Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Российский экономический университет имени Г.В. Плеханова»

Высшая школа кибертехнологий, математики и статистики

Базовая кафедра цифровой экономики института развития информационного общества

КУРСОВОЙ ПРОЕКТ

по дисциплине «Технологии распределённых реестров»

на тему «Разработка децентрализованной торговой платформы на основе технологии блокчейн»

Выполнила
обучающаяся группы
15.27Д-ИВТ02/23б
очной формы обучения

Петрушина Юлия Дмитриевна

Научный руководитель:
д.э.н., доцент, профессор кафедры

Колесник Георгий Всеволодович

Москва – 2024

Содержание

Введение

Актуальность темы исследования обусловлена тем, что сама технология блокчейн за последние года приобрела огромную популярность. В мире появляется всё больше продуктов, созданных с ее использованием. Хотя основной сферой, где блокчейн применяется, остается криптоиндустрия, но это не значит, что он не выходит за ее рамки. В российском секторе ФинТех уже разрабатывает свою платформу на основе блокчейна - Masterchain[1][2]. Она работает на основе протокола Ethereum. Другой пример работы с технологиями распределенных реестров на нашем рынке, это ООО “Системы Распределенного Реестра”. Среди учредителей этой компании есть ВТБ, Газпромбанк, Промсвязьбанк, Национальная система платёжных карт, Московская биржа и ФинТех. Компания как раз была создана для развития на нашем рынке проектов, основанных на технологиях распределенных реестров. Как видно из представленных выше фактов, блокчейн интересен крупным игрокам на рынке, и это интерес растет. А значит сама технология блокчейн пользуется большим спросом и привлекает к себе много внимания. Поэтому тема разработки торговой платформы на основе технологии блокчейн сейчас как никогда актуальна.

Цель курсового проекта - разработать децентрализованную торговую платформу на основе технологии блокчейн.

Задачи, которые нужно будет решить для достижения этой цели, можно обозначить так:

1) Исследовать, что такое торговые платформы, какие они бывают, как они работают;
2) Изучить, как устроена технология блокчейн;

3) Выдвинуть требования к нашей будущей платформе. Она должна быть децентрализованной, а также использовать технологию блокчейн;
4) Определить, с помощью чего будем разрабатывать нашу программу, и поподробнее в этом разобраться;
5) Написать код для торговой платформы;
6) Протестировать нашу платформу, чтобы удостовериться, что она соответствует всем требованиям.

Предмет исследования - разработка решения с использованием технологии блокчейн для работы с финансовыми продуктами.

Объект исследования - применение технологии блокчейн в финансовых электронных решениях.

Основная часть

1 Торговая платформа

1.1 Общее понятие “торговая платформа”

Перед тем, как разбираться с децентрализацией и как она может пригодиться торговой платформе, нужно понять, что из себя эта торговая платформа представляет. Сразу отметим, что термины “торговая платформа” и “торговая площадка” в нашем случае являются синонимами. Торговая платформа - это электронная платформа для размещения заказов финансовых продуктов, то есть какого-то экономического блага, которое приносит своему владельцу выгоду.[3][5] К финансовым благам относятся разные кредиты, страховые полисы, договора страхования, акции, облигации и т. п. С помощью торговой платформы можно продавать и покупать эти блага, причем не обязательно присутствуя лично. Ведь такие торговые платформы позволяют работать удаленно. То, что они выполнены в электронном виде, делает их невероятно удобными по сравнению с их физическими аналогами. Для покупателей торговые платформы позволяют практично и оперативно сравнивать предложение на товары, а также экономить время. А поставщикам они помогают удобно размещать товары и проводить торги. Торговые платформы уже использует для оптимизации своих закупок большое количество крупнейших российских компаний. К примеру, с торговой платформой “Росэлторг” уже работают ГК «Росатом», «Трансстрой» и «VK» [4].

1.2 Функции, которые должны быть у торговой платформы

Вся работа торговых платформ построена на взаимодействии двух групп: поставщиков и закупщиков. [8] Закупщики размещают требования к закупке, а поставщики ищут интересные им запросы и отвечают на них. Причем лично они могут не взаимодействовать, ведь мы говорим об электронных

платформах. Заказы, которые размещают закупщики, также называются лотами.

Теперь рассмотрим, что вообще должна уметь выполнять электронная торговая платформа. Причем как со стороны заказчика, так и со стороны поставщика.

Самое важное, что она должна уметь делать, это хранить список всех заказов от закупщиков и список всех ответов к ним от поставщиков. Программа должна уметь обращаться к заказам и ответам и получать информацию о них. Для каждого заказа нам важно знать какую-то информацию о заказчике, чтобы поставщик имел возможность с ним связаться, а для каждого ответа - информацию о поставщике. Также важно хранить информацию о статусе заказа, чтобы его можно было отслеживать. Это имеет большое значение для всей работы программы, ведь некоторые действия можно выполнять только с заказами определенного статуса. Помимо этого, важно уметь сопоставить, к какому заказу - какой ответ. Причем обладать такой информацией может только сам контракт, ведь то, что закупщик не знает, от каких поставщиков к нему пришли ответы, и видит только условия сделок, очень важно для поддержания здоровой конкуренции на платформе. Поэтому данные о том, какие ответы принадлежат каким сделкам, и сами данные ответов, должны храниться отдельно. Заказчик должен получать информацию о поставщике, с которым ему нужно связаться, только после того, как определился с выбором.

Чтобы хранить какие-то заказы и ответы, нужно их сначала создать. Поэтому у нашей платформы должна быть функция публикации заказа для закупщиков и ответа для поставщиков. При создании ответа, поставщик обязан указать, к какому заказу он ссылается. При этом нельзя отвечать на уже завершенные или удаленные заказы. Помимо этого, должна быть возможность удалять заказы и ответы, но, опять же, только если у них подходящие статусы. Заказы, которые уже завершились или где закупщик уже договорился с заказчиком

отменить нельзя. Тоже самое с ответами. Кроме того, нельзя удалять уже удаленные ответы и заказы. При выполнении этой операции должны меняться статусы заказов и ответов, причем, если мы удаляем заказ, нужно также удалить и все связанные с ним ответы.

Программа должна предоставлять возможность всем пользователям посмотреть список активных на данный момент заказов и ответов. При этом, как уже говорилось, нельзя показывать авторов ответов на заявки. Пользователи могут видеть только количество ответов на заказ, их описания и цены. Также должна быть возможность посмотреть все заявки и ответы, включая завершенные и отмененные. Еще важной для платформы является возможность узнать именно опубликованные тобой заказы и ответы на них, а также информацию о них. Поэтому должна быть еще функция, которая будет возвращать пользователю принадлежащие ему заказы и ответы.

Еще одна функция, играющая большую роль в работе платформы, это функция выбора одного ответа на заказ. Ее будут использовать только закупщики для того, чтобы получить информацию о выбранном поставщике, и далее связаться с ним. Только после выбора ответа заказчик сможет увидеть, кто поставщик. После выбора статус заказа и выбранного ответа должен сменится, показывая, что заказчик и поставщик выбраны. А те ответы, которые выбраны не были, должны быть удалены.

Также наша платформа должна иметь возможность каким-то образом идентифицировать пользователей. Это нужно для того, чтобы только те, кто владеют заказом или ответом могли его изменять. Без этого любой сможет изменить какой ему вздумается документ, что приведет к нечестным манипуляциям со стороны злоумышленников и неразберихе среди пользователей. К примеру, недобросовестный поставщик сможет сам поддвердить, что заказ выполнен, даже не отсылая товар, и получить

незаконно деньги. Чтобы не допустить этого, все пользователи должны быть идентифицированы.

Ну и последнее, что стоит упомянуть, это то, что программа должна хранить размер пошлины, которую контракт должен взимать за свое использование [3], и то, куда собранные пошлины отсылать. Эти переменные может изменять только тот, кто владеет платформой. Программа должна уметь отсылать пошлины, полученные с выполнения заказов, своему создателю.

2 Технологии распределенных реестров

2.1 Децентрализация, блокчейн и распределенный реестры

Для начала рассмотрим, что вообще такое “технологии распределенных реестров”. Часто понятия “технология распределенных реестров” и “блокчейн” являются синонимами. Главная особенность этой технологии заключается в том, что данные хранятся в последовательной цепочке блоков. [9] Поэтому ее и называют технологией “блокчейн” - что буквально переводиться с английского как цепь блоков. Каждый блок содержит в себе информацию о предыдущем блоке, а генерация нового является вычислительно сложным процессом. Поэтому злоумышленники не могут изменить блок в цепочке, так как для этого придется изменять и все последующие блоки тоже, и чем больше их уже есть после изменяемого, тем труднее это сделать. Такой принцип хранения информации позволяет держать ее не на одном сервере, а на всех компьютерах участников сети. Таким образом появляется децентрализация. Это отражено и в названии технологии - распределенные реестры.

Теперь посмотрим, каким же образом осуществляется связывание блоков. Для того, чтобы понять это, нужно разобраться, как устроено хэширование. Хеширование - это какая-то функция, которая преобразует некоторый массив данных в строку определенной длины, называемую хешем. При этом,

гарантированно при одинаковых вводных данных получение одинакового хеша. А чем менее вероятно, что при разных вводных мы получим одинаковый хеш, тем лучше и надежней хеш-функция. Конечно, в идеале хеш функция должна давать разные хеши для любых разных данных, но на практике такое не реализуемо. Еще особенность хеширования в том, что мы не можем получить исходные данные, зная их хеш. Именно с помощью хэша блоки хранят информацию о друг друге. Каждый блок считает свой хеш, при этом учитывая хэш предыдущего блока.

