Ethereum:

работа с сетью, контракты и распределенные приложения

12+

Автор: А.В. Бурков

Алексей Бурков

Ethereum: работа с сетью, смарт-контракты и распределенные приложения

«ЛитРес: Самиздат»

2020

Бурков А.

Ethereum: работа с сетью, смарт-контракты и распределенные приложения / А. Бурков — «ЛитРес: Самиздат», 2020

В представленном учебнике рассматривается создание смарт-контрактов для блокчейн-сети Ethereum на языке программирования Solidity в операционной системе Windows. Мы также опишем развертывание окружения для создания и тестирования смарт-контрактов и децентрализованных приложений (DApps). Более того, в завершение курса мы создадим свою собственную блокчейн-сеть.Данный учебный курс создан на базе ООО «Лаборатория цифровой трансформации» при поддержке ООО «Цифровые технологии».

Содержание

Введение

В настоящее время технологии распределенных реестров (блокчейн-технологии) проникают во многие сферы человеческой деятельности. Изначально технология блокчейн использовалась в финансовой сфере для создания криптовалют. Затем была разработана технология защищенного хранения небольших объемов информации. И наконец, после появления блокчейна Ethereum стало возможно создавать программы в блокчейн-сетях.

Блокчейн Ethereum обладает своей виртуальной машиной – EVM (Ethereum Virtual Machine). Данное программное обеспечение позволяет децентрализованно хранить и запускать программы внутри блокчейн-сети Ethereum. В такой роли блокчейн-сеть работает как некий суперкомпьютер, где программное обеспечение хранится и выполняется на множестве компьютеров (узлов), подключенных к блокчейн-сети.

Программы, выполняемые в EVM, называются смарт-контрактами. Наиболее популярным языком программирования смарт-контрактов в настоящее время является язык программирования Solidity. В основу языка программирования Solidity был положен язык Java Script. Поэтому если вы знаете такие языки программирования, как Java Script, Java или C++, то изучение Solidity будет для вас достаточно простым.

В представленном учебнике рассматривается создание смарт-контрактов для блокчейн-сети Ethereum на языке программирования Solidity в операционной системе Windows. Мы также рассмотрим развертывание окружения для создания и тестирования смарт-контрактов и децентрализованных приложений (DApps). Более того, в завершение курса мы создадим свою собственную блокчейн-сеть.

Весь учебник разбит на шесть недель. Неделя – это глава учебника, посвященная определенному разделу создания смарт-контрактов. Каждая неделя разбита на уроки. Урок – это определенная тема в изучении программирования смарт-контрактов.

Учебник состоит из следующих глав-недель.

• Неделя № 1. Развертывание рабочего окружения («песочницы») для создания и тестирования смарт-контрактов на языке программирования Solidity.

• Неделя № 2. Создание и тестирование простейших смарт-контрактов.

• Неделя № 3. Хранение и обработка данных в распределенных реестрах.

• Неделя № 4. Реализация игровых смарт-контрактов.

• Неделя № 5. Финансовые смарт-контракты.

• Неделя № 6. Интерфейс, тестирование и публикация смарт-контрактов.

Теперь рассмотрим применяемые в учебнике обозначения.

1. В учебнике применяется сквозная нумерация рисунков. То есть «рис. 3.5.1» обозначает первый рисунок пятого урока третьей недели.

2. На рисунках важные места интерфейса выделены красными стрелками.

3. В тексте учебника встречаются замечания, выделенные серым цветом. Замечания – это важная или справочная информация, непосредственно не связанная с темой урока.

4. Некоторые большие блоки кода вынесены в приложение.

5. В тексте ссылки на источники информации обозначаются в квадратных скобках. Например, [4].

Для разработки смарт-контрактов нам необходимо установить следующее программное обеспечение (ПО).

1. Visual Studio Code и расширение для работы с языком программирования Solidity (https://code.visualstudio.com/).

2. Node.js – компилятор JavaScript в машинный код (https://Node.js.org/ru/).

3. Фреймворк Truffle (https://www.trufflesuite.com/truffle).
4. Эмулятор Ganache (https://www.trufflesuite.com/ganache).
5. Криптокошелек MetaMask (https://MetaMask.io/).
6. Офлайн-криптокошелек MyEtherWallet (https://github.com/kvhnuke/etherwallet/releases).
7. Установочный пакет блокчейн-сети Geth (https://geth.ethereum.org/downloads/).

Все описание установки вышеперечисленного ПО описано в уроках первой недели.

Данный учебник предназначен для читателей, желающих освоить разработку смарт-контрактов на языке программирования Solidity. Учебник также подойдет тем, кто планирует развернуть собственную блокчейн-сеть и создавать свои децентрализованные приложения (Dapp).

Для успешного изучения материала, представленного в учебнике, желательно иметь начальные знания по технологии распределенных реестров (технологии блокчейн), желательно иметь базовый опыт программирования в таких языках программирования, как Java, Java Script или C++.

> **Замечание.** В данном учебнике не приведены основы технологии блокчейн и основы программирования, а рассматривается только технология создания смарт-контрактов для блокчейна Ethereum в операционной системе Windows. Для изучения основ технологии блокчейн можно воспользоваться нашим курсом на учебном портале Stepik по ссылке: https://stepik.org/54926.
>
> Также можно сдать аттестационный тест и получить сертификат по основам технологии блокчейн. Аттестационный тест расположен по адресу https://stepik.org/57910.
>
> Электронная версия данного учебного курса размещена на учебном портале Stepik по адресу https://stepik.org/course/60331/syllabus?auth=registration. В конце каждого урока электронной версии добавлен небольшой аттестационный тест, а в конце каждой недели – практические задания для самостоятельного выполнения. Тем, кто сдаст все тесты и выполнит все практические задания, выдается сертификат по разработке смарт-контрактов и распределенных приложений (DApps) для блокчейн-сети Ethereum в операционной системе Windows.

Итак, давайте начнем изучать создание смарт-контрактов для блокчейна Ethereum на языке программирования Solidity.

Неделя № 1. Развертывание рабочего окружения («песочницы») для создания и тестирования смарт-контрактов на языке программирования Solidity

Введение

Эта неделя будет посвящена установке и настройке окружения для создания и тестирования смарт-контрактов для блокчейна Ethereum. В качестве языка программирования будем использовать язык программирования смарт-контрактов Solidity, а в качестве среды разработки – Visual Studio Code (VS Code). Для создания проектов будем использовать фреймворк Truffle, для запуска и тестирования наших смарт-контрактов – эмулятор блокчейна Ethereum Ganache, а в качестве криптокошелька – расширение для браузера MetaMask и офлайн-криптокошелек MyEtherWallet.

Рассмотрим пошагово установку перечисленного ПО. После установки необходимого ПО мы протестируем его работу на тестовом смарт-контракте из фреймворка Truffle.

Урок 1. Установка Visual Studio Code

Аннотация. В данном уроке мы рассмотрим процесс установки среды разработки (IDE) Visual Studio Code [1].

Среда разработки (IDE) Visual Studio Code – это удобный инструмент для написания кода смарт-контрактов. Он позволяет как создавать сам код, так и производить его отладку.

Для начала нам необходимо скачать установочный пакет VS Code. Это можно сделать с официального сайта Visual Studio Code, расположенного по адресу https://code.visualstudio.com/.

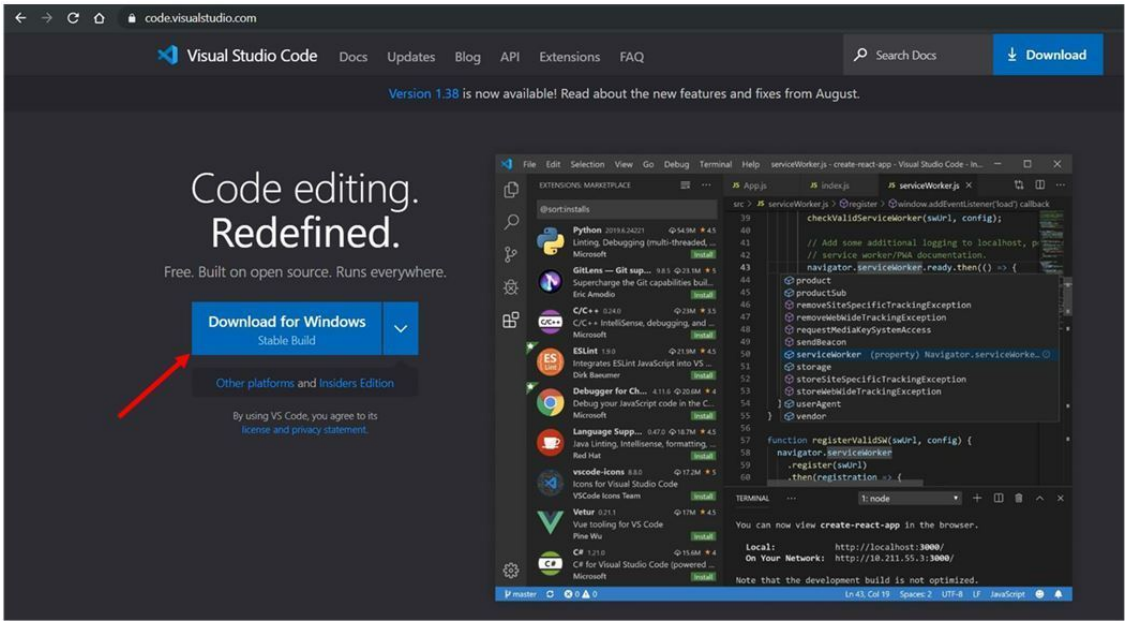

Рис. 1.1.1

Для скачивания версии для операционной системы Windows необходимо нажать кнопку Download for Windows (рис. 1.1.1).

В появившемся окне укажите место, куда будет сохранен установочный пакет VS Code, и нажмите кнопку «Сохранить» (рис. 1.1.2).

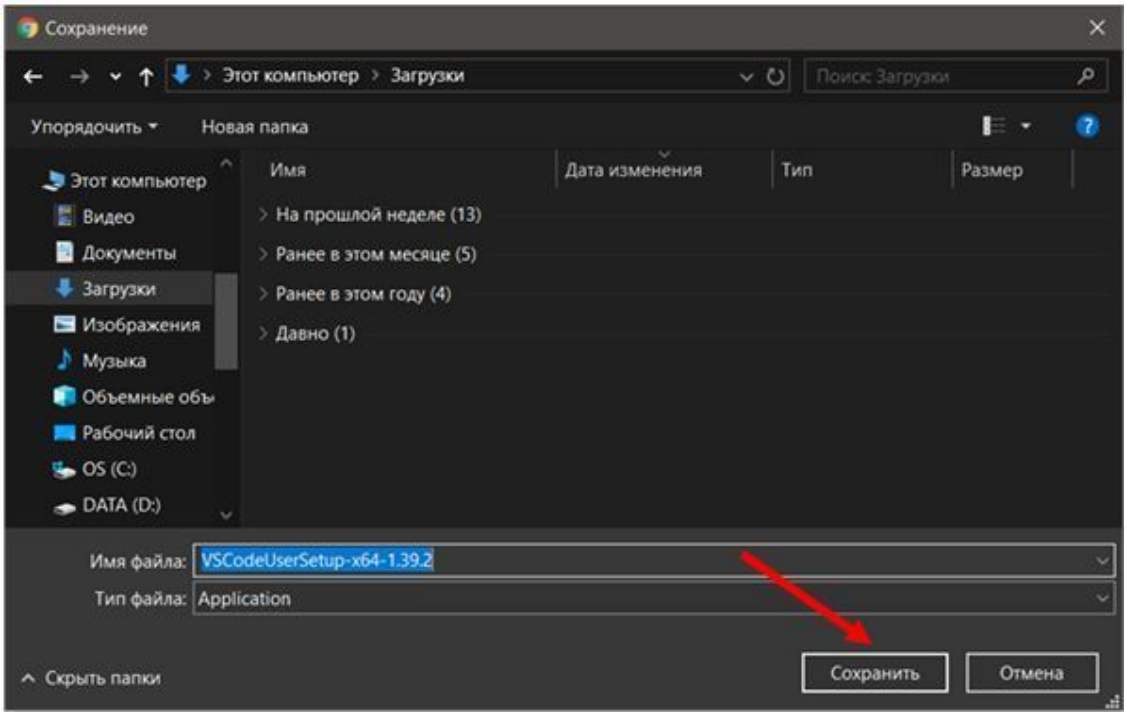

Рис. 1.1.2

Запустите скачанный установочный пакет. Появится окно с лицензионным соглашением (рис. 1.1.3).

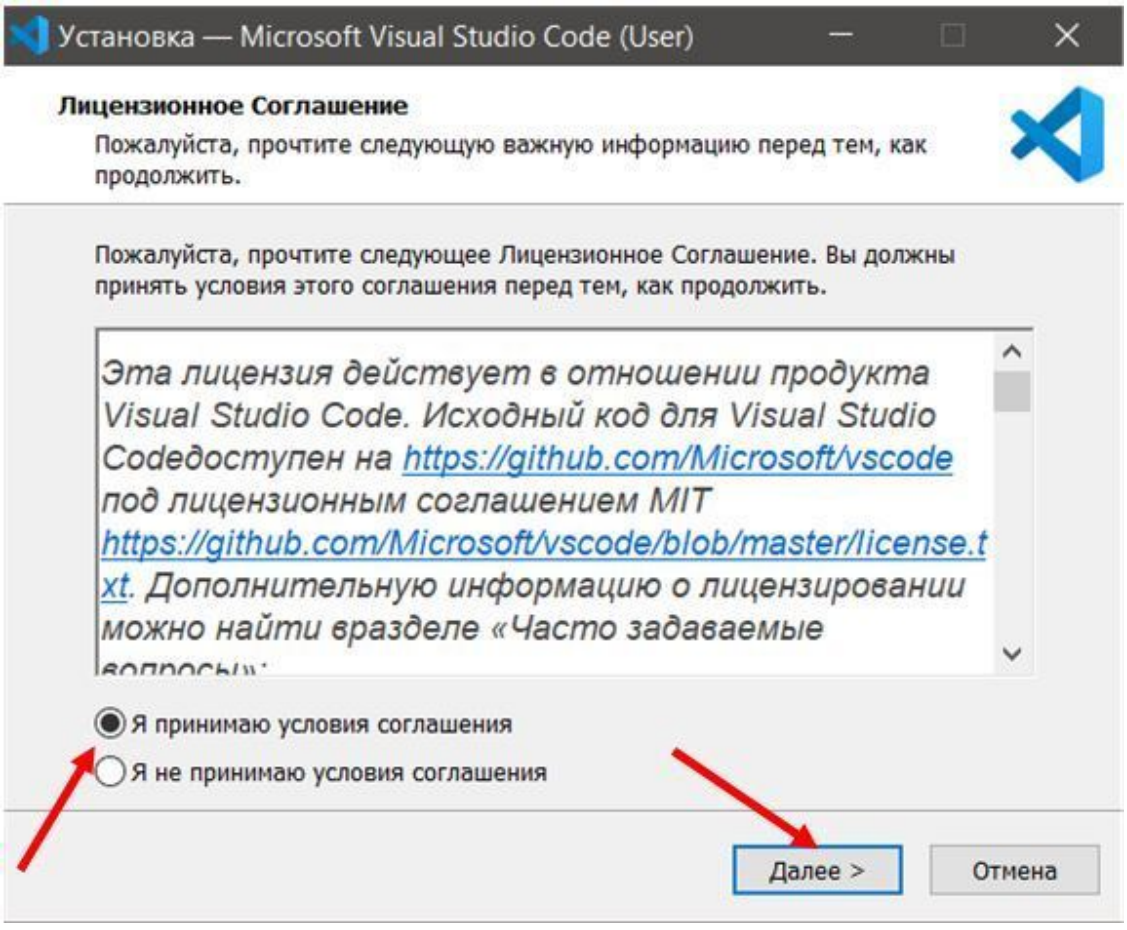

Рис. 1.1.3

Выберите пункт «Я принимаю условия соглашения» и нажмите кнопку «Далее». Появится окно с выбором места установки (рис. 1.1.4).

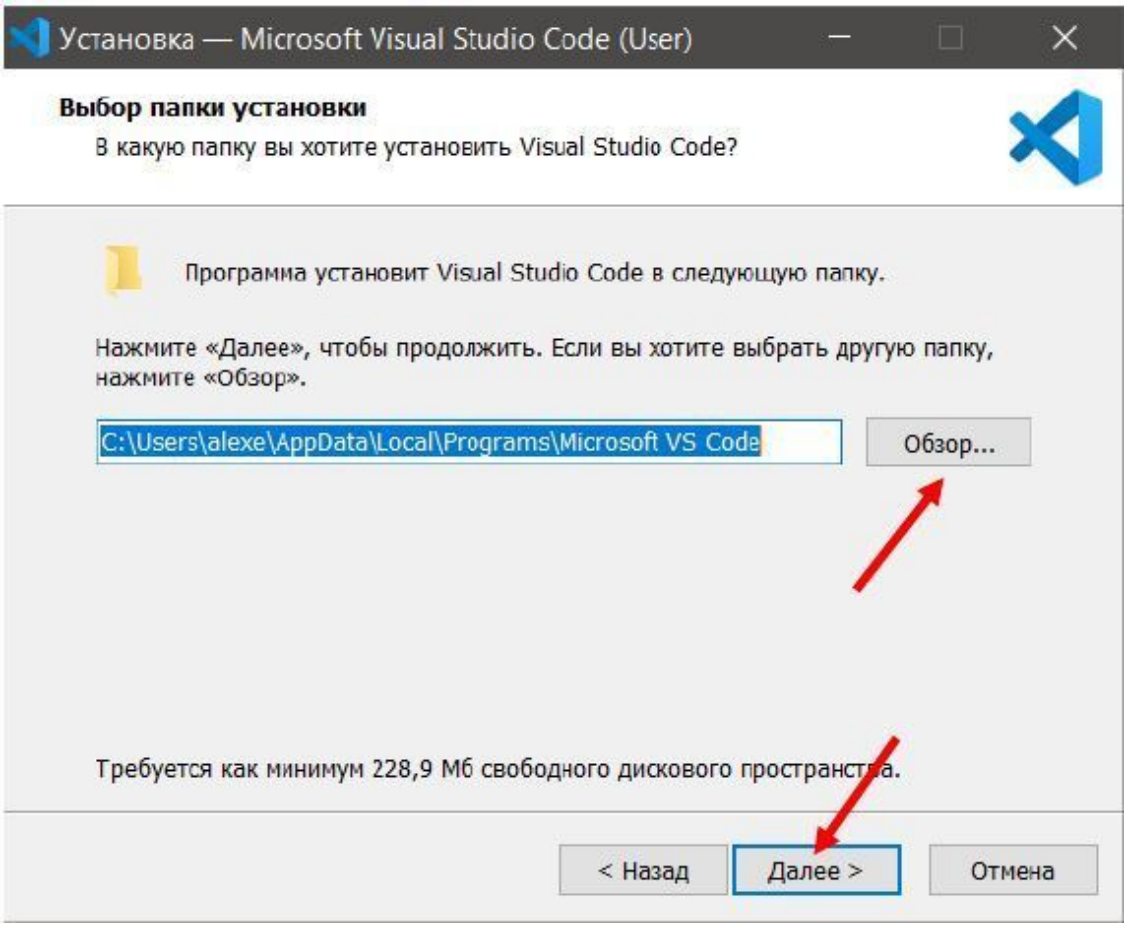

Рис. 1.1.4

Здесь можно оставить все по умолчанию либо изменить место установки, нажав кнопку «Обзор…». Затем нажмите кнопку «Далее». Появится окно настройки папки в меню «Пуск» (рис. 1.1.5).

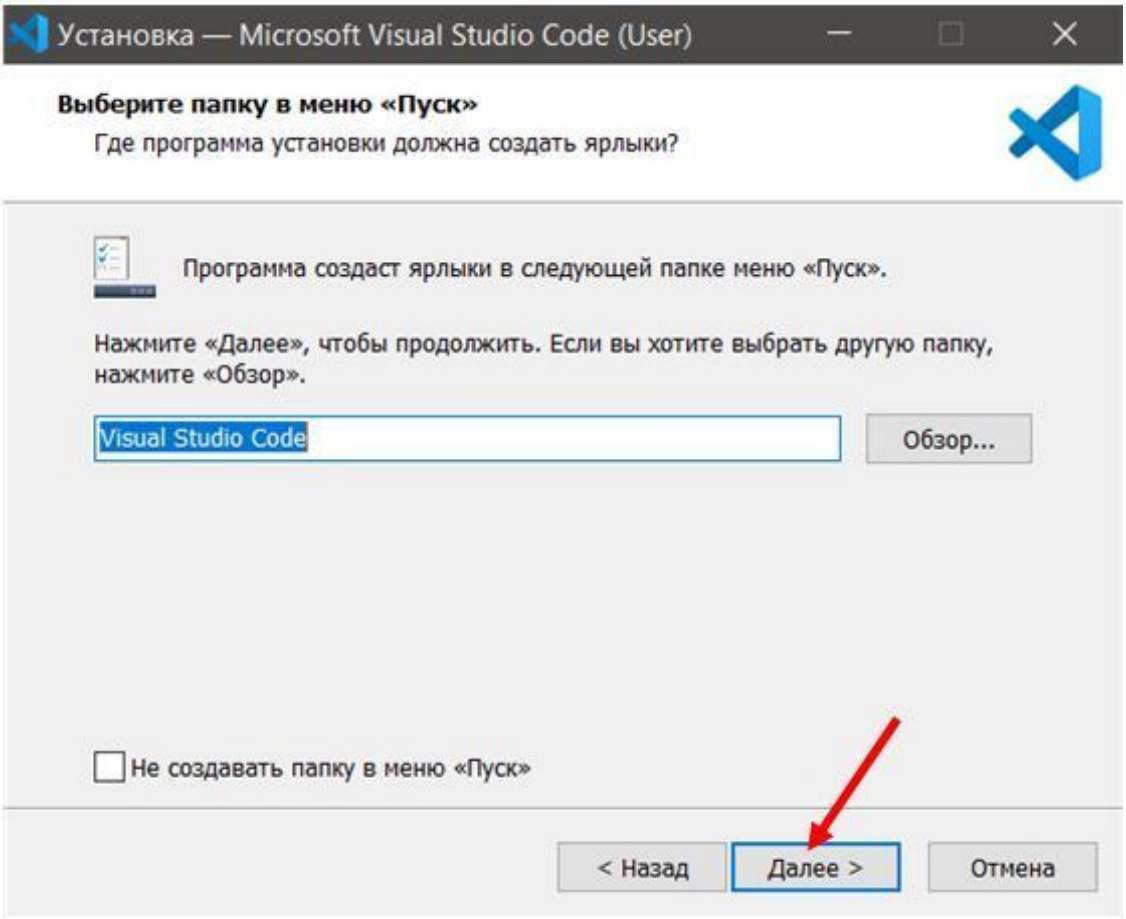

Рис. 1.1.5

В данном окне просто нажмите кнопку «Далее». Появится окно с дополнительными настройками установки (рис. 1.1.6).

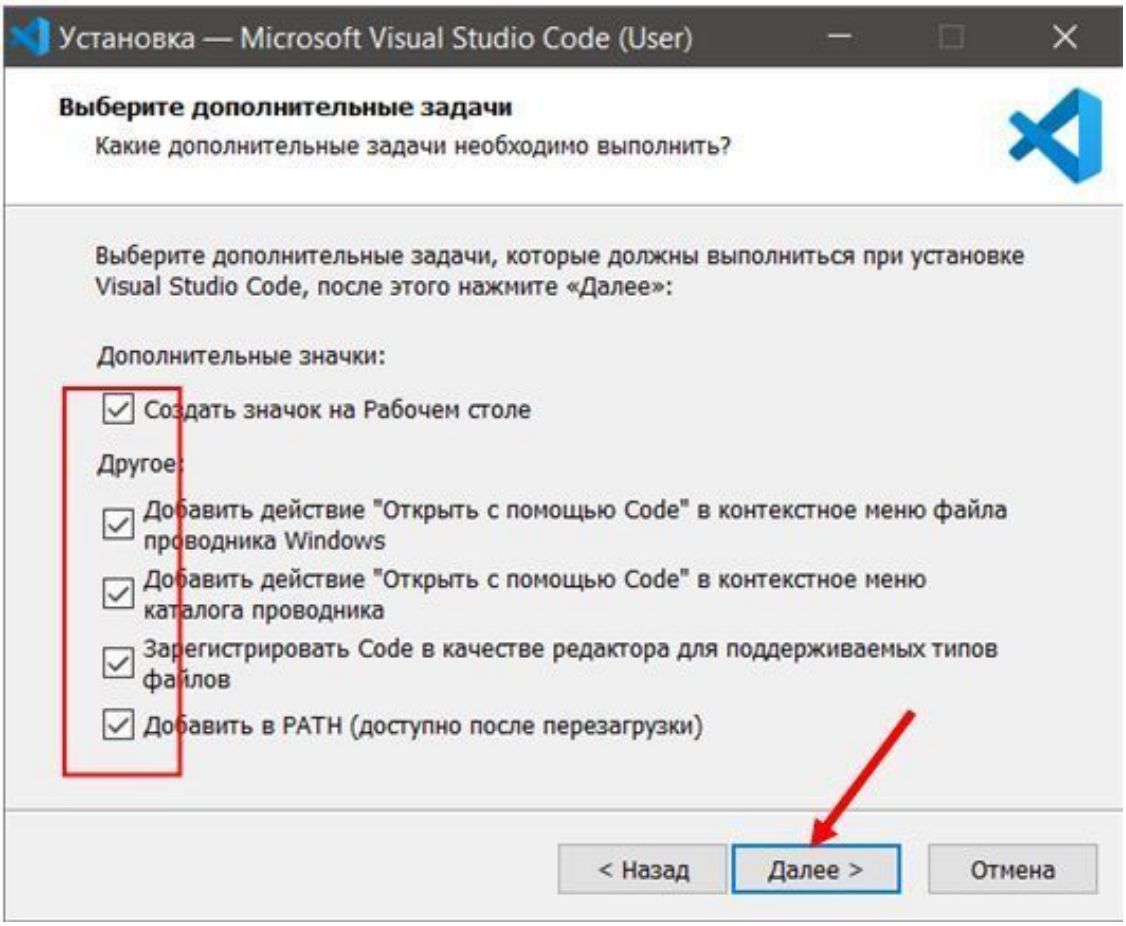

Рис. 1.1.6

Установите настройки как показано на рис. 1.1.6 и нажмите кнопку «Далее». Появится окно (рис. 1.1.7).

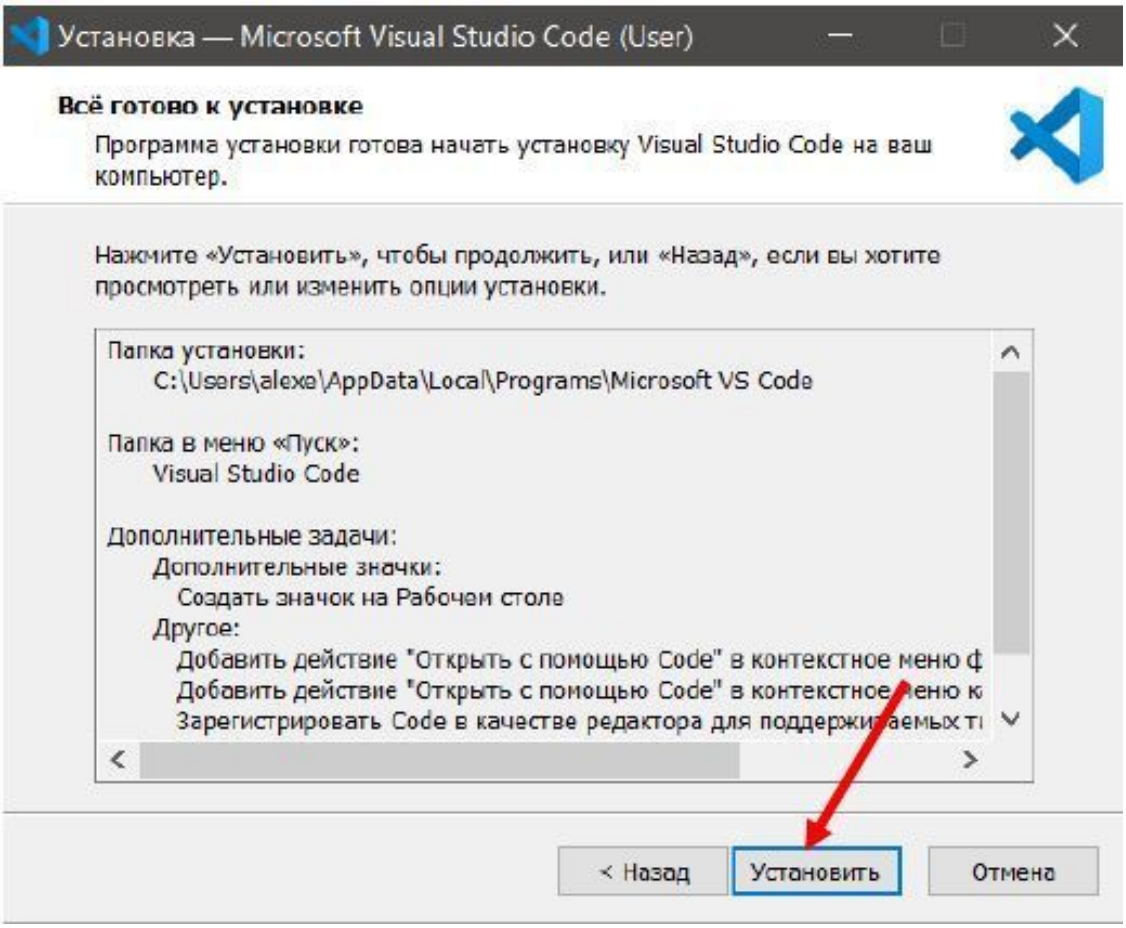

Рис. 1.1.7

Нажмите кнопку «Установить», начнется процесс установки. Затем появится окно завершения установки (рис. 1.1.8).

