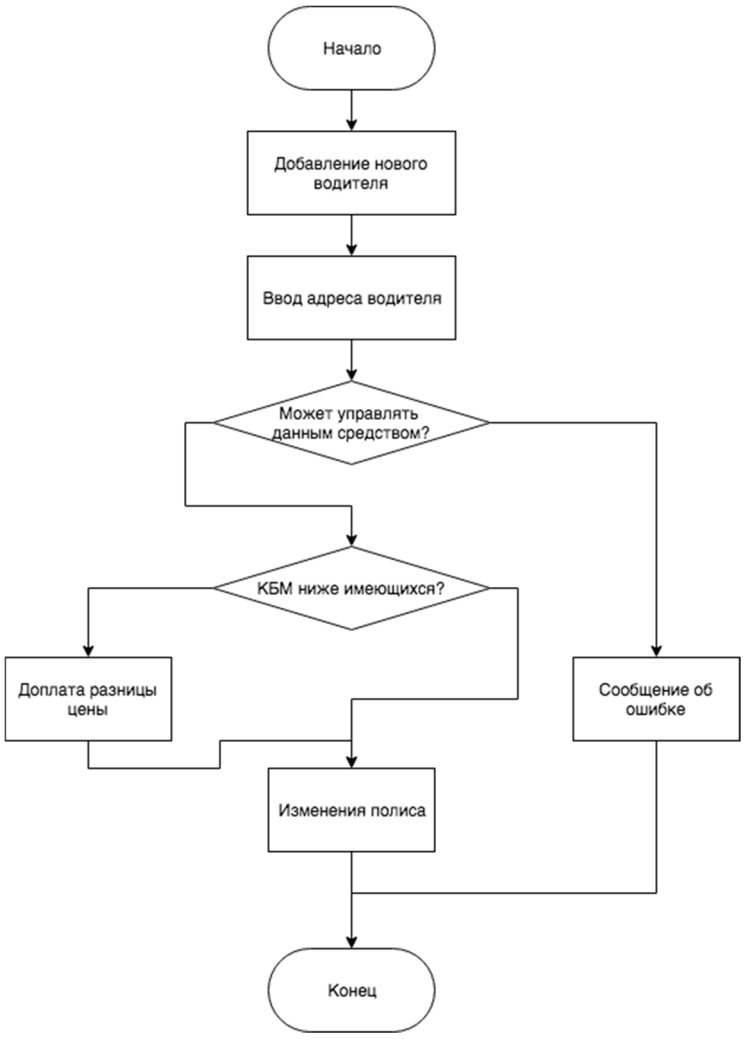

Рисунок 3.13 – процесс добавления нового водителя

В дополнение к сайту, было решено сделать мобильное приложение, которое не только повторяет функциональность сайта, но и дополняет. При выборе полиса предоставляется QR-код, который можно сканировать любым устройством. QR-код - это товарный знак, изначально разработанный для автомобильной промышленности Японии. Штрих код — считываемая машиной оптическая метка, содержащая информацию об объекте, к которому она привязана. QR-код использует четыре стандартизированных режима кодирования для эффективного хранения данных; могут также использоваться расширения. Это будет полезно при проверке документов на посту ДПС. Нет необходимости возить с собой бумажный вариант полиса, а сотруднику набирать номер полиса вручную. Дополнительной функцией является возможность видео фиксации автомобиля. Эта функция может понадобится,

например, при свершении дорожно-транспортного происшествия. Видеопоток будет напрямую транслироваться в систему, гарантируя тем самым отсутствие монтажа записи и сохранение временных меток. А с развитием нейронных сетей и дополненной реальности, в будущем можно будет автоматизировать оформление евро протокола, так как сегодня эта процедура довольно сложная и не всегда можно понять, какая сумма потребуется для устранения повреждений.