2.2 Свойства успешных децентрализованных моделей

Теперь рассмотрим свойства успешных децентрализованных моделей. Первое из них - автоматическая синхронизация данных между пользователями. Причем происходить это должно без контроля сверху. Это сильно увеличит эффективность программы. Второе свойство - монетизация приложения осуществляется через внутреннюю криптовалюту. Третье - эффективность приложения не должна зависеть от того, на скольких устройствах оно открыто, а также ни у кого не должно быть контрольного пакета акций, так как это будет мешать децентрализации. Также хорошим качеством системы будет открытость ее кода. Так клиентам будет легче нам доверять. Помимо прочего, хорошие системы всеми силами пытаются избежать ситуации, в которой у них будет единая точка отказа. Это значит, что у продукта не должно быть места, при поломке которого, будет выходить из строя вся система в целом. Последнее, что стоит отметить, это то, что системы должны стараться достигнуть максимальной децентрализации.

Но при всем этом, система не обязано удовлетворять всем требованиям, чтобы быть хорошей. Это лишь параметры идеального кейса, так что мы просто должны постараться найти решение, которое реализовывает в себе как можно больше пунктов.

3 Средства, с помощью которых будем разрабатывать нашу платформу

3.1 Ethereum

Теперь рассмотрим, с помощью чего мы будем создавать нашу торговую платформу. Было выбрано написать смарт-контракт для Ethereum [11] с помощью языка Solidity. [12] Ethereum - это второй по популярности блокчейн проект в мире, он удобен тем, что на нем можно создавать децентрализованные приложения на базе его блокчейна, что как раз подходит для нашей задачи. Так как Ethereum работает на виртуальной машине EVM, где все транзакции хранятся одновременно на каждом узле и выполняются параллельно, то запускаться приложение будет не на одном сервере, а сразу у всех майнеров, что хорошо скажется на его децентрализации. Также, открытость и прозрачность системы позволят ей получить больше доверия от пользователей. Кроме того, в Ethereum у каждого пользователя уже есть свой уникальный адрес, привязанный к нему, поэтому, используя его, мы решим проблему с идентификацией пользователей. Еще в Ethereum уже есть функции для работы с денежными балансами, поэтому мы сможем использовать их в своей платформе. Так нам не придется думать, как же организовать платежи между закупщиками, поставщиками и создателем платформы. Конечно, с другой стороны, к минусам Ethereum можно отнести то, что запуск контрактов на ней платный, поэтому часть средств будет уходить. Но плюсы Ethereum полностью нивелируют минусы, поэтому был выбран именно он.

Вся работа Ethereum основана на взаимодействии аккаунтов. Адрес любого аккаунта представляет из себя двадцати байтовую величину. С помощью этих адресов один аккаунт может обращаться к другим и взаимодействовать с ними. Аккаунты бывают двух типов: кошельки и смарт-контракты. Кошельками управляют непосредственно сами пользователи, а вот

смарт-контрактами руководит их программа. Кошельки могут инициировать транзакции, которые являются взаимодействием между аккаунтами. Транзакции могут выполнять три вещи:

1) Переводить средства с одного кошелька на другой. При этом можно также перевести деньги на смарт-контракт, важно лишь то, что в начале цепи стоит кошелек;
2) Создавать новый смарт-контракт, а потом записать его код в блокчейн для того, чтобы запускать дальше;
3) Исполнять смарт-контракты, которые были до этого загружены в блокчейн.

Важно, что за выполнение смарт-контрактов пользователям нужно платить монетами Gas, причем цена зависит от сложности выполняемых операций. Также в Ethereum есть ограничение на размер контракта, он должен быть меньше 24 килобайтов. Поэтому в наших же интересах попытаться как можно больше оптимизировать наш код.

3.2 Обзор языка Solidity и смарт-контрактов

Для того, чтобы создать свое приложение для Ethereum, нужно написать смарт-контракт. Solidity создан специально для этого. Solidity - это объектно-ориентированный язык программирования, спроектированный для того, чтобы переводить код, написанный в удобной для человека форме, в байт-код для EVM. Во многом, этот язык похож на C++ и JavaScript.

В начале любого смарт-контракта на Solidity нужно указать версию языка, которая подходит для компиляции данного кода. Одно из отличительных свойств Solidity - это частый выпуск новых версий компилятора, поэтому важно следить, чтобы на выбранных версиях нашего кода программа компилировалась правильно. Основная смысловая часть контракта хранится в блоках, которые называются contract, их принцип работы практически идентичен принципу работы классов в других языках программирования. У

каждого contract есть свое название, а помимо этого внутренности. Внутри него могут лежать переменные, функции, структуры и перечисления. Они очень похожи на свои аналоги в других языках программирования, хотя у них есть свои специфики. Важным отличием является то, что среди атрибутов функций и переменных есть те, что указывают, с какими ресурсами эти объекты работают и влияют на то, столько монет Gas будет потрачено на работу с ними. Есть атрибут view, он показывает, что такая функция только смотрит на уже существующие объекты, при этом никак не изменяя. Такие функции дешевле других, поэтому, если есть возможность указать такой атрибут, его лучше указывать. Еще есть атрибут pure, он говорит, что какая-то работа с переменными будет, но браться они будут из локальной памяти, поэтому затраты ресурсов будут меньше. Последний важный атрибут - это атрибут payable. Он показывает, что функция или переменная будут использоваться для работы с балансами. Такие объекты очень дорогие, поэтому их нужно применять аккуратно.

Важным в Solidity является перевод средств из одного аккаунта на другой. Для этого в принимающей функции нужно добавить функцию с пустым набором вводных аргументов receive или fallback. Перед их объявлением указывать ярлык function не нужно. Receive применяется, когда в функцию не поступает дополнительно никаких данных, а fallback в противном случае. А вот в контракте, посылающим деньги, нужно запустить функцию, синтаксис который выглядит как адрес, на который посылаем средства, причем он должен быть с ярлыком payable, а после него через точку функция transfer. В скобках после transfer нужно указать сумму, которая будет переведена, также можно указать в каких единицах, к примеру, ether или wei.

4 Создание схемы продукта и написание ее кода

4.1 Структуры и переменные смарт-контракта

Теперь, когда мы поняли, что должна из себя представлять наша система, какие у нее должны быть функции, а также с помощью каких инструментов мы будем ее создавать, мы можем приступить к ее реализации. Сначала рассмотрим структуры и переменные внутри нашего смарт-контракта по отдельности, а потом как они взаимодействуют между собой.

Начнем с закупщиков и поставщиков. Так как ничто не мешает кому-то быть одновременно и тем, и другим, у нас не будет деления пользователей, просто будет набор функций, часть из которых нужна только закупщикам, а часть - только поставщикам. Но ничего не мешает пользоваться и теми, и другими. Каждый пользователь представляет из себя адрес кошелька, с которого он отправляет запрос в наш смарт-контракт.

Про каждый заказ и ответ будет храниться его статус, для этого создадим enum или перечисление. В Solidity это такой пользовательский тип данных, который позволяет переменным принимать значения только из ограниченного списка. Называться наш enum будет Status, и он может принимать пять значений:

1) “Posted”, что значит, что заказ или ответ был размещен на платформе;
2) “Chosen”, что значит, что заказчик определился с поставщиком в случае с заказом, и, в случае с ответом, что конкретно этот ответ был выбран закупщиком;
3) “Sent”. Это значит, что товар, который запрашивали в заказе, был отправлен;
4) “Completed”, что свидетельствует о том, что заказ, а следовательно и ответ на него, был завершен;
5) “Canceled”. Это значит, что заказ или ответ отменили.

Для того, чтобы в информации внутри смарт-контракта было удобнее ориентироваться, создадим несколько вспомогательных структур. Структуры в Solidity, это еще один пользовательский тип данных, который хранит в себе какой-то набор указанных в нем переменных. У нас будет две структуры для хранения информации внутри массивов, одна для заказов, назовем ее Order, вторая для ответов, ее назовем Answer. В Order будут лежать переменные, хранящие информацию о id заказа (uint id), об адресе закупщика (address addr_buyer), об имени заказа (string name), об описании (string description), о статусе (Status status), тут нам как раз пригодиться enum, который мы описали выше, о телефоне закупщика (string buyer_phone_number) и его почте (string buyer_mail). В Answer будет храниться информация об id ответа (uint id), об id заказа, к которому этот ответ принадлежит (uint id_order), о имени ответа (string name), об описании (string description), о цене, которую предлагает поставщик (uint price), о статусе ответа (Status status), о телефоне поставщика (string provider_phone_number) и о его почте (string provider_mail).

Для удобства, будем хранить статистику всей платформы и ее пользователей по отдельности. Для этого будет своя структура. Это поможет нам отслеживать количество активных заказов и ответов, чтобы формировать выводы функций, а еще так участникам платформы будет легче выбирать, с кем работать. Эту структуру назовем TPStats. В статистике будет количество всех заказов, независимо от статуса (uint all_orders), количество только активных заказов (uint current_orders), количество всех ответов (uint all_answers), количество активных ответов (uint current_answers). Кроме того будем хранить как отдельные переменные адрес владельца торговой платформы и размер пошлины, которую он будет получать с выполненых заказов.

Нам нужно показывать закупщикам информацию о том, какие ответы поступили на их заказы, но при этом мы не можем оглашать, какие конкретно

поставщики эти ответы отправили. Поэтому мы создадим специальную структуру для вывода только тех данных, которые мы можем показывать. Данная структура будет называться Answer_Out, она будет использоваться для вывода только доступной всем информации об ответах. В ней будут лежать данные о цене, указанной в ответе, названии ответа и его описание.