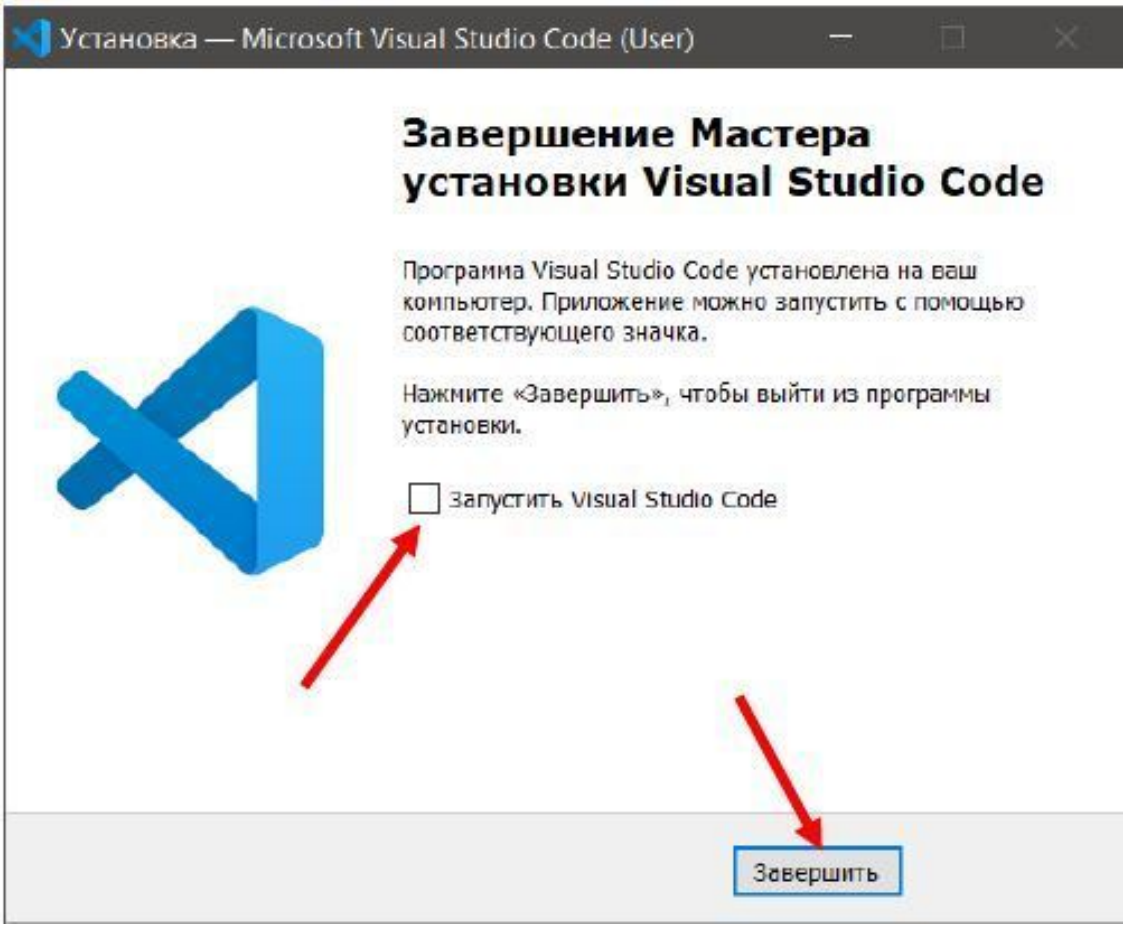

Рис. 1.1.8

Отключите опцию «Запустить Visual Studio Code» и нажмите кнопку «Завершить». **Перезагрузите компьютер!** На этом установка VS Code завершена. Перейдем к установке расширения для работы с Solidity.

Урок 2. Установка расширения Visual Studio Code для работы с Solidity

Аннотация. В данном уроке мы рассмотрим, как установить в VS Code расширение для работы с языком программирования смарт-контрактов Solidity [1].

Изначально VS Code не поддерживает язык программирования смарт-контрактов, поэтому нам необходимо установить специальное расширение.

Запустите Visual Studio Code, дважды щелкнув по значку на «Рабочем столе» или в меню «Пуск» (рис. 1.2.1).

Рис. 1.2.1

Откроется окно VS Code, где на панели слева необходимо открыть раздел EXTENSIONS: MARKETPLACE («Магазин расширений»), щелкнув по нему (рис. 1.2.2).

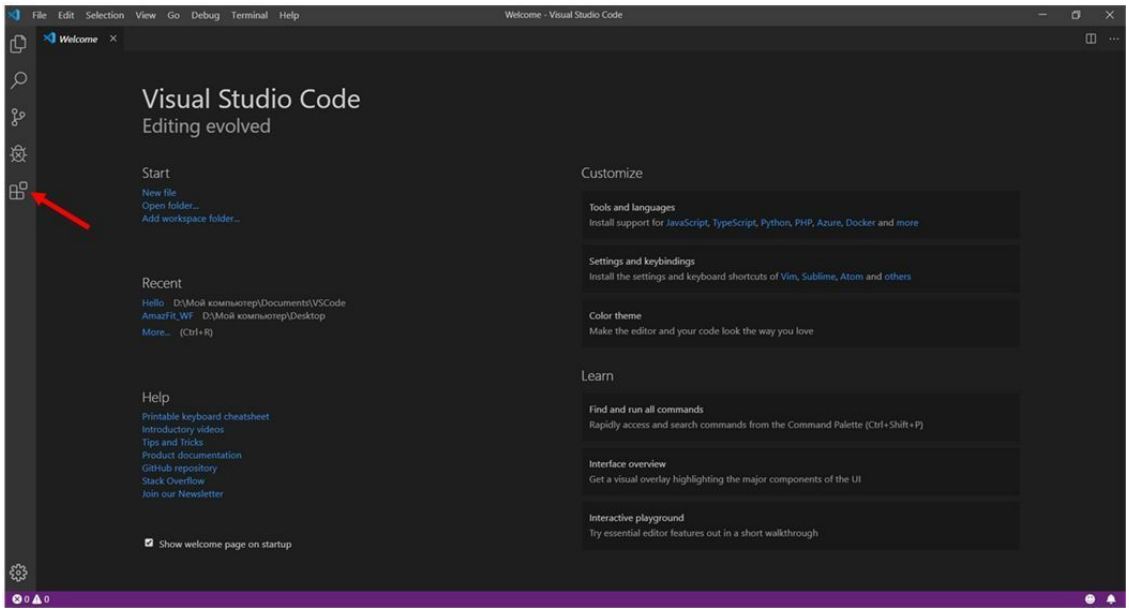

Рис. 1.2.2

В строке поиска панели EXTENSIONS: MARKETPLACE набираем слово Solidity и нажимаем на клавиатуре клавишу «Enter». В результатах поиска выбираем первый пункт «solidity… Juan Blanco» (может быть не первым). Затем на вкладке Extension: solidity щелкаем по ссылке Install (рис. 1.2.3).

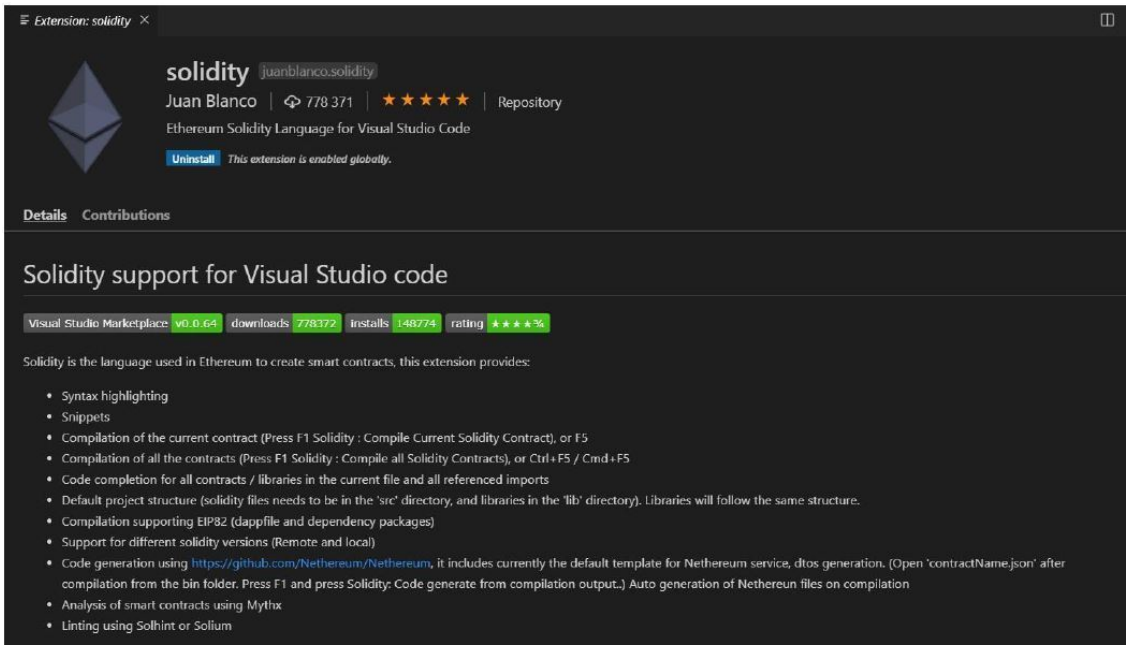

Рис. 1.2.3

Начнется установка расширения. По окончании установки вкладка Extension: solidity будет выглядеть как на рис. 1.2.4.

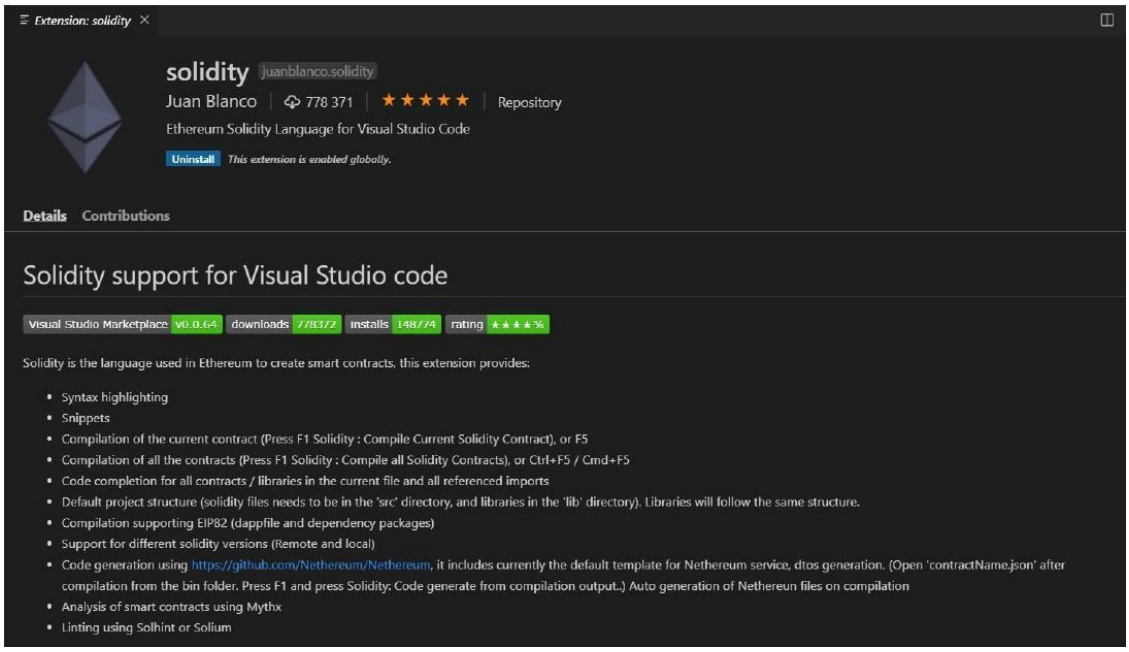

Рис. 1.2.4

На этом мы завершаем установку расширения VS Code для работы с Solidity и переходим к установке компилятора Node.js. **Закройте VS Code!**

Урок 3. Установка компилятора Node.js

Аннотация. В данном уроке рассматривается установка компилятора смарт-контрактов Node.js [2].

Мы будем создавать наши смарт-контракты на языке программирования Solidity, похожем на JavaScript. Но блокчейн Ethereum не понимает JavaScript. Нам необходимо конвертировать наш смарт-контракт на Solidity в машинный (бинарный) код. Для этого мы будем использовать компилятор Node.js.

Для установки компилятора в браузере откройте сайт https://Node.js.org/ru/ (рис. 1.3.1).

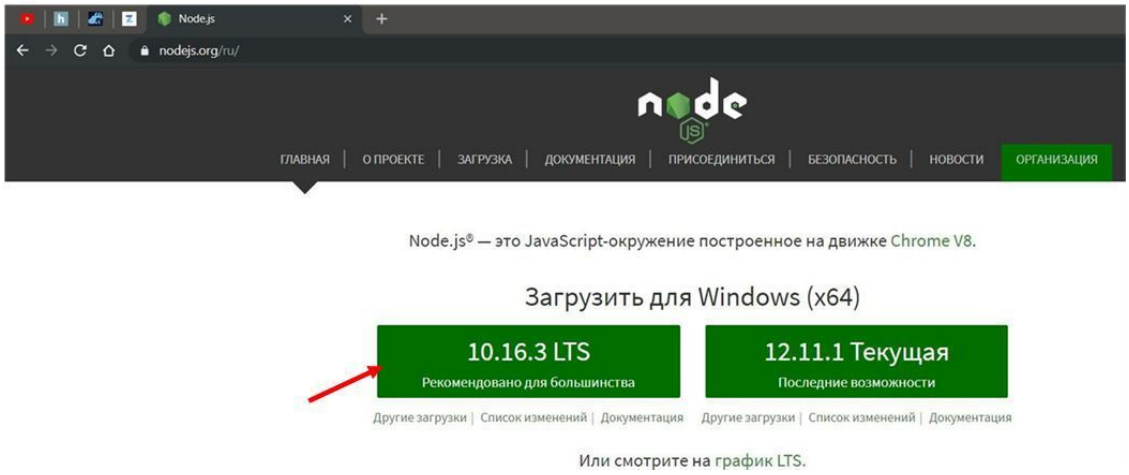

Рис. 1.3.1

На странице Node.js нажмите ссылку «10.16.3 LTS» для загрузки стабильной версии компилятора или ссылку «12.11.1 Текущая» для загрузки последней версии компилятора. После окончания загрузки установочного пакета его необходимо запустить. Появится окно начала установки (рис. 1.3.2).

Рис. 1.3.2

После нажатия кнопки Next появится окно с лицензионным соглашением (рис. 1.3.3).

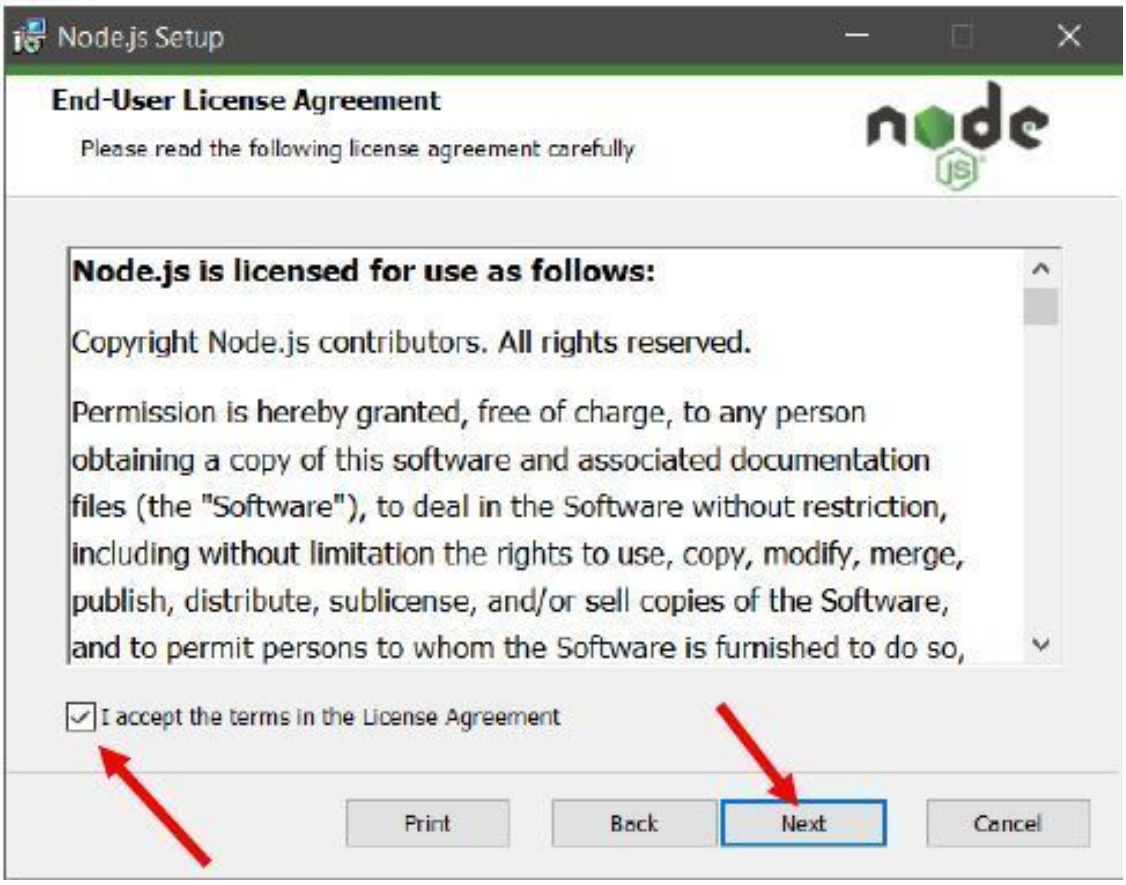

Рис. 1.3.3

Включите переключатель «I agree the terms…» и нажмите кнопку Next. Появится окно выбора папки для установки компилятора (рис. 1.3.4).

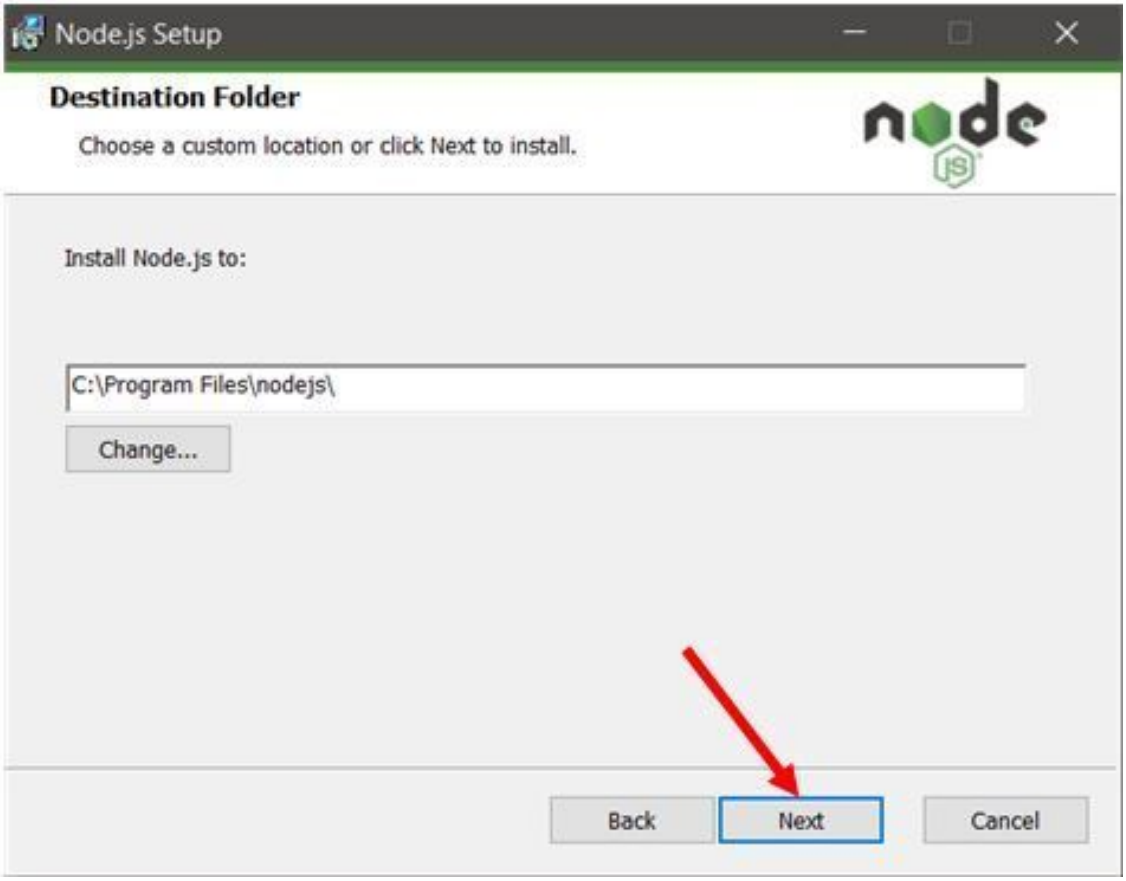

Рис. 1.3.4

Здесь можно задать папку, нажав кнопку «Change…». Затем нажмите кнопку Next. Появится окно выбора устанавливаемых компонентов компилятора (рис. 1.3.5).

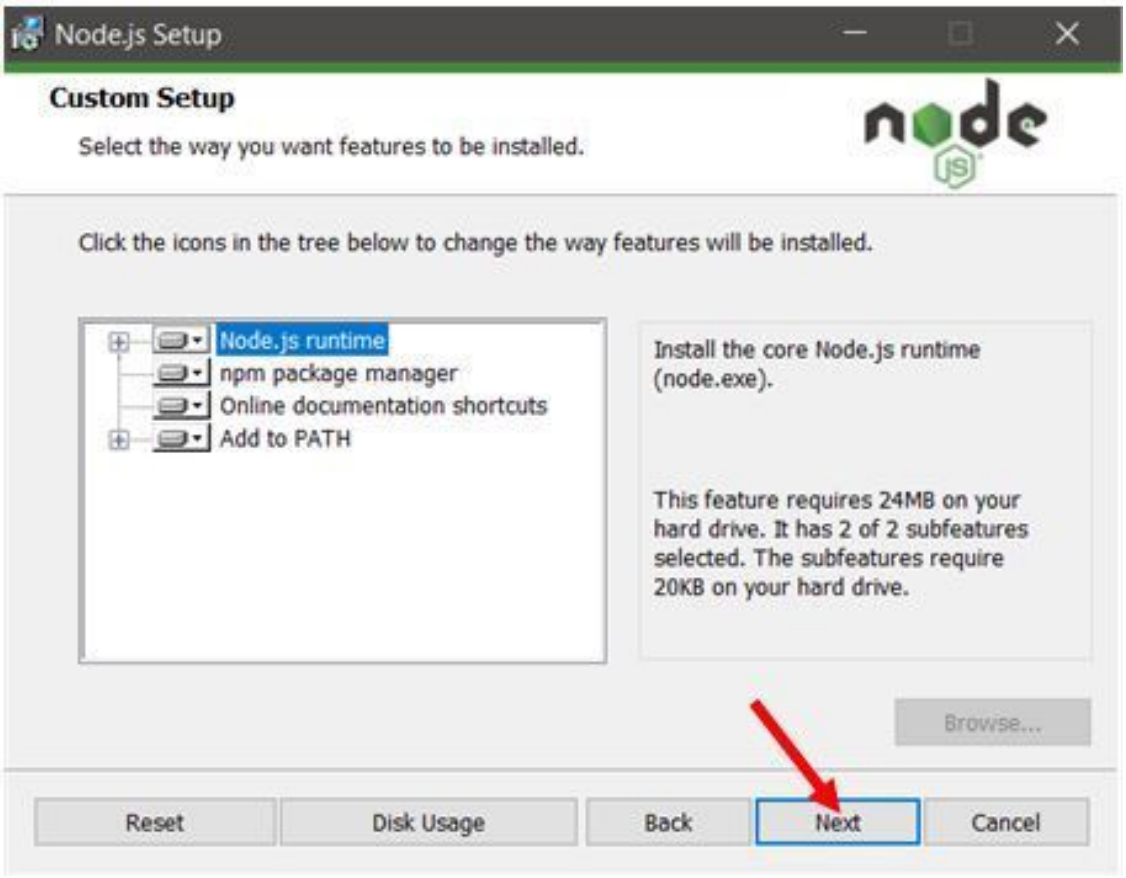

Рис. 1.3.5

Оставьте эти настройки без изменений и нажмите кнопку Next. Появится окно начала установки (рис. 1.3.6).

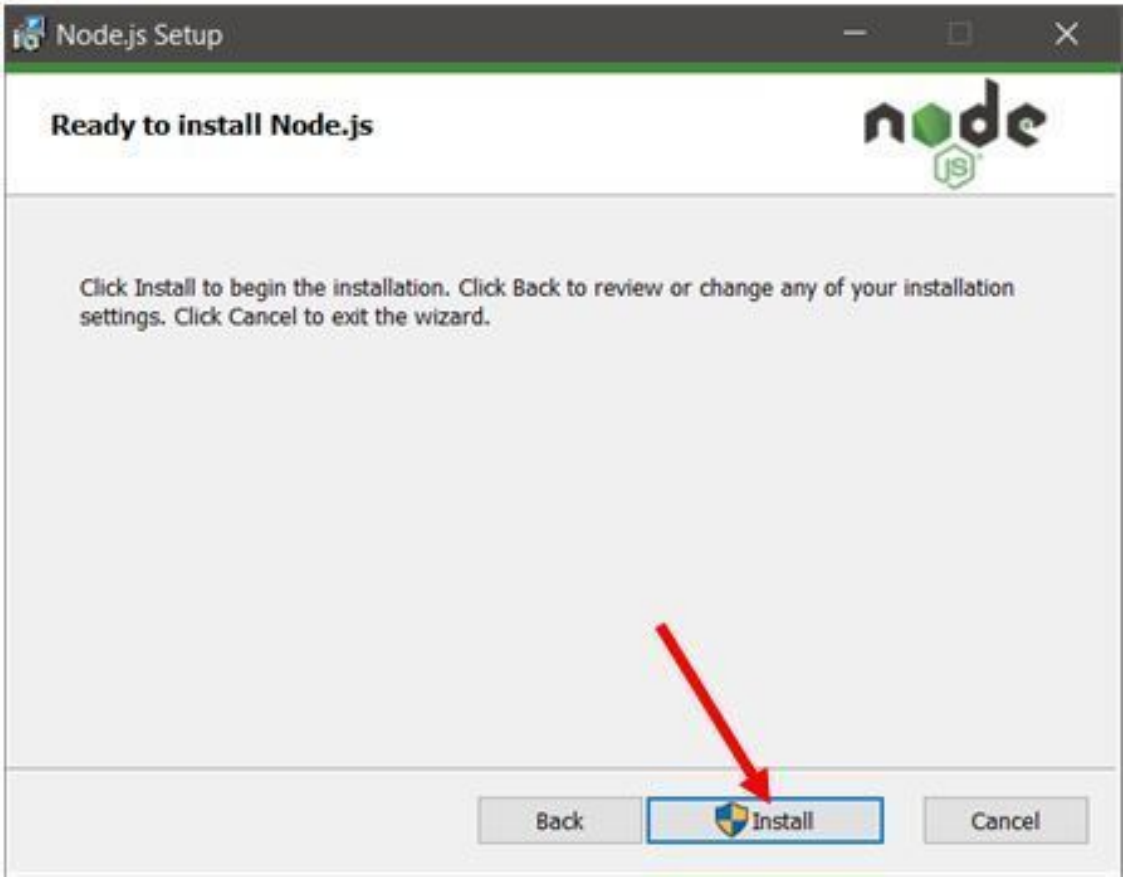

Рис. 1.3.6

В данном окне нажмите кнопку Install. Начнется процесс установки компилятора. По окончании установки появится финальное окно (рис. 1.3.7).

