ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет Компьютерных наук
Департамент Программной инженерии

СОГЛАСОВАНО
Профессор департамента
программной инженерии факультета
компьютерных наук канд. техн. наук
_____/Авдошин С.М.
«__» _____2018 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
_____/Шилов В.В.
«__» _____2018 г.

ДЕЦЕНТРАЛИЗОВАННАЯ БИРЖА КРИПТОВАЛЮТ

Исполнитель
Студент группы БПИ 141 НИУ ВШЭ
_____/Кондрашов А. А
«___»_____2018 г.

2018

Аннотация

На сегодняшний день блокчейн является невероятно популярной технологией. Однако к бизнесу, приносящему прибыль, в данной сфере можно отнести лишь криптовалютную торговлю и предоставление платформ для ее осуществления. Существует множество сервисов, реализующих данные идеи, но большая их часть имеет различные недостатки, такие как централизованность и медлительность, не обеспечивающие пользователю адекватный уровень удобства. Те же, кто лишен данных изъянов, не предоставляют мобильной версии приложений. Мое решение устраняет все вышеперечисленные проблемы.

Ключевые слова — блокчейн, криптовалютная биржа, криптотокен, криптовалюта, 0x протокол.

Abstract

There is much hype around blockchain and cryptocurrency today. The only known profitable use of it is trading cryptocurrency and providing traders with a place for that. There are many solutions that provide exchange services but most of them are either centralized or too slow to be convenient for the customer. Those that lack these negative features do not have a mobile version of applications. This paper proposes an approach that grants enough speed and decentralization.

Keywords — blockchain, crypto exchange, crypto token, cryptocurrency, 0x protocol

СОДЕРЖАНИЕ

Определения

Блокчейн - постоянно растущий список записей, называемых блоками, которые связаны и защищены с помощью криптографии. Он копируется его пользователями и устойчив к модификации. Машина с рабочей копией называется узлом.

Смарт-контракт - компьютерный протокол, предназначенный для цифровой поддержки, проверки или обеспечения выполнения исполнения контракта. Смарт-контракты позволяют выполнять надежные транзакции без третьих сторон. Эти транзакции являются отслеживаемыми и необратимыми [1].

Ethereum - открытая, общедоступная, распределенная вычислительная платформа на основе технологии блокчейн и операционная система с функциональностью смарт-контрактов [2].

Токен - криптовалюта, основанная на смарт-контрактах Ethereum.

ERC20 - стандарт токена описывает функции и события, которые должен выполнить контракт токена Ethereum [3].

Протокол 0x - протокол, определяющий частично децентрализованный способ обмена токенами, совместимыми с ERC20, созданными первыми членами сообщества проекта 0x [4].

Биржа криптовалют - бизнес и / или приложение, которое позволяет клиенту торговать криптовалютами.

SDK (Software Development Kit) - часть программного обеспечения, которая может быть интегрирована в приложение для облегчения разработки определенной компьютерной системы в ее различных проявлениях.

Реле - промежуточная точка между клиентом и блочной цепью. Его использование позволяет ускорить взаимодействие с пользователем.

Легкий клиент - версия узла цепочки блоков, которая не содержит полный список записей, а только те, которые необходимы для его полезной работы. Он обычно используется для мобильных устройств

Газ – условная единица измерения вычислительных затрат, покупаемая за криптовалюту Ether без фиксированной цены, используемая для оплаты майнинга, поддерживающего работу платформы Ethereum.

Введение

«Без обмена Биткойн и все другие криптовалюты были бы рынком без ликвидности» [5]. Цель моего проекта - создать децентрализованное приложение обмена, которое можно использовать на смартфонах. Основное отличие от других решений заключается в том, что он не будет использовать только цепочку блоков, что приводит к очень низкой скорости транзакций, и не будет использовать взаимодействие только через реле, что приводит к централизации. Он объединяет в себе обе практики для достижения лучшего пользовательского опыта и независимости от регуляторов. Несмотря на то, что существует множество подобных решений, они упускают вышеупомянутые свойства, а в сфере блокчейна по-прежнему нет подходящих и надежных решений. Более того, мои планы заключаются в том, чтобы сделать его бесплатным или только самообеспечивающим. Моя личная цель - не прибыль, а эксперимент с блокчейном на мобильных устройствах.

1. Обзор источников и решений

В данной главе описаны возможные векторы создания криптовалютной биржи с приведением примеров на основе существующих приложений. Все существующие решения можно разделить на централизованные, децентрализованные и частично децентрализованные.

1.1. Централизованные решения

На данный момент эти решения являются наиболее прибыльными конкурентами. Это потому, что они используют те же старые технологии, которые были в использовании и отлаживались уже более двух десятилетий. Тем не менее, они могут устареть в будущем, так как их децентрализованные конкуренты все больше продвигаются к достижению аналогичного уровня скорости и безопасности, предоставляя при это преимущества децентрализации.

1.1.1. Веб-приложения

Этот тип включает GDAX [6] Coinbase, Gemini [7], Changelly [8], Cryptopia [9] и мириады других. Все они почти одинаковы с технической точки зрения. Они могут быть дифференцированы по пользовательскому интерфейсу, цене и функциональности, но технология везде одинаковая.

1.1.2. Чат-боты Телеграм

Наиболее популярной является группа ботов, созданных одной компанией. Они могут обменять 6 криптовалют на фиатные деньги. Ограниченное количество монет включает Bitcoin, Ether, T, Dogcoin, Bitcoin Cash и Litecoin [10, 11, 12, 13, 14, 15]. Все указанные валюты существуют на разных блокчейнах. Вот почему их сложно обменять при помощи смарт-контрактов.

1.1.3. Мобильные приложения

Все мобильные биржи, которые существуют, используют только централизованный способ связи. Ни один из них не пишет прямо в блок-цепочку, даже если основное решение децентрализовано. Причиной этого является отсутствие легких клиентов даже для самых популярных блокчейн-приложений.

1.2. Децентрализованные решения

У всех указанных решений либо нет мобильных клиентов, либо они являются децентрализованными

1.2.1. BitShares

Высокопроизводительная децентрализованная платформа блокчеин со смарт-контрактами [16]. Команда, ответственная за это решение, также построила биржу на ней. Возможная проблема с BitShares заключается в том, что они используют механизм делегированного доказательства ставки, что делает решение довольно быстрым, но он недостаточно изучен, и пока нельзя сказать, что насколько же безопасен, как и механизм доказательства работы.