Теперь, когда мы обозначили все пользовательские структуры, которые будут использоваться в нашем смарт-контракте, мы можем перейти к переменным. Все переменные, которые будут лежать внутри контракта будут приватными, так как мы не хотим, чтобы пользователи извне могли как-то изменить их. Начнем с базовых данных о торговой платформе. В смарт-контракте будет переменная address owner, в которой будет храниться адрес владельца торговой платформы. Сюда наша программа будет отправлять пошлины, полученные с выполнения заказов. Еще будет переменная uint fee, она будет хранить размер пошлины. Обе эти переменные будут задаваться в конструкторе контракта. Также будет переменная TPStats stats, она будет хранить статистику всей торговой платформы. Помимо этих переменных у нас еще будут словари и массивы. Массивов будет два, и они будут хранить информацию о каждом заказе и ответе. Первый будет называться orders и будет хранить объекты типа Order, а второй - answers, он будет хранить объекты типа Answer. Еще будет два словаря, также один для заказов, второй для ответов, которые будут хранить по id информацию о том, существует ли такой объект. Они будут называться order_exist и answer_exist, соответственно. Будет два словаря, хранящих информацию о заказах один (orders_of) и об ответах другой (answers_of), созданных конкретным пользователем. В качестве ключа он будут принимать объекты типа address, а выдавать будут массивы типа uint, в которых будут лежать id искомых объектов. Еще будет словарь, который будет хранить статистики пользователей. В качестве ключа он будет принимать address, а выдавать будет TPStats. Последний словарь будет хранить информацию о том, как

связаны между собой заказы и ответы. Он по id заказа будет выдавать массив типа uint, в котором будут лежать все id ответов, связанных с этим заказом.

4.2 Функции смарт-контракта для создания или изменения информации

Теперь посмотрим, как взаимодействуют эти объекты через функции смарт-контракта. Начнем с функции создания заказа, она будет называться create_order [Приложение 1]. На вход закупщик вводит имя заказа, его описание, а также свои номер телефона и почту. В начале проверим, что заказчик ввел все нужные данные и не оставил не одно поле пустым. Id добавляемого нами заказа равен stats.all_orders на данный момент. Добавим в словарь order_exist информацию о том, что заказ с таким id существует, а в словаре orders_of добавим id данного заказа к массиву всех заказов, созданных пользователем, который отправил этот запрос. В словарь stats_of в информации о пользователе, создающем данный заказ, добавим единицу в all_orders и current_orders, а также увеличим на один all_orders и current_orders в статистике всей торговой платформы. В список orders добавим заказ с введенными закупщиком данными, id этого заказа, адресом закупщика и со статусом Status.Posted.

Принцип работы функции создания ответа и заказа похожи. Поставщик отправляет программе id заказа, на который он отвечает, имя ответа, его описание, свои телефон и почту, а также цену, за которую он готов этот заказ выполнить. Дальше программа проверяет, что поставщик ввел имя, описание, телефон, почту, а еще что заказ под введенным id существует, это делается при помощи order_exist, а также что заказ находится в статусе “Posted”. Id ответа, который мы сейчас добавляем, будет равен stats.all_answers. Добавим в answer_exist, что ответ с нашем id сушествует. В stats_of по адресу поставщика, который отправил этот запрос, увеличим all_answers и current_answers на один. В answers добавим ответ с id данного ответа, данными, которые ввел поставщик, его адресом, с которого был отправлен

запрос, и со статусом Status.Posted. Также добавим в answers_of информацию о том, что этот ответ тоже принадлежит данному поставщику, а также в answers_to информацию о том, что этот ответ принадлежит к заказу, чей id поставщик указал в начале. Помимо этого, увеличим в stats current_answers и all_answers на один. Эту функцию назовем create_answer [Приложение 2].

Теперь рассмотрим функцию удаления заказа, она будет называться delete_order [Приложение 3]. На вход этой функции пользователь дает id заказа, который хочет уничтожить. Вначале, программа должна проверить, что заказ с таким id существует, что адреса владельца этого заказа и того, кто отправил запрос, совпадают, а также что у заказа подходящий статус. Если всё верно, уменьшим в stats current_answers на длину массива, который лежит в answers_to по ключу id удаляемого заказа, и all_orders заказов на один. В stats_of у владельца удаляемого заказа уменьшим current_orders на один. А дальше пройдемся по всем элементам массива id ответов, который соответствует id удаляемого заказа в answers_to, у каждого найденного ответа в answers меняем статус на Status.Canceled, а в stats_of у владельца этого ответа уменьшаем current_answers на один. В конце меняем статус у самого заказа на Status.Canceled.

Механизм удаления ответа похож на механизм удаления заказа. Пользователь вводит id ответа, который хочет удалить. Дальше программа проверяет, что ответ с введенным id существует, что адрес создателя ответа и того, кто отправил запрос, совпадают, а также что у ответа подходящий статус. После этого, если все проверки прошли успешно, в stats и в stats_of по ключу, равному id удаляемого ответа, current_answers уменьшается на один. В конце статус самого ответа изменяется на Status.Canceled. Эту функцию назовем delete_answer [Приложение 4].

Еще одна важная функция - это функция выбора заказчиком ответа из предложенных, она будет называться choose_answer [Приложение 5]. Эту

функцию должен вызвать заказчик, которому принадлежит заказ, а на вход ей он вводит id заказа и id ответа на него, который хочет выбрать. Важно, что на этом шаге у закупщика спишутся деньги за заказ, которые указаны в выбранном ответе, учитывая пошлину. Вначале программа должна проверить, что заказ и ответ с введенными закупщиком id существуют, что ответ принадлежит именно введенному заказу, что тот, кто прислал запрос является владельцем заказа, что у заказа и ответа статусы “Posted”, а также что у заказчика достаточно денег в кошельке для оплаты. Если все проверки были пройдены успешно, заказчик переводит нужную сумму на адрес торговой платформы, статус заказа и выбранного ответа меняется на Status.Chosen. Уменьшаем в stats current orders на один и current_answers на длину массива, который лежит в answers_to по ключу id удаляемого заказа. Потом пройдемся по всем ответам на заказ, которых не выбрали. Для владельца каждого ответа в stats_of уменьшим current_orders на один. Помимо прочего, мы оставляем в answers_to по запросу о данном id заказа только один id ответа - того, который мы выбрали. На выход эта функция должна отдать объект типа Provider_info, в котором будут лежать данные о поставщике, которого выбрал закупщик.

Для поставщика должна быть функция, чтобы отмечать, что товар по заказу был отправлен закупщику. Ее назовем product_sent [Приложение 6]. Поставщик подает на вход функции id ответа. Функция проверяет, что ответ с таким id существует, что данный ответ принадлежит тому, кто ее вызвал, и что статус ответа Status.Chosen. Если всё верно, то половина цены заказа отправляется от адреса смарт-контракта торговой платформы на кошелек поставщика. Статус ответа и заказа, соответствующего ему, меняем на Status.Sent.

Также для закупщика должна быть функция подтверждения получения оговоренного товара, она будет называться complete_order [Приложение 7]. На вход ей заказчик подает id заказа, который хочет отметить как

выполненный. Программа проверяет, что заказ с таким id существует, что тот, кто отправил запрос, владеет этим заказом и что статус заказа “Sent”. Если все проверки пройдены, то мы отправляем оставшуюся часть оплаты поставщику, отправляем пошлину на кошелек владельца платформы, а также меняем статусы заказа и ответа, который в нем выбран, на Status.Completed.

4.3 Функции смарт-контракта для вывода информации

Помимо всех перечисленных выше функций, которые связаны непосредственно с работой нашего смарт-контракта, нам нужны еще функции вывода, которые позволят пользователям получать актуальную информацию о том, что происходит на торговой платформе. Должны быть функции, которые будут выводить такие переменные: адрес владельца торговой платформы (get_owner) , текущую пошлину (get_fee) , текущую статистику по торговой платформе (get_tp_stats), текущую статистику по пользователю (get_user_stats).

Помимо этих простых функций, должны быть еще более сложные, которые мы опишем далее. Нам потребуется функция вывода всех текущих заказов, она будет называться get_current_orders. Она должна будет вернуть массив заказов у которых статусом не является Status.Canceled или Status.Completed. Также должны быть функции, которые возвращают вообще все и только текущие заказы, опубликованные конкретным пользователем, их назовем get_all_orders_of и get_current_orders_of. Причем смотреть текущие заказы конкретного пользователя могут все, а вот удаленные и выполненные только он сам. Такие же две функции должны быть и для ответов, только тут пользователь может узнать только свои ответы. Их назовем get_all_answers_of и get_current_answers_of. Еще заказчикам потребуется функция просмотра всех ответов на определенный заказ. Такую функцию назовем get_answers_to. Вначале эта функция будет проверять, что заказ с введенным id существует. А

потом выдавать массив типа Answer_out со всеми ответами, принадлежащими этому заказу.

5 Тестирование платформы

После того, как мы написали код нашей торговой платформы [Приложение 8], нам нужно протестировать его. Для этого попробуем провести разные сценарии работы смарт-контракта с разными вводными данными. Для начала выделим самые важные сценарии. Сначала проверим их положительные исходы, а затем те, где программа должна выдавать ошибку.

В итоге, получился такой список текстов, которые должны проходить без ошибок:

1) создать контракты
2) создать ответы на существующие заказы
3) удалить опубликованные заказы
4) выбрать подходящий ответ на существующий заказ
5) отметить продукт в заказе подходящего статуса как отправленный
6) отметить продукт в несуществующем заказе как отправленный
7) отметить заказы подходящего статуса как выполненные
8) разные функции вывода информации из смарт-контракта, которые мы прописывали до этого.