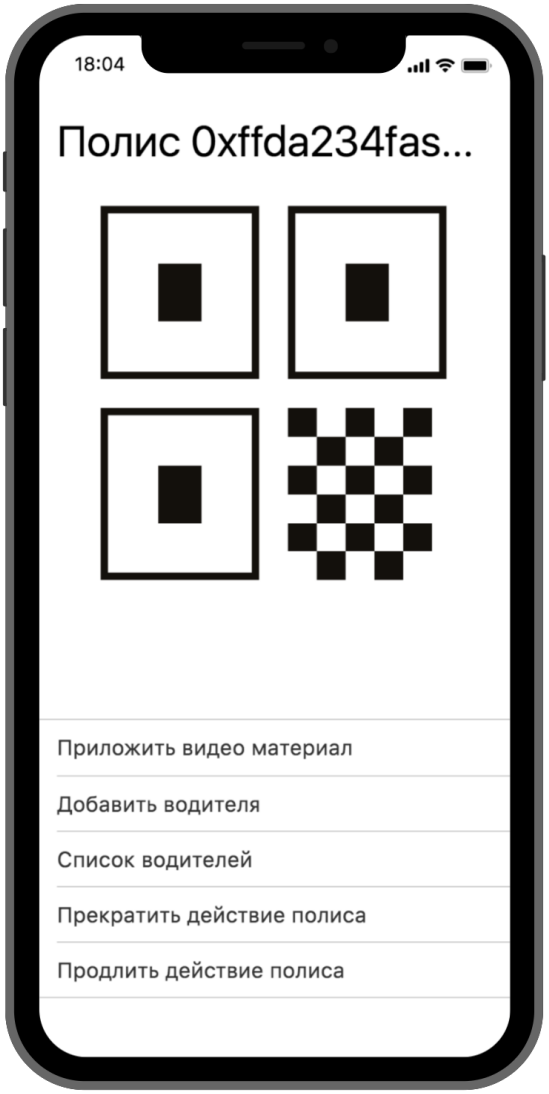

Рисунок 3.14 – экран полиса в мобильном приложении

На этом этапе, завершается демонстрация разработки и можно сделать вывод, что разработанная программа, успешно работает и выполняет все поставленные задачи.

С точки зрения страховой компании тоже стало лучше. Им теперь не требуется подтверждать достоверность данных, которые передают страхователи, а следовательно, нет необходимости привлекать к этому рабочую силу.

Так как мы работаем с распределенным реестром, перед нами открывается большая возможность коммуникаций между всеми государственными органами. Например, когда страхователь добавил нового водителя в страховой полис. Новый водитель сразу может сесть за руль, и если вдруг случится страховой случай, то все инстанции будут осведомлены о том, что расширение действия полиса произошло успешно и своевременно.

Оформление и работа с полисом ОСАГО у нас носит обязательный характер, то с точки зрения государства необходимо реализовать максимально удобный инструмент, которым будет пользоваться подавляющее большинство населения. Новая система является простой и удобной в использовании – начиная от оформления полиса и заканчивая добавлением новых водителей. Дизайн выполнен в минималистичном виде, чтобы не отвлекать пользователя от основных функций. Стоит отметить, что данная система может быть расширена до работы с полисами КАСКО. В этом случае потребуются доработки ввиду вариативности полисов. При оформлении полисов КАСКО, необходим личный визит в офис или же выезд сотрудника компании для оценки состояния авто. А уже реализованная в новой системе, функция видео фиксации позволит упросить эту процедуру и тоже сделать все удаленно. Модульная архитектура проекта позволит совершить это с минимальными усилиями.

3.2 Технико-экономическая часть программного продукта

Система, разработанная в рамках данной работы призвана снизить затраты труда, ввиду автоматизации бизнес-процессов страховых компаний. На данный момент, с каждого страхового полиса резервируется 20% на

расходы выпуска полиса ОСАГО. С применением этой системы эти средства можно применить для маркетинговых целей или же снизить стоимость на полис. С точки зрения потребителя, у него экономится время, так как многие операции можно производить удаленно. Косвенно экономятся и деньги потребителя, так как исключаются транспортные расходы. Дополнительно исключается вероятность оформления страхового полиса по недостоверным данным, так как все данные не вводятся вручную, а находятся в смартконтрактах.

В рамках ВКР разработано специфическое программное обеспечение для государственного и муниципального сектора, связанное с автоматизацией социальных проектов, где экономическая выгода от внедрения программного обеспечения не очевидна и не предполагается дальнейшее распространение данного программного продукта на рынке программного обеспечения. Ввиду этого решено провести SWOT-анализ сильных и слабых сторон разработанного программного продукта. Название данного метода анализа представляет собой аббревиатуру английских слов Strength (сила), Weakness (слабость), Opportunities (возможности), Threats (угрозы).

Схема SWOT-анализа включает следующие этапы:

– Выявление: сильных сторон разработки; слабых сторон разработки; возможностей разработки; угроз;

– Формирование и анализ SWOT-матрицы;

– Выработка заключения о перспективности разработки.

Сильные стороны:

– Децентрализация. система устойчива к внешним факторам, неработоспособность одного из участников сети не приведет к ликвидации всей системы.

– Удобство использования в новой системе. Многие процессы жизненного цикла полиса ОСАГО оптимизированы, что привело к меньшему количеству требуемых действий.

– Открытость информации обо всех транзакциях, что полезно как для контролирующих функций, так и для получения статистики

– Низкая стоимость обслуживания. Так как система в основном функционирует в сети Ethereum, то нам нет необходимости иметь сервера для обслуживания смартконтрактов, а расходы на их выполнение можно заложить в имеющиеся 20% стоимости полиса.

– Масштабируемость. Сеть Ethereum является глобальной и не привязана к конкретному государству.

Слабые стороны:

– Высокая стоимость разработки ввиду того, что разработка децентрализованных приложений является достаточно новой областью. Квалифицированных специалистов мало, а спрос большой.

– Масштабность требуемых изменений. Данную систему нельзя внедрить только в сектор страхования. Для нормального функционирования системы требуются изменения в смежных секторах.

Возможности:

– Интеграция с другими системами. Самое близкое к теме ОСАГО, это полис КАСКО. Его также можно будет перевести на блокчейн технологии.