Рис. 1.3.7

Для завершения установки компилятора Node.js нажмите кнопку Finish.

Урок 4. Тестирование Node.js и подключение фреймворка Truffle

Аннотация. В данном уроке мы протестируем работу компилятора Node.js, а также установим и протестируем фреймворк Truffle [3].

Теперь протестируем работу компилятора. Запустите VS Code от имени администратора. Для этого на «Рабочем столе» или в меню «Пуск» щёлкните правой кнопкой мыши по значку Visual Studio Code и в появившемся меню выберите пункт «Запуск от имени администратора» (рис. 4.1).

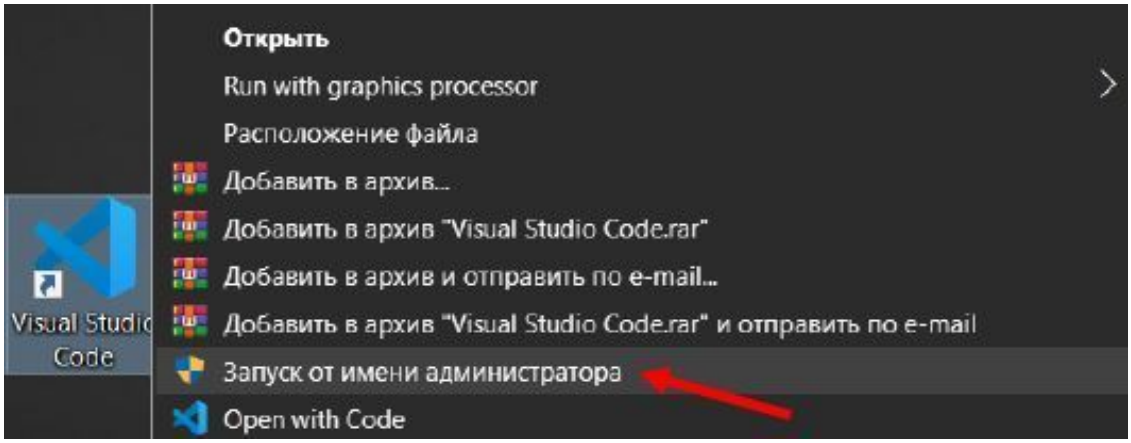

Рис. 1.4.1

В появившемся окне VS Code откроем терминал. Терминал нам будет необходим для ввода различных команд. Например, для управления компилятором и другими инструментами.

> **Замечание.** В качестве альтернативы терминалу в VC Code можно использовать утилиту Windows PowerShell (впрочем, она и запускается внутри VC Code в виде терминала).

Для того чтобы запустить терминал в оконном меню VS Code, выберите пункт Terminal / New Terminal (рис. 1.4.2).

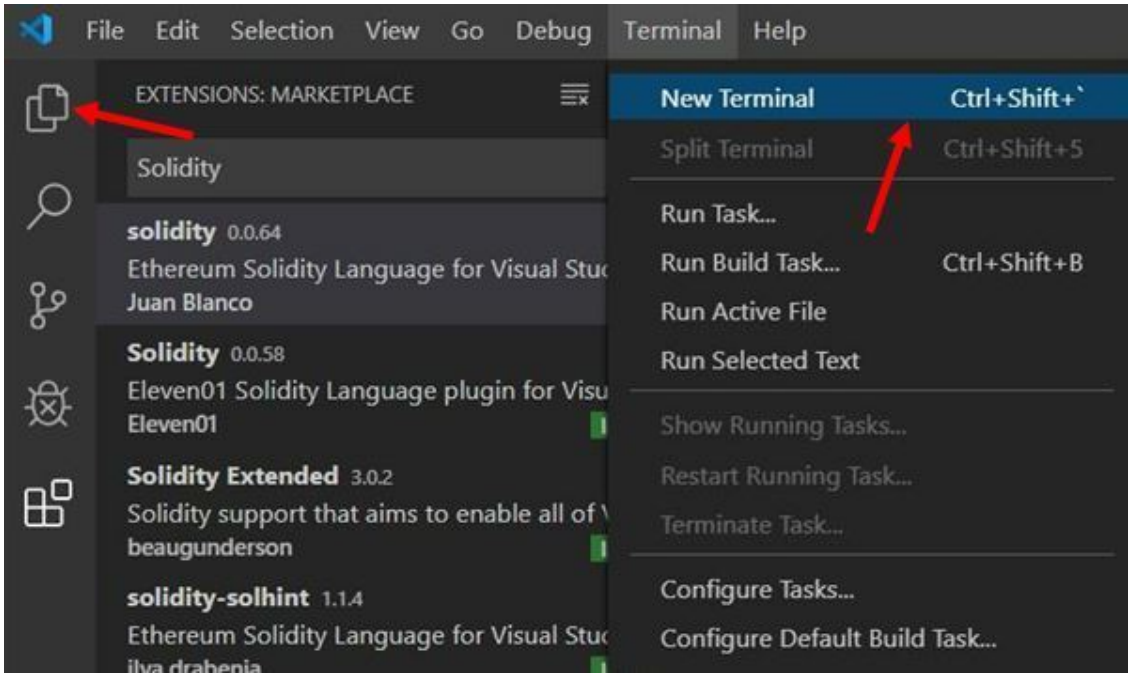

Рис. 1.4.2

Панель терминала откроется в нижней части окна VS Code. Для работы с новым проектом откройте раздел EXPLORER, нажав самую верхнюю кнопку на панели слева (рис. 1.4.2). В итоге окно VS Code будет выглядеть как на рис. 1.4.3.

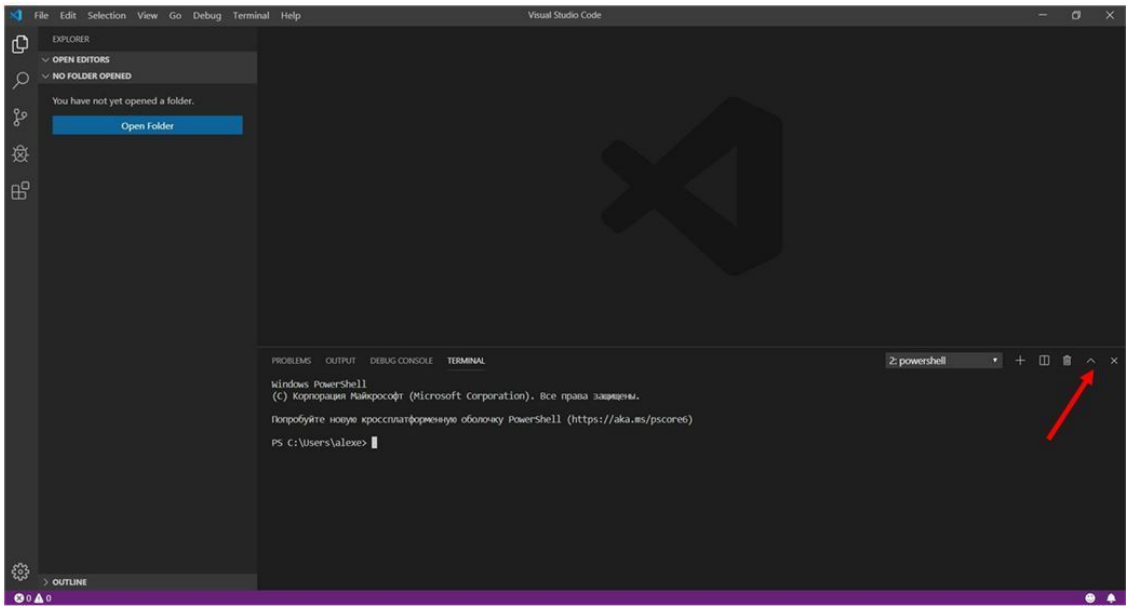

Рис. 1.4.3

Разверните терминал на всю панель, нажав кнопку «^» в верхнем правом углу панели терминала (рис. 1.4.3). Окно VS Code примет вид как на рис. 1.4.4.

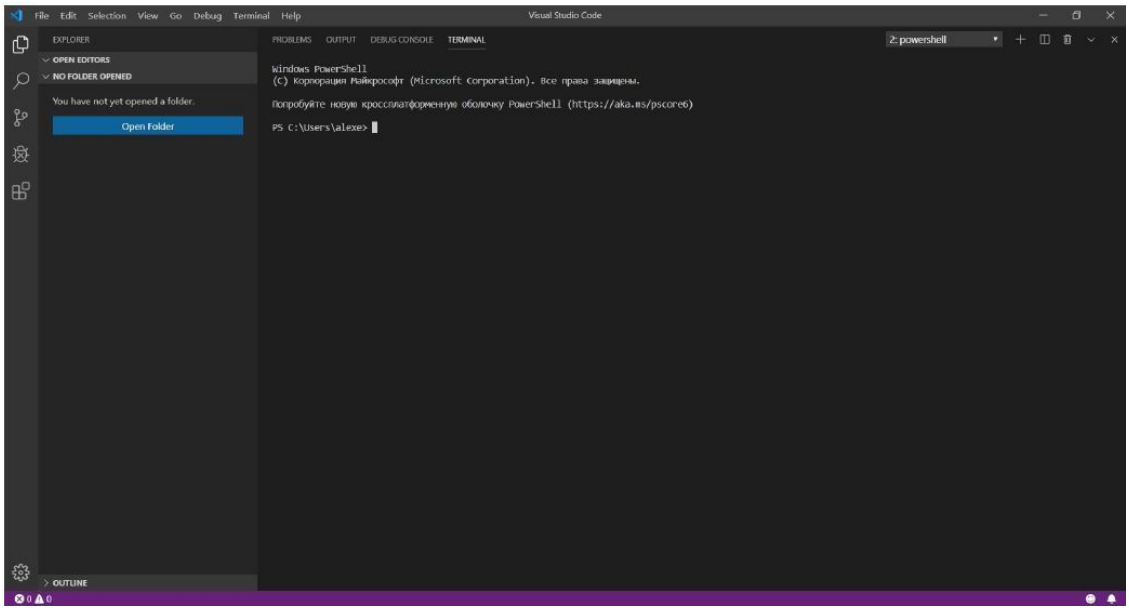

Рис. 1.4.4

Проверим подключение компилятора к терминалу VS Code. Для этого в терминале наберите команду «npm» и нажмите клавишу Enter (рис. 1.4.5).

Если компилятор работает, то мы увидим справку о команде «npm», как на рис. 1.4.5.

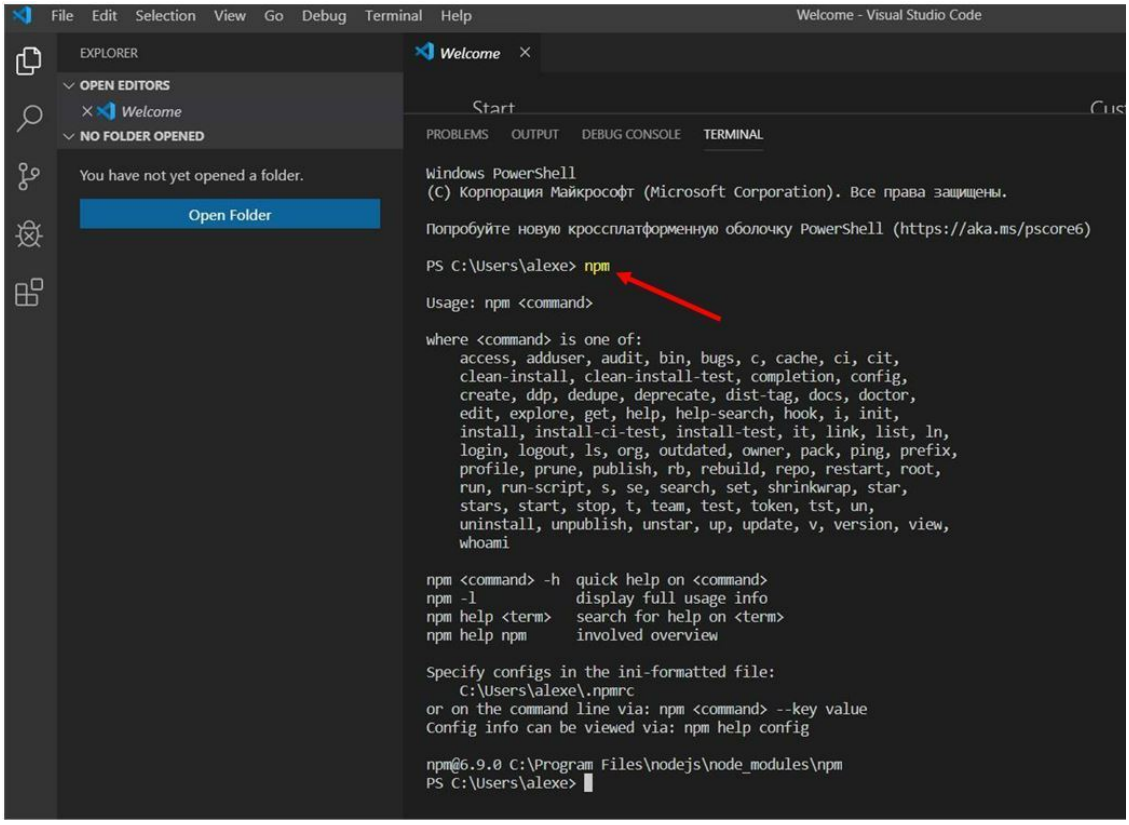

Рис. 1.4.5

Теперь установим фреймворк Truffle. Truffle – это набор инструментов и библиотек для создания смарт-контрактов на языке программирования Solidity.

Замечание. Конечно, можно обойтись и без Truffle и все делать вручную. Это сложно и занимает много времени. Поэтому давайте автоматизируем создание смарт-контрактов с использованием инструментов фреймворка Truffle.

Для установки фреймворка в терминале наберите команду «npm install –g truffle» и нажмите клавишу Enter (рис. 1.4.6).

Если все работает, терминал будет выглядеть как на рис. 1.4.6.

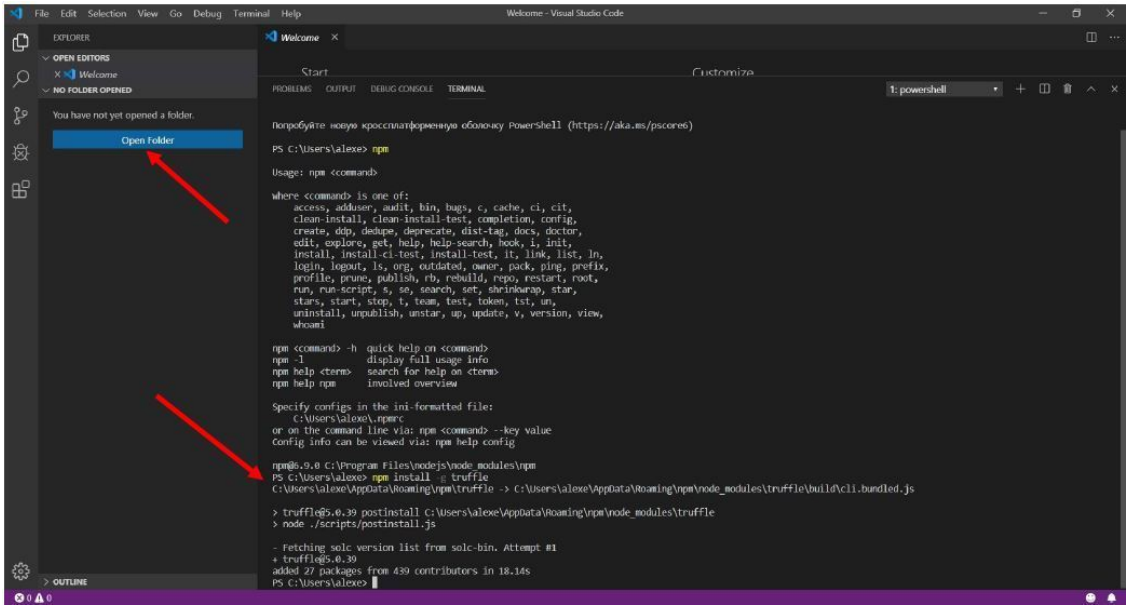

Рис. 1.4.6

Теперь для проверки работы компилятора Node.js при помощи фреймворка Truffle мы развернем тестовый проект Solidity и откомпилируем его.

Для начала создадим папку проекта. Для этого в разделе EXPLORER в левой части окна VS Code нажмите синюю кнопку Open Folder (рис. 1.4.6). В окне Open Folder нажмите кнопку «Новая папка». Создайте папку MetaCoin, выделите ее и нажмите кнопку «Выбор папки» (рис. 1.4.7).

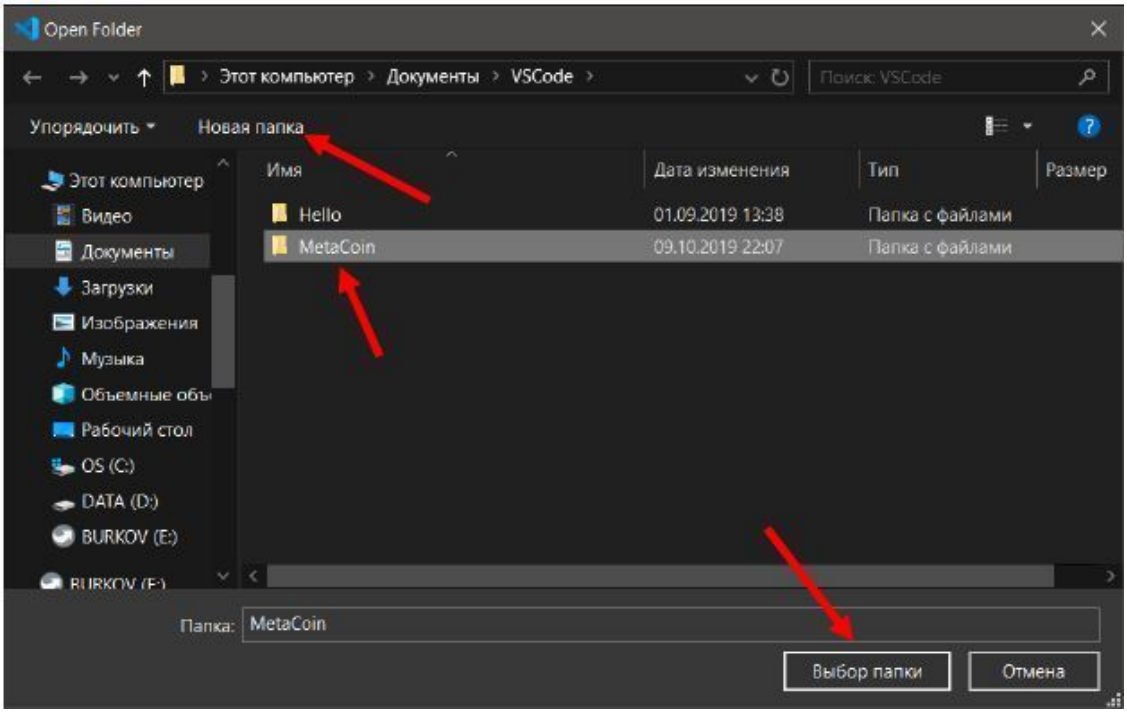

Рис. 1.4.7

Для проверки работы нашего окружения – «песочницы» – создадим тестовый проект при помощи Truffle. Для этого в окне терминала наберите команду «truffle unbox metacoin», как на рис. 1.4.8, и нажмите кнопку Enter. После развертывания тестового контракта MetaCoin окно VS Code будет выглядеть как на рис. 1.4.8.

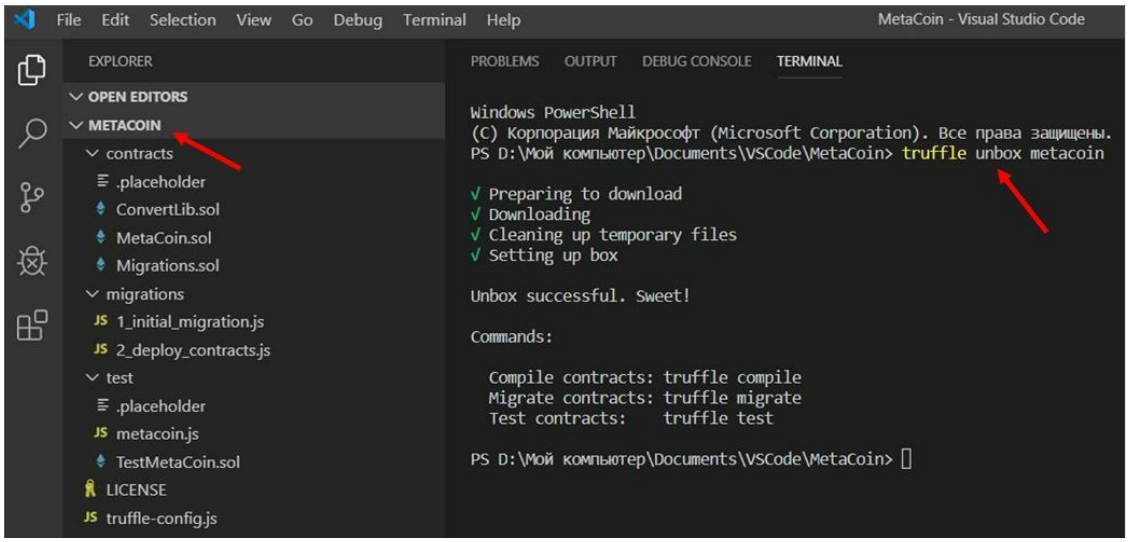

Рис. 4.8

Обратите внимание на изменения в разделе EXPLORER слева (рис. 1.4.8). Здесь появился проект MetaCoin, содержащий папки contracts, migrations и test. В папке contracts расположены наши смарт-контракты. Это файлы с расширением sol. В папке migrations располагаются настройки компиляции смарт-контрактов. Это файлы с расширением js. В папке test расположены файлы для отладки смарт-контрактов. Это файлы с расширением js и sol. Также обратите внимание на файл truffle-config.js, в нем мы прописываем настройки блокчейн-сети,

где будем публиковать и запускать наши смарт-контракты, и при помощи данного файла мы будем подключать наш проект к эмулятору.

В заключение для проверки работы компилятора Node.js произведем компиляцию проекта MetaCoin. Для начала компиляции проекта в окне терминала наберите команду «truffle compile» и нажмите кнопку Enter. Окно VS Code примет вид как на рис. 1.4.9.

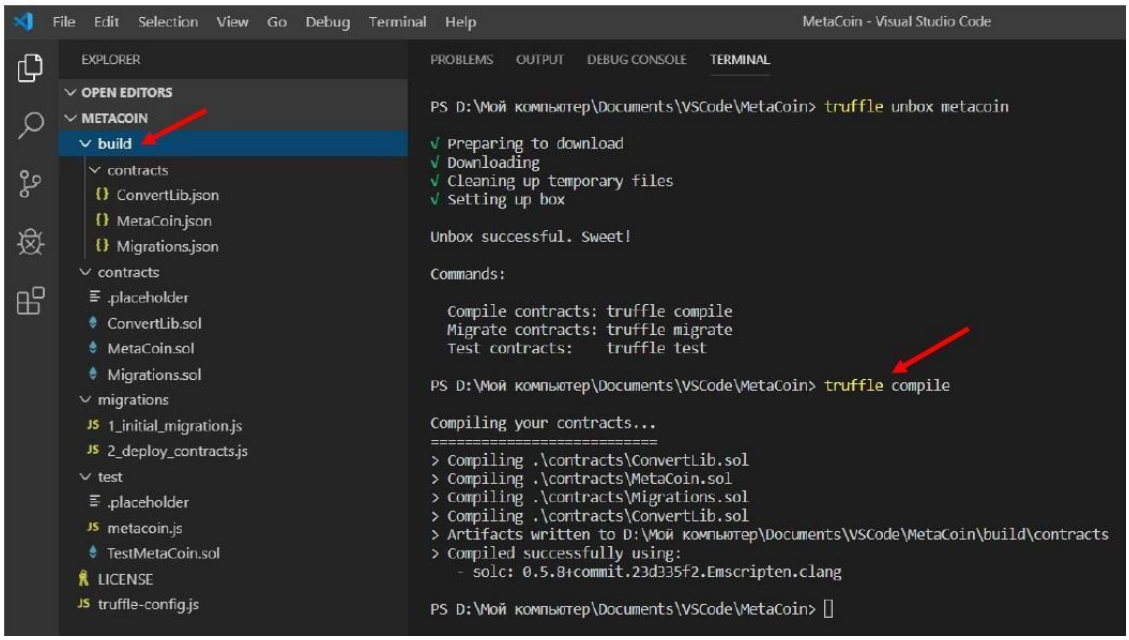

Рис. 1.4.9

Обратите внимание на то, что в проекте MetaCoin в разделе EXPLORER появилась папка build. В данной папке мы можем увидеть файлы из папки contracts, откомпилированные в формат json. Позднее мы рассмотрим, как опубликовать эти файлы в эмуляторе сети блокчейн. Успешная компиляция тестовых смарт-контрактов в формат json подтверждает функционирование компилятора Node.js.

Теперь перейдем к установке эмулятора сети блокчейн Ganache.

Урок 5. Установка эмулятора Ganache

Аннотация. В уроке рассматривается установка эмулятора блокчейн сети Ganache [4].

Рассмотрим установку эмулятора блокчейн-сети Ethereum – Ganache. Мы будем использовать его для тестирования наших смарт-контрактов. Для начала скачаем установочный пакет эмулятора. Это можно сделать с официального сайта, расположенного по адресу http://trufflesuite.com/ganache (рис. 1.5.1).

Рис. 1.5.1

Для начала скачивания версии Ganache для Windows необходимо нажать на ссылку DOWNLOAD (WINDOWS) (рис. 1.5.1). Появится окно загрузки установочного пакета (рис. 1.5.2).

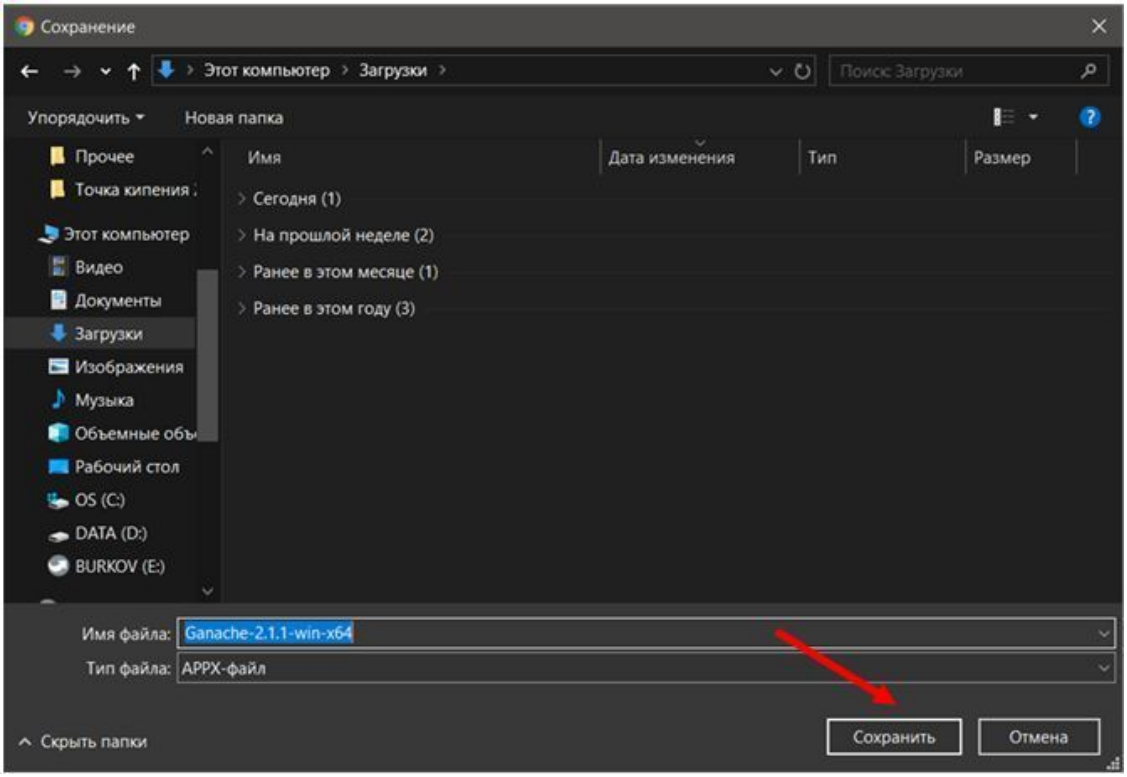

Рис. 1.5.2

Нажмите кнопку «Сохранить» (рис. 1.5.2). По окончании скачивания запустите установленный файл. Появится окно начала установки, где необходимо нажать кнопку «Установить» (рис. 1.5.3).