И такой список тестов, которые должны выдавать ошибку:

1) создать ответы на несуществующие заказы
2) удалить несуществующие заказы
3) удалить уже удаленные заказы
4) удалить уже выполненные заказы
5) выбрать ответ от другого заказа на существующий заказ
6) выбрать ответ неподходящего статуса на существующий заказ
7) выбрать ответ на несуществующий заказ

8) отметить продукт в заказе неподходящего статуса как отправленный
9) отметить заказы неподходящего статуса как выполненные
10) выбрать поставщика переводя в смарт-контракт недостаточную сумму

В ходе этого были выявлены незначительные баги, к примеру была обнаружена структура, которая не использовалась в коде. Она была убрана. В итоге, ошибок в финальной версии продукта, которая была получена после всех тестов, установлено не было.

Заключение

В ходе выполнения курсового проекта по теме "Разработка децентрализованной торговой платформы на основе технологии блокчейн", получилось выполнить поставленные задачи. Вначале было изучено, что торговые платформы из себя представляют электронные платформы, на которых закупщики размещают заказы, а поставщики на них отвечают. Таким образом они заключают между собой сделки. Торговые платформы невероятно удобнее своих физических аналогов, ведь они позволяют сделкам совершаться удаленно. Сами платформы получают прибыль с комиссий от сделок. Далее были обозначены ключевые функции, которые должны быть у торговой платформы. Закупщики должны иметь возможность размещать заказы, поставщики - размещать ответы на них. Все эти данные также должны храниться внутри платформы.

Затем мы углубились в тему децентрализации и блокчейна. Стало понятно, что блокчейн из себя представляет технологию децентрализованного хранения данных, в которой данные хранятся в цепочке. Каждый блок считает свой хэш, при этом учитывая хэш предыдущего блока. Именно так осуществляется связь в цепочке блокчейна. В успешных децентрализованных системах синхронизация данных осуществляется автоматически между пользователями, а также чаще всего них открытый код.

Было принято решение реализовывать свою программу, используя платформу Ethereum, которая является одной из ведущих на рынке блокчейна, и язык Solidity, который был создан специально для создания смарт-контрактов в Ethereum. Ethereum предлагает хорошую инфраструктуру для создания децентрализованных приложений, позволяет удобно управлять цифровыми активами, а также обеспечивает безопасность транзакций. Также плюсом

является то, что по Ethereum уже есть достаточный объем документации, что упростит нашу разработку.

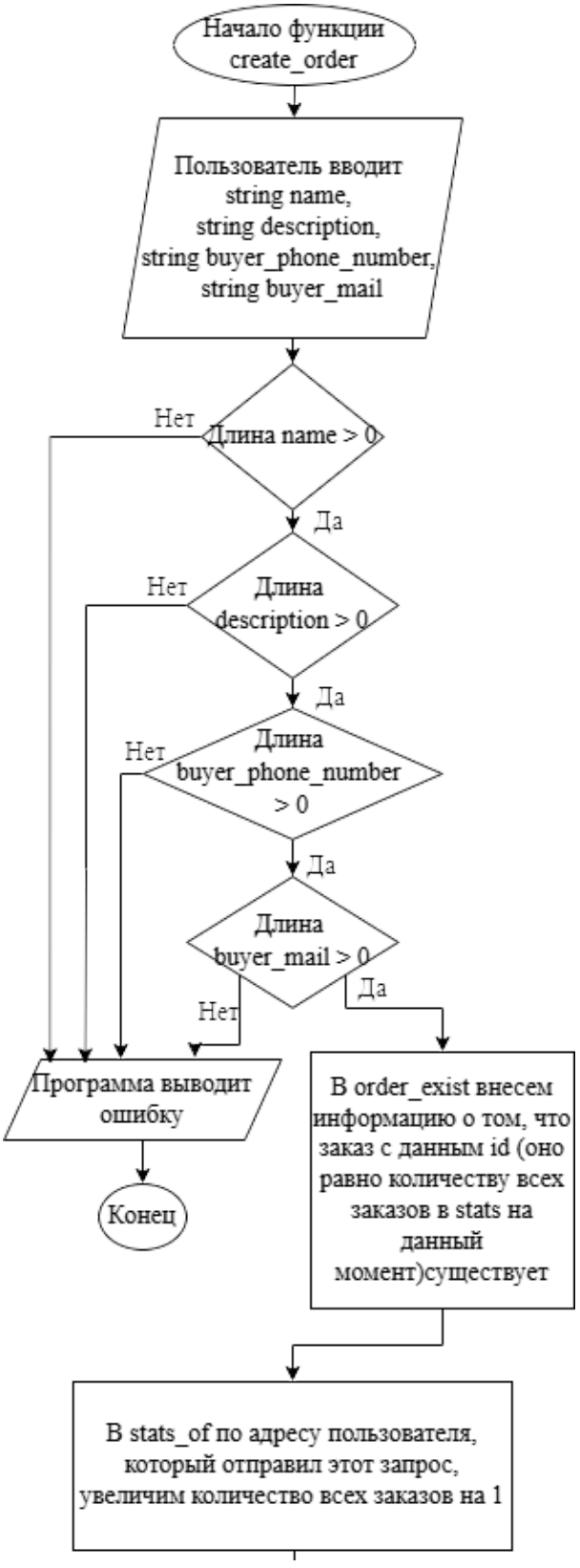
В итоге, был написан код торговой платформы. Децентрализация обеспечила больший уровень прозрачности ее работы и повысила ее безопасность. Также мы свели к минимуму участие третьей стороны в совершении сделок на нашей платформе, так что пользователи могут не волноваться о третьей стороне. Кроме того, работа смарт-контракта не будет зависеть от нас, он будет выполняться сам, когда пользователь будет к нему обращаться, что еще больше уменьшает влияние третьей стороны. Так пользователям легче доверится нашей платформе, а значит они охотнее будут ей пользоваться.

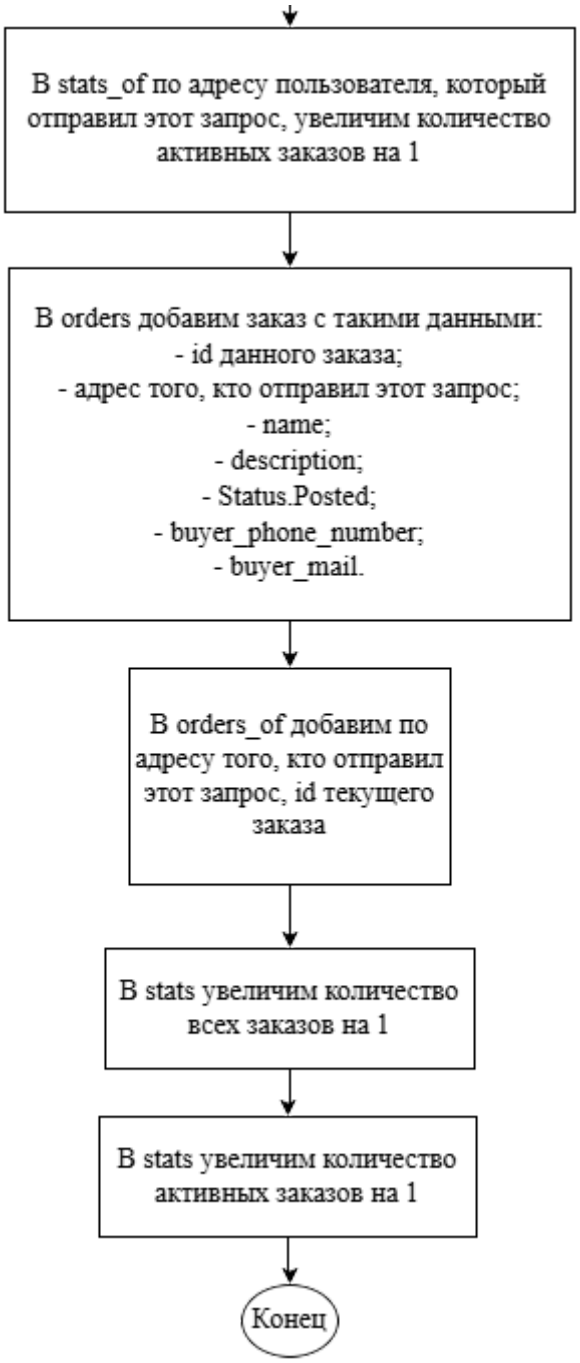
Список использованных источников

1) Публикация на сайте ассоциации “ФинТех” “Внедрение сервиса для цифровых банковских гарантий на базе блокчейн-платформы от компании Мастерчейн. Система для получения банковских гарантий” [Электроный ресурс] - ФинТех - 16.12.2020 - URL:https://www.fintechru.org/publications/assotsiatsiya-fintekh-zapustila-servis-dlya-tsifrovykh-bankovskikh-garantiy-na-mastercheyn/
2) Официальный сайт проекта “Masterchain” [Электронный ресурс] - URL:https://www.masterchain.ru
3) Чен, Д. What Is a Trading Platform? Definition, Examples, and Features [Электронный ресурс] - Investopedia - URL:https://www.investopedia.com/terms/t/trading-platform.asp
4) Официальный сайт проекта “Росэлторг” [Электронный ресурс] - URL:https://www.roseltorg.ru/
5) Афанасьева, Ю. Что такое торговая платформа? [Электронный ресурс] - Финам - 21.01.22 - URL:https://www.finam.ru/publications/item/chto-takoe-torgovaya-platforma-20220121-131500/
6) Солабуто, Н. Децентрализованные биржи: что нужно знать инвестору [Электронный ресурс] - Финам - 15.11.22 - URL:https://www.finam.ru/publications/item/decentralizovannye-birzhi-chto-nuzhno-znat-investoru-20221115-211500/
7) Азаренко, Н. Электронная торговая площадка [Электронный ресурс] - Unisender - 14.08.2023 - URL:https://www.unisender.com/ru/glossary/chto-takoe-elektronnaya-torgovaya-ploshhadka/
8) Смирнова, В. Электронные торговые площадки: гид для начинающих [Электронный ресурс] - КонтурШкола - 22.03.2024 - URL:https://school.kontur.ru/publications/1835