– Полная автономность системы. В будущем возможно исключение человека из этой цепочки. Например, когда водитель только садится за руль автомобиля, на него сразу оформляется полис. А если происходит ДТП, на основании датчиков в авто или же с помощью автономных сканеров будет происходить оценка ущерба и производится выплаты.

– Глобализация. Так как Ethereum является глобальным, то и разработанная система является таковой. В перспективе возможно создание единого полиса ОСАГО, который будет действовать по всему миру.

Угрозы:

– Все децентрализованные системы имеют проблему 51%. Это когда злоумышленник обладает большей частью вычислительных мощностей сети и в этом случае он может мешать ее работе. Но стоит отметить, что это никак не отразится на уже внесенных данных в блокчейн, а только лишь будет препятствовать созданию новых транзакций.

– Недостаточная производительность сети. Так как сеть является глобальной и количество участников не предсказуемо, возможны ситуации, когда производительности сети будет недостаточно для обработки транзакций.

Для анализа полученных данных составим SWOT – матрицу:

	Возможности			Угрозы		
Сильные стороны	Интеграция с другими системами	Глобализация	Автономность	Атака 51%	Недостаточная производительность сети	Итого
1. Децентрализация	2	2	2	1	0	7
2. Удобство использования	2	2	2	-1	-1	4
3. Открытость информации	2	2	1	0	0	5
4. Низкая стоимость обслуживания	2	2	2	0	-2	4
Итого	8	8	7	0	-3	20
Слабые стороны						0
1. Высокая стоимость разработки	-1	0	-1	0	0	-2
2. Масштабность требуемых изменений	-1	-2	-2	-1	-1	-7
Итого	-2	-2	-3	-1	-1	-9
Общий итог	6	6	4	-1	-4	11

Рисунок 4.1 – SWOT-матрица

Проанализировав полученную SWOT-матрицу, можно сделать следующие выводы:

– Трудно выделить самые важные сильные стороны, они все в достаточной степени важны, но, по-видимому, наиболее важным

достоинством разработки является децентрализация. В дальнейшем необходимо обращать особое внимание на обеспечение и расширение этой стороны разработки;

– Наибольшей слабой стороной является масштабность требуемых изменений. Но это возможно решить продвижением возможностью системы.

– Из рассмотренных возможностей стоит выделить интеграцию с другими системами и секторами. Это позволит системе более гибко работать и увеличивать функциональность.

– Наиболее опасной угрозой является недостаточная производительность сети. Так как это влечет за собой не только медленное выполнение транзакций, но и увеличением стоимости транзакций. Но стоит отметить, что работа в этом направлении уже ведется, переходом от модели Proof of Work к Proof of Stake.

– Заключение о перспективности разработки. Разработка является перспективной и имеет широкий диапазон для направлений развитий.

– В ближайшее время стоит сконцентрироваться на интеграции с другими системами, чтобы уменьшить влияние масштабных изменений.

В итоге делаем вывод, что разработка является перспективной и достаточно масштабной. Есть определенные точки роста и интеграции со сторонними системами, а угрозы минимальны. Например, атаку 51% можно решить, воспользовавшись не главной сетью Ethereum, а запустив свою, но тем самым подымится стоимость поддержки.

ЗАКЛЮЧЕНИЕ

Обозначенные во введении работы задачи были выполнены. Было изучено определение и характеристики блокчейна, исследован принцип оформления страхового полиса ОСАГО, проанализировано текущее состояния автострахования и применение блокчейна в нем. Благодаря разработанной системе была достигнута цель работы – оптимизированы текущие бизнес-процессы при оформлении и поддержке полиса ОСАГО через автоматизацию валидации входящих запросов от страхователя, для снижения издержек у страховых компаний и привнесения большего удобства для страхователей.

В целом технология блокчейн в современном виде вполне подходит для изучения и исследования страховыми компаниями. Однако ее полноценная эксплуатация еще далека. Это связано с тем, что блокчейн является распределенной системой, и поэтому его ценность в большей степени зависит от эффективного сотрудничества игроков с конкурентами, поставщиками и другими сторонами. По сути, блокчейн-это инвестиция в информационные технологии с перспективой полной реализации всех ее преимуществ. В некоторых приложениях, которые не требуют распределенных механизмов blockchain так много, есть альтернативные решения, которые могут обеспечить аналогичные преимущества гораздо быстрее. С другой стороны, даже при отсутствии эталонных примеров внедрения технологии блокчейн открывает уникальные возможности для тех игроков отрасли, которые смогут продуктивно участвовать в обмене данными с другими стейкхолдерами с целью улучшения выявления мошенничества или автоматизации процесса рассмотрения претензий.

Для отрасли в целом это означает, что пришло время создать консорциум с технологическими экспертами, стартапами, регуляторами и другими участниками рынка. Целью этой Ассоциации должно быть выявление

проблем, которые блокчейн и его открытый и децентрализованный характер ставит перед страховой индустрией. К числу других проблем относятся технологические ограничения, а также рыночные, правовые, нормативные (которые будут регулироваться в отсутствие посредника или границ между рынками) и оперативные требования, касающиеся, например, защиты и стандартизации данных.