1.2.2. Waves Dex

Еще одна платформа с механизмом доказательства работы. Его основным отличием является пользовательский алгоритм под названием «Matcher», который, очень эффективен в сопоставлении покупателей и продавцов в цепочке блоков [17].

1.2.3. BitSquare

Проект с открытым исходным кодом, за который не отвечает ни одна компания. Он разработан и поддерживается сообществом и использует сеть Tor как инфраструктуру [18]. Все транзакции проверяются вручную, и размещение вашей заявки занимает в среднем один час. И это так медленно не только из-за ручной работы, но и потому, что сама сеть Tor очень медленная.

1.3. Частично децентрализованные решения

Все следующие примеры поддерживают только обмен токенами, а не всеми криптовалютами. Это определяется протоколами, реализованными в этих решениях, которые не поддерживаются всеми криптовалютами.

1.3.1. 0x protocol

Биржи со свободным в использовании реле, использующим протокол 0x. Примером может служить Radar [19].

1.3.2. Bancor

Биржа с закрытым реле, протоколом и блокчейном [20]. Примечательна высокими темпами развития и набора аудитории. Для обмена на криптовалюты других бирж платформа вводит свои токены-заменители, которые идейно должны иметь ту же стоимость, что и реальные, на основании доверия к компании.

1.3.3. Ether Delta

Обмен с смарт-контрактами без открытого протокола на блокчейне Ethereum и ужасным пользовательский интерфейсом по утверждению самих авторов [21, 22]. Примечательна данная биржа тем, что без инвестиций разработала собственный смарт-контракт для осуществления обмена и набрала популярность.

Наиболее интересной является третья категория решений, так как она является наиболее быстрой из всех децентрализованных решений.

1.4. Техническая сторона решений

Все приложения, упомянутые ранее, разрабатывались командами гораздо дольше, чем у меня для первой реализации моей идеи. Вот почему моему первому решению необходимо использовать многие SDK, которые уже прошли тестирование и доказали свою работоспособность. Мое приложение может находиться в категории «Децентрализованная» или «Частично децентрализованная». Вот почему я провел исследование технологий, которые мог бы использовать.

Я узнал, что наиболее перспективный подход - создание узла легкого клиента блокчейна Ethereum на устройстве и использование SDK 0x протокола, разработанного сообществом в сотрудничестве с авторами протокола.

1.4.1. Узел цепи блоков Ethereum

Самый очевидный способ децентрализовать приложение - использование первой самой популярной цепочки блоков, которая позволяет использовать смарт-контракты – блокчейн Ethereum. Я реализовал тестовый образец для создания узла на мобильном устройстве, используя официальный фреймворк Ethereum Geth для iOS [23], завершившийся ошибкой, гласившей, что платформа не поддерживается. После многих дополнительных тестов с изменением кода я решил, что стоит попробовать другие технологии.

1.4.2. SDK 0x протокола

Сначала я изучил документацию по протоколу 0x и пришел к выводу, что у меня недостаточно времени для реализации всего протокола на нативных языках: Swift или Objective C. Поэтому я решил интегрировать уже созданные SDK. Единственная проблема заключается в том, что он реализован только на одном языке – TypeScript, являющемся строго типизированной надстройкой над языком JavaScript и компилирующимся в него. К счастью, в операционной системе iOS есть встроенная среда [24], которая может обрабатывать и интерпретировать JavaScript. Тем не менее, я мог столкнуться с необходимостью ручного редактирования фреймворка.

Если бы это понадобилось, проект мог стать нестабильным, поскольку изменения в SDK протокола могут привести к новым трудностям в обнаружении ошибок и дальнейшем непрерывном тестировании.

Однако после нескольких безуспешных попыток поместить и запустить скомпилированную в JavaScript существующую библиотеку, я решил, что лучше реализовать имплементацию на хорошо работающем в iOS языке Swift.

Использование протокола 0x включает в себя реализацию реле. Я выбрал реле биржи Radar. Это наиболее стабильное, если не единственное работающее приложение, обеспечивающее реле протокола 0x.

1.4.3. Другие способы

Я также нашел другие способы, которые могли бы привести к аналогичному результату, но они занимают больше времени. Все же они заслуживают упоминания, поскольку показывают, как я получил результаты, которые у меня есть. Решение с узлом может быть изменено таким образом, чтобы использовался сторонний SDK. Например, Swiftheureum [25]. Недостатками этого метода являются отсутствие глубокого тестирования и слабая поддержка. Второй альтернативой является обратное проектирование протокола Bancor и его использование. Этот вариант может быть незаконным и его довольно сложно выполнить.

1.5. Пользовательские интерфейсы решений

Большинство существующих обменных приложений позволяют работать только в одном из режимов: режиме торговли или режиме обмена. В режиме торговли есть функции, необходимые профессионалам, такие как функции остановки убытков и взятия прибыли,

вычисление графика и индикаторов. Режим обмена позволяет только сделать заказ двух типов: покупку и продажу. Мое решение будет второго типа. Оно будет включать только те функции, которые необходимы для получения текущей цены валютной пары, размещения заказа и совершения транзакции. Оно также не включает внутренний кошелек для валюты, поскольку это сделает приложение уязвимым для хакерских атак с возможной потерей активов.

1.6. Обзор whitepaper 0x протокола

Так как выбранным способом для создания децентрализованной биржи является 0x протокол, является важным подробный обзор данного технологического решения, что будет произведено далее.

1.6.1. Введение

В данной главе описывается проблема, существующая в сообществе разработчиков децентрализованных приложений. Она заключается в том, что уже создано большое количество решений с варьирующимися методами взаимодействия, зачастую заточенными под каждое отдельное приложение, даже внутри одной бизнес-области, например, среди децентрализованных бирж.

Помимо этого, приводится в пример множество успешно проведенных хакерских атак на аналогичные централизованные решения: Mt. Gox, Shapeshift и Bitfinex. В децентрализованных приложениях риску подвергаются активы отдельных пользователей, а не многомиллионные активы биржи-посредника, что уменьшает риск и делает хакерские атаки менее привлекательными.

Далее говорится о том, что закрытые протоколы и решения являются менее конкурентоспособными и эффективными в силу того, что открытые решения быстрее развиваются за счет объединения ресурсов сообщества, использующего свободно распространяемое решение. Кроме того, протоколы не должны быть привязаны к логике одного конкретного приложения, а скорее ко всей области.