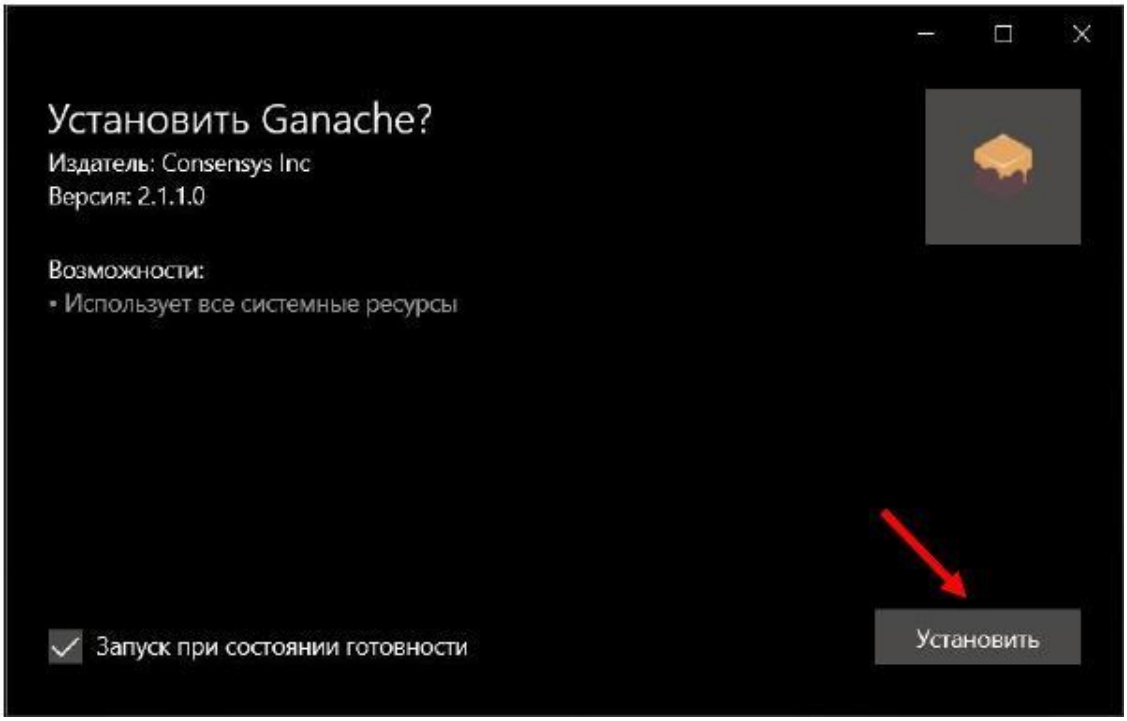

Рис. 1.5.3

После окончании установки эмулятора появится окно с вопросом о сборе статистики. В данном окне нажмите кнопку CONTINUE (рис. 1.5.4).

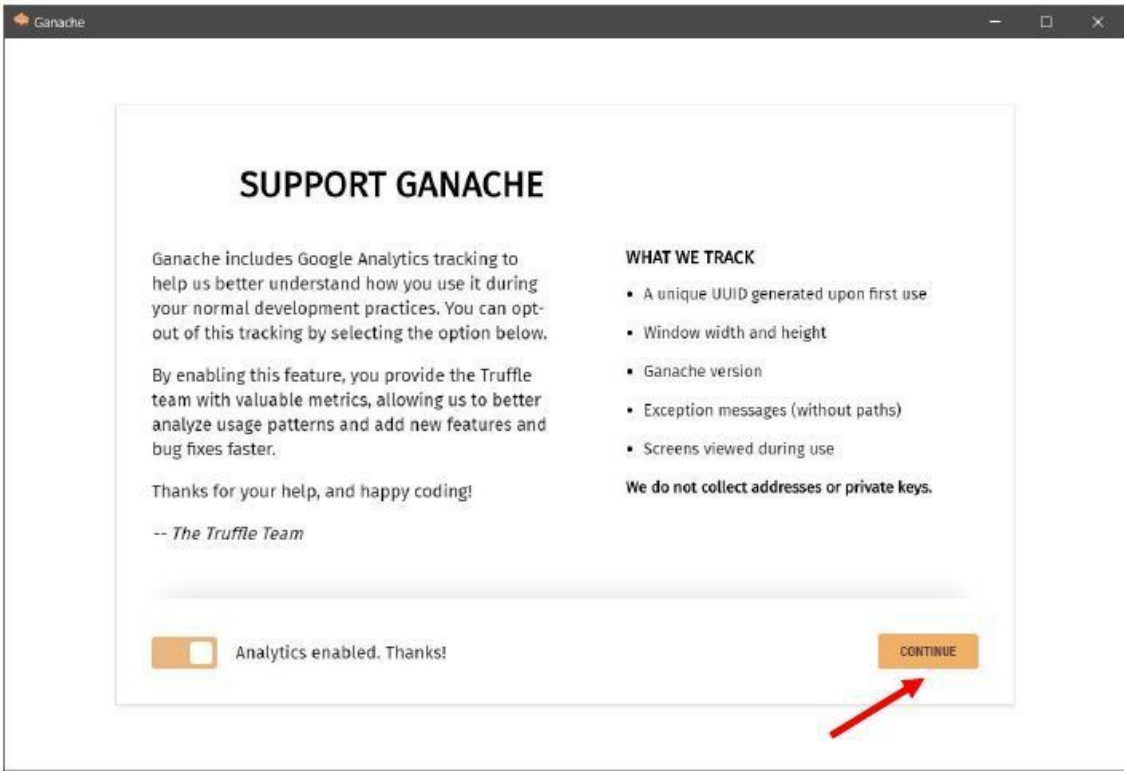

Рис. 1.5.4

Теперь произведем настройку рабочего окружения. Для начала активируем опцию QUICKSTART. Это автоматическая настройка окружения. Если выбрать опцию NEW WORKSPACE, то появятся расширенные настройки окружения. Если вы продвинутый пользователь, то можно их настроить. Всем остальным я советую предпочесть опцию QUICKSTART (рис. 1.5.5).

> **Замечание.** Мы для простоты остановили свой выбор на опции QUICKSTART. Однако в данном случае все настройки эмулятора будут сброшены при последующем запуске эмулятора. Для текущих задач этого достаточно. Позже мы посмотрим, как создать постоянное окружение.

Рис. 1.5.5

Появится рабочее окно Ganache. Рассмотрим его более подробно. Первое, что мы здесь видим, – это вкладка ACCOUNTS. В центре окна расположено десять криптокошельков с виртуальными 100 ETH на счету (рис. 1.5.7). Их можно добавлять и удалять. Обратите внимание, что в столбце ADDRESS отображается адрес кошелька, а знак ключа предоставляет доступ к адресу и закрытым ключам кошелька – эта информация понадобится нам позже (рис. 1.5.6).

Рис. 1.5.6

В верхней части окна расположены кнопки для открытия других вкладок и текущая информация по состоянию нашей «виртуальной» сети, «Цена газа», «Остаток газа», «Состояние майнинга» и т. д. (рис. 1.5.7).

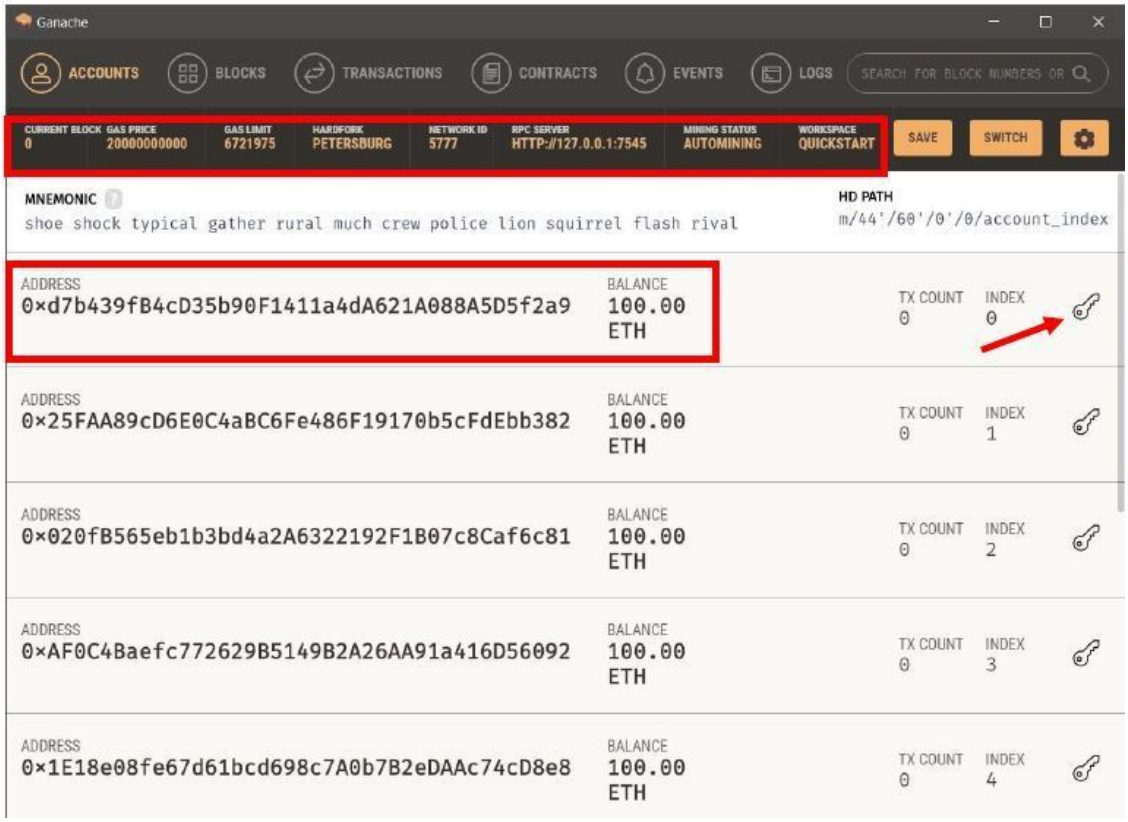

Рис. 1.5.7

Рассмотрим другие вкладки. Вкладка BLOCKS отображает состояние майнинга (рис. 1.5.8), т. е. сколько блоков было создано вашим компьютером, пока был запущен эмулятор. Блоки будут создаваться в результате каких-либо действий в эмуляторе.

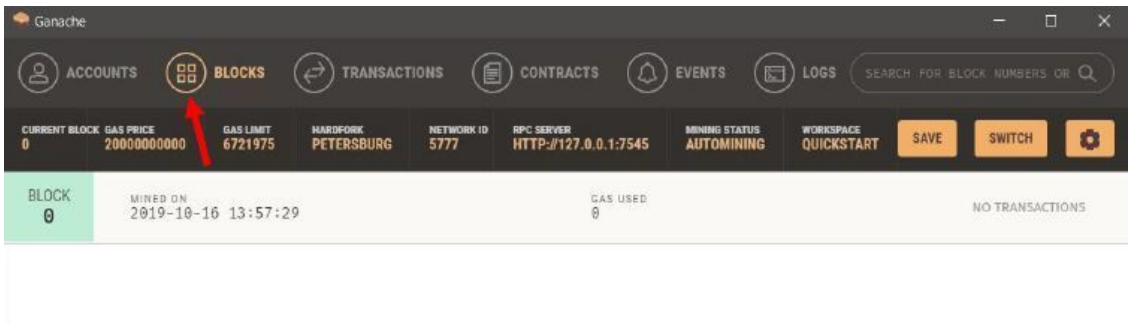

Рис. 1.5.8

Перейдем на вкладку TRANSACTIONS. Здесь отображаются все транзакции в нашей виртуальной сети. Поскольку мы только установили эмулятор и не производили никаких транзакций, эта вкладка будет пуста (рис. 1.5.9).

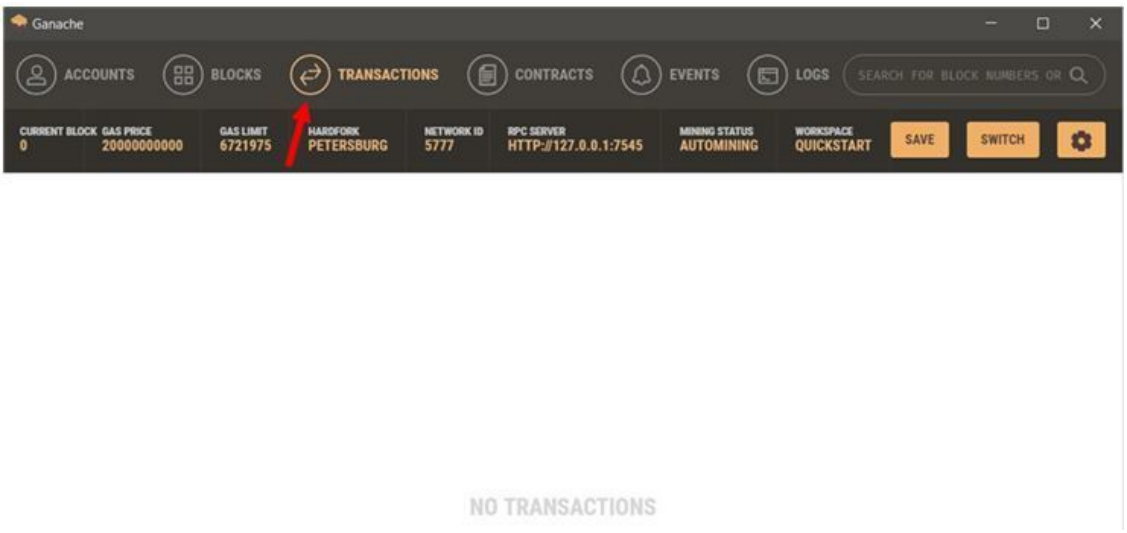

Рис. 1.5.9

Вкладка CONTRACTS отображает опубликованные в нашей виртуальной сети смарт-контракты, их состояние и действия (рис. 1.5.10). Далее мы будем часто работать с этой вкладкой.

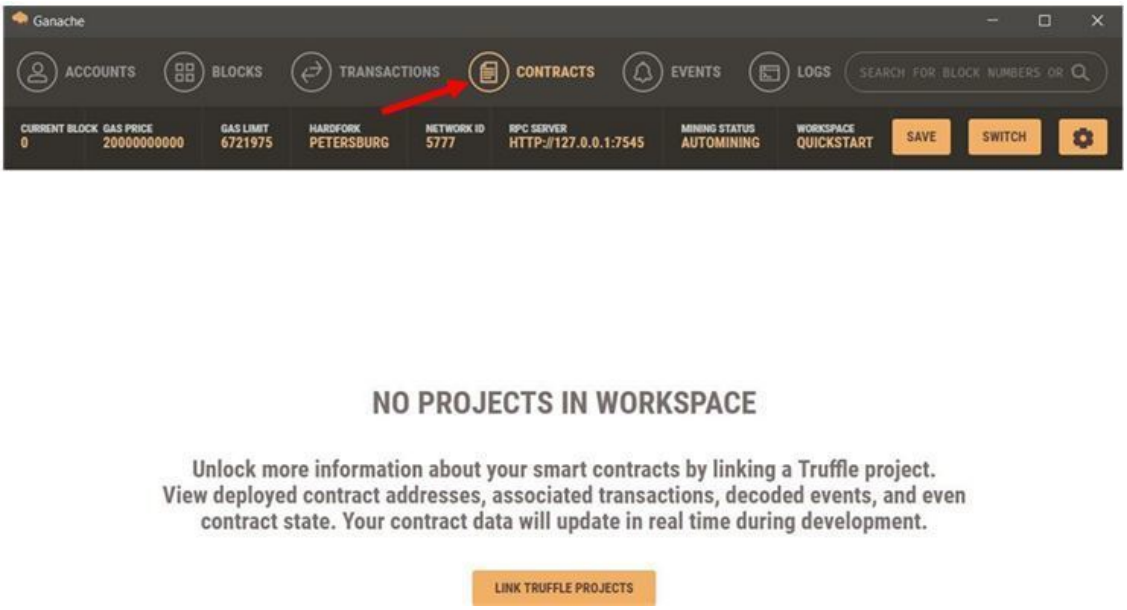

Рис. 1.5.10

На вкладке EVENTS отображаются различные события, происходящие в нашей виртуальной блокчейн-сети (рис. 1.5.11).

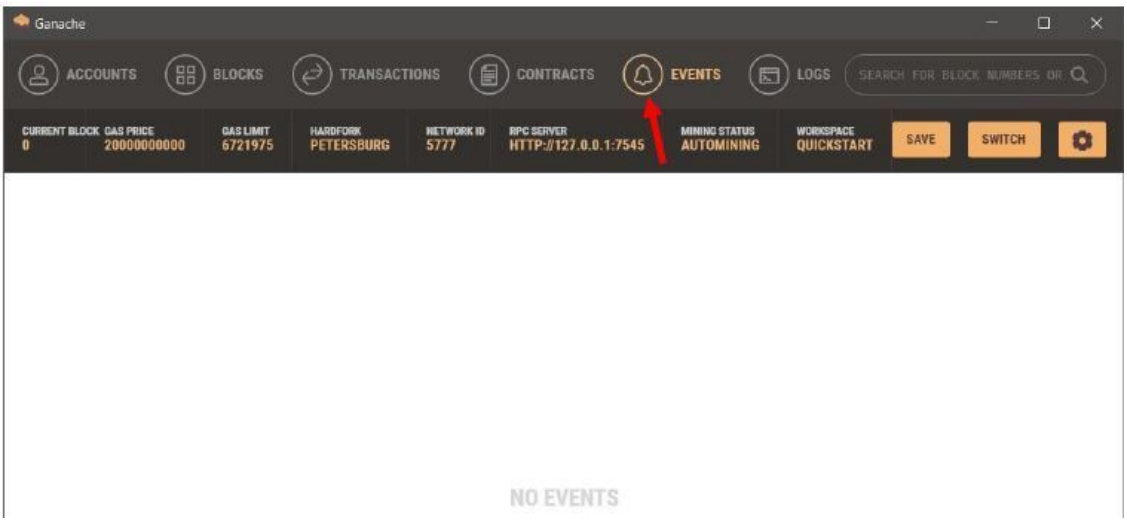

Рис. 1.5.11

Ну и наконец, последняя вкладка LOGS. На данной вкладке отображаются все действия и события в нашей блокчейн-сети. Данный «журнал» можно очистить, нажав кнопку CLEAR LOGS, расположенную в верхнем левом углу окна (рис. 1.5.12).

Рис. 1.5.12

Если нажать кнопку с шестеренкой, расположенную в верхнем правом углу (рис. 1.5.12), появится окно с настройками эмулятора (рис. 1.5.13).

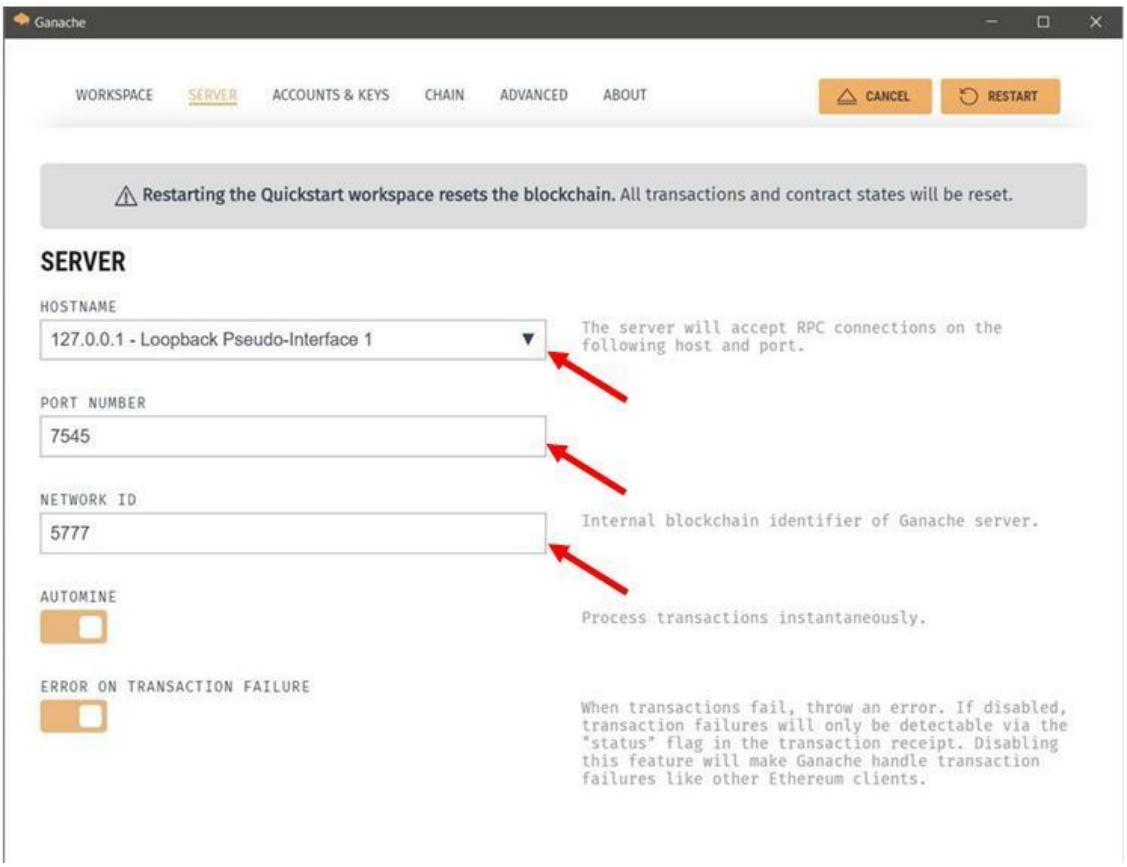

Рис. 1.5.13

В данном окне для нас наиболее интересна вкладка «Server». На данной вкладке находятся параметры подключения нашего проекта на Solidity в VS Code к нашей виртуальной блокчейн-сети в Ganache. Для подключения проекта нам понадобятся параметры HOSTNAME, PORT NUMBER и NETWORK ID (рис. 1.5.13).

Урок 6. Подключение тестового проекта в VS Code к эмулятору Ganache и проверка работы эмулятора

Аннотация. В данном уроке мы подключим наш тестовый проект в VS Code к эмулятору блокчейн-сети, опубликуем его в нашей виртуальной сети и протестируем его работу.

Теперь перейдем к подключению нашего тестового проекта, созданного нами ранее в VS Code. **Сверните окно Ganache, но не закрывайте его!** Запустите VS Code и на панели EXPLORER в нашем тестовом проекте METACOIN откройте файл truffle-config.js, в центре окна отобразится его содержимое (рис. 1.6.1).

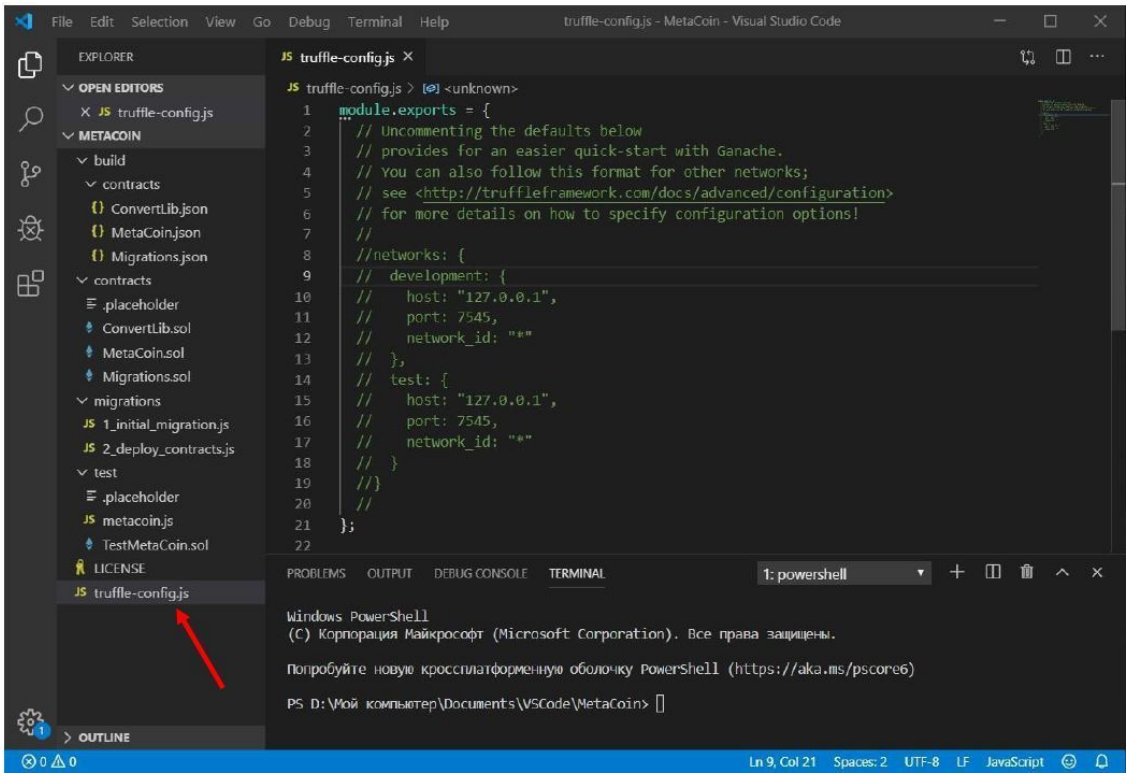

Рис. 1.6.1

Файл truffle-config.js содержит настройки подключения проекта к блокчейн-сети. Обратите внимание на то, что почти все строки в этом файле сейчас отключены. Они помечены символами «//», как комментарии. Давайте включим некоторые строки. Удалите символы «//» у строк с номерами 8–12, 18, 19 и 21, как на рис. 1.6.2.