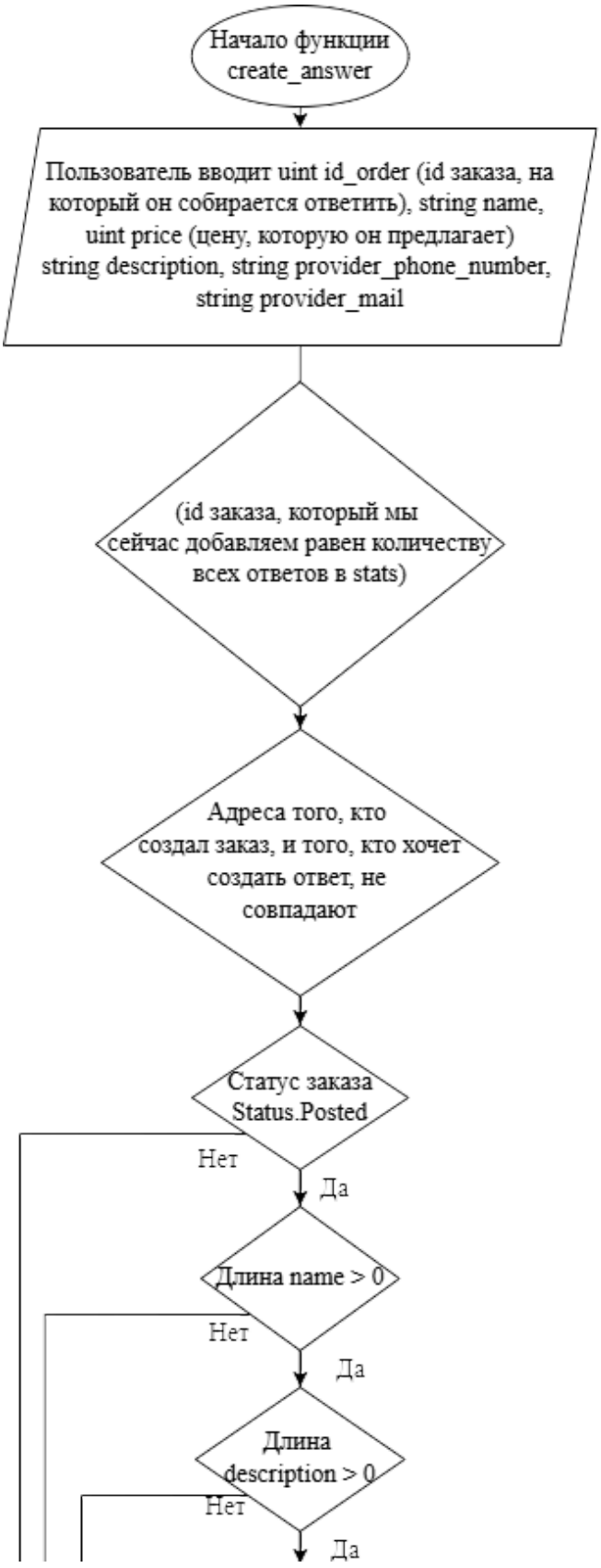
9) Генкин, А. и Михеев, Ф. Блокчейн: Как это работает и что ждет нас завтра / Альпина Паблишер / Москва; 2018
10) Распоряжение Минпросвещения России от 18.05.2020 N Р-44 "Об утверждении методических рекомендаций для внедрения в основные общеобразовательные программы современных цифровых технологий"
11) Официальный сайт Ethereum [Электронный ресурс] - URL:https://ethereum.org/ru/
12) Документация по языку программирования Solidity v0.8.25 [Электронный ресурс] - URL:https://docs.soliditylang.org/en/v0.8.25/

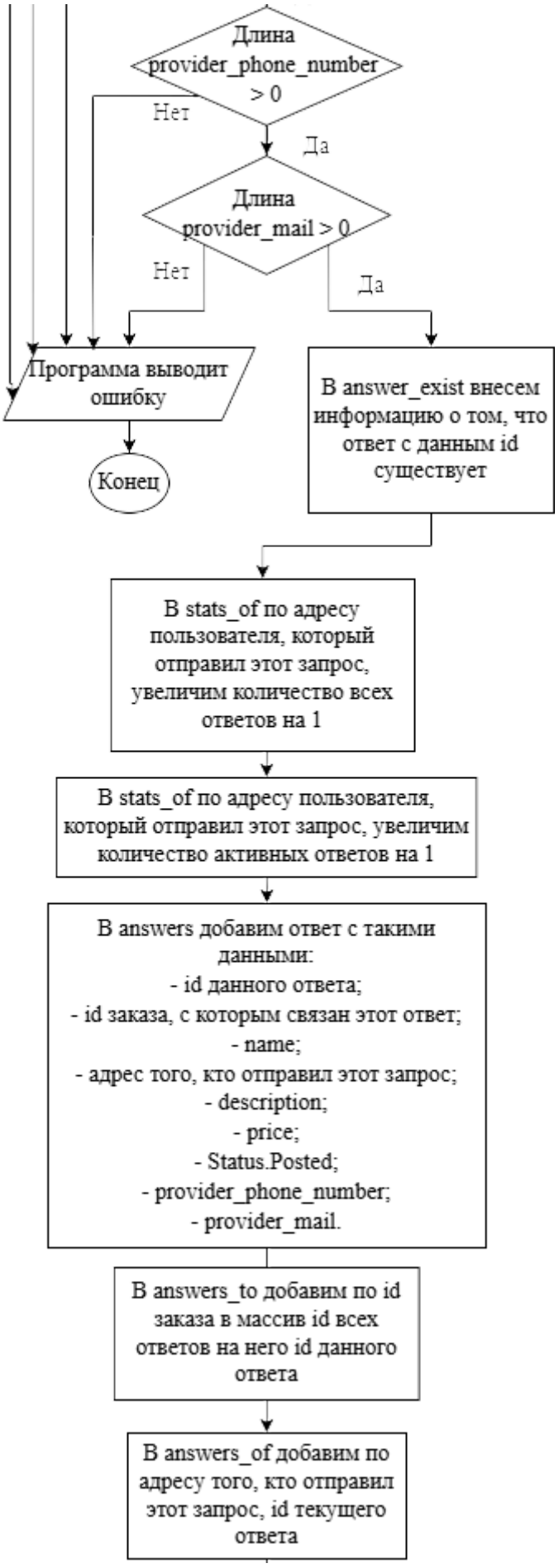
Приложение 1: Схема функции create_order




В stats_of по адресу пользователя, который отправил этот запрос, увеличим количество активных заказов на 1
В orders добавим заказ с такими данными:
- id данного заказа;
- адрес того, кто отправил этот запрос;
- name;
- description;
- Status.Posted;
- buyer_phone_number;
- buyer_mail.
В orders_of добавим по адресу того, кто отправил этот запрос, id текущего заказа
В stats увеличим количество всех заказов на 1
В stats увеличим количество активных заказов на 1
Конец

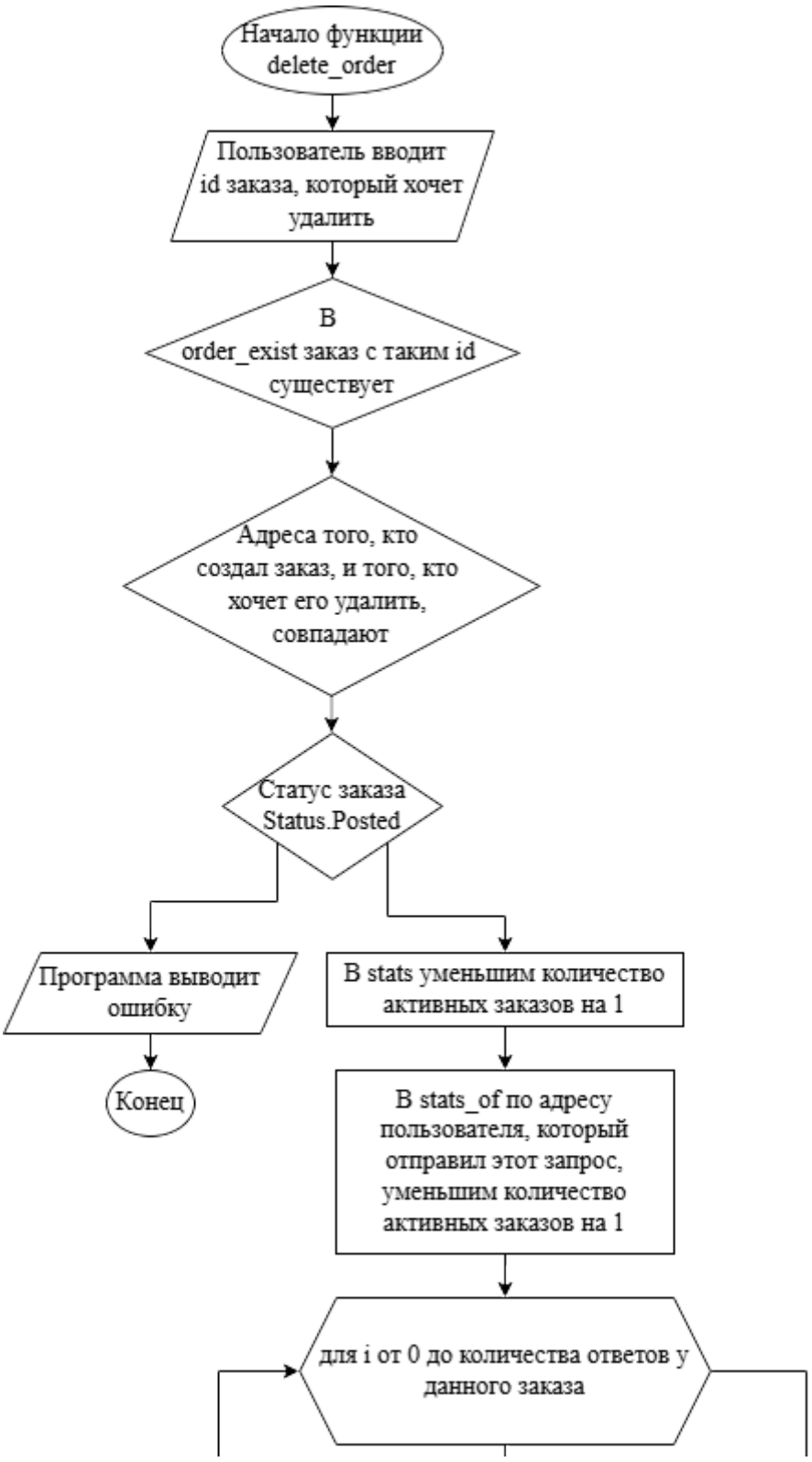
Приложение 2: Схема функции create_answer