Индивидуальные страховые компании должны начать с целостного взгляда на реальные потребности клиентов и их собственные уязвимости. Это позволит оценить, когда применение новых технологий может принести им реальную пользу. Как инновационная технология, блокчейн представляет угрозу для многих игроков рынка, так как создает возможности для новых революционных бизнес-моделей и / или значительного снижения операционных издержек для тех компаний, которые смогут успешно добиться его практического применения. Тем не менее, существует ряд возможностей для устранения этой угрозы.

Блокчейн-это технология цифровой трансформации, представляющая стратегический интерес для игроков страхового рынка. Наиболее серьезными препятствиями на пути широкого использования технологий в страховой отрасли являются необходимость поощрения сотрудничества между участниками рынка и технологическими лидерами, необходимость успешных оперативных реформ и создание благоприятной нормативно-правовой базы. Закладка фундамента для решения этих задач сегодня позволит страховым компаниям реализовать полномасштабные сценарии собственного практического применения технологии и извлечь выгоду из ее преимуществ в течение ближайших пяти лет.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 «Business innovation through blockchain», Vincenzo Morabito, Springer International Publishing, 2017.

2 «Proof-of-Work» Proves Not to Work, Ben Laurie, Richard Clayton, 2004.

3 Scoping SIG, Tokenization Taskforce PCI Security Standards Council. Info Supplement:PCI DSS Tokenization Guidelines, 2011.

4 Finance and Economics Discussion SeriesDivisions of Research & Statistics and Monetary AffairsFederal Reserve Board, Washington, D.C., 2017 [https://ru.scribd.com/document/333308566/Distributed-ledger-technology-in-payments-clearing-and-settlement#from_embed?content=10079&ad_group=Online+Tracking+Link&campaign=Skimbit%2C+Ltd.&keyword=ft500noi&source=impactradius&medium=affiliate&irgwc=1].

5 «The future of financial infrastructure», 2016 [https://bravenewcoin.com/assets/Industry-Reports-2016/WEF-The-future-of-financial-infrastructure.pdf].

6 «The future of financial infrastructure», 2016 [https://bravenewcoin.com/assets/Industry-Reports-2016/WEF-The-future-of-financial-infrastructure.pdf].

7 «On Public and Private Blockchains» [https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/].

8 Блог компании Bitfury [http://bitfury.com/blog].

9 «Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto», 2009 [https://bitcoin.org/bitcoin.pdf].

10 «Форум «Блокчейн и открытые платформы», 2016 [https://www.slideshare.net/Bankir_Ru/block-chain-19042016v2].

11 «Проблема перебора», А.Шень, 2015 [https://postnauka.ru/faq/43795].

12 «С чем это едят»: что такое блокчейн [https://habrahabr.ru/company/bitfury/blog/326340/].

13 «Банки адаптируют блокчейн быстрее, чем можно было бы предположить» [https://habrahabr.ru/company/ibm/blog/312862/].

14 «М.Видео» и «Сбербанк» провели пилотный проект с использованием технологии блокчейн [http://forklog.com/m-video-i-sberbank-proveli-pilotnyj-proekt-s-ispolzovaniem-tehnologii-blokchejn/].

15 «Как понять нужно ли интегрировать blockchain в ваш продукт?» [https://habrahabr.ru/company/web_payment_ru/blog/301972/].

16 «Сбербанк» в I квартале 2017 года запустит документооборот на основе blockchain [http://www.rbc.ru/finances/19/01/2017/587de3d49a7947533915ad51].

17 «SWIFT and Accenture outline path to Distributed Ledger Technology adoption within financial services», 2016 [https://www.swift.com/insights/press-releases/swift-and-accenture-outline-path-to-distributed-ledger-technology-adoption-within-financial-services].

18 Biella M, Zinetti V Blockchain technology and applications from a financial perspective. Unicredit technical report, 2016 [http://www.the-blockchain.com/docs/UNICREDIT%20-%20Blockchain-Technology-and-Applications-from-a-Financial-Perspective.pdf].

19 «Токены vs Пароли» [https://habrahabr.ru/post/126828/].

20 «Использование блокчейн-технологии в банковской системе. Взгляд лидера рынка» [https://www.youtube.com/watch?v=JuMancRkj0A].

21 «Как банки будут развивать блокчейн-решения в 2017 году» [https://geektimes.ru/company/wirex/blog/284556/].

22 «Distributed Ledger Technology: What We Can Learn from Recent Blockchain Attacks» [http://www.securitylab.ru/bitrix/exturl.php?goto=https://www.greenwich.com/blog/distributed-ledger-technology-what-we-can-learn-recent-blockchain-attacks].

23 «Лекция 12: Экранирование, анализ защищенности» [http://www.intuit.ru/studies/courses/10/10/lecture/318].