1.6.2. Существующие работы

Приводится несколько вариантов решения проблемы медленного взаимодействия с цепочкой блоков Ethereum, вызываемой низкой пропускной способностью данного блокчейна, высокой стоимости и большом количестве транзакций, необходимых для поддержания работы децентрализованной биржи. Среди этих решений смарт-контракты AMM [28, 29], основной

проблемой которых является неспособность быстрого реагирования на изменение цены, каналы состояний, которые являются уязвимыми к распределённым атакам типа «отказ в обслуживании».

Лучшим решением выбирается вынесение ордеров на реле за пределами блокчейна. Данное решение не имеет тех же проблем, что и предыдущие, но при этом так же уменьшает стоимость торговли за счет уменьшения количества транзакций на цепочке блоков, так как они производятся только при успешном выполнении ордера. Оно расширяется за счет уменьшения привязанности к конкретным приложениям.

1.6.3. Спецификация

Каждый ордер представляет собой пакет данных, содержащий параметры ордера и подпись. Параметры заказа объединяются и хэшируются до 32 байтов с помощью функции Keccak SHA3. Создатель ордера подписывает хэш его секретным ключом для создания подписи ECDSA.

Название	Тип данных	Описание
Version	address	Адрес смарт контракта обмена
Maker	address	Адрес создателя ордера
tokenA	address	Адрес токена A, поддерживающего ERC20
tokenB	address	Адрес токена B, поддерживающего ERC20
valueA	uint256	Количество единиц токена A, предлагаемого создателем
valueB	uint256	Количество единиц токена B, запрашиваемых создателем
Expiration	uint256	Дата срока годности (unix)
feeRecipient	address	Адрес Реле
feeA	uint256	Количество токенов протокола, выплачиваемых создателем ордера
feeB	uint256	Количество токенов протокола, выплачиваемых исполнителем ордера
V	uint8	Подпись ECDSA
R	bytes32	
S	bytes32	

Таблица 1. Параметры ордера.

Магистральный пользовательский сценарий, согласно протоколу 0x выглядит следующим образом (см. рис. 1):

1. Реле ссылается на свою ценовую политику и адрес, которые они используют для сбора выплат за транзакции;

2. Создатель создает ордер, устанавливая поля feeA и feeB значениям, которые соответствуют ценовой политике Реле, присваивая полю feeRecipient адрес, выбираемый Реле и подписывает заказ своим личным ключом;

3. Создатель передает подписанный заказ Реле;

4. Реле получает ордер, проверяет, что ордер действителен, и что он предоставляет требуемые значения выплат. Если ордер недействителен или не соответствует требованиям Реле, порядок отклоняется. Если ордер удовлетворительный, Реле отправляет ордер в свою книгу ордеров;

5. Исполнители получают обновленную версию книги заказов, которая включает заказ изготовителя;

6. Исполнитель исполняет ордер Создателя, отправив его на биржевой контракт на блокчейн Ethereum.

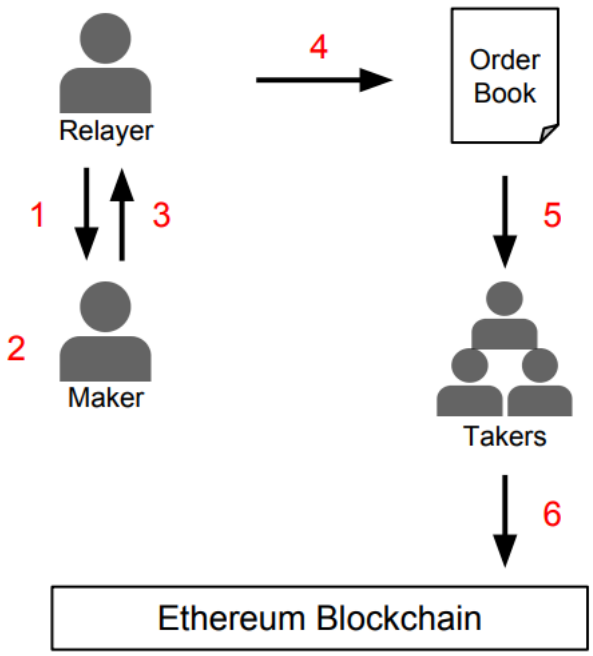

Рисунок 1. Алгоритм исполнения ордера

Важной деталью является то, что описанный алгоритм предполагает, что выбор наиболее выгодного ордера происходит не на реле, а на клиенте, однако, большинство открытых реле предоставляет книги ордеров в упорядоченном по цене порядке, что существенно снижает нагрузку на девайс. В нашем случае это особенно важно, так как обработка происходит на мобильном устройстве, не располагающем большими ресурсами.

Протокол осуществления обмена реализуется на смарт-контракте. Данный контракт должен содержать всего две функции: исполнение ордера (fill) и отмена ордера (cancel). Исполнение стоит примерно 90 тысяч единиц газа.

Возможно, как полное, так и частичное исполнение ордеров. Частичные исполнения могут осуществляться до тех пор, пока сумма частей не превышает не превышает число указанную в ордере стоимость.

Время в виртуальной машине Ethereum задается метками времени, которые устанавливаются каждый раз, когда майнится новый блок. Следовательно, срок действия ордера зависит не от времени, в которое исполнитель ордера передает изъявляет намерение исполнить ордер, а от времени, когда функция исполнения (fill) выполняется на машине майнера. Временная метка текущего блока не может быть более ранней, чем временная метка предыдущего блока.

Неисполненные и не просроченные блоки могут быть отменены создателем ордера. Однако использование этой возможности предполагается создателями 0x протокола только в исключительных случаях, так как операции на цепи блоков оплачиваются.

1.6.4. Другие главы

В остальных главах статьи объясняется механизм обновления смарт-контрактов, приводится описание протокола ERC20, а также обобщается весь материал, они не рассматриваются подробно в данной работе, так как не имеют прямого отношения к конечному решению.

2. Проектирование

2.1. Архитектурные паттерны

Наиболее популярными архитектурными паттернами в мобильной и, в частности, iOS разработке (возможны другие названия, но аналогичные принципы):

- MVC (Model View Controller);
- MVP (Model View Presenter);
- MVVM (Model View ViewModel);
- VIPER (View Interactor Presentation Entity Router).

Далее будут рассмотрены их ключевые качества и осуществлен выбор наиболее подходящего варианта.

2.1.1. Model View Controller

MVC – наиболее часто используемый архитектурный паттерн. За счет простоты определяемых абстракций он завоевал репутацию удобного подхода к разделению сущностей.