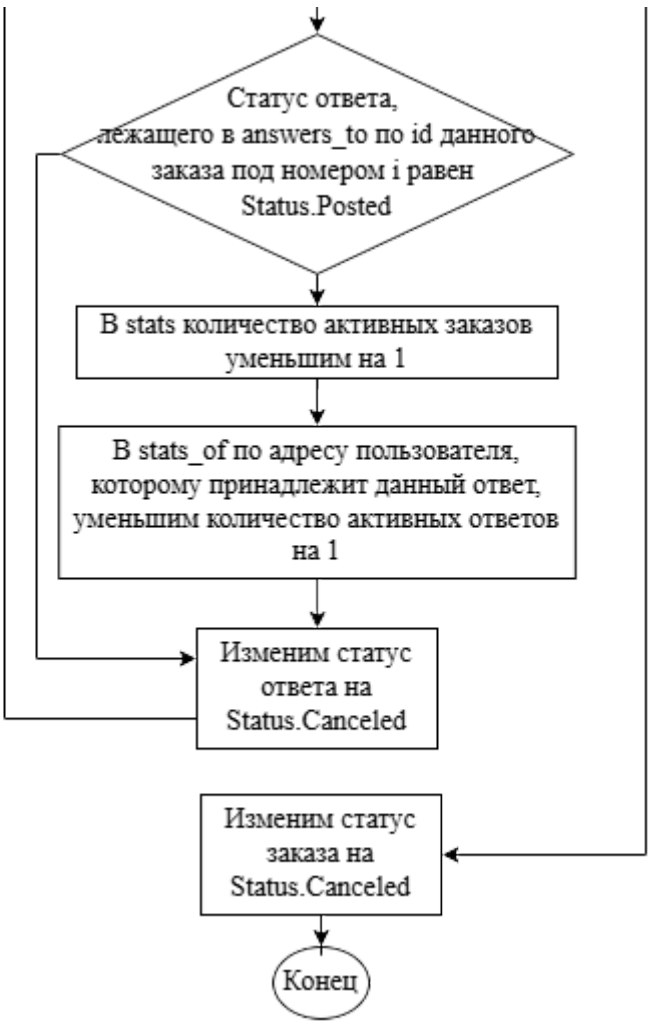



Длина provider_phone_number > 0
Нет
Да
Длина provider_mail > 0
Нет
Да
Программа выводит ошибку
Конец
В answer_exist внесем информацию о том, что ответ с данным id существует
В stats_of по адресу пользователя, который отправил этот запрос, увеличим количество всех ответов на 1
В stats_of по адресу пользователя, который отправил этот запрос, увеличим количество активных ответов на 1
В answers добавим ответ с такими данными:
- id данного ответа;
- id заказа, с которым связан этот ответ;
- name;
- адрес того, кто отправил этот запрос;
- description;
- price;
- Status.Posted;
- provider_phone_number;
- provider_mail.
В answers_to добавим по id заказа в массив id всех ответов на него id данного ответа
В answers_of добавим по адресу того, кто отправил этот запрос, id текущего ответа

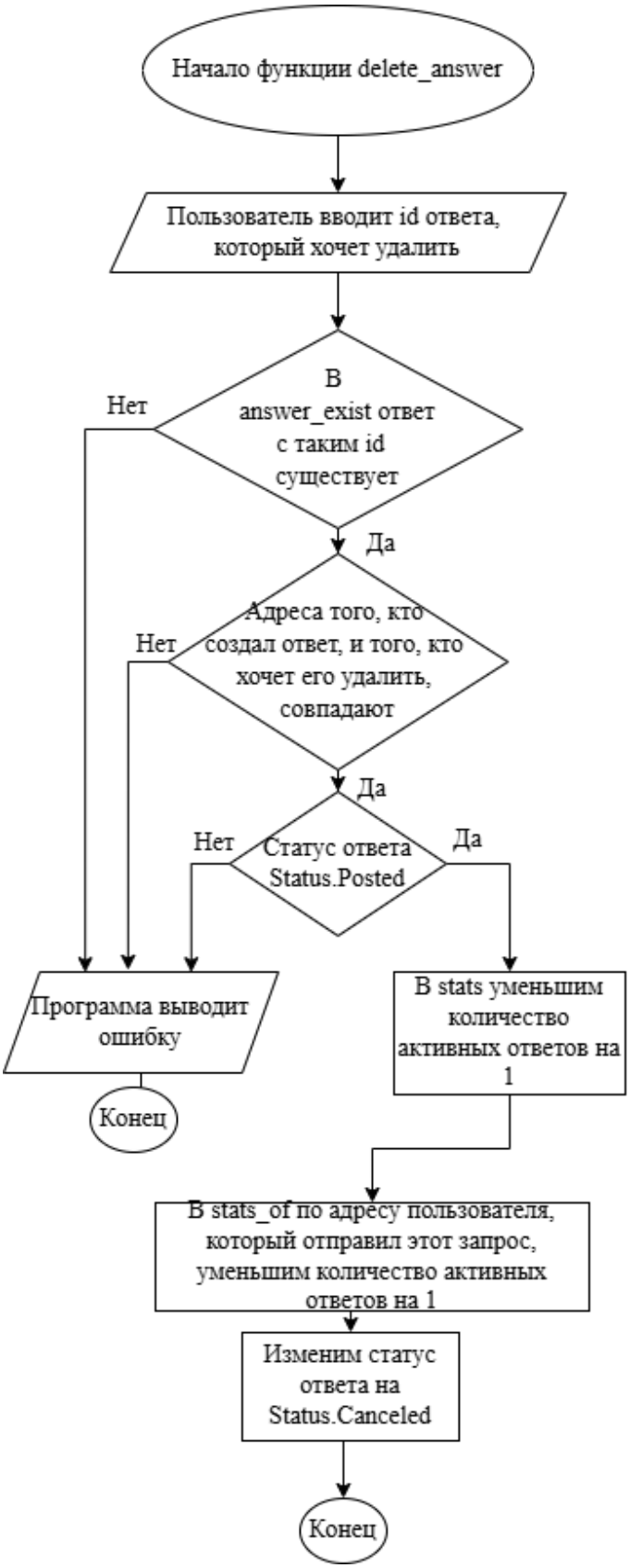

В stats увеличим количество всех ответов на 1
В stats увеличим количество активных ответов на 1
Конец