ПРИЛОЖЕНИЕ А

Смартконтракты

Company.sol

```
pragma solidity ^0.4.23;

import "./RUMI.sol";
import "./Insurance.sol";

contract Company {
  address public owner;
  string public name;
  address public RUMIAddress;
  uint public netPremium;
  uint public issuanceCosts;
  mapping(address => bool) public insurances;

  event CreateInsurance(address _insurance);
  event ActivateInsutance(
    address _insurance,
    uint _warrantyReserve,
    uint _reserveForCompensationPayments,
    uint _issuanceCosts,
    uint _netPremium
  );
  modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
  }

  constructor (address _owner, string _name, address _RUMIAddress)
public {
    owner = _owner;
    name = _name;
    RUMIAddress = _RUMIAddress;
  }

  function createInsurance (
    address _vehicle,
    address _passport,
    string _className,
    uint _basePrice
  ) onlyOwner public {
    Insurance insurance = new Insurance(_vehicle, _passport, _className,
_basePrice);
    insurances[insurance] = true;
    emit CreateInsurance(insurance);
  }

  function activateInsurance (address _insuranceAddress) onlyOwner
public {
    require(insurances[_insuranceAddress]);
    RUMI rumi = RUMI(RUMIAddress);
    Insurance insurance = Insurance(_insuranceAddress);

    insurance.activate();
```

```
    uint price = insurance.price();

    uint warrantyReserve = price / 100;
    uint reserveForCompensationPayments = (price * 2) / 100;

    uint _issuanceCosts = (price * 20) / 100;
    uint _netPremium = (price * 77) / 100;

    rumi.addInsurance(
      insurance.passportAddress(),
      _insuranceAddress,
      this,
      warrantyReserve,
      reserveForCompensationPayments
    );


    issuanceCosts += _issuanceCosts;
    netPremium += _netPremium;

    emit ActivateInsutance(
      _insuranceAddress,
      warrantyReserve,
      reserveForCompensationPayments,
      _issuanceCosts,
      _netPremium
    );
  }
}
```

Insurance.sol

```
pragma solidity ^0.4.23;

import "./Vehicle.sol";
import "./KBM.sol";
import "./KM.sol";

contract Insurance {
  address public owner; // Адрес Страховой компании
  address public vehicleAddress; // Адрес ПТС
  address public passportAddress; // Адрес паспорта
  uint public basePrice; // базовая ставка
  uint public price;
  string public className;

  uint public endDate;
  uint public startDate;

  // Коэффициенты
  uint public kbm; // по классу скидки
  uint public km; // по мощности двигателя
  uint public constant ko = 180; // по допуску лиц к управлению

  constructor (
    address _vehicleAddress,
    address _passportAddress,
    string _className,
    uint _basePrice
```

```
  ) public {
    require(_basePrice > 3432 * 1000000 && _basePrice < 4118 * 1000000,
"Базовая ставка в рамках");

    require(KBM.hasClassName(_className));
    owner = msg.sender;
    vehicleAddress = _vehicleAddress;
    passportAddress = _passportAddress;
    basePrice = _basePrice;
    className = _className;

    kbm = KBM.getKBM(_className);

    Vehicle vehicle = Vehicle(vehicleAddress);
    uint power = vehicle.power();
    // Коэффициент по мощности
    km = KM.getKM(power);

    price = (_basePrice * kbm * ko * km) / 100**3;
  }

  function activate () public {
    require(msg.sender == owner);
    // solium-disable security/no-block-members
    startDate = now;
    endDate = now + 365 days;
  }

  function isActive () public view returns(bool) {
    return now >= startDate && now <= endDate;
  }
}
```

KBM.sol

```
pragma solidity ^0.4.23;

library KBM {
  function getKBM(string _className) internal pure returns (uint) {
    bytes32 kbmHash = keccak256(abi.encodePacked(_className));
    if (kbmHash == keccak256("M")) {
      return 2.45*100;
    } else if (kbmHash == keccak256("0")) {
      return 2.3*100;
    } else if (kbmHash == keccak256("1")) {
      return 1.55*100;
    } else if (kbmHash == keccak256("2")) {
      return 1.4*100;
    } else if (kbmHash == keccak256("3")) {
      return 1*100;
    } else if (kbmHash == keccak256("4")) {
      return 0.95*100;
    } else if (kbmHash == keccak256("5")) {
      return 0.9*100;
    } else if (kbmHash == keccak256("6")) {
      return 0.85*100;
    } else if (kbmHash == keccak256("7")) {
      return 0.8*100;
```

```
    } else if (kbmHash == keccak256("8")) {
      return 0.75*100;
    } else if (kbmHash == keccak256("9")) {
      return 0.7*100;
    } else if (kbmHash == keccak256("10")) {
      return 0.65*100;
    } else if (kbmHash == keccak256("11")) {
      return 0.6*100;
    } else if (kbmHash == keccak256("12")) {
      return 0.55*100;
    } else if (kbmHash == keccak256("13")) {
      return 0.5*100;
    }

    return 0;
  }
  function hasClassName(string _className) internal pure returns (bool)
{
    return getKBM(_className) != 0;
  }

}
```