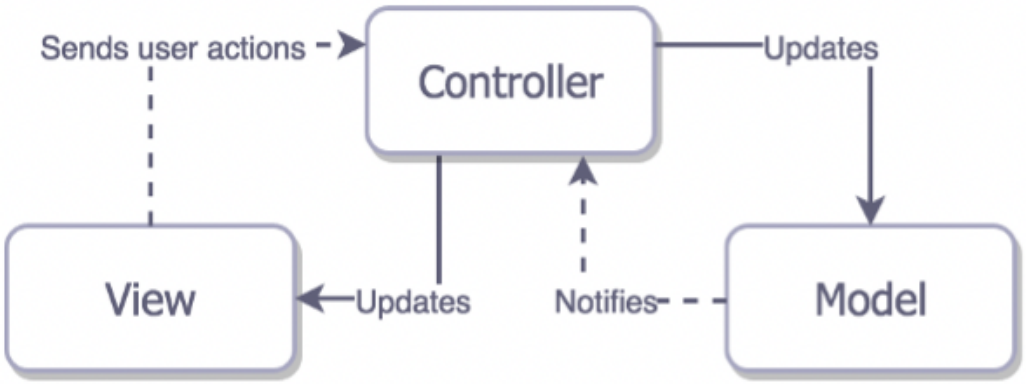

Рисунок 2. MVC

Паттерн MVC определяет 3 компонента:

1. Модель (Model) – отвечает за бизнес-логику и отображение доменной области, ничего не знает об Отображении, передает Контроллеру необходимые данные через выделенные интерфейсы или системные механизмы;

2. Отображение (View) - отвечает за отображение визуальных компонентов, не знает о существовании модели, о действиях пользователя сообщает Контроллеру через выделенные интерфейсы или системные механизмы;
3. Контроллер (Controller) – отвечает за обеспечение взаимодействие Модели и Отображения, знает о них обоих.

   Уже из определения абстракций можно заметить, что ответственность Контроллера имеет больший объем, чем у других двух компонентов. Однако это усугубляется устройством фреймворков компании Apple, которое приводит сильной привязке Отображения и Контроллера, что приводит к проблеме, часто называемой “Massive View Controller” (см. рис. 3)

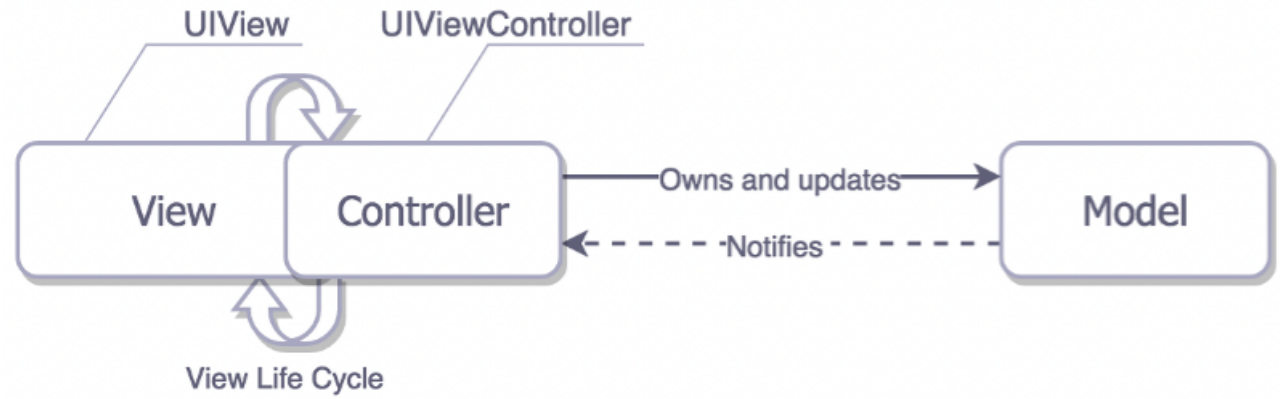

Рисунок 3. Причина возникновения “Massive View Controller”

2.1.2. Model View Presenter

Если слегка переопределить ответственности тех же абстракций, что и в MVC получится более удобный вариант архитектуры. В iOS сообществе его называют MVP.

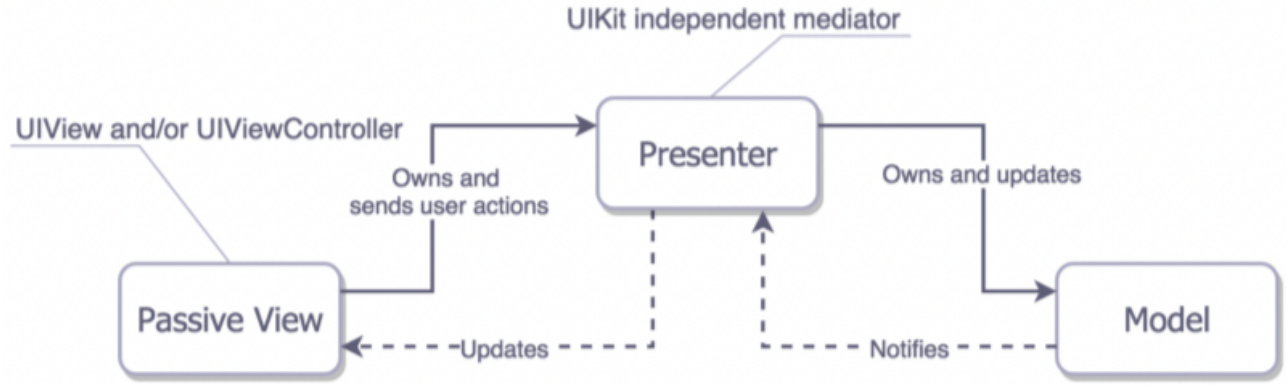

Рисунок 4. MVP

Переопределение ответственности происходит следующим образом:

1. Модель (Model) – роль полностью идентична роли Модели в MVC;

2. Отображение (View) – в отличие от MVC здесь Отображение знает о существовании аналога Контроллера – Представления (Presenter);
3. Представление – так же знает о существовании Отображения и Модели, отвечает за активацию бизнес логики и преобразование данных для передачи отображению.

Такое распределение абстракций приводит к облегчению тестирования бизнес-логики и уменьшению ответственности, находящейся в классе-наследнике UIViewController, который в случае MVC содержит часть ответственности Отображения и всю ответственность Контроллера, а в случае MVP – только Отображения.

2.1.3. Model View ViewModel

В MVVM распределение ролей компонентов полностью идентично MVP с точностью до переобозначения (ViewModel = Presenter). Единственным отличием является использование методов реактивного программирования. Для того, чтобы быстро, удобно и надежно использовать указанные методы, необходимо использование сторонних фреймворков, что приведет к увеличению размера приложения.