Приложение 3: Схема функции delete_order

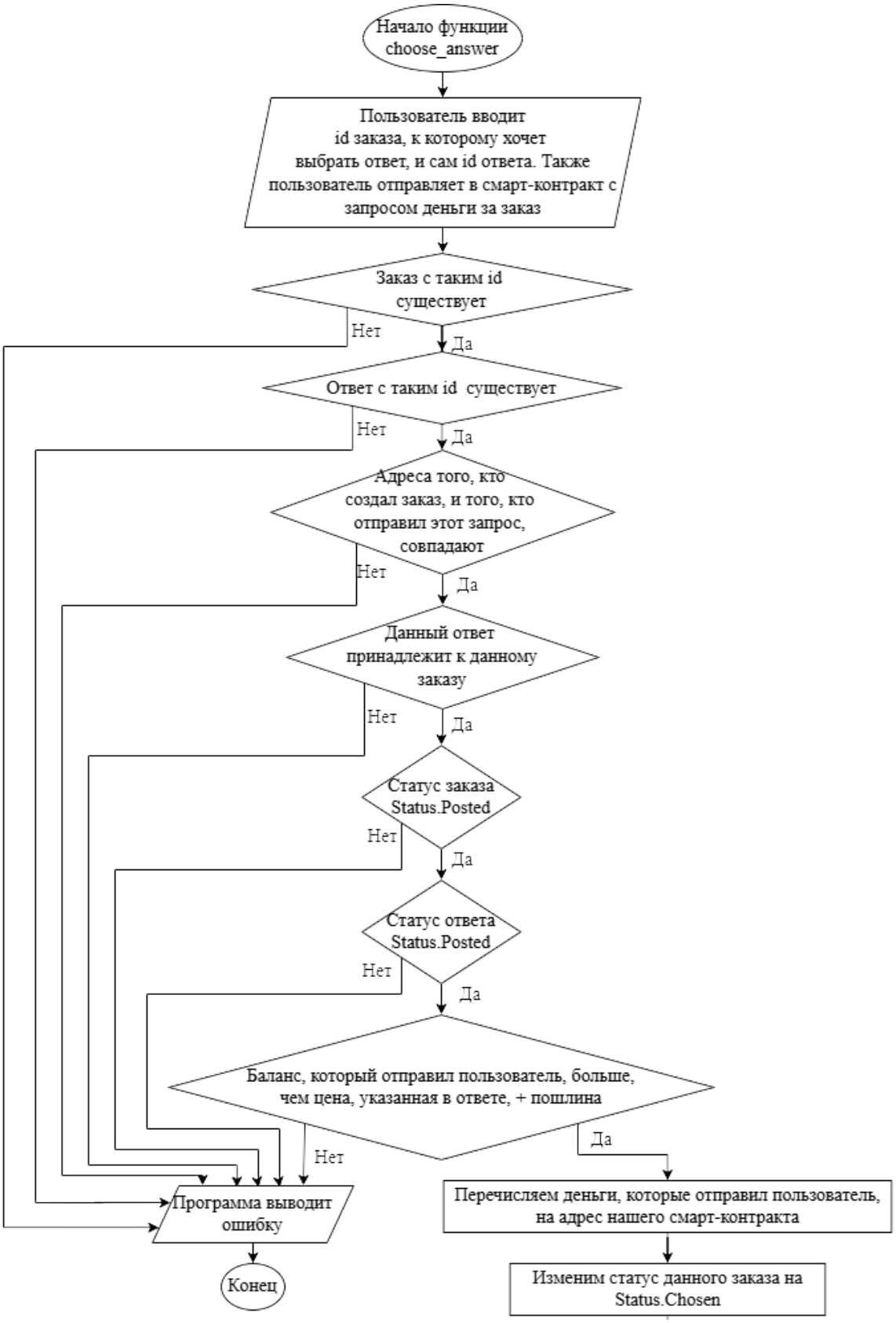


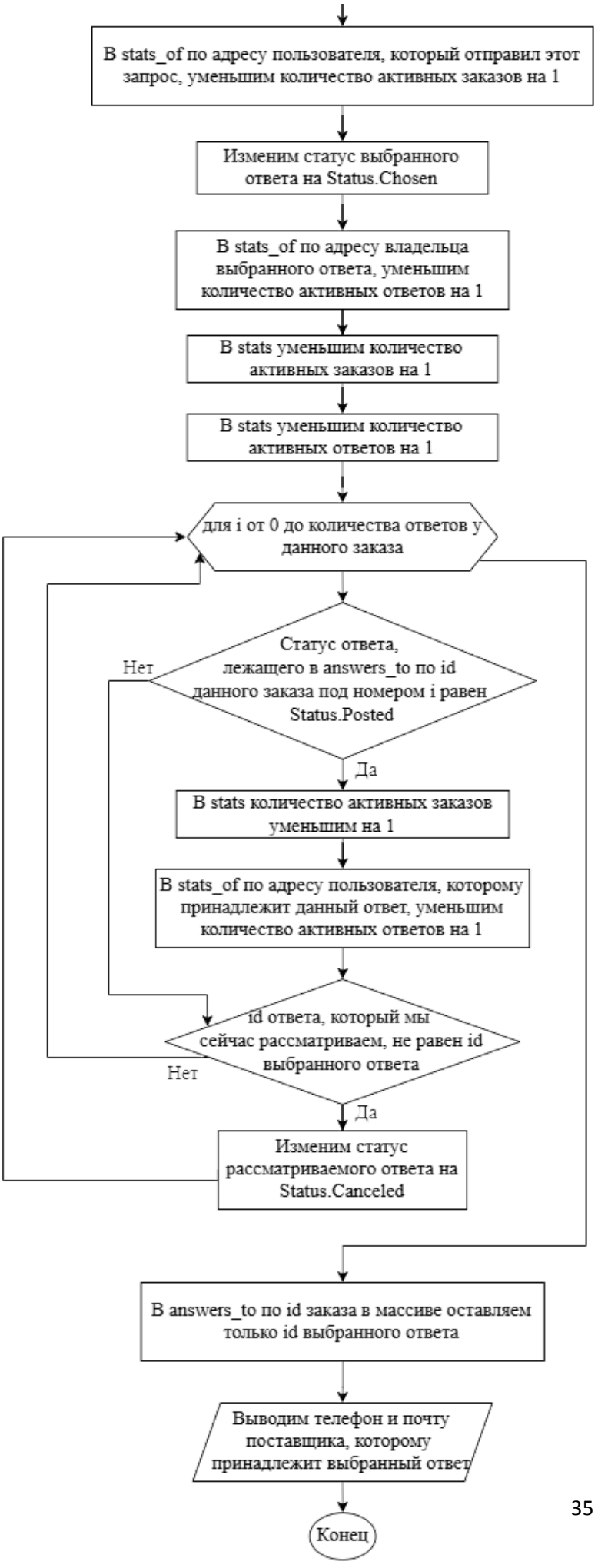

Статус ответа, лежащего в answers_to по id данного заказа под номером i равен Status.Posted
В stats количество активных заказов уменьшим на 1
В stats_of по адресу пользователя, которому принадлежит данный ответ, уменьшим количество активных ответов на 1
Изменим статус ответа на Status.Canceled
Изменим статус заказа на Status.Canceled
Конец

Приложение 4: Схема функции delete_answer

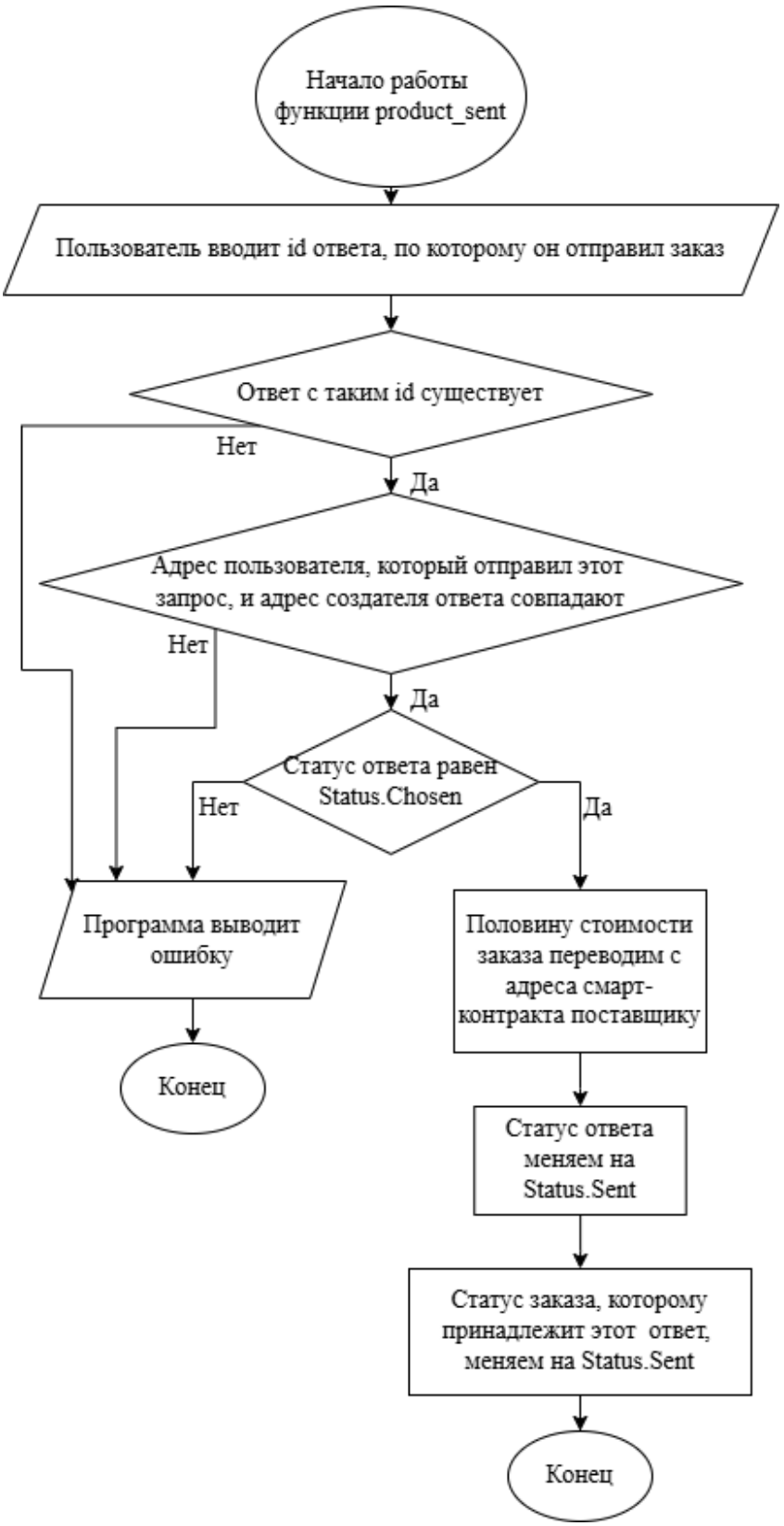


Приложение 5: Схема функции choose_answer




В stats_of по адресу пользователя, который отправил этот запрос, уменьшим количество активных заказов на 1
Изменим статус выбранного ответа на Status.Chosen
В stats_of по адресу владельца выбранного ответа, уменьшим количество активных ответов на 1
В stats уменьшим количество активных заказов на 1
В stats уменьшим количество активных ответов на 1
для i от 0 до количества ответов у данного заказа
Статус ответа, лежащего в answers_to по id данного заказа под номером i равен Status.Posted
Нет
Да
В stats количество активных заказов уменьшим на 1
В stats_of по адресу пользователя, которому принадлежит данный ответ, уменьшим количество активных ответов на 1
id ответа, который мы сейчас рассматриваем, не равен id выбранного ответа
Нет
Да
Изменим статус рассматриваемого ответа на Status.Canceled
В answers_to по id заказа в массиве оставляем только id выбранного ответа
Выводим телефон и почту поставщика, которому принадлежит выбранный ответ
Конец