KM.sol

```
pragma solidity ^0.4.23;

library KM {
  function getKM(uint _power) internal pure returns (uint) {
    if (_power <= 50 * 100) {
      return 0.6 * 100;
    } else if (_power <= 70 * 100) {
      return 1 * 100;
    } else if (_power <= 100 * 100) {
      return 1.1 * 100;
    } else if (_power <= 120 * 100) {
      return 1.2 * 100;
    } else if (_power <= 150 * 100) {
      return 1.4 * 100;
    }
    return 1.6 * 100;
  }
}
```

Passport.sol

```
pragma solidity ^0.4.23;


contract Passport {
  string public firstName;
  string public lastName;
  string public middleName;
  uint public dateOfBirth;
  address public owner;
```

```
  constructor (
    string _firstName,
    string _lastName,
    string _middleName,
    uint _dateOfBirth
  ) public {
    firstName = _firstName;
    lastName = _lastName;
    middleName = _middleName;
    dateOfBirth = _dateOfBirth;
    owner = msg.sender;
  }
}
```

RUMI.sol

```
pragma solidity ^0.4.23;

import "./Company.sol";

contract RUMI {
  address public owner;
  uint public warrantyReserve;
  uint public reserveForCompensationPayments;
  mapping(address => bool) public companies;

  event AddCompany(address newCompany);
  event AddInsurance(address indexed _passport, address indexed
_insurance);

  modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
  }

  modifier onlyRegistredCompany (address companyAddress) {
    require(companies[companyAddress]);
    Company company = Company(companyAddress);
    require(company.owner() == msg.sender);
    _;
  }

  constructor () public {
    owner = msg.sender;
  }

  // Добавляем страховую компанию в реестр
  function addCompany(address companyOwner, string nameCompany)
onlyOwner public {
    Company company = new Company(companyOwner, nameCompany, this);
    companies[company] = true;
    emit AddCompany(company);
```

```
  }

  function addInsurance(
    address passport,
    address insurance,
    address companyAddress,
    uint _warrantyReserve,
    uint _reserveForCompensationPayments
  ) onlyRegistredCompany(companyAddress) public {
    warrantyReserve += _warrantyReserve;
    reserveForCompensationPayments += _reserveForCompensationPayments;
    emit AddInsurance(passport, insurance);
  }
}
```

Vehicle.sol

```
pragma solidity ^0.4.23;


contract Vehicle {
  string public name;
  string public vin;
  uint public power;
  uint public lastTI;
  address public owner;


  modifier onlyOwner {
    require(msg.sender == owner, "Only owner can call this function.");
    _;
  }
  event TransferOwnership(address from, address to);


  constructor (
    string _vin,
    uint _power,
    string _name
  ) public {
    name = _name;
    vin = _vin;
    power = _power;
    owner = msg.sender;
  }


  function updateLastTI(uint _time) public {
    lastTI = _time;
```

```
  }

  // Передача авто
  function transferOwnership(address newOwner) onlyOwner public {
    address oldOwner = owner;
    owner = newOwner;
    emit TransferOwnership(oldOwner, newOwner);
  }
}
```

Тесты

Company.js

```
const Company = artifacts.require('./Company.sol');
const Vehicle = artifacts.require('./Vehicle.sol');
const RUMIMock = artifacts.require('./RUMIMock.sol');

async function getVehicleAddress(mainOwner) {
  const _power = Math.ceil(162.2 * 100);
  const _vin = '4USBT53544LT26841';
  const _name = 'Mondeo';

  const vehicle = await Vehicle.new(_vin, _power, _name, {
    from: mainOwner
  });
  return vehicle.address;
}

async function getRUMIAddress(mainOwner) {
  const rumi = await RUMIMock.new({
    from: mainOwner
  });
  return rumi.address;
}

contract('Company', accounts => {
  let company;

  const _name = 'Ингострах';
  let _RUMI;
  const _owner = accounts[1];
  beforeEach(async function() {
    _RUMI = await getRUMIAddress(accounts[3]);

    company = await Company.new(_owner, _name, _RUMI, {
      from: accounts[0]
    });
  });
  it('should has a name', async () => {
```

```
  const name = await company.name.call();
  assert.equal(name, _name);
});
it('should has a netPremium', async () => {
  const netPremium = await company.netPremium.call();
  assert.equal(netPremium, 0);
});
it('should has a owner', async () => {
  const owner = await company.owner.call();
  assert.equal(owner, _owner);
});

it('should create insurance', async () => {
  const vehicleAddress = await getVehicleAddress(accounts[3]);
  const transaction = await company.createInsurance(
    vehicleAddress,
    accounts[3],
    '3',
    4000000000,
    {
      from: _owner
    }
  );

  const insurance = transaction.logs[0].args._insurance;

  const hasInMapper = await company.insurances.call(insurance);
  assert.equal(hasInMapper, true);
});

it('should active insurance', async () => {
  const vehicleAddress = await getVehicleAddress(accounts[3]);
  const transaction = await company.createInsurance(
    vehicleAddress,
    accounts[3],
    '3',
    4000000000,
    {
      from: _owner
    }
  );
  const insurance = transaction.logs[0].args._insurance;
  const { logs } = await company.activateInsurance(insurance, {
    from: _owner
  });

  assert.equal(logs[0].args._insurance, insurance);
  assert.equal(logs[0].args._warrantyReserve.toNumber(), 115200000);
  assert.equal(
    logs[0].args._reserveForCompensationPayments.toNumber(),
    230400000
  );
  assert.equal(logs[0].args._issuanceCosts.toNumber(), 2304000000);
  assert.equal(logs[0].args._netPremium.toNumber(), 8870400000);

  const netPremium = await company.netPremium.call();
  const issuanceCosts = await company.issuanceCosts.call();
```

```
    assert.equal(issuanceCosts.toNumber(), 2304000000);
    assert.equal(netPremium.toNumber(), 8870400000);
  });
});
```