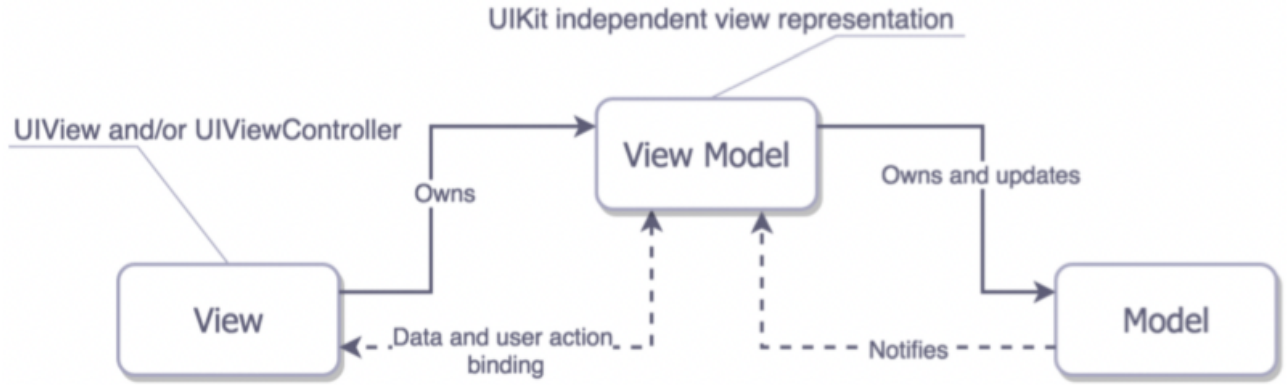

Рисунок 5. MVVM

2.1.4. VIPER

VIPER определяет наибольшее количество сущностей, что идейно должно приводить к наиболее сбалансированному распределению ответственностей. Обязательными абстракциями являются:

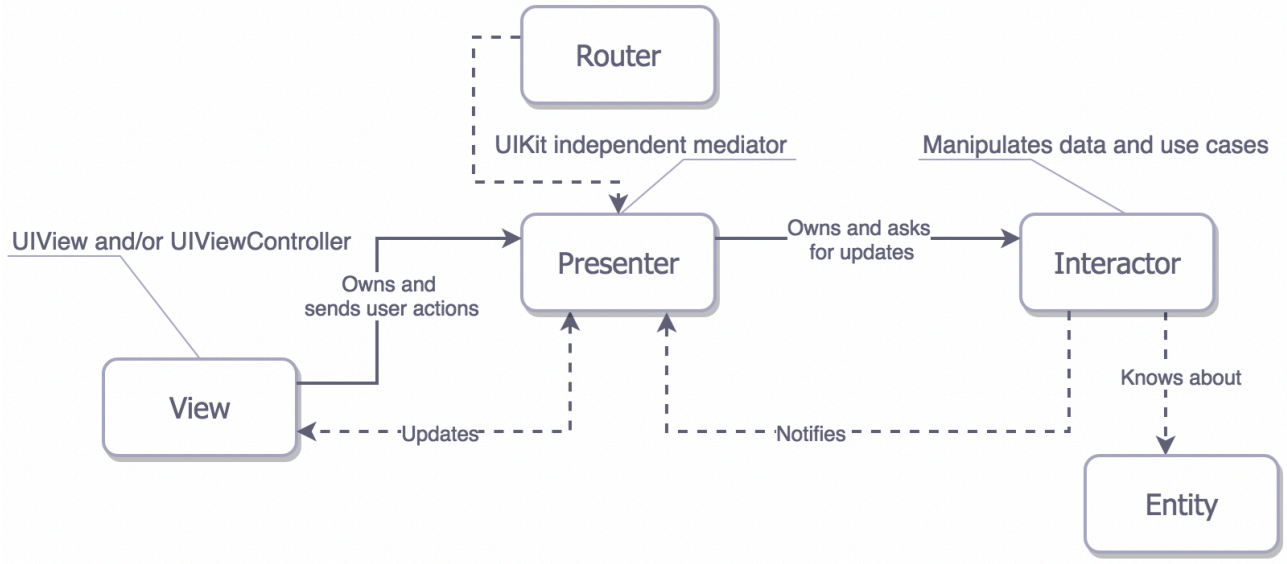

Рисунок 6. VIPER

1. Отображение (View) - идентично Отображению в MVP;
2. Интерактор (Interactor) - отвечает за бизнес-логику приложения, знает о Сущности и общается с Представлением;
3. Представление (Presenter) – отвечает за приведение формата данных к нужному для отображения;
4. Сущность (Entity) – идентично роли Модели в MVP;
5. Роутер (Router) - отвечает за осуществление навигации между экранами.

Дополнительной сущностью является Каркас (Wireframe), который собирает все остальные воедино, согласно правилам их отношений. В теории, такое распределение ответственности позволяет полностью тестировать все компоненты. Однако поддержка такого количества классов требует большого количество выделяемых на разработку ресурсов, и обычно используется командами от 3 человек и более.

2.1.5. Выбор архитектурного паттерна

Из указанных выше описаний можно сделать вывод, что MVC быстро приводит к усложнению поддержки приложения, VIPER недоступен ввиду ограниченного количества ресурсов.

Таким образом, остается сделать выбор между MVP и MVVM. В моем приложении будет большое количество обратных вызовов, а MVVM хорошо решает проблему высокого уровня вложенности. Поэтому будет использоваться данная архитектура.

2.2. Используемые паттерны проектирования

В данной секции описаны основные паттерны проектирования, которые используются в реализации приложения.