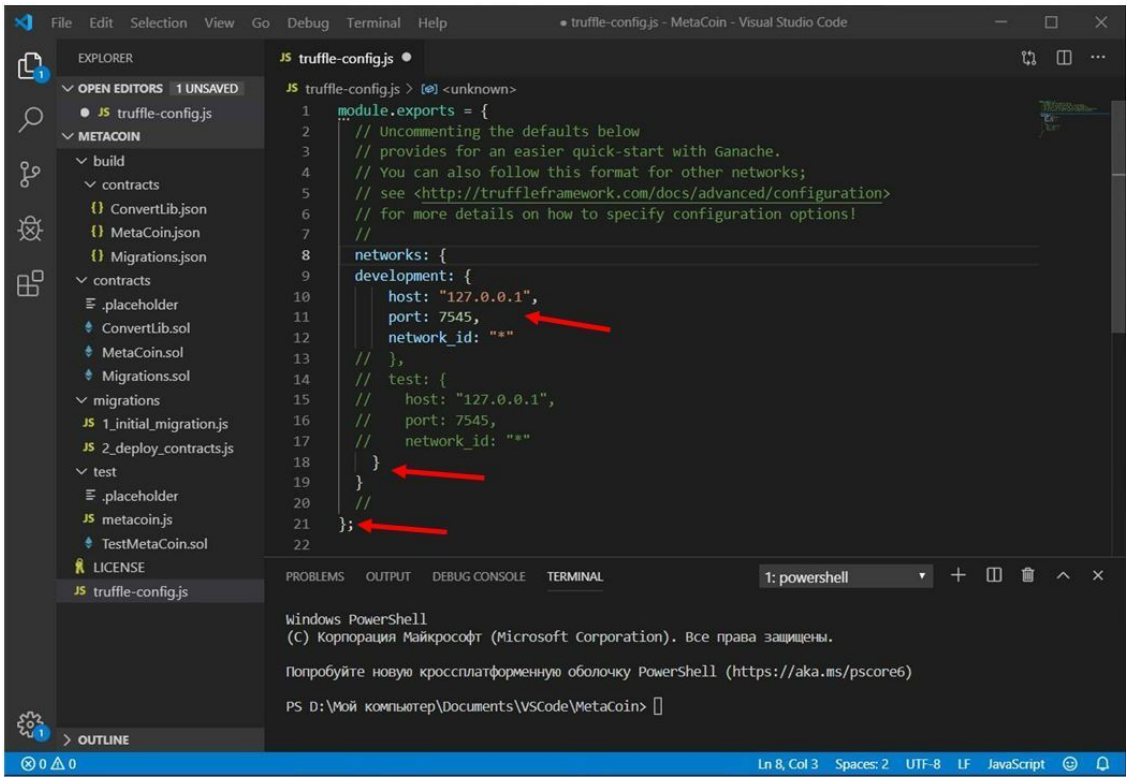

Рис. 1.6.2

Мы можем видеть, что параметры подключения к сети «host» и «port» совпадают с параметрами HOSTNAME и PORT NUMBER из окна настроек Ganache (урок 5, см. рис. 1.5.13). Параметр network_id мы оставили как «*» (любой), хотя его можно было задать как параметр NETWORK ID из окна настроек Ganache, т. е. «5777». Сохраните изменения в файле с настройками, выбрав в оконном меню VS Code пункт File/Save (рис. 1.6.3).

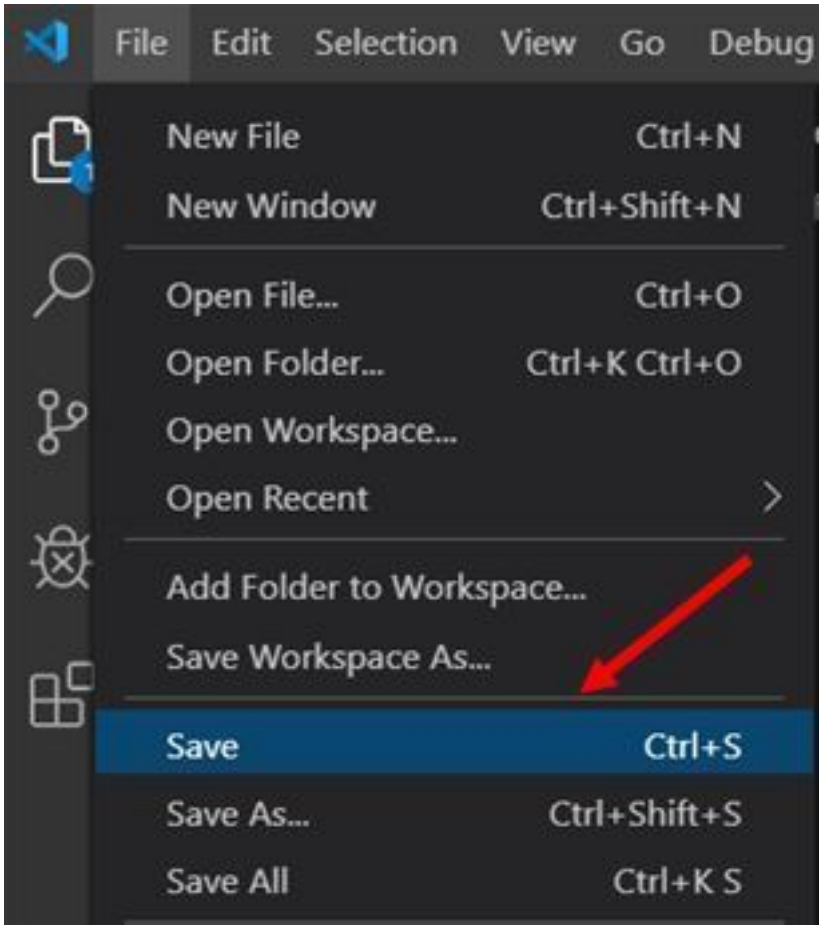

Рис. 1.6.3

После подключения нашего проекта к эмулятору необходимо подключить эмулятор к нашему проекту, т. к. пока проект «видит» эмулятор, а эмулятор «не видит» наш проект. Для подключения эмулятора к проекту в окне Ganache откройте вкладку CONTRACTS. Мы видим, что эмулятор «не видит» наши тестовые смарт-контракты. Для подключения проекта нажмите кнопку LINK TRUFFLE PROJECTS в центре вкладки CONTRACTS (рис. 1.6.4).

Рис. 1.6.4

Появится окно с настройками эмулятора и открытой вкладкой WORKSPACE (рис. 1.6.5).

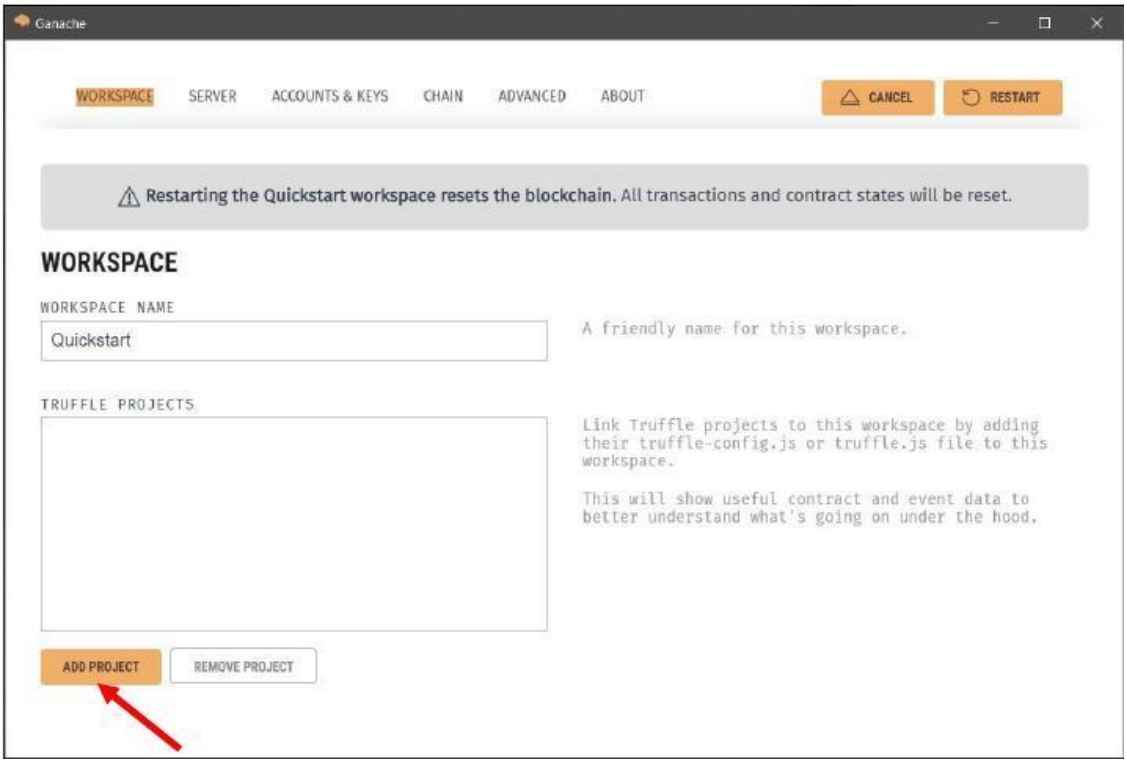

Рис. 1.6.5

Теперь нажмите кнопку ADD PROJECT (рис. 1.6.5), в появившемся окне выбора файла выберите файл «truffle-config.js» нашего проекта METACOIN и нажмите кнопку «Открыть» (рис. 1.6.6).

Замечание. По умолчанию все проекты VS Code сохраняются в папке «Мои документы / VS Code».

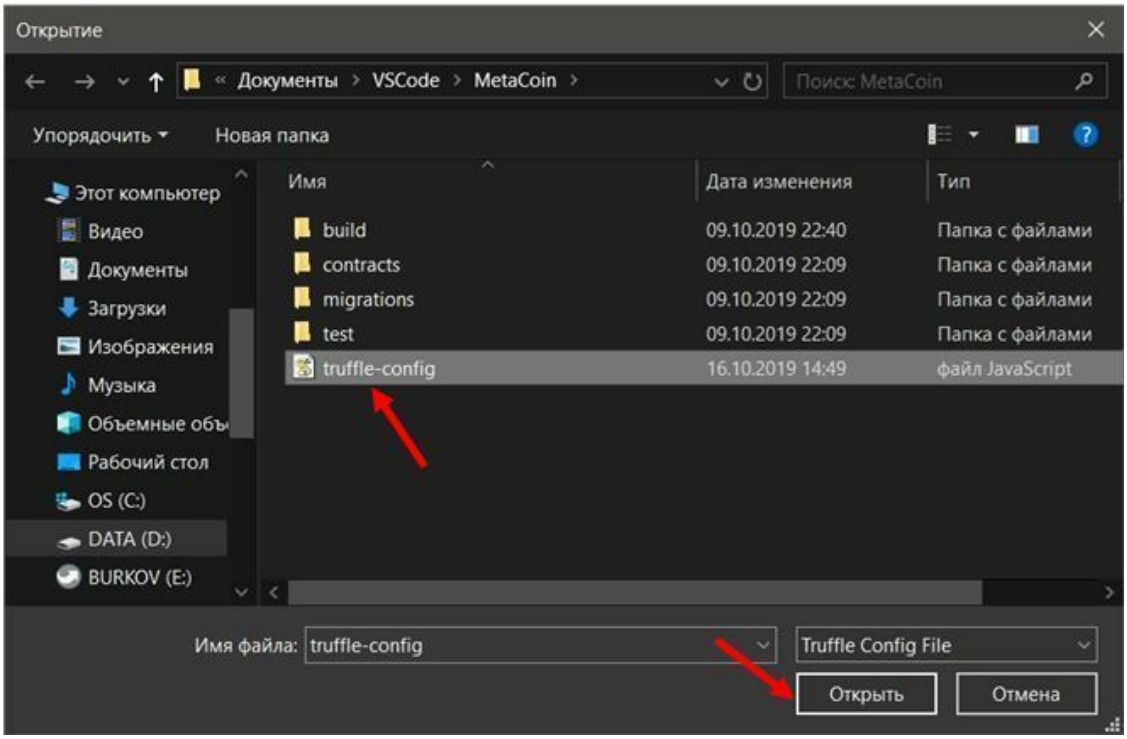

Рис. 1.6.6

Окно настроек Ganache примет вид как на рис. 1.6.7. В поле «TRUFFLE PROJECTS» появится путь к нашему тестовому проекту MetaCoin (рис. 1.6.7).

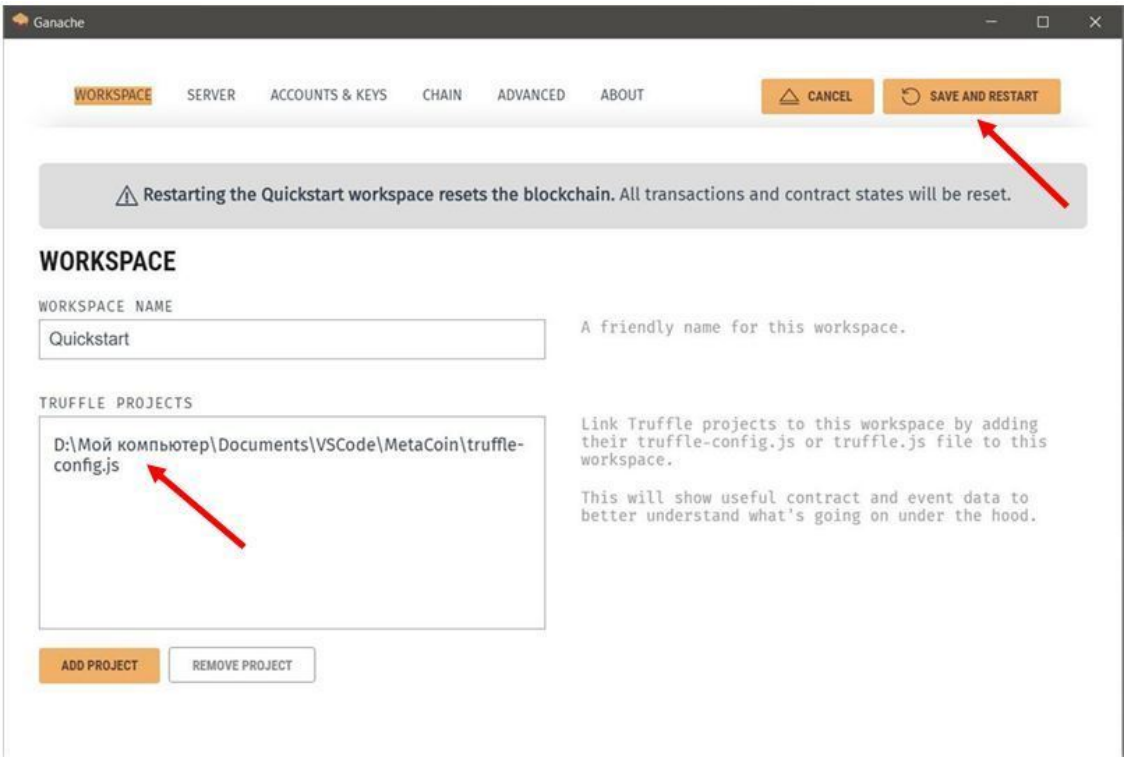

Рис. 1.6.7

Для сохранения результатов нажмите кнопку SAVE AND RESTART (рис. 1.6.7), расположенную в верхнем правом углу окна настроек. Затем в окне Ganache перейдите на вкладку CONTRACTS. Она примет вид как на рис. 1.6.8.

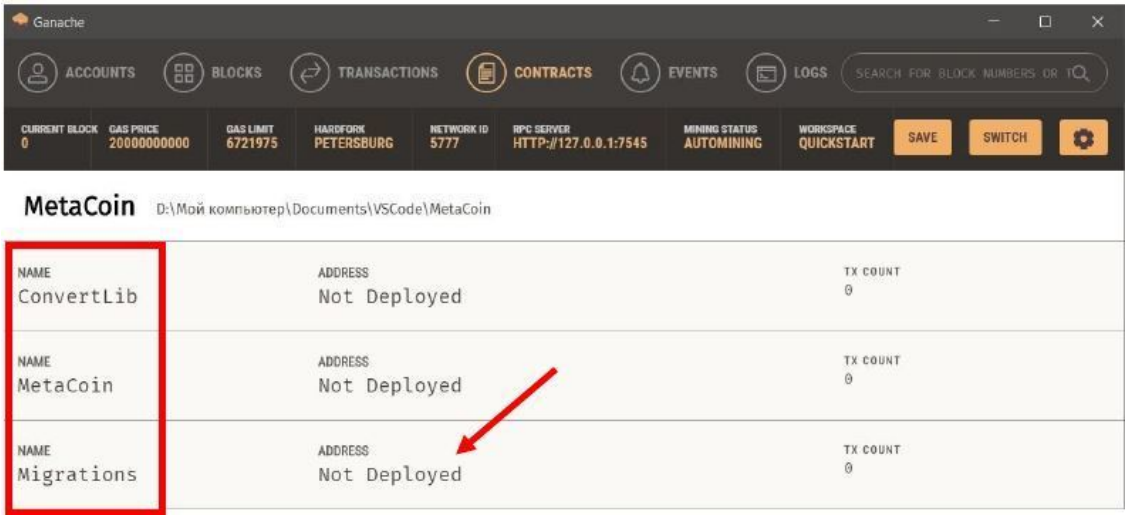

Рис. 1.6.8

Видно, что наш эмулятор «увидел» наш проект. На вкладке CONTRACTS появились смарт-контракты нашего тестового проекта: ConvertLib, MetaCoin и Migrations. Мы видели файлы этих смарт-контрактов в окне VS Code на панели EXPLORER (рис. 1.6.2). Однако наш проект не опубликован, около наших смарт-контрактов стоит надпись Not Deployed (рис. 1.6.8).

Наш следующий шаг – это публикация нашего тестового проекта в виртуальной блокчейн-сети. Для этого откройте панель терминала и в ней выполните команду публикации проекта в блокчейн-сети «truffle migrate». Окно VS Code примет вид как на рис. 1.6.9.

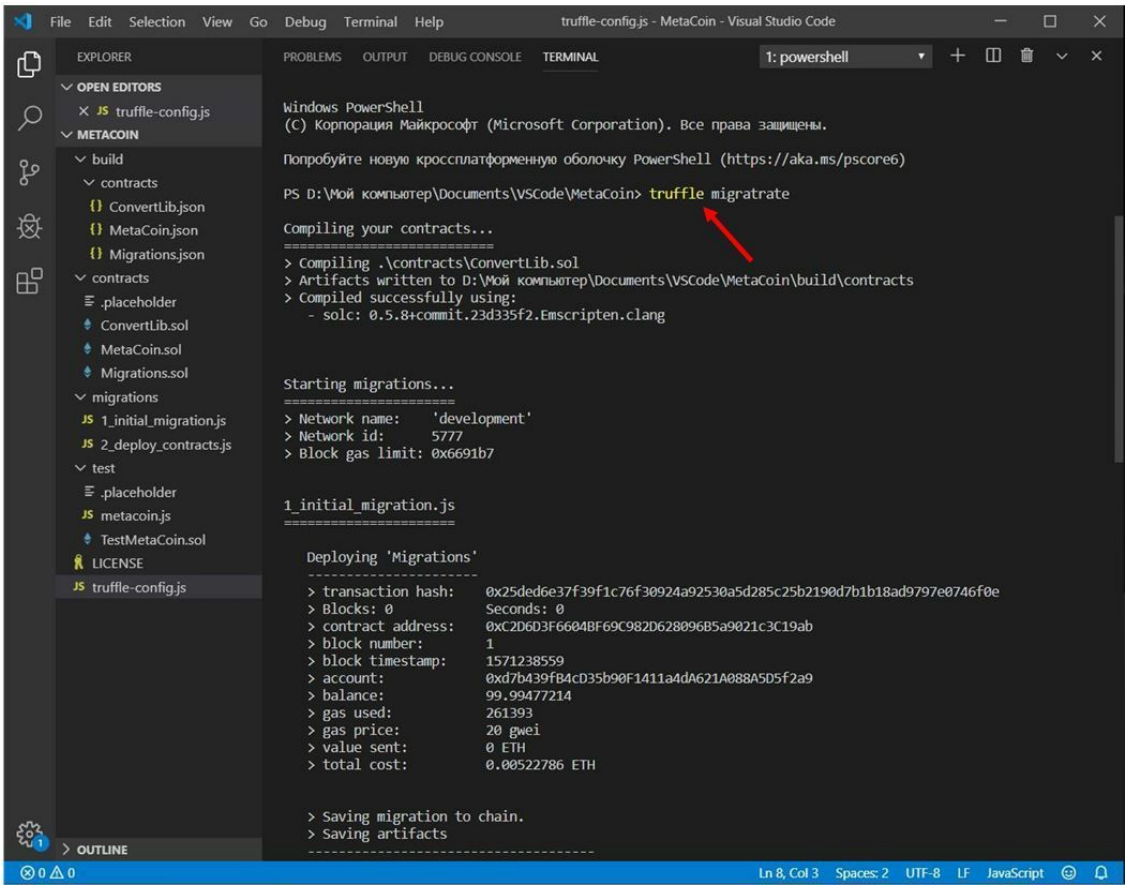

Рис. 1.6.9

Здесь можно видеть, что наши тестовые смарт-контракты были опубликованы и в окне терминала отобразились параметры публикации. Боле того, если посмотреть на вкладку CONTRACTS эмулятора, то можно видеть, что контракты опубликованы, т. е. перешли в статус DEPLOYED и получили свои адреса (рис. 1.6.10).

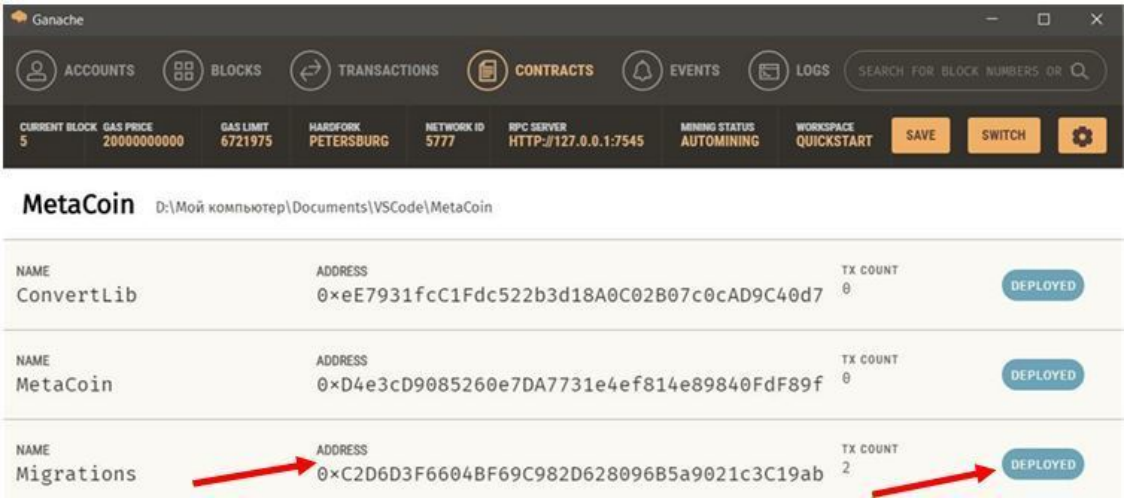

Рис. 1.6.10

Замечание. Если на вкладке CONTRACTS нажать на кнопку DEPLOYED, расположенную справа от смарт-контракта, то можно посмотреть статистику его работы.

Если открыть вкладку TRANSACTIONS, то мы увидим список транзакций, которые опубликовали наши тестовые смарт-контракты (рис. 1.6.11).

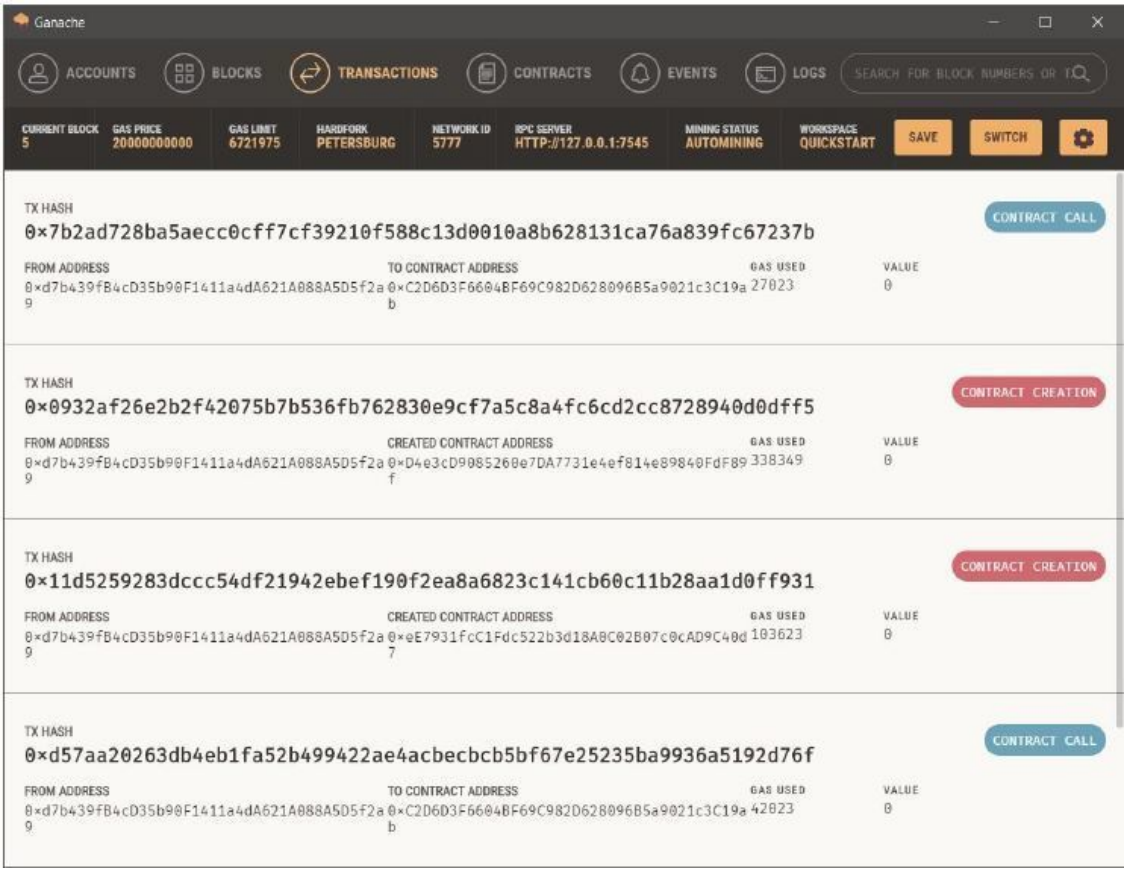

Рис. 1.6.11

На вкладке BLOCKS можно увидеть, что произошел процесс майнинга шести блоков и было потрачено определенное количество газа (рис. 1.6.12).

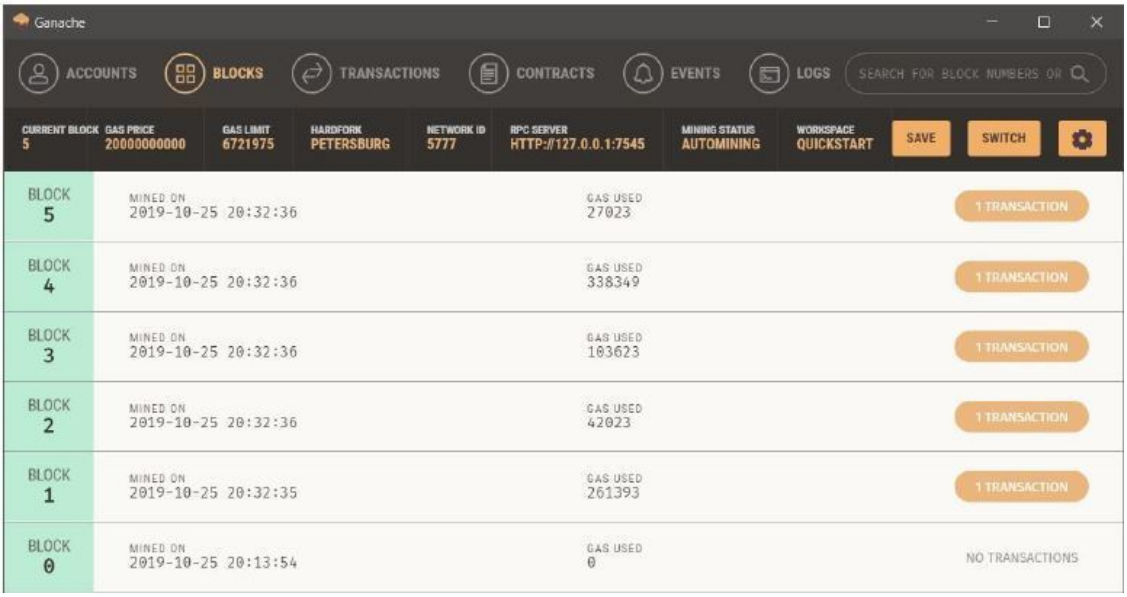

Рис. 1.6.12

И наконец, на вкладке ACCOUNTS мы видим, что с нашего первого кошелька было списано 0,02 ETH в качестве оплаты пяти транзакций, т. е. публикации наших трех тестовых смарт-контрактов (рис. 1.6.13).