Insurance.js

```
const Insurance = artifacts.require('./Insurance.sol');
const Vehicle = artifacts.require('./Vehicle.sol');

async function getVehicleAddress(mainOwner) {
  const _power = Math.ceil(162.2 * 100);
  const _vin = '4USBT53544LT26841';
  const _name = 'Mondeo';

  const vehicle = await Vehicle.new(_vin, _power, _name, {
    from: mainOwner
  });
  return vehicle.address;
}
contract('Insurance', accounts => {
  let insurance;
  let _className = '3';
  let _basePrice = 4000 * 1000000;
  let _vehicleAddress;
  let _passportAddress = accounts[2];
  let _companyAddress = accounts[0];
  beforeEach(async function() {
    _vehicleAddress = await getVehicleAddress(accounts[3]);
    insurance = await Insurance.new(
      _vehicleAddress,
      _passportAddress,
      _className,
      _basePrice,
      {
        from: _companyAddress
      }
    );
  });
  it('should has a vehicleAddress', async () => {
    const vehicleAddress = await insurance.vehicleAddress.call();
    assert.equal(vehicleAddress, _vehicleAddress);
  });
  it('should has a passportAddress', async () => {
    const passportAddress = await insurance.passportAddress.call();
    assert.equal(passportAddress, _passportAddress);
  });
  it('should has a base price', async () => {
    const basePrice = await insurance.basePrice.call();
    assert.equal(basePrice, _basePrice);
  });
  it('should has a className', async () => {
    const className = await insurance.className.call();
    assert.equal(className, _className);
  });
  it('should has a kbm', async () => {
    const kbm = (await insurance.kbm.call()).toNumber();
    assert.equal(kbm, 100);
```

```
  });
  it('should has a km', async () => {
    const km = (await insurance.km.call()).toNumber();
    assert.equal(km, 160);
  });
  it('should has a price', async () => {
    const price = (await insurance.price.call()).toNumber();
    assert.equal(price, 6400 * 1000000 * 1.8);
  });
  it('should has not active status', async () => {
    const isActive = await insurance.isActive.call();
    assert.equal(isActive, false);
  });
  it('should change activate status', async () => {
    await insurance.activate({
      from: _companyAddress
    });
    const isActive = await insurance.isActive.call();
    assert.equal(isActive, true);
  });
});
```

KBM.js

```
const KBMMock = artifacts.require('./KBMMock.sol');
contract('KBM', accounts => {
  let kbmUtils;
  const classNameList = [
    'M',
    ...Array.apply(null, { length: 14 }).map(String.call, String)
  ];
  before(async function() {
    kbmUtils = await KBMMock.new();
  });

  it('should has all class name from list', async () => {
    for (let className of classNameList) {
      const kbm = (await kbmUtils.getKBM.call(className)).toNumber();
      assert.isAbove(kbm, 0);
    }
  });
  it('should return zero when existant class name', async () => {
    const kbm = (await kbmUtils.getKBM.call('S')).toNumber();
    assert.equal(kbm, 0);
  });

  it('should return false when existant class name ', async () => {
    const hasClassName = await kbmUtils.hasClassName.call('S');
    assert.equal(hasClassName, false);
  });
  it('should return true when existing class name ', async () => {
    const              hasClassName              =              await
kbmUtils.hasClassName.call(classNameList[0]);
    assert.equal(hasClassName, true);
  });
```

```
});
```

Passport.js

```
const Passport = artifacts.require('./Passport.sol');
contract('Passport', accounts => {
  let passport;
  const _firstName = 'Левенсон';
  const _lastName = 'Семен';
  const _middleName = 'Яковлевич';
  const _dateOfBirth = Date.now();
  const _owner = accounts[0];
  beforeEach(async function() {
    passport = await Passport.new(
      _firstName,
      _lastName,
      _middleName,
      _dateOfBirth,
      {
        from: _owner
      }
    );
  });
  it('should has a firstName', async () => {
    const firstName = await passport.firstName.call();
    assert.equal(firstName, _firstName);
  });
  it('should has a lastName', async () => {
    const lastName = await passport.lastName.call();
    assert.equal(lastName, _lastName);
  });
  it('should has a middleName', async () => {
    const middleName = await passport.middleName.call();
    assert.equal(middleName, _middleName);
  });
  it('should has a dateOfBirth', async () => {
    const dateOfBirth = await passport.dateOfBirth.call();
    assert.equal(dateOfBirth, _dateOfBirth);
  });
  it('should has a owner', async () => {
    const owner = await passport.owner.call();
    assert.equal(owner, _owner);
  });
});
```