2.2.1. Внедрение зависимостей (Dependency Injection)

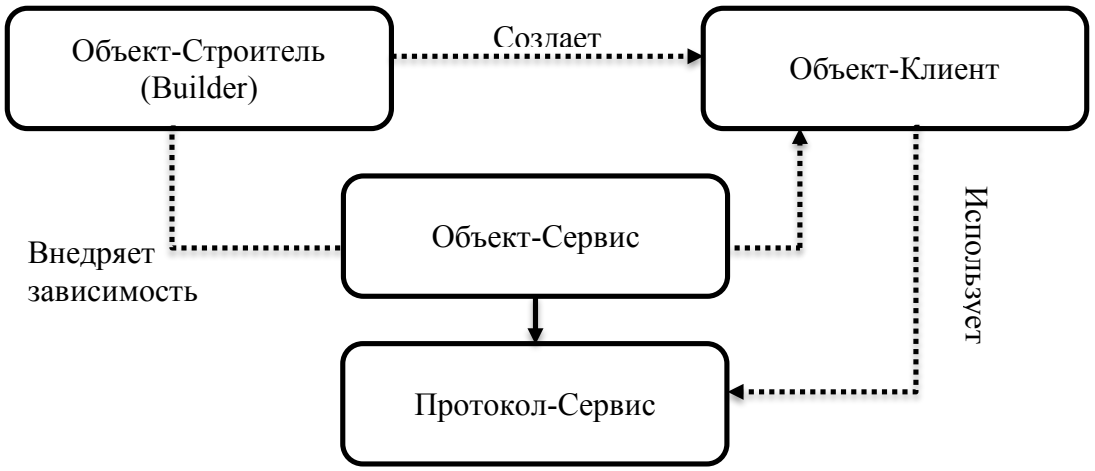

Рисунок 7. Внедрение зависимостей

Данный паттерн служит для уменьшения связности отдельных модулей и компонентов, а также для повышения удобства тестирования. Это производится посредством внедрения зависимостей через конструктор объекта, где параметр и внутреннее свойство объекта имеют тип протокола, а внедряемая зависимость только реализует этот протокол. Таким образом можно реализовать несколько вариантов класса внедряемой зависимости и использовать их в разных случаях. Например, одну имплементацию для стандартной работы приложения, и одну для реализации тестирования.

2.2.2. Делегат (Delegate)

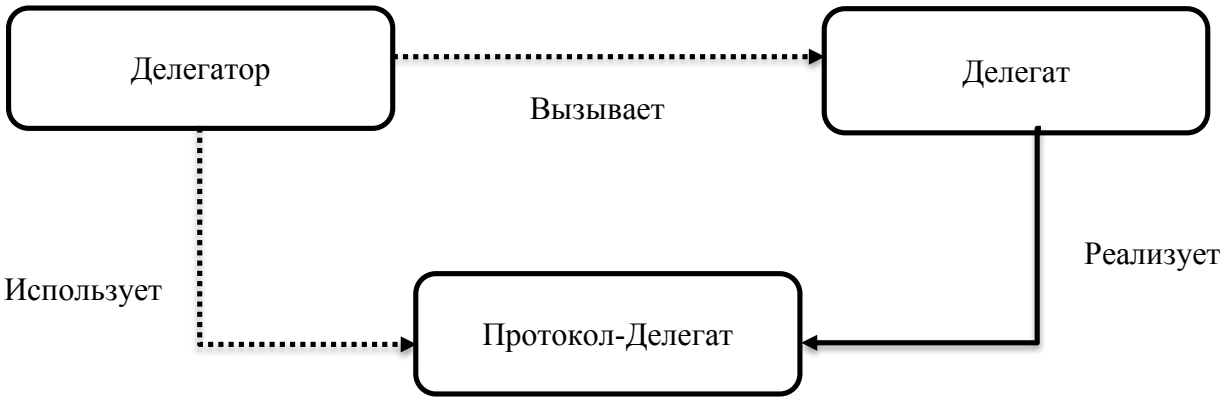

Рисунок 8. Делегат

Паттерн «Делегат» служит для реализации обратного вызова. В практике iOS разработки он используется для работы с таблицами и при любой делегации контроля дочернему объекту. Так как в языке Swift функции являются объектами первого класса, а так же есть механизм замыканий, то иногда для удобства данный паттерн заменяется передачей функций для обратного вызова в качестве параметра.

2.2.3. Наблюдатель (Observer)

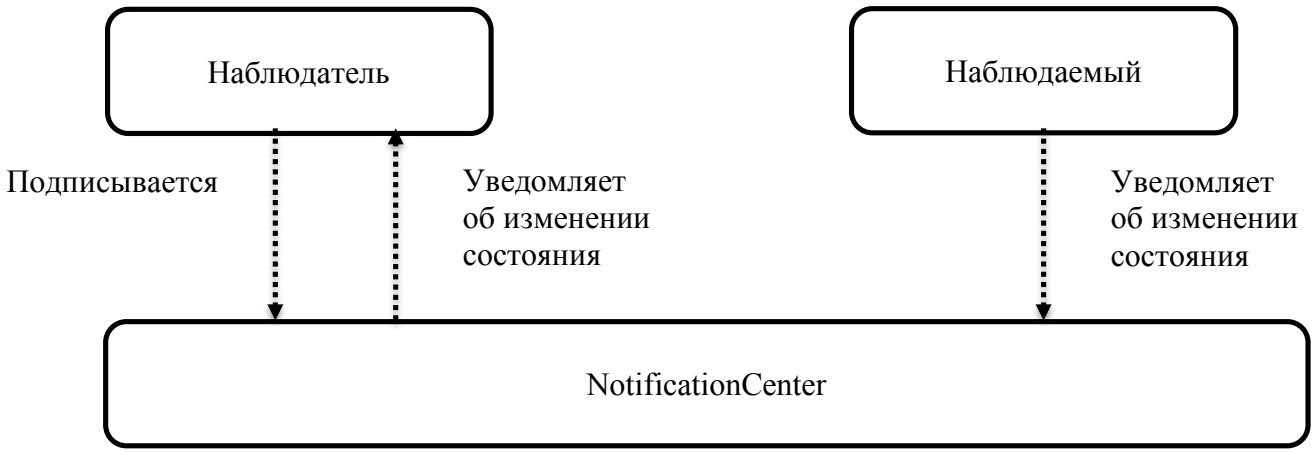

Рисунок 9. Реализация паттерна наблюдатель
с использованием системных механизмов iOS

Паттерн Наблюдатель используется во избежание дублирования одинаковых вызовов при изменении состояния объектов из разных мест. В сочетании с внутренними функциями операционной системы iOS он создает возможность вызова функций объектов, которые не являются зависимыми друг от друга. Данный механизм используется через внутренний фреймворк NotificationCenter.

2.3. Сетевое взаимодействие

Далее описывается взаимодействия с двумя другими информационными системами посредством сети интернет: реле и блокчейном.