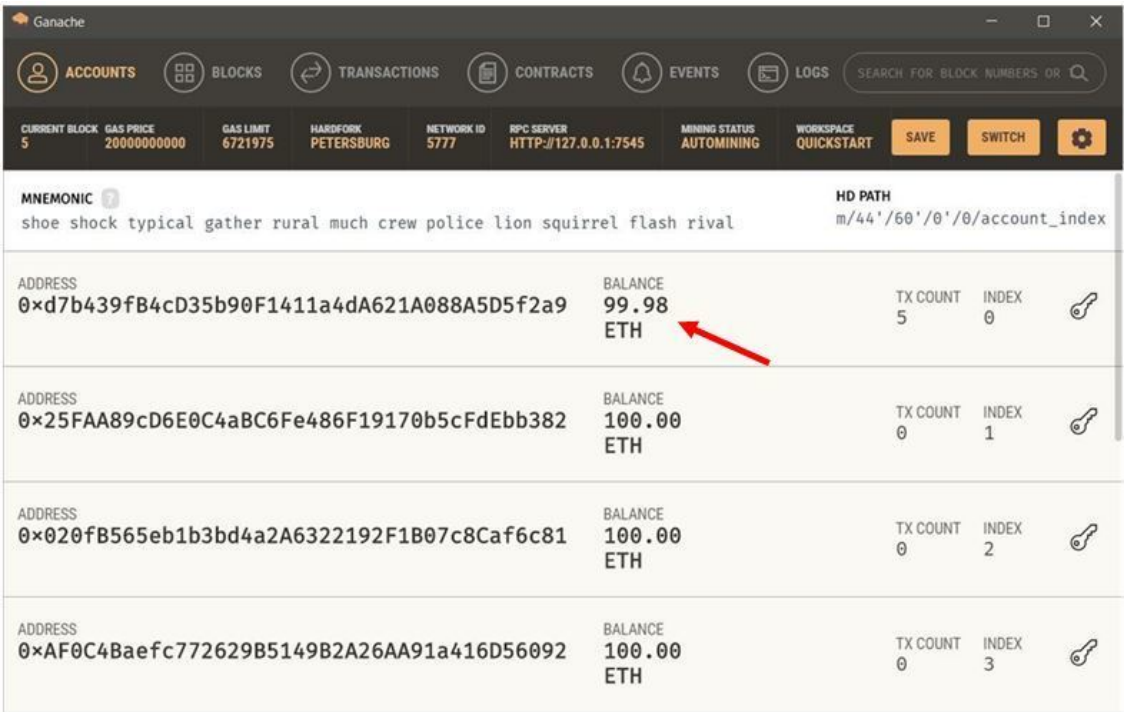

Рис. 1.6.13

В заключение данной темы проведем тестирование смарт-контрактов нашего проекта MetaCoin. В этом случае будет запущен специальный тестирующий смарт-контракт TestMetaCoin.sol, находящийся в папке test нашего проекта. Для запуска теста выполните в терминале команду «truffle test» (рис. 1.6.14).

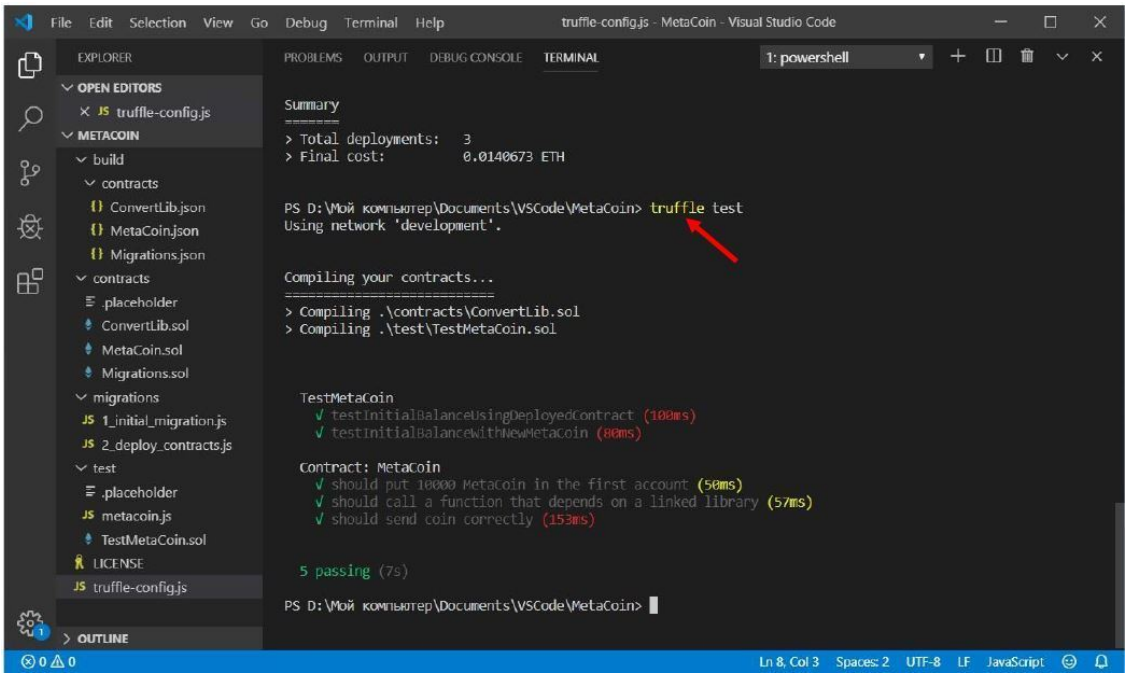

Рис. 1.6.14

Произойдет компиляция и выполнение смарт-контракта TestMetaCoin.sol, а на вкладке EVENTS эмулятора появится событие (рис. 1.6.15).

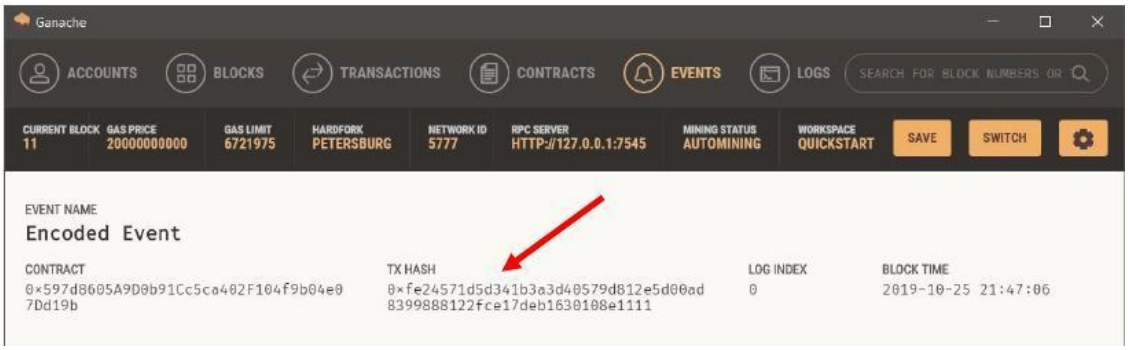

Рис. 1.6.15

Урок 7. Установка плагина MetaMask для работы с криптокошельками

Аннотация. В данном уроке рассматривается процедура установки плагина для работы с криптокошельками MetaMask для браузера Chrome [5]. Также рассматривается подключение и тестирование плагина с эмулятором блокчейн-сети Ganache.

В заключение настройки нашего окружения установим специальный плагин для браузера MetaMask, при помощи которого мы будем совершать транзакции в нашей виртуальной блокчейн-сети. Для начала откройте в браузере веб-страницу, расположенную по адресу https://MetaMask.io/ (рис. 1.7.1).

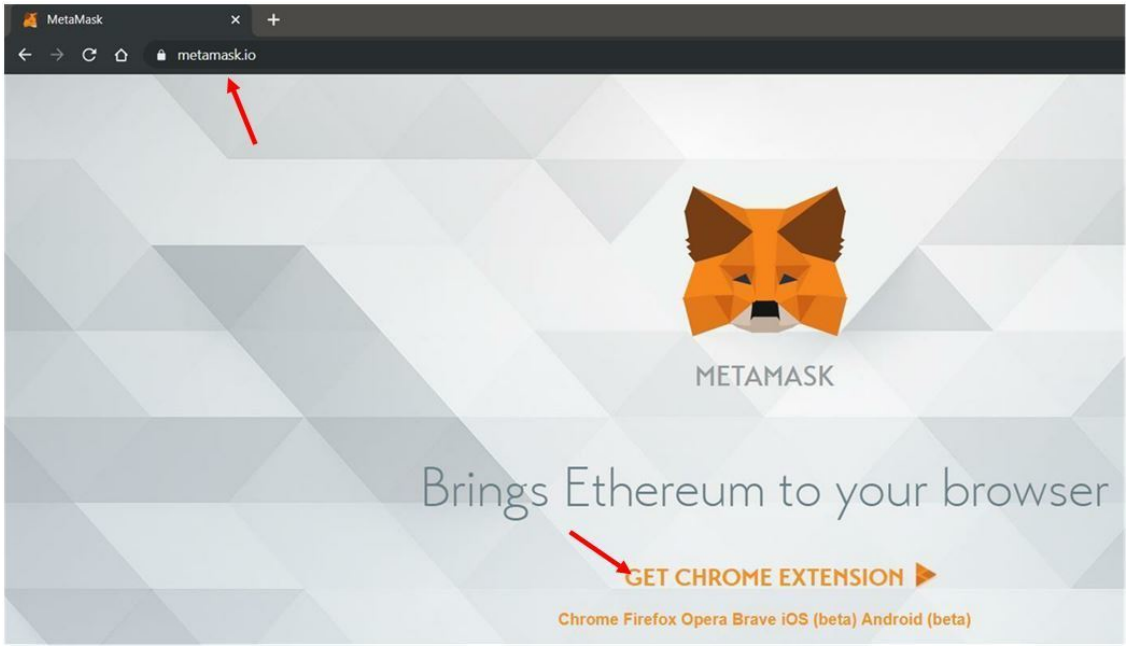

Рис. 1.7.1

В данном уроке мы будем работать в браузере Chrome. Поэтому на странице сайта MetaMask перейдите по ссылке GET CHROME EXTENSION (рис. 1.7.1). Откроется страница для скачивания расширения MetaMask, расположенная в интернет-магазине Chrome. Нажмите кнопку «Установить» (рис. 1.7.2).

> **Замечание.** Если у вас браузер, отличный от Chrome, то для установки расширения необходимо перейти по соответствующим ссылкам, расположенным под надписью GET CHROME EXTENSION (см. рис. 1.7.1)

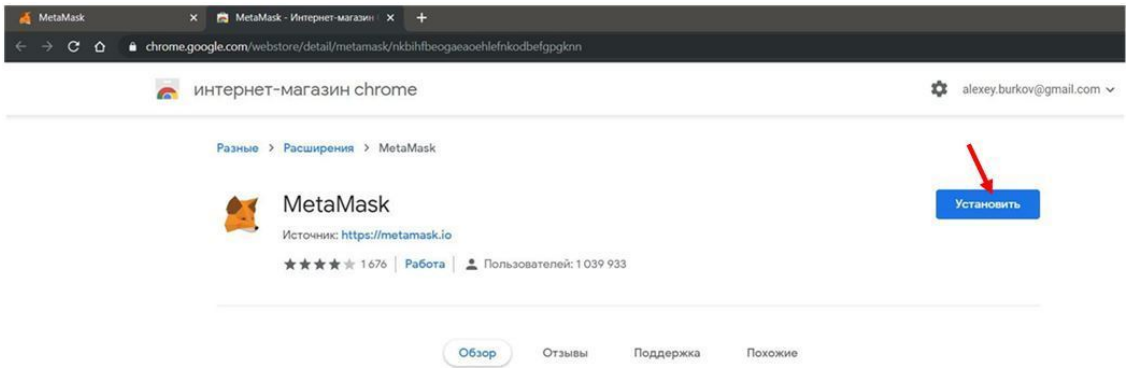

Рис. 1.7.2

Появится окно с запросом разрешения на установку расширения. Нажмите кнопку «Установить расширение» (рис. 1.7.3).

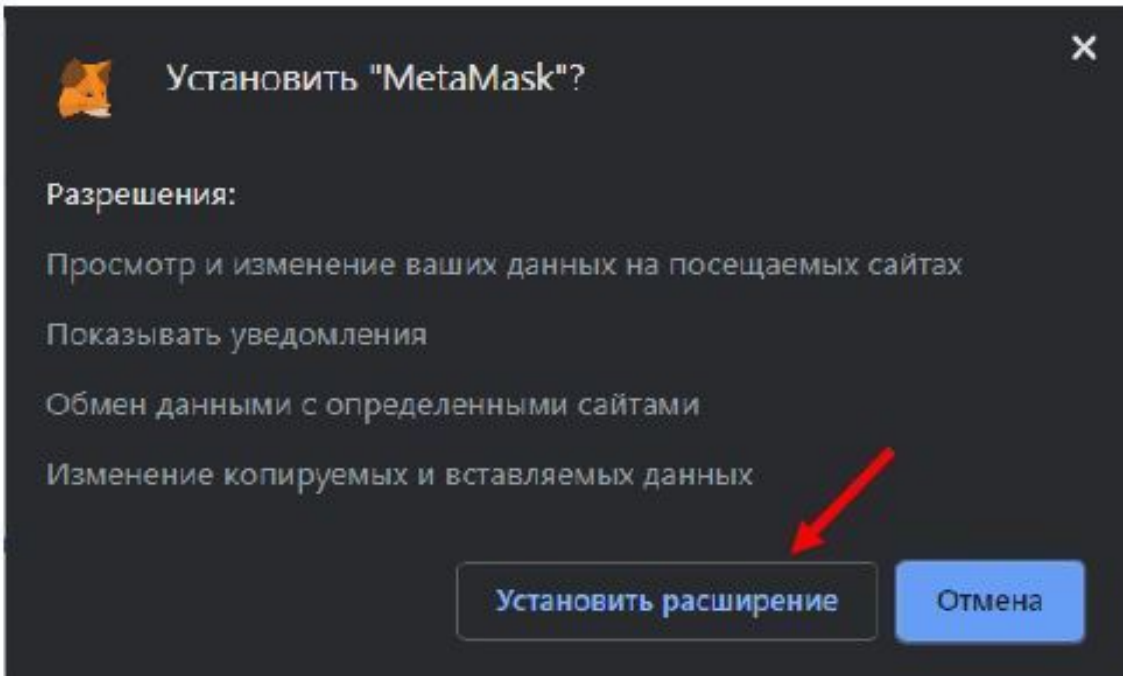

Рис. 1.7.3

После завершения установки расширения откроется стартовая страница MetaMask, на которой необходимо нажать кнопку «Начать» (рис. 1.7.4).

Замечание. Получить доступ к плагину также можно через иконку плагина в верхнем правом углу окна Chrome (рис. 1.7.4).

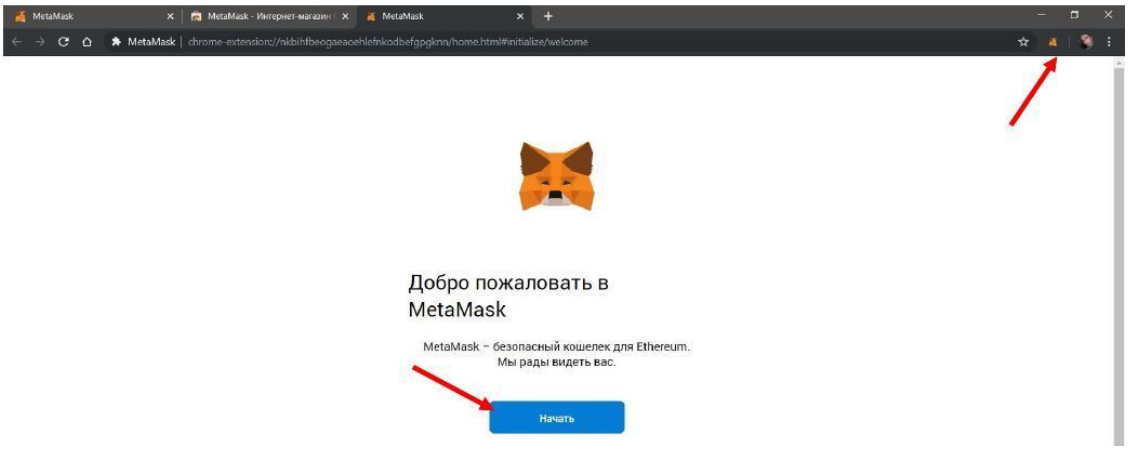

Рис. 1.7.4

Далее появится страница, где можно создать новый кошелек или импортировать существующий при помощи кодовой фразы. Давайте создадим новый кошелек, нажав кнопку «Создать кошелек» (рис. 1.7.5).

> **Замечание.** Вновь созданный кошелек будет работать в публичной открытой сети Ethereum. Мы же будем работать в эмуляторе Ganache, поэтому далее мы импортируем наши кошельки из эмулятора, и этот новый кошелек нам будет не нужен. Однако для продолжения работы с MetaMask нам необходимо его создать или импортировать существующий.

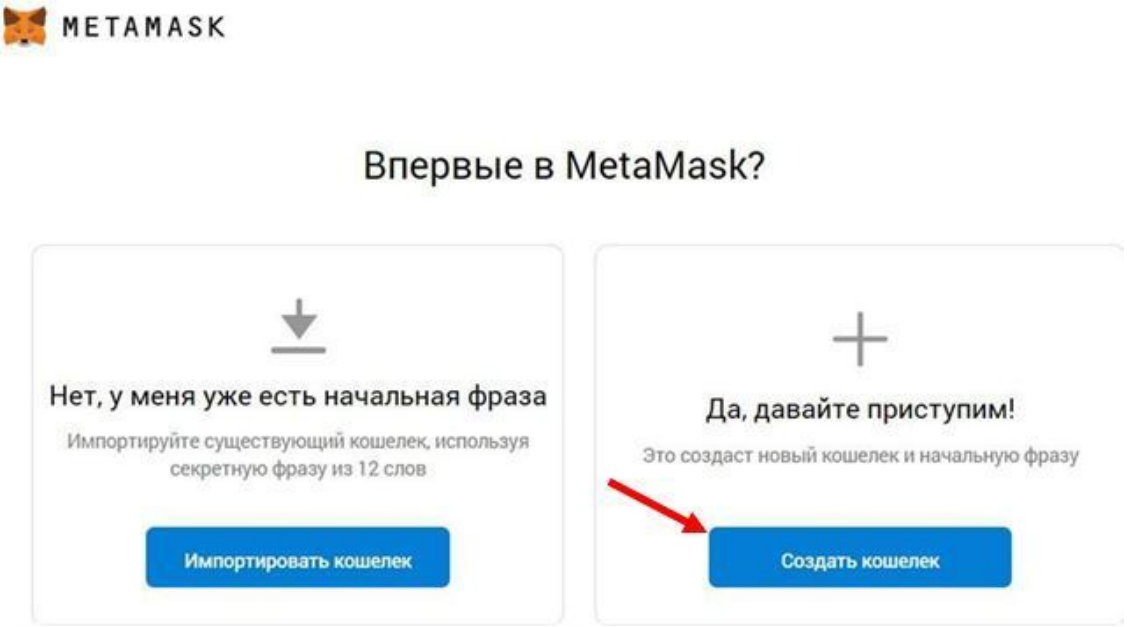

Рис. 1.7.5

Появится страница с вопросом о сборе статистики для улучшения плагина. Здесь необходимо нажать кнопку I agree (рис. 1.7.6).

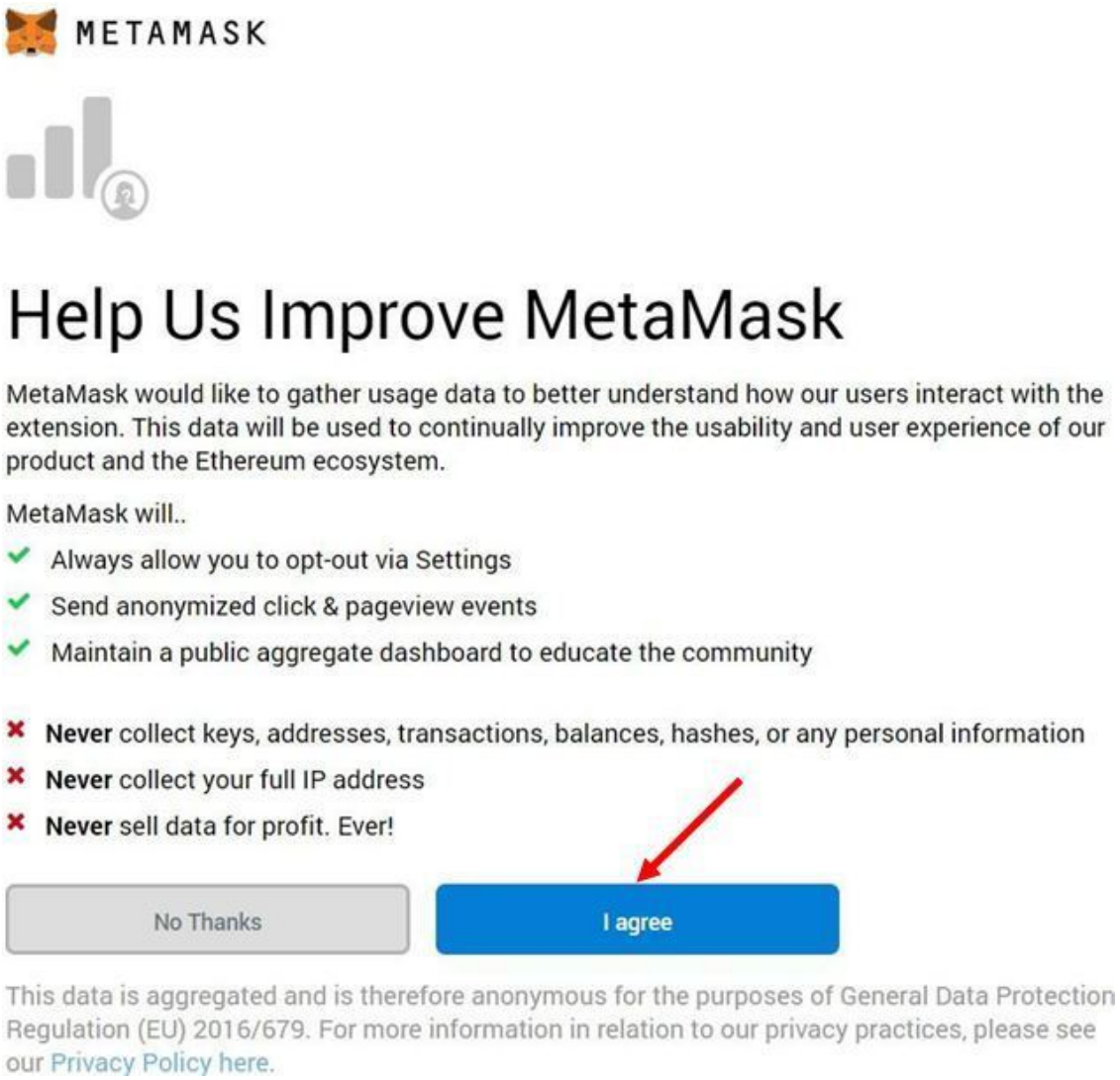

Рис. 1.7.6

Для начала задайте пароль вашего нового криптокошелька, согласитесь с условиями использования плагина и нажмите кнопку «Создать» (рис. 1.7.7).

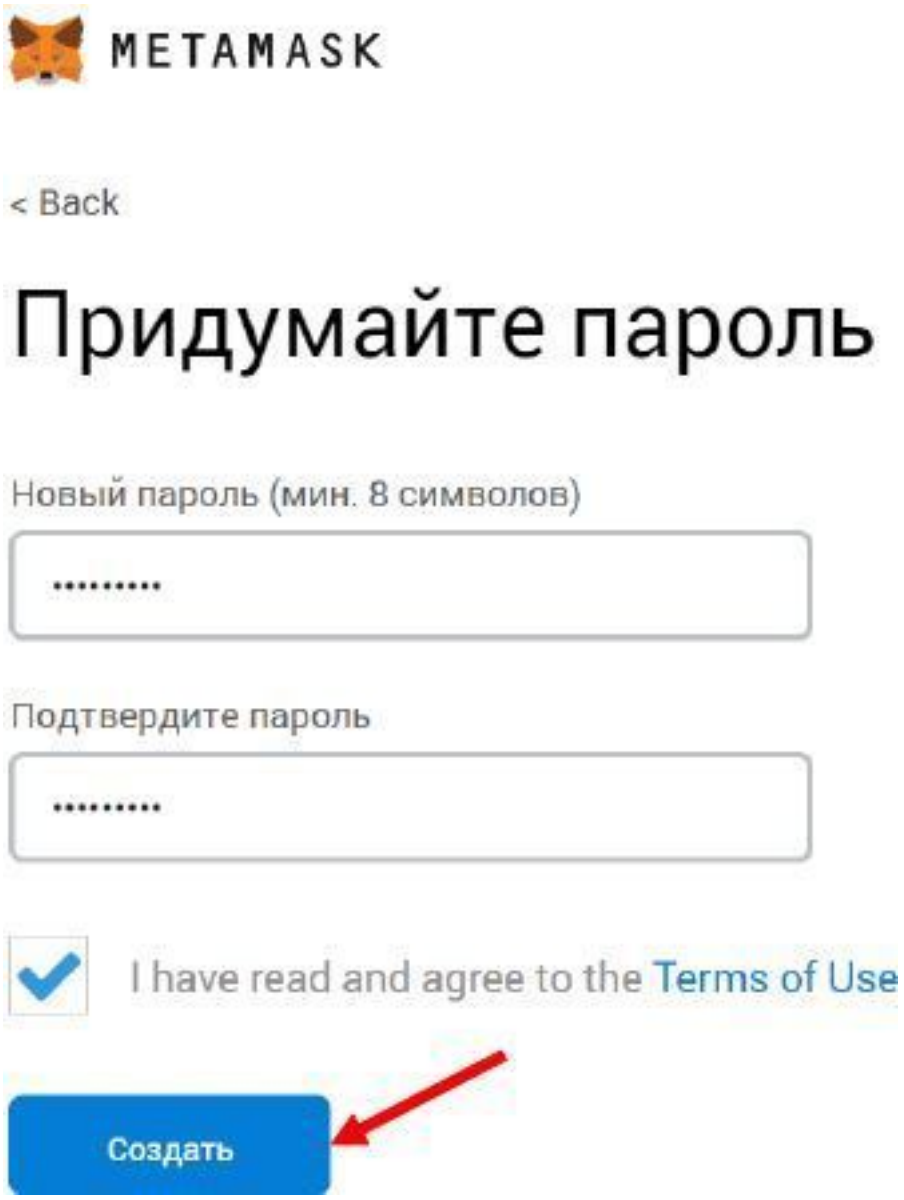

Рис. 1.7.7

Появится страница с секретной фразой для подключения нового кошелька. Для того чтобы получить доступ к данному кошельку с другого компьютера или при переустановке плагина, необходима секретная фраза кошелька. Ее можно увидеть, нажав на изображение замка на данной странице. Поскольку мы будем работать с кошельками эмулятора Ganache, нам эта фраза неинтересна. Нажмите кнопку «Напомнить позже» (рис. 1.7.8).