RUMI.js

```
const RUMI = artifacts.require('./RUMI.sol');
contract('RUMI', accounts => {
  let rumi;
  const _owner = accounts[0];
  beforeEach(async function() {
    rumi = await RUMI.new({
      from: _owner
    });
  });
  it('should normal initialization', async () => {
```

```
    const owner = await rumi.owner.call();
    const warrantyReserve = await rumi.warrantyReserve.call();
    const reserveForCompensationPayments = await
rumi.reserveForCompensationPayments.call();
    assert.equal(owner, _owner);
    assert.equal(warrantyReserve, 0);
    assert.equal(reserveForCompensationPayments, 0);
  });

  it('should add insurance company', async () => {
    const { logs } = await rumi.addCompany(accounts[1], 'Ингострах');
    assert.equal(logs.length, 1, 'should be 1 event');
    let eventArgs = logs[0].args;
    const hasCompany = await rumi.companies.call(eventArgs.newCompany);
    assert.equal(hasCompany, true);
  });
  it('should add insurance from company', async () => {
    const companyOwner = accounts[1];
    const { logs: addCompanyLog } = await rumi.addCompany(
      companyOwner,
      'Ингострах'
    );

    const _passport = accounts[2];
    const _insurance = accounts[3];

    const { logs: addInsuranceLog } = await rumi.addInsurance(
      _passport,
      _insurance,
      addCompanyLog[0].args.newCompany,
      1,
      1,
      {
        from: companyOwner
      }
    );
    let eventAddIncurance = addInsuranceLog[0].args;
    const warrantyReserve = await rumi.warrantyReserve.call();
    const reserveForCompensationPayments = await
rumi.reserveForCompensationPayments.call();
    assert.equal(warrantyReserve, 1);
    assert.equal(reserveForCompensationPayments, 1);
    assert.equal(eventAddIncurance._passport, _passport);
    assert.equal(eventAddIncurance._insurance, _insurance);
  });
});
```

Vehicle.js

```
const Vehicle = artifacts.require('./Vehicle.sol');
contract('Vehicle', accounts => {
  const _power = Math.ceil(162.2 * 100);
  const _vin = '4USBT53544LT26841';
  const _name = 'Mondeo';
  let vehicle;
  const mainOwner = accounts[0];
  beforeEach(async function() {
    vehicle = await Vehicle.new(_vin, _power, _name, {
      from: mainOwner
```

```
    });
  });
  it('should has a name', async () => {
    const name = await vehicle.name.call();
    assert.equal(name, _name);
  });
  it('should has a vin', async () => {
    const vin = await vehicle.vin.call();
    assert.equal(vin, _vin);
  });
  it('should has a power', async () => {
    const power = await vehicle.power.call();
    assert.equal(power, _power);
  });
  it('should has a owner', async () => {
    const owner = await vehicle.owner.call();
    assert.equal(owner, mainOwner);
  });
  it('should has a lastTI', async () => {
    const lastTI = (await vehicle.lastTI.call()).toNumber();
    assert.equal(lastTI, 0);
  });
  it('should update lastTI', async () => {
    const time = Math.floor(new Date().getTime() / 1000);
    await vehicle.updateLastTI(time);
    const lastTI = (await vehicle.lastTI.call()).toNumber();
    assert.equal(lastTI, time);
  });
  it('should transfer vehicle', async () => {
    const futureOwner = accounts[1];
    const { logs } = await vehicle.transferOwnership(futureOwner);
    const owner = await vehicle.owner.call();
    const eventArgs = logs[0].args;

    assert.equal(eventArgs.from, mainOwner);
    assert.equal(eventArgs.to, futureOwner);
    assert.equal(owner, futureOwner);
  });
});
```

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
1. 11070026.09.03.02.113.ПЗВКР	Пояснительная записка	76 с.
	Графические документы	
2. 11070026.09.03.02.113.ДМВКР	Демонстрационные материалы (презентация)	15 сл.
	Демонстрационные материалы (пл. ф. А4)	15 лист.*5 экз
	Другие документы	
3.	Документы на компакт-диске	1 CD

Изм.	Лист.	Номер докум.	Подп.	Дата	11070026.09.03.02.113.ПЗВКР			
Разработал		Левенсон С.Я			Тема ВКР Система подтверждения и выдачи страховых полисов для автомобильного транспорта	Лит.	Лист	Листов
Проверил						У	75	76
Н.контр.		Нестерова Е.В.				НИУ «БелГУ» гр.07001353		
Утвердил								

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

_____ _____

(подпись) (Ф.И.О.)