2.3.1. Реле

Реле по своей сути является сервером, который предоставляет услуги по хранению ордеров и обновлению смарт-контрактов. Как упоминалось раньше, было принято решение

использовать для этой цели Radar Relay. Стандарт того, какие запросы можно отправлять на реле описан в документации к SDK 0x протокола. В приложении используются почти все возможные запросы:

- GET /token_pairs – данный запрос используется для того, чтобы определить, с какими парами токенов могут быть пары, выкладываемые на данное реле;
- GET /orders – позволяет получать ордера отфильтрованные по различным признакам, таким как адрес создателя, токена и т.п. Использовался только во время разработки, но не в финальном решении;
- GET /order – позволяет получить ордер по его хешу, если ордер все еще присутствует на реле;
- GET /orderbook – позволяет получить все ордеры на покупку и продажу по конкретной паре токенов, в приложении используется для определения рыночной цены и осуществления обмена по рыночной цене;
- POST /fees – предоставляет информацию по выплатам, которые будут осуществлены создателем и исполнителем ордера, в случае его успешного исполнения;
- POST /order – осуществляет размещение передаваемого ордера на реле. Используется в приложении для размещения отложенного ордера.

Radar Relay использует названия параметров, отличные от указанных в протоколе. Однако это не влияет на общую работу системы, так как названия параметров не используются в создании хеша ордера или передаче параметров на смарт-контракт.

2.3.2. Блокчейн

Взаимодействие с блокчейном Ethereum осуществляется через публичный узел, предоставляемый сервисом Infura [31]. Оно осуществляется посредством вызова процедур на узле в соответствии с протоколом JSON-RPC [32], разработанным компанией Ethereum, независимым от транспортной инфраструктуры и платформы, легковесным и не учитывающим состояния. В документации данного протокола определяется стандарт обмена сообщениями (структур данных, способов осуществления вызовов). В данном приложении используется ограниченный набор процедур, описанных стандартом.

Используемые процедуры:

- *eth_sendRawTransaction* – данная процедура создает новую транзакцию в блокчейне. В приложении с ее помощью осуществляются вызовы смарт-контракта. В

частности, метод выполнения ордера и его отмены. Сигнатуры методов можно посмотреть в публично доступном смарт контракте 0x протокола [33]. В случае успешного вызова процедура возвращает хеш созданной транзакции;

- *eth_getTransactionReceipt* – по хешу транзакции данная процедура возвращает статус транзакции, логи, созданные при запуске смарт-контракта на машине майнера, номер и хеш блока, в котором находится транзакция, количество потраченного газа и другая, менее интересная информация. В приложении данная процедура используется для определения статуса ордера после его размещения в блокчейне.

Для вызова процедур необходимо передавать параметры в соответствии с бинарным интерфейсом приложения (Application Binary Interface) [34]. Он определяет базовые типы (uint256, uint8, address, bool, function), и то как они должны быть представлены для передачи в сообщении.

3. Особенности реализации

3.1. Пользовательский интерфейс

Существует несколько вариантов построения пользовательского интерфейса для iOS устройств:

- XIB
- Storyboard
- Средствами встроенных фреймворков из кода
- Средствами сторонних фреймворков из кода

3.1.1. XIB

Данная технология представляет весь интерфейс в виде XML по определенным стандартизованным правилам. Для редактирования данного файла используются интерактивные инструменты, входящие в состав интегрированной среды разработки XCode. Расширение файлов называется так же как технология «.xib».

Ограничением данной технологии является то, что каждый создаваемый файл может содержать только интерфейс одного экрана.

3.1.2. Storyboard

Эта технология так же представляет интерфейс в виде XML, однако может включать в себя все или несколько экранов приложения. Помимо этого, она предполагает определение навигации между экранами при нажатии на интерактивные элементы (кнопки, переключатели и т.п.). Добавление навигации таким образом увеличивает связность отдельных модулей, отвечающих за разные экраны. Однако возможно использование данной технологии для каждого отдельного экрана или отдельных пользовательских историй, что уменьшит уровень связности, однако это увеличивает количество файлов и усложняет систему. Внедрение зависимостей тоже усложняется.

3.1.3. Встроенные фреймворки

Использование встроенных в операционную систему фреймворков из кода делает изменения уже существующего интерфейса более контролируемыми, так как языки разметки,

автоматически генерируемые инструментами интегрированной среды разработки менее читаемы и редко исправляются вручную. Другой особенностью данного подхода является то, что кодовая база приложения увеличивается за счет отказа от языка разметки.

3.1.4. Сторонние фреймворки

Использование сторонних фреймворков обычно служит для внедрения функций, специфичных для доменной области приложения, например, игры. Другой частой причиной использования сторонних фреймворков является возможность использовать написанный с их помощью код на других платформах. Внедрение стороннего фреймворка увеличивает размер занимаемой приложением памяти.

3.1.5. Выбор технологии

Настоящая работа требует автоматического тестирования как минимум ключевых функций, отвечающих за взаимодействие с другими программными комплексами. В связи с этим очень важно, чтобы внедрение зависимостей происходило как можно более просто. Однако, нет смысла не использовать интерактивные технологии в тех местах приложения, которые обычно не содержат большого количества логики. Такими элементами, например, являются отдельные клетки таблиц. Для них используется формат XIB. Для входа в приложение удобнее использовать Storyboard, а построение интерфейсов отдельных экранов производить кодом при помощи встроенных фреймворков. В сторонних фреймворках необходимости нет.

3.2. Решенные проблемы

Во время разработки программного продукта автор данной работы столкнулся с рядом проблем. Большинство из них связаны с низким уровнем развития области приложений, основанных на технологии цепочки блоков, к которой относится решение.

3.2.1. Уровень документирования

Как следствие, существуют не описанные стандарты, которые знают сообщества разработчиков отдельных решений. Примером может служить отсутствие упоминания того, что в сообщении подписываемом цифровой подписью SECP256k1, исползуемой для транзакций в блокчейне Ethereum, а также для хешей ордеров для отправки на реле, обязательно должен присутствовать префикс «\u{19}Ethereum Signed Message:\n32», где 32 – это длина изначального сообщения (в данном случае хеша ордера). В документации 0x протокола этот префикс обозначается, как необязательный. Разработчики Radar Relay ничего не знали о нем и только

ссылались на существующую реализацию, которая не подходила мне по используемому языку программирования. Как итог, об этом знали разработчики самого протокола, и только от них самих можно было узнать о том, что такой префикс обязателен, и используется всеми, кто использует протокол 0x.

Вторым подобным «негласным стандартом» оказался порядок конкатенируемых полей ордера перед хешированием. В таблице 1 данной работы повторен порядок полей из документа «0x protocol whitepaper». Однако, как было выяснено в ходе разработки, в документе порядок приводится наобум, и других документов, упоминающих данную тематику, нет. В данном случае помогло то, что код смарт-контрактов, используемых на блокчейне Ethereum, всегда можно посмотреть. Удобный интерфейс для этого предоставляет сервис Etherscan [35]. Из кода смарт-контракта удалось выяснить порядок хеширования элементов.