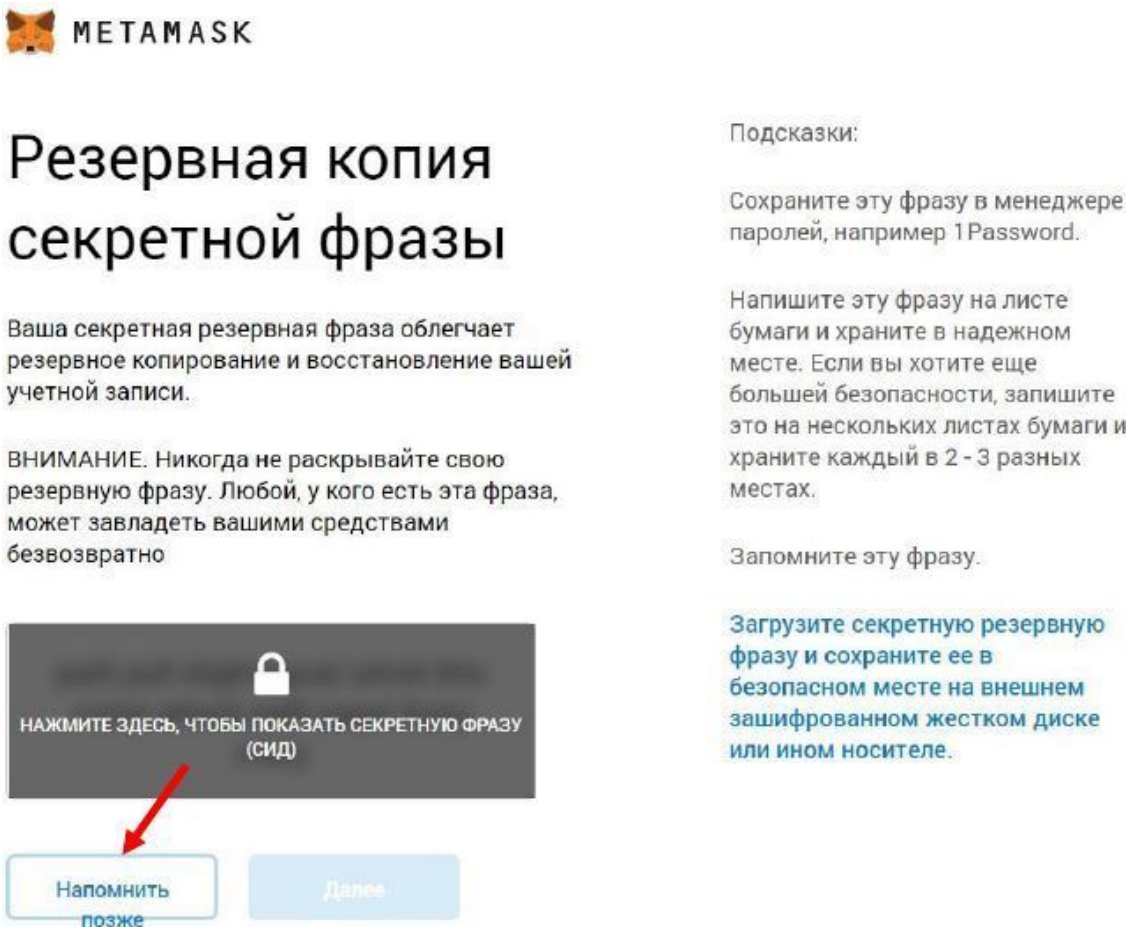

Рис. 1.7.8

Далее откроется страница нашего нового криптокошелька Account 1 (рис. 1.7.9). Мы видим, что в данном кошельке 0 ETH. Если нажать выпадающий список в верхнем правом углу страницы, можно увидеть, что кошелек подключен к основной сети Ethereum. Но нам необходимо подключиться к нашей виртуальной сети, созданной в эмуляторе Ganache. Для этого в списке сетей выберите пункт «Пользовательский RPC» (рис. 1.7.9).

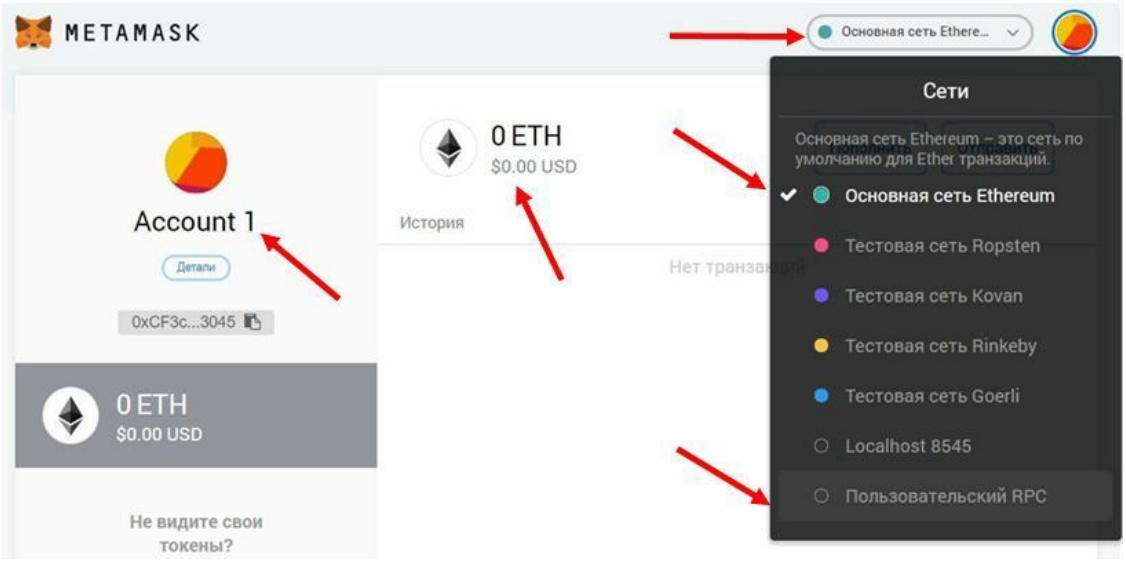

Рис. 1.7.9

Откроется страница с настройками для подключения к новой сети. Здесь необходимо ввести имя подключаемой сети на ваше усмотрение, я задал MyNet. Затем необходимо определить адрес и порт нашей виртуальной сети «Новый RPC URL». Адрес и порт отображаются в верхней части окна эмулятора Ganache (рис. 1.7.10).

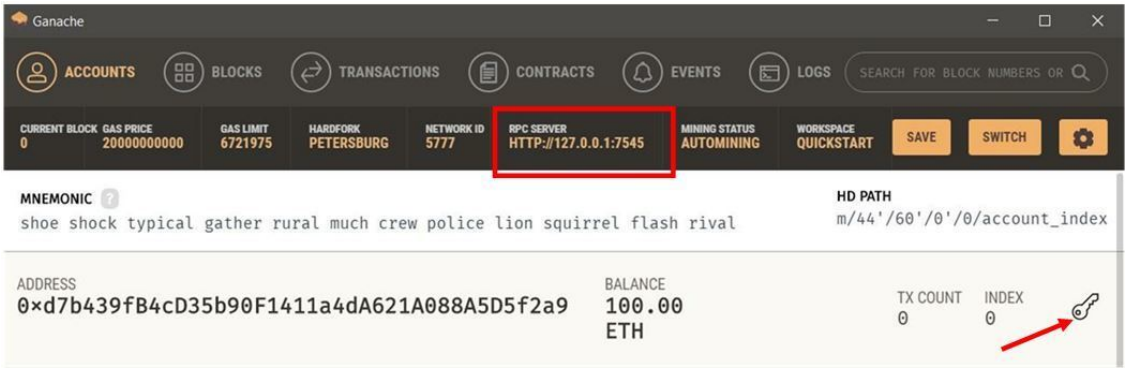

Рис. 1.7.10

«Закрыть» (крестик в верхнем правом углу) (рис. 1.7.11).

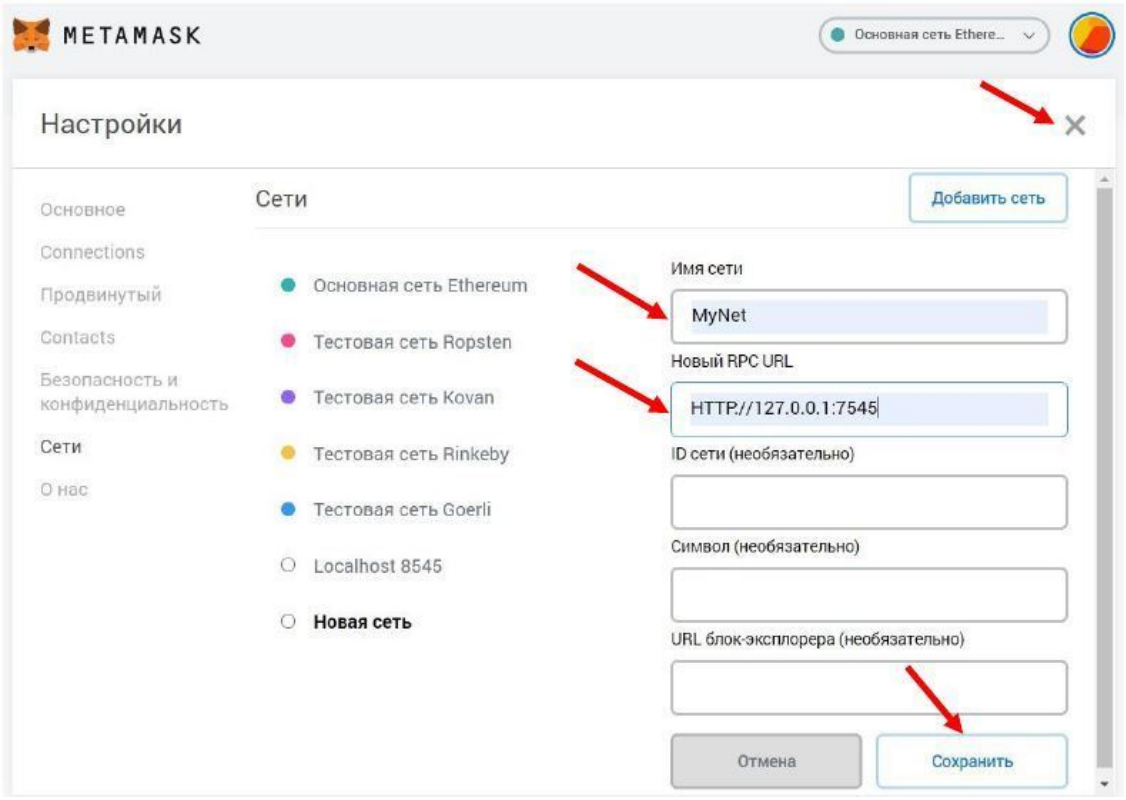

Рис. 1.7.11

Теперь импортируем наши кошельки из эмулятора. Для этого щелкните по цветному кругу в верхнем правом углу страницы нового кошелька и в появившемся меню выберите пункт «Импортировать счет» (рис. 1.7.12).

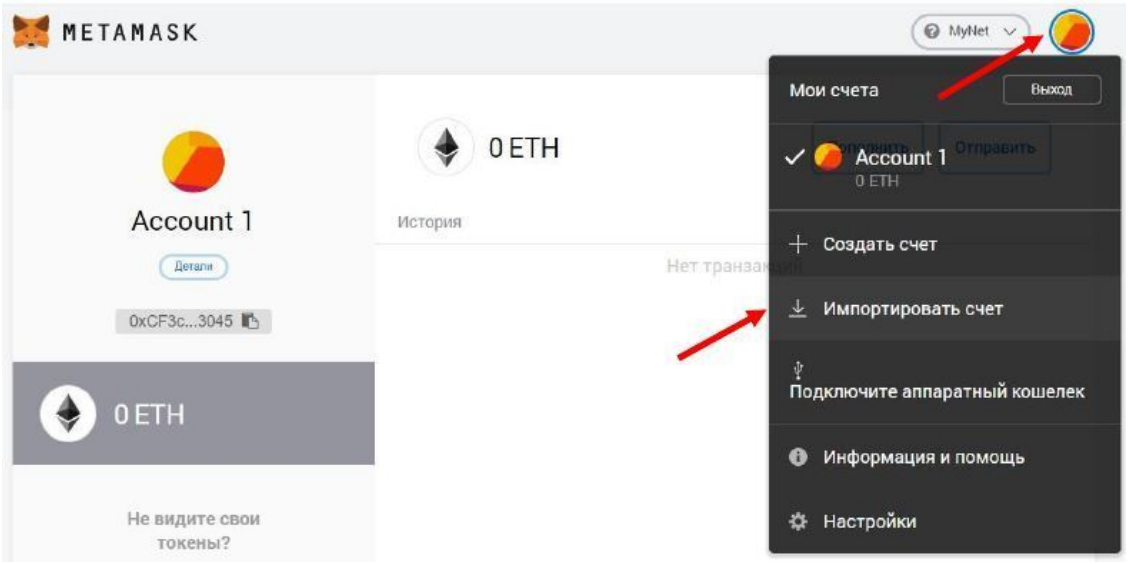

Рис. 7.12

Появится страница импортирования «Новый счет» (рис. 1.7.13).

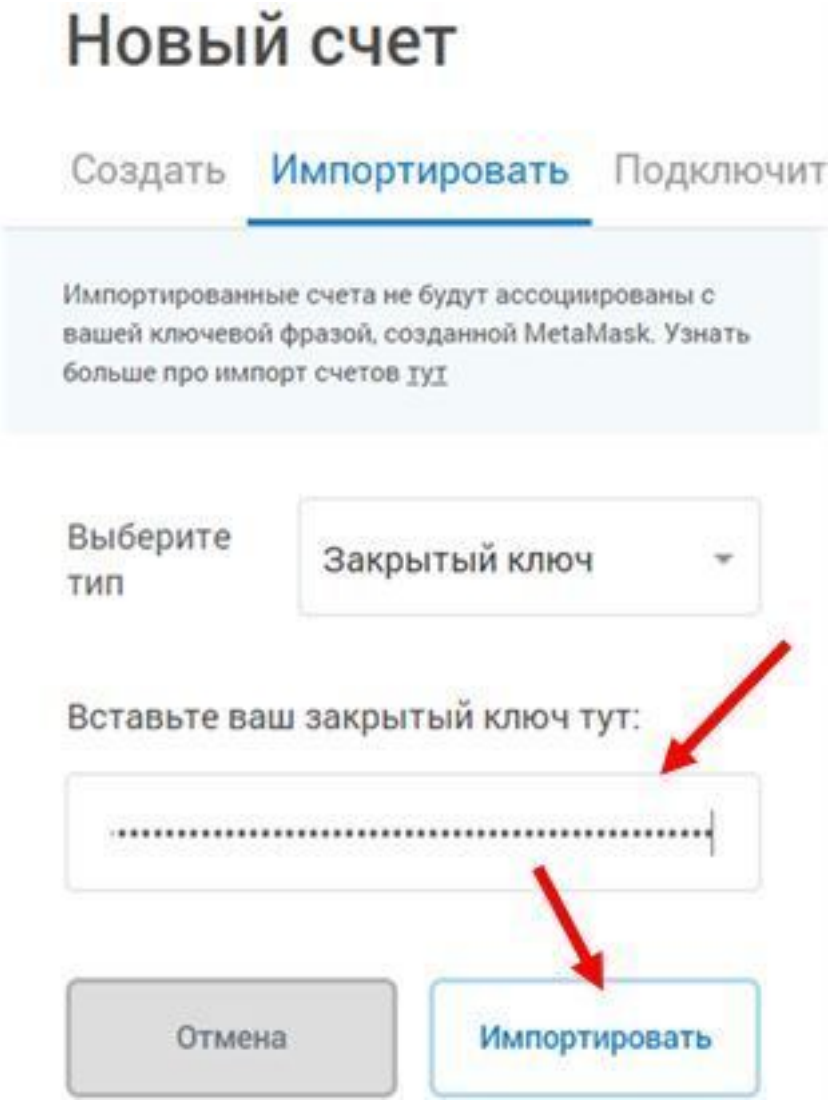

Рис. 1.7.13

На данной странице необходимо указать закрытый ключ нашего кошелька из эмулятора. Для получения ключа в окне эмулятора Ganache щелкните по ключу напротив первого кошелька (рис. 1.7.10) и в появившемся окне ACCOUNT INFORMATION скопируйте в буфер обмена параметр Private key (рис. 1.7.14), а затем вставьте его из буфера обмена в поле «Вставьте ваш закрытый ключ тут:» на странице MetaMask (рис. 1.7.13).

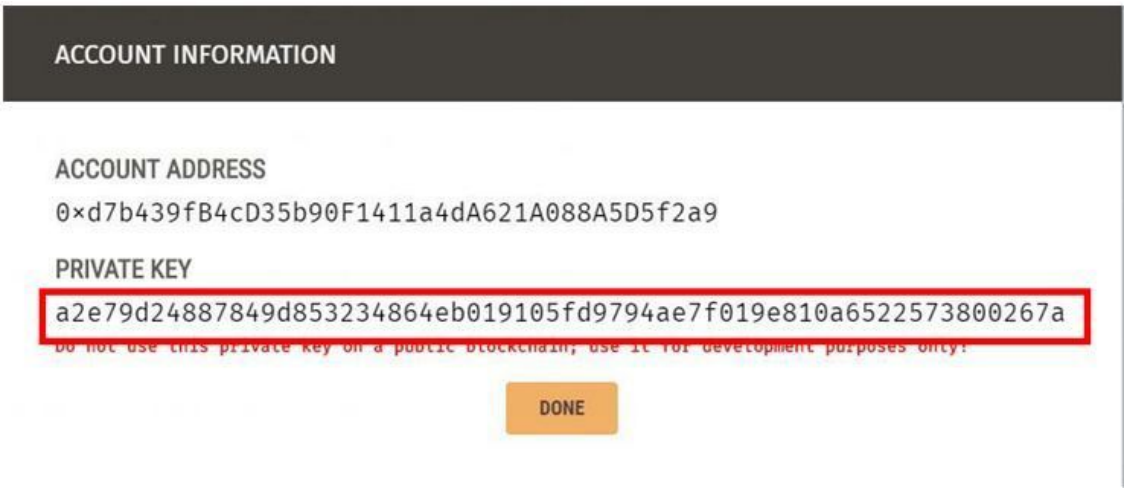

Рис. 1.7.14

Для импорта на странице MetaMask нажмите кнопку «Импортировать» (рис. 1.7.13). Итак, мы импортировали наш первый кошелек из эмулятора. При импортировании он получил имя Account 2. Импортируйте второй кошелек из нашего эмулятора в MetaMask. Для этого повторите все действия начиная с рис. 1.7.12. Только скопируйте Private key не первого кошелька, а второго. В итоге при щелчке по цветному кругу на странице MetaMask выпадающее меню будет выглядеть как на рис. 1.7.15.

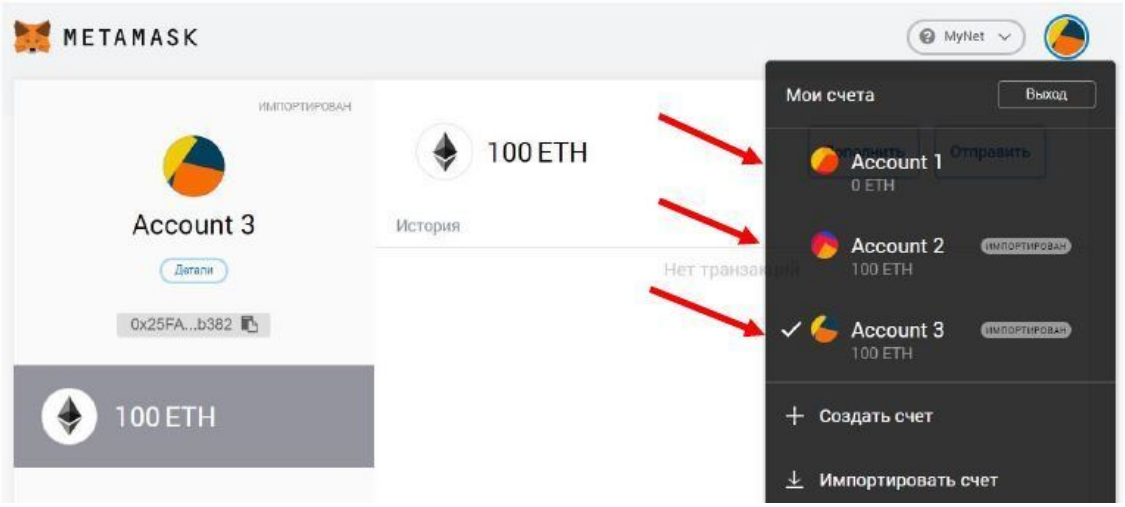

Рис. 1.7.15

Обратите внимание на то, что сейчас мы имеем три счета. Один, Account 1, подключен к публичной сети Ethereum, а два других, Account 2 и Account 3, подключены к эмулятору Ganache и на этих счетах находится по 100 ETH. **Окно эмулятора закрывать нельзя!**

Замечание. Если мы не перезапускали эмулятор, то на наших счетах будет не 100 ETH, а несколько меньше, так как ранее мы тестировали демонстрационный проект.

Теперь протестируем работу плагина MetaMask в связке с эмулятором Ganache. Для этого совершим транзакцию, то есть переведем 10 ETH со счета Account 3 на счет Account 2. Перевод средств инициализируется нажатием кнопки «Отправить» на странице MetaMask (рис. 1.7.16).

Рис. 1.7.16

Появится окно выбора получателя. Здесь можно указать адрес кошелька получателя, например, с рис. 1.7.14. Однако мы переводим между своими кошельками, поэтому перейдем по ссылке «Перевод между моими аккаунтами» (рис. 1.7.17).

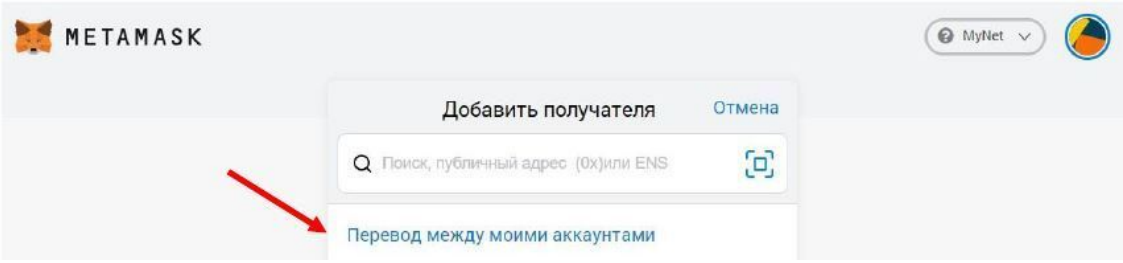

Рис. 1.7.17

Появится список наших счетов, где выбираем Account 2 (рис. 1.7.18).

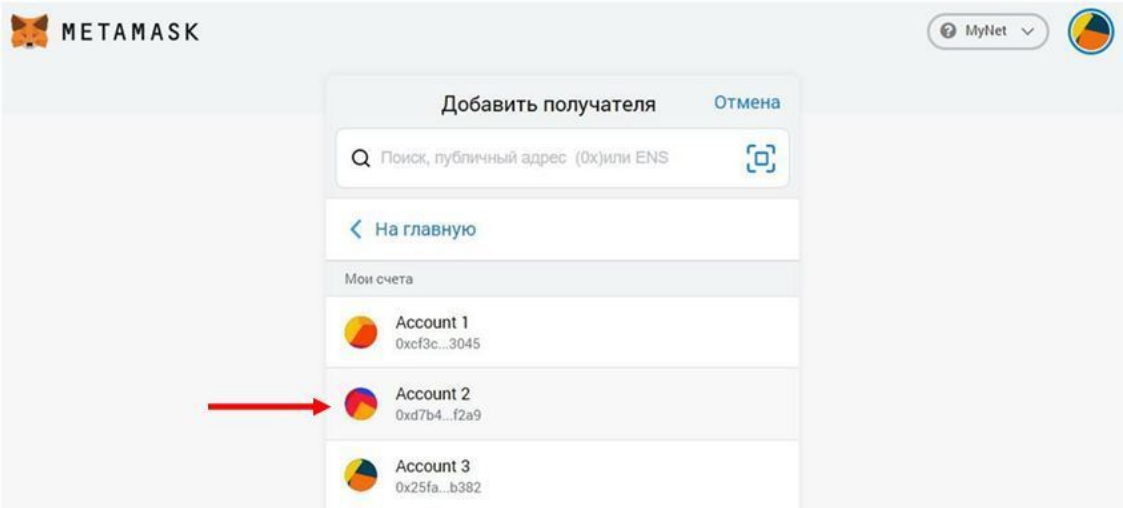

Рис. 1.7.18

Появится страница параметров транзакции (рис. 1.7.19). Здесь мы определим сумму 10 ETH и комиссию за перевод как «Средний» (чем быстрее перевод, тем он дороже). Нажмите кнопку «Далее».

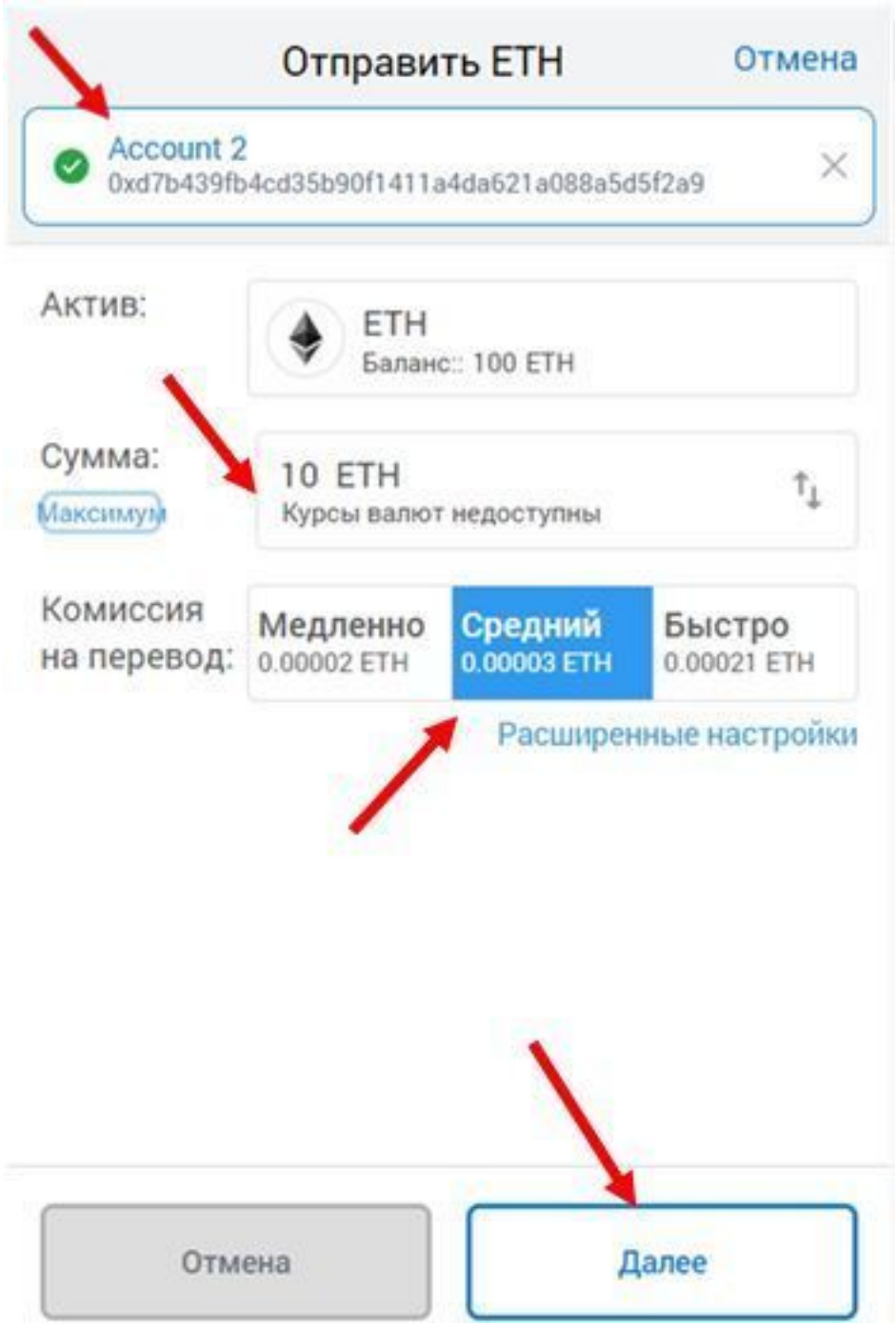

Рис. 1.7.19

Далее появится страница подтверждения транзакции (рис. 1.7.20). На данной странице мы видим, что мы переводим 10 ETH со счета Account 3 на счет Account 2. Комиссия за перевод будет равна 0,000034 ETH. Для подтверждения транзакции нажмите кнопку «Подтвердить» (рис. 1.7.20).