Приложение 6: Схема функции product_sent



Приложение 7: Схема функции complete_order

Приложение 8: Текст смарт-контракта торговой платформы

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract Trading_Platform {

    fallback() external payable { }

    receive() external payable {  }


    struct TPStats {

        uint all_orders;

        uint current_orders;

        uint all_answers;

        uint current_answers;

    }

    enum Status {

        Posted,

        Chosen,

        Sent,

        Completed,

        Cancelled

    }

    struct Order {
```

```
    uint id;
    address addr_buyer;
    string name;
    string description;
    Status status;
    string buyer_phone_number;
    string buyer_mail;
}
struct Answer {
    uint id;
    uint id_order;
    string name;
    address addr_provider;
    string description;
    uint price;
    Status status;
    string provider_phone_number;
    string provider_mail;
}
struct Answer_out{
    uint id;
    string name;
```

```
    uint price;
    string description;
}
address payable  private owner;
TPStats private stats;
uint private fee;
Order[] private orders;
Answer[] private answers;
mapping(address => uint[]) private orders_of;
mapping(address => uint[]) private answers_of;
mapping(uint => uint[]) private answers_to;
mapping(address => TPStats) private stats_of;
mapping(uint => bool) private order_exist;
mapping(uint => bool) private answer_exist;
function change_owner(address payable new_owner) public{
    require(msg.sender == owner);
    owner = new_owner;
}
function get_owner() public view returns(address){
    return owner;
}
function get_balance() public view returns(uint){ return address(this).balance;}
```

```
function get_tp_stats() public view returns(TPStats memory){
    return stats;
}
function get_user_stats(address addr) public view returns(TPStats memory){
    return stats_of[addr];
}
function get_fee() public view returns(uint){
    return fee;
}
constructor(uint _fee){
    owner = payable(msg.sender);
    fee = _fee;
}
function create_order(
    string memory name,
    string memory description,
    string memory buyer_phone_number,
    string memory buyer_mail
) public {
    require(bytes(name).length > 0);
    require(bytes(description).length > 0);
    require(bytes(buyer_phone_number).length > 0);
```

```
    require(bytes(buyer_mail).length > 0);

    order_exist[stats.all_orders] = true;
    stats_of[msg.sender].current_orders++;
    stats_of[msg.sender].all_orders++;
    Order memory order;
    order.id = stats.all_orders;
    orders_of[msg.sender].push(stats.all_orders);

    stats.all_orders++;
    stats.current_orders++;

    order.addr_buyer = msg.sender;
    order.name = name;
    order.description = description;
    order.status = Status.Posted;
    order.buyer_phone_number = buyer_phone_number;
    order.buyer_mail = buyer_mail;
    orders.push(order);
}
function create_answer(
    uint id_order,
```

```
        string memory name,
        uint price,
        string memory description,
        string memory provider_phone_number,
        string memory provider_mail
    ) public{
        require(order_exist[id_order], "This order doesn't exist");
        require(orders[id_order].addr_buyer != msg.sender, "You don't have access to this order");
        require(orders[id_order].status ==  Status.Posted, "Wrong order status");
        require(bytes(name).length > 0);
        require(bytes(description).length > 0);
        require(bytes(provider_phone_number).length > 0);
        require(bytes(provider_mail).length > 0);

        answer_exist[stats.all_answers] = true;
        stats_of[msg.sender].current_answers++;
        stats_of[msg.sender].all_answers++;
        Answer memory answer;
        answer.id = stats.all_answers;
        answer.id_order = id_order;
        answers_of[msg.sender].push(stats.all_answers);
```

```
        answers_to[id_order].push(stats.all_answers);

        stats.all_answers++;
        stats.current_answers++;

        answer.addr_provider = msg.sender;
        answer.name = name;
        answer.status = Status.Posted;
        answer.price = price;
        answer.description = description;
        answer.provider_phone_number = provider_phone_number;
        answer.provider_mail = provider_mail;
        answers.push(answer);
    }
    function get_current_orders() public view returns(Order[] memory ret){
        ret = new Order[](stats.current_orders);
        uint elem = 0;
        for(uint id_order=0; id_order<stats.current_orders; id_order++){
            if(orders[id_order].status != Status.Cancelled && orders[id_order].status !=
Status.Completed){
                ret[elem] = orders[id_order];
                elem++;
```

```
            }
        }
        return ret;
    }
    function get_current_orders_of(address addr) public view returns(Order[]
memory ret){
        ret = new Order[](stats_of[addr].current_orders);
        uint elem = 0;
        for(uint i=0; i<stats_of[addr].all_orders; i++){
            if(orders[orders_of[addr][i]].status != Status.Cancelled &&
orders[orders_of[addr][i]].status != Status.Completed){
                ret[elem] = orders[orders_of[addr][i]];
                elem++;
            }
        }
        return ret;
    }
    function get_all_orders_of() public view returns(Order[] memory ret){
        ret = new Order[](stats_of[msg.sender].all_orders);
        uint elem = 0;
        for(uint i=0; i<stats_of[msg.sender].all_orders; i++){
            ret[elem] = orders[orders_of[msg.sender][i]];
```

```
            elem++;
        }
        return ret;
    }
    function get_all_answers_of() public view returns(Answer[] memory ret){
        ret = new Answer[](stats_of[msg.sender].all_answers);
        uint elem = 0;
        for(uint i=0; i<stats_of[msg.sender].all_answers; i++){
            ret[elem] = answers[answers_of[msg.sender][i]];
            elem++;
        }
        return ret;
    }
    function get_current_answers_of() public view returns(Answer[] memory ret){
        ret = new Answer[](stats_of[msg.sender].current_answers);
        uint elem = 0;
        for(uint i=0; i<stats_of[msg.sender].current_answers; i++){
            ret[elem] = answers[answers_of[msg.sender][i]];
            elem++;
        }
        return ret;
    }
```

```
    function get_answers_to(uint id_order) public view returns(Answer_out [] memory ret){
        require(order_exist[id_order], "14");
        ret = new Answer_out[](answers_to[id_order].length);
        for(uint i=0; i<answers_to[id_order].length; i++){
            ret[i] = Answer_out(answers_to[id_order][i], answers[answers_to[id_order][i]].name, answers[answers_to[id_order][i]].price, answers[answers_to[id_order][i]].description);
        }
        return ret;
    }
    function delete_order(uint id_order) public {
        require(order_exist[id_order], "This oreder doesn't exist");
        require(orders[id_order].addr_buyer == msg.sender, "You don't have access to this order");
        require(orders[id_order].status == Status.Posted, "Wrong order status");
        stats.current_orders --;
        stats_of[msg.sender].current_orders--;
        for(uint i=0; i<answers_to[id_order].length; i++){
            if(answers[answers_to[id_order][i]].status == Status.Posted){
                stats.current_answers--;

stats_of[answers[answers_to[id_order][i]].addr_provider].current_answers--;
```

```
            }
            answers[answers_to[id_order][i]].status = Status.Cancelled;
        }
        orders[id_order].status = Status.Cancelled;
    }
    function delete_answer(uint id_answer) public {
        require(answer_exist[id_answer], "This answer doesn't exist");
        require(answers[id_answer].addr_provider == msg.sender, "You don't have access to this answer");
        require(answers[id_answer].status == Status.Posted, "Wrong answer status");
        stats.current_answers --;
        stats_of[msg.sender].current_answers--;
        answers[id_answer].status = Status.Cancelled;
    }
    function choose_answer(uint id_order, uint id_answer) public payable returns(Answer memory){//Закупщик выбирает ответ поставщика и платит платформе
        require(order_exist[id_order], "Error 21");
        require(answer_exist[id_answer], "Error 22");
        require(msg.sender == orders[id_order].addr_buyer, "Error 23");
        require(answers[id_answer].id_order == id_order, "Error 24");
        require(orders[id_order].status == Status.Posted, "Error 25");
        require(answers[id_answer].status == Status.Posted, "Error 26");
```

```
    require(msg.value >= answers[id_answer].price + fee, "Error 27");


    payable(address(this)).transfer(msg.value); //Платим торговой платформе
    orders[id_order].status = Status.Chosen;
    stats_of[orders[id_order].addr_buyer].current_orders --;
    answers[id_answer].status = Status.Chosen;
    stats_of[answers[id_answer].addr_provider].current_answers --;
    stats.current_orders --;
    stats.current_answers --;
    for(uint i=0; i<answers_to[id_order].length; i++){
        if(answers[answers_to[id_order][i]].status == Status.Posted){
            stats.current_answers--;
            stats_of[msg.sender].current_answers--;
        }
        if(answers_to[id_order][i] != id_answer){
            answers[id_answer].status = Status.Cancelled;
        }
    }
    answers_to[id_order] = [id_answer];
    return answers[id_answer];
}
function pay_address(address payable addr, uint amount) private {
```

```
        payable(addr).transfer(amount);
    }
    function product_sent(uint id_answer) public {
        require(answer_exist[id_answer], "Error 28");
        require(msg.sender == answers[id_answer].addr_provider, "Error 29");
        require(answers[id_answer].status == Status.Chosen, "Error 30");
        pay_address(payable(msg.sender), answers[id_answer].price/2); //Половину стоимости заказа отправляем поставщику
        orders[answers[id_answer].id_order].status = Status.Sent;
        answers[id_answer].status = Status.Sent;
    }
    function complete_order(uint id_order) public {
        require(order_exist[id_order], "Q");
        require(msg.sender == orders[id_order].addr_buyer, "W");
        require(orders[id_order].status == Status.Sent, "E");
        pay_address(payable(msg.sender), answers[answers_to[id_order][0]].price - answers[answers_to[id_order][0]].price/2);//Отправляем поставщику вторую половину стоимости заказа
        pay_address(owner, fee); //Отправляем пошлину владельцу платформы
        orders[id_order].status = Status.Completed;
        answers[answers_to[id_order][0]].status = Status.Completed;
    }
}
```