В документации API Radar Relay говорится об использовании своих стандартов точности передачи чисел (количества знаков после запятой), однако на самом деле они сохраняют стандартную точность в 18 знаков, и свой стандарт используется только для отображения пользователю.

3.2.2. Метаинформация токенов

Еще одной проблемой, которую нужно было решить во время разработки оказалось отсутствие имен токенов в отдаваемой реле информации. Вместо них используются 20-байтные адреса смарт-контрактов токенов, которые совершенно неудобны для восприятия, запоминания и идентификации пользователем приложения. К счастью, информацию по соответвию адресов читаемым названиям токенов можно достать из вышеупомянутого сервиса Etherscan. Она была выкачана скриптом и собрана в JSON-файл, который поставляется в сборке приложения. Далее все это собирается в словарь, где ключом является адрес, а значением – объект, содержащий полное имя, символ и ссылку на логотип токена.

ПРИЛОЖЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1) Smart contract [Электронный ресурс] // Wikipedia [официальный сайт] URL: https://en.wikipedia.org/wiki/Smart_contract (Дата обращения: 22.01.2018)
2) Ethereum [Электронный ресурс] // Wikipedia [официальный сайт] URL: https://en.wikipedia.org/wiki/Ethereum (Дата обращения: 22.01.2018)
3) ERC-20 Token Standard [Электронный ресурс] // Ethereum Wiki [официальный сайт] URL: https://theethereum.wiki/w/index.php/ERC20_Token_Standard (Дата обращения: 22.01.2018)
4) 0x project [Электронный ресурс] // 0x project [официальный сайт] URL: https://0xproject.com/ (Дата обращения: 22.01.2018)
5) BitSquare Philosophy [Электронный ресурс] // BitSquare [официальный сайт] URL: https://bisq.network/philosophy/ (Дата обращения: 22.01.2018)
6) GDAX [Электронный ресурс] // GDAX [официальный сайт] URL: https://www.gdax.com/ (Дата обращения: 22.01.2018)
7) Gemini [Электронный ресурс] // Gemini [официальный сайт] URL: https://gemini.com/ (Дата обращения: 22.01.2018)
8) Changelly [Электронный ресурс] // Changelly [официальный сайт] URL: https://changelly.com/ (Дата обращения: 22.01.2018)
9) Cryptopia [Электронный ресурс] // Cryptopia [официальный сайт] URL: https://www.cryptopia.co.nz/ (Дата обращения: 22.01.2018)
10) Bitcoin Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/BTC_CHANGE_BOT (Дата обращения: 22.01.2018)
11) Ethereum Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/ETH_CHANGE_BOT (Дата обращения: 22.01.2018)
12) Litecoin Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/LTC_CHANGE_BOT (Дата обращения: 22.01.2018)
13) Dogecoin Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/DOGE_CHANGE_BOT (Дата обращения: 22.01.2018)
14) Bitcoin Cash Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/BCC_CHANGE_BOT (Дата обращения: 22.01.2018)
15) DASH Exchange Bot [Электронный ресурс] // Telegram [официальный сайт] URL: https://telegram.me/DASH_CHANGE_BOT (Дата обращения: 22.01.2018)

16) Bit Shares [Электронный ресурс] // Bit Shares [официальный сайт] URL: https://bitshares.org/technology (Дата обращения: 22.01.2018)

17) Waves Dex [Электронный ресурс] // Waves [официальный сайт] URL: https://wavesplatform.com (Дата обращения: 22.01.2018)

18) BitSquare [Электронный ресурс] // BitSquare [официальный сайт] URL: https://bisq.network (Дата обращения: 22.01.2018)

19) Radar [Электронный ресурс] // Radar [официальный сайт] URL: https://app.radarrelay.com/ (Дата обращения: 22.01.2018)

20) Bancor [Электронный ресурс] // Bancor [официальный сайт] URL: https://www.bancor.network/discover (Дата обращения: 22.01.2018)

21) Ether Delta [Электронный ресурс] // Ether Delta [официальный сайт] URL: https://etherdelta.com (Дата обращения: 22.01.2018)

22) Ether Delta Description [Электронный ресурс] // Reddit [официальный сайт] URL: https://www.reddit.com/r/EtherDelta/comments/6hrwnu/etherdelta_is_a_decentralized_exchange/ (Дата обращения: 22.01.2018)

23) Geth for iOS Documentation [Электронный ресурс] // GitHub [официальный сайт] URL: https://github.com/ethereum/go-ethereum/wiki/Mobile:-Introduction (Дата обращения: 22.01.2018)

24) JavaScriptCore Documentation [Электронный ресурс] // Apple [официальный сайт] URL: https://developer.apple.com/documentation/javascriptcore (Дата обращения: 22.01.2018)

25) Swifthereum [Электронный ресурс] // GitHub [официальный сайт] URL: https://github.com/IndisputableLabs/Swifthereum (Дата обращения: 22.01.2018)

26) The Swift Programming Language (Swift 4.0 Edition) – Apple Inc., 2018

27) Will Warren, Amir Bandeali, 0x: An open protocol for decentralized exchange on the Ethereum blockchain [Электронный ресурс] // 0x protocol [официальный сайт] URL: https://0xproject.com/pdfs/0x_white_paper.pdf (Дата обращения 22.03.2018)

28) Euler [Электронный ресурс] // URL: https://www.reddit.com/r/ethereum/comments/54l32y/euler_the_simplest_exchange_and_currency/ (Дата обращения 01.02.2017)

29) Galia Benartzi Guy Benartzi, Eyal Hertzog. Bancor protocol: A hierarchical monetary system and the foundation of a global decentralized autonomous exchange. 2017

30) Module-View-Controller [Электронный ресурс]// URL: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (Дата обращения: 02.03.2018)

31) Infura [Электронный ресурс] // URL: https://infura.io/ (Дата обращения: 10.05.2018)

32) Документация JSON-RPC // URL: https://github.com/ethereum/wiki/wiki/JSON-RPC (Дата обращения: 15.05.2018)

33) Пример смарт-контракта 0x протокола // URL: https://etherscan.io/address/0x12459c951127e0c374ff9105dda097662a027093#code (Дата обращения (20.05.2018)

34) Ethereum Contract ABI // URL: https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI Дата обращения (20.03.2018)

35) Etherscan [Электронный ресурс] // URL: https://etherscan.io/ (Дата обращения: 15.05.2018)

36) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

37) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

38) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

39) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001. 5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

40) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

41) ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.