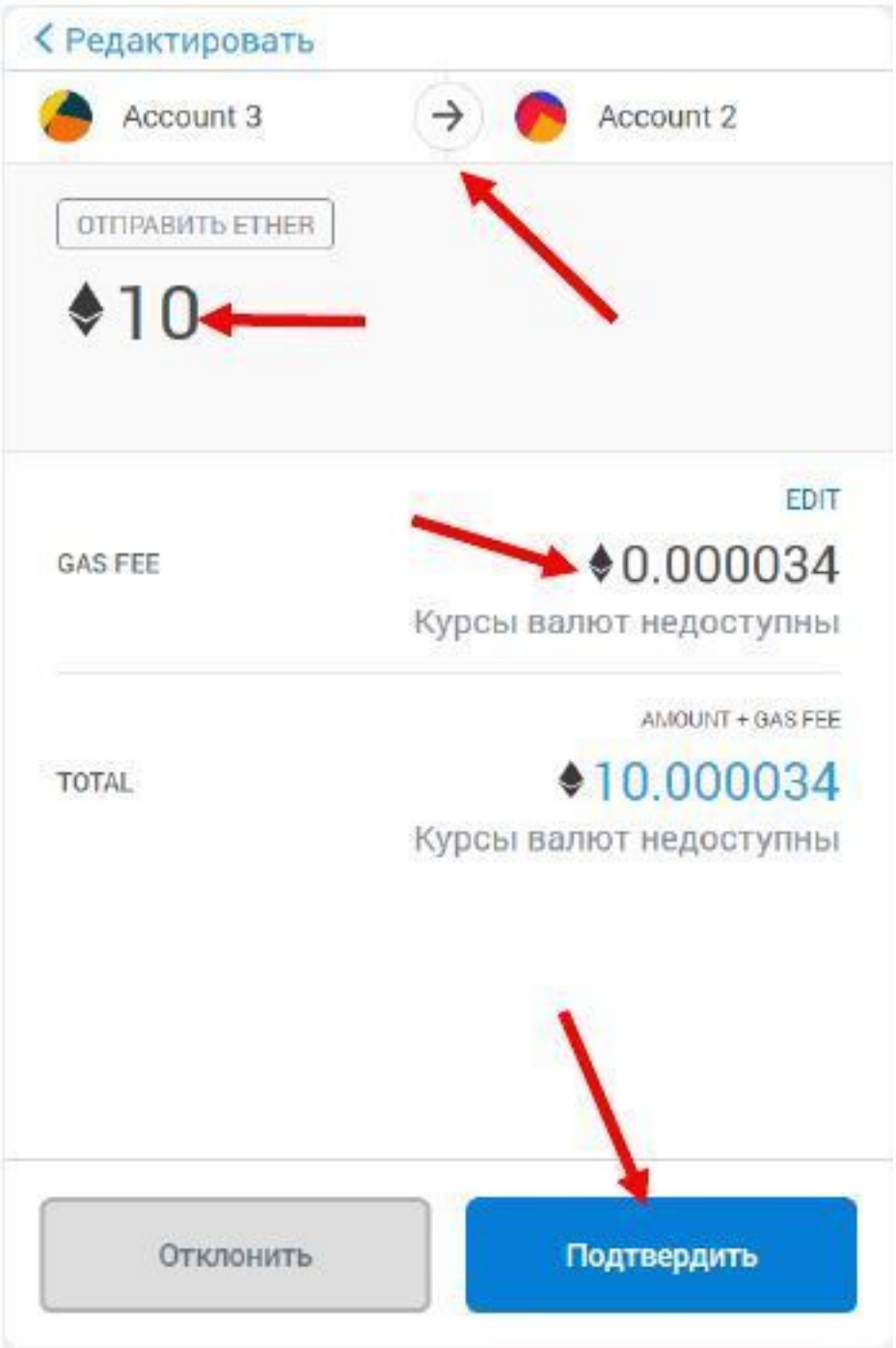

Рис. 1.7.20

Посмотрим на результат нашей транзакции. Если зайти в меню наших счетов, то мы можем увидеть, что на счету Account 2 стало 110 ETH, а на счету Account 3 – 89,999966 ETH (рис. 1.7.21).

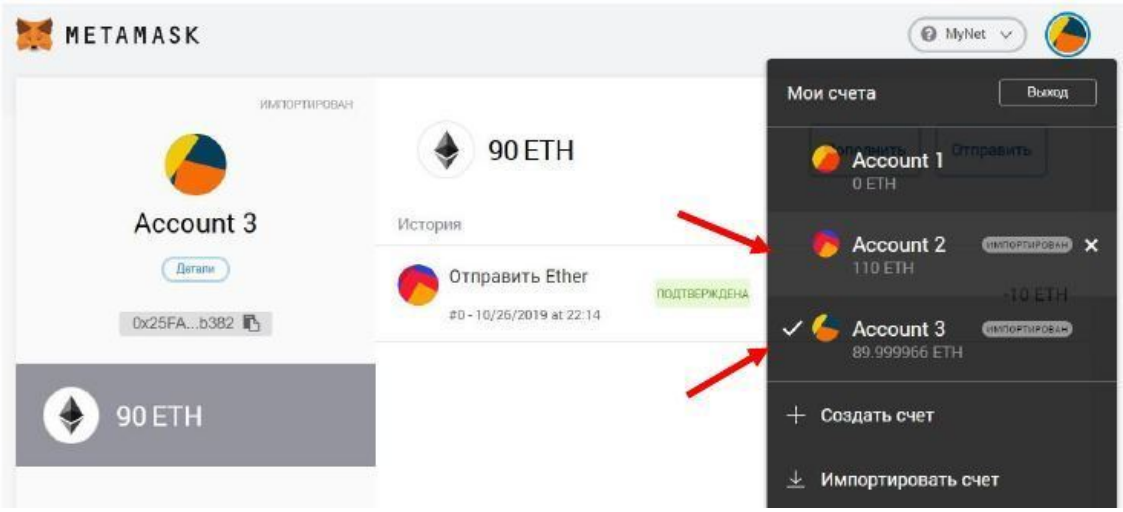

Рис. 1.7.21

Если открыть окно эмулятора, то мы видим аналогичную картину. С одного кошелька на другой кошелек было переведено 10 ETH (рис. 1.7.22).

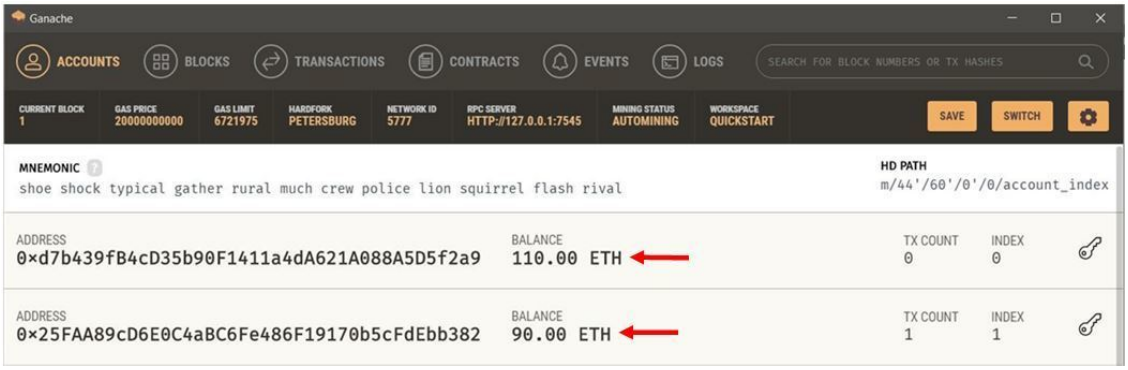

Рис. 1.7.22

Урок 8. Установка офлайн-криптокошелька MyEtherWallet

Аннотация. В данном уроке мы рассмотрим инструмент для запуска наших смарт-контрактов – офлайн-криптокошелек MyEtherWallet [6].

Для перевода ETH с одного криптокошелька на другой нам вполне хватит и плагина MetaMask. Однако для тестовых запусков наших смарт-контрактов нам понадобится офлайн-версия электронного криптокошелька MyEtherWallet. Этот криптокошелек позволяет как переводить ETH, так и публиковать, и тестировать смарт-контракты.

> **Замечание.** В принципе MyEtherWallet может полностью заменить MetaMask, но он гораздо сложней в использовании.

Для установки MyEtherWallet перейдите по адресу https://github.com/kvhnuke/etherwallet/releases и скачайте архив с офлайн-версией криптокошелька etherwallet-v.3.40.0.zip (версия может быть иной) (рис. 1.8.1).

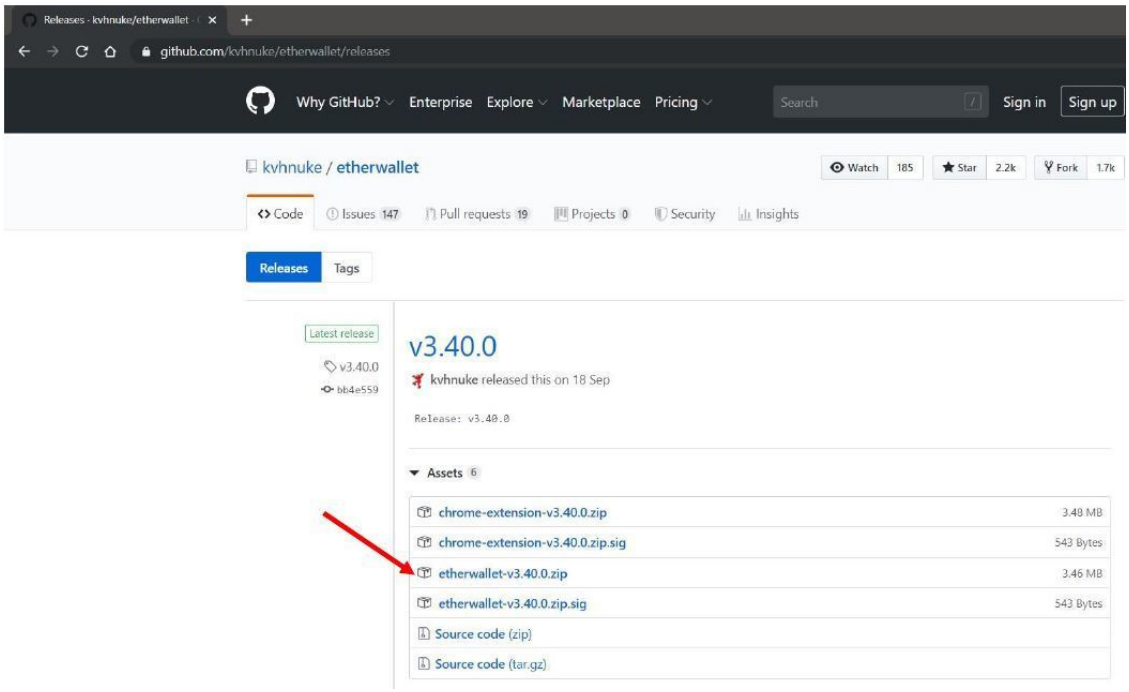

Рис. 1.8.1

После окончания скачивания распакуйте архив в любую папку. Для запуска MyEtherWallet в распакованной папке откройте файл index.htm (рис. 1.8.2).

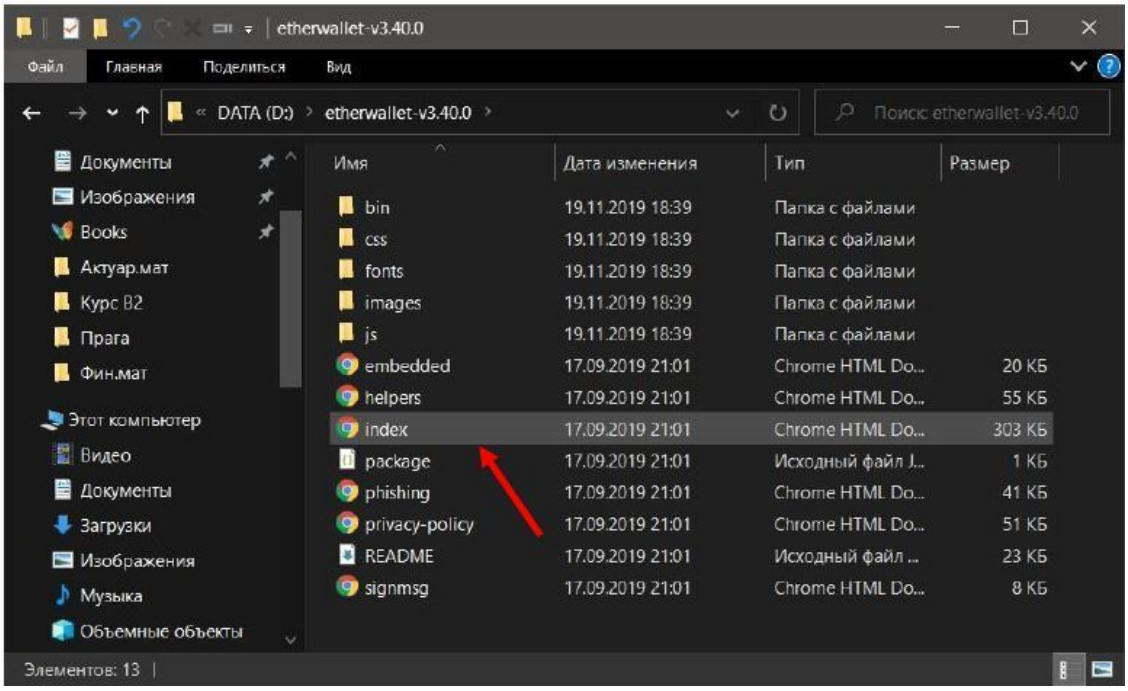

Рис. 1.8.2

После этого запустится веб-браузер с начальной страницей и сообщением о выходе новой онлайн-версии криптокошелька (рис. 1.8.3).

Рис. 1.8.3

Просто закройте окно с сообщением, щелкнув по значку «X» в верхнем правом углу сообщения. Мы попадем на начальную страницу криптокошелька.

Теперь подключим криптокошелек к эмулятору Ganache. Для этого щелкните по выпадающему списку выбора сети блокчейн, расположенному в верхнем правом углу страницы, и выберите последний пункт в списке Add Custom Network / Node (рис. 1.8.4).

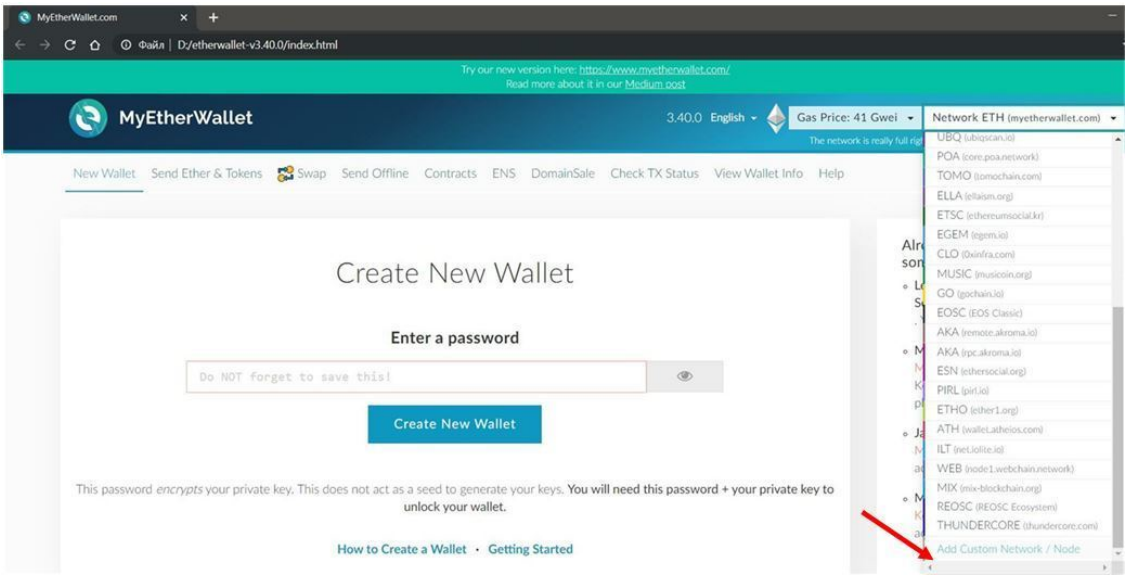

Рис. 1.8.4

Для подключения криптокошелька к эмулятору блокчейна Ganache нам необходимо узнать адрес и порт нашего эмулятора Ganache. Для этого запустите Ganache, на стартовом экране выберите вариант запуска QUICKSTART. Затем в окне эмулятора обратите внимание на параметр RPC SERVER. Здесь мы видим запись вида «HTTP://127.0.0.1:7545». Это значит, что адрес сервера – http://127.0.0.1, а порт – 7545 (рис. 1.8.5).

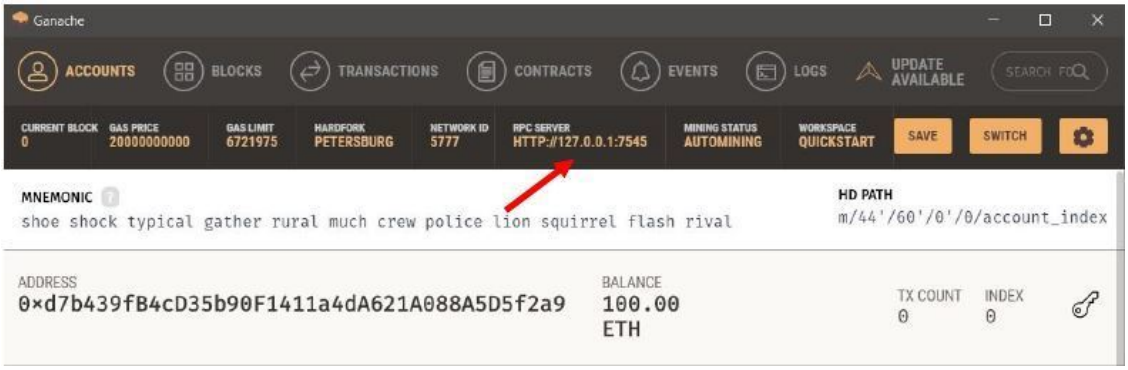

Рис. 1.8.5

Замечание: Не закрывайте окно эмулятора Ganache. Просто сверните его – он нам еще понадобится.

Теперь вернемся в окно криптокошелька MyEtherWallet. После выбора настройки Add Custom Network / Node (рис. 1.8.4) появится окно с настройками сервера и порта блокчейн-сети (рис. 1.8.6).

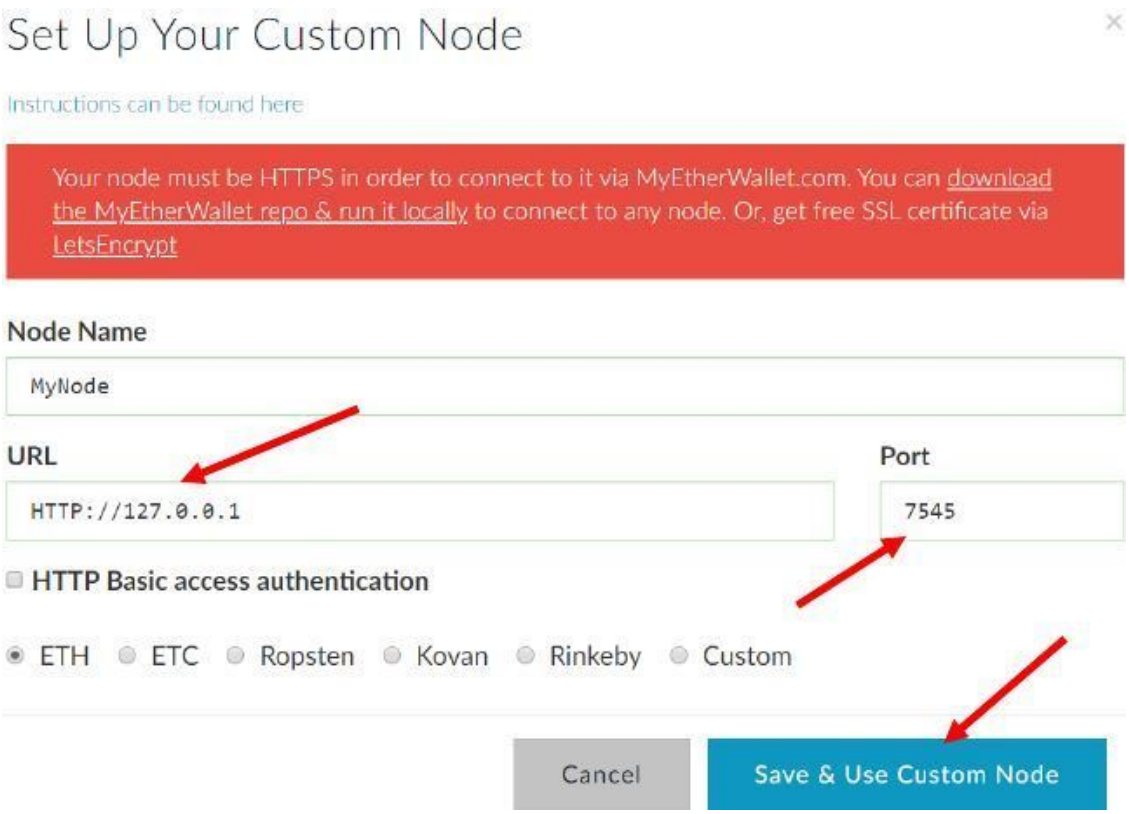

Рис. 1.8.6

В данном окне задаем следующие настройки: Node Name – любое имя без пробелов (мы задали MyNode), URL – http://127.0.0.1, Port – 7545 (рис. 1.8.6). Мы их получили из эмулятора Ganache (рис. 1.8.5). Для сохранения настроек нажмите кнопку Save & Use Custom Node. Страница MyEtherWallet примет вид как на рис. 1.8.7.

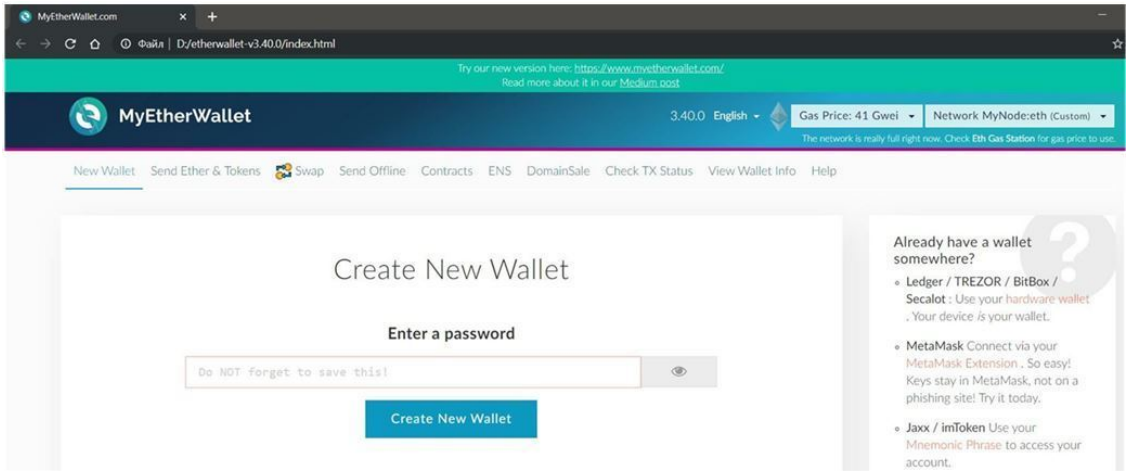

Рис. 1.8.7

Проверим работу криптокошелька, проверим баланс ETH на одном из наших счетов в эмуляторе Ganache. На странице MyEtherWallet нажмите ссылку View Wallet Info (рис. 1.8.8).

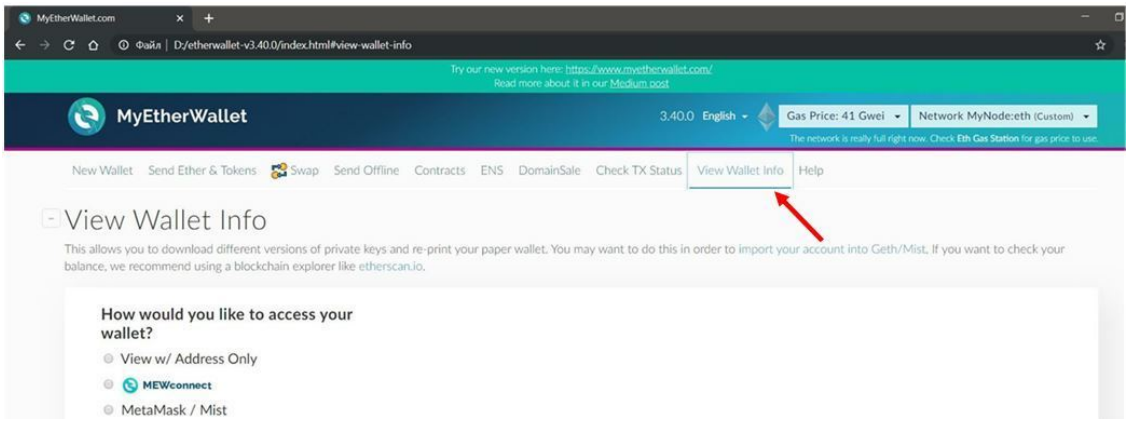

Рис. 1.8.8

Разверните окно Ganache и скопируйте из него адрес первого счета (рис. 1.8.9).

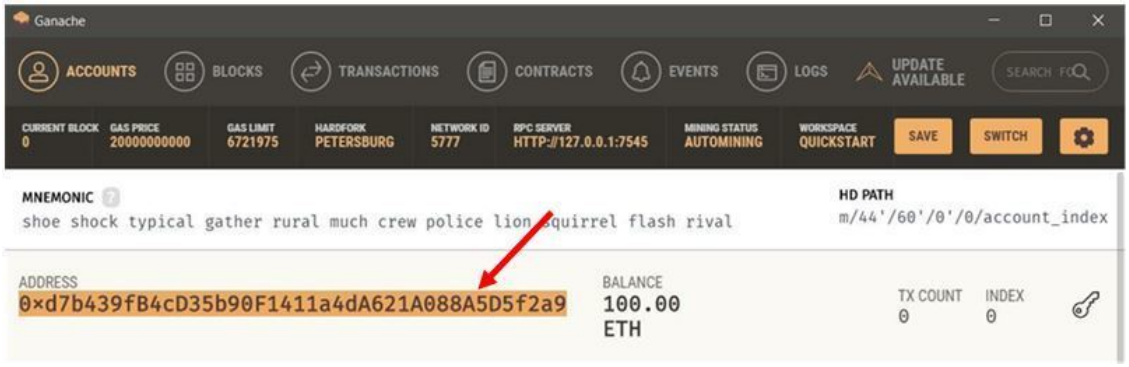

Рис. 1.8.9

Вернитесь на страницу MyEtherWallet и выберите способ доступа к криптокошельку как «View w / Address Only». В поле «Your Address» вставьте адрес, скопированный из окна Ganache (рис. 1.8.9).

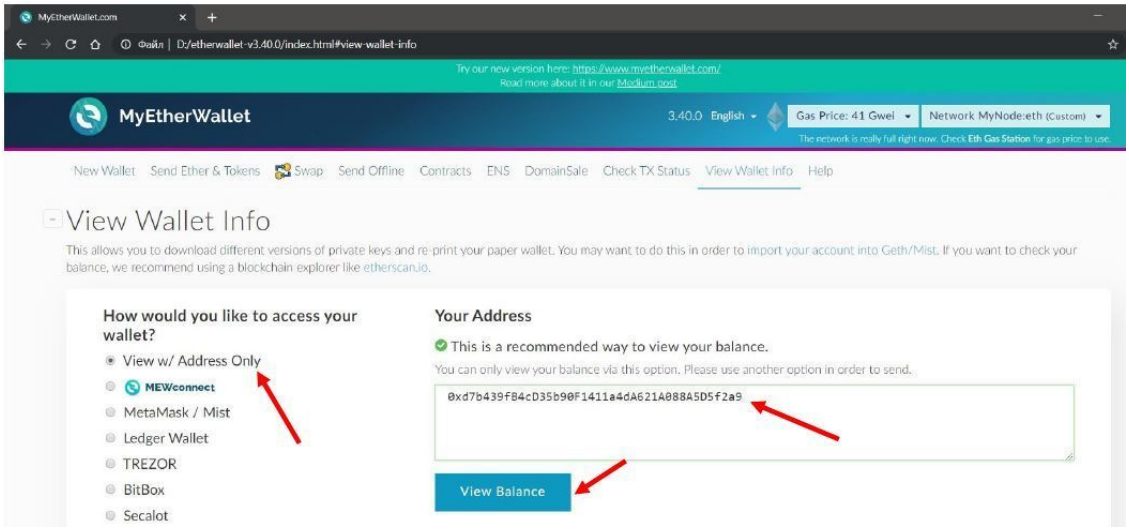

Рис. 1.8.10

Для просмотра баланса на нашем счете нажмите кнопку View Balance (рис. 1.8.10). Откроется страница с данными о нашем счете в Ganache, где мы видим, что наш баланс равен 100 ETH (рис. 1.8.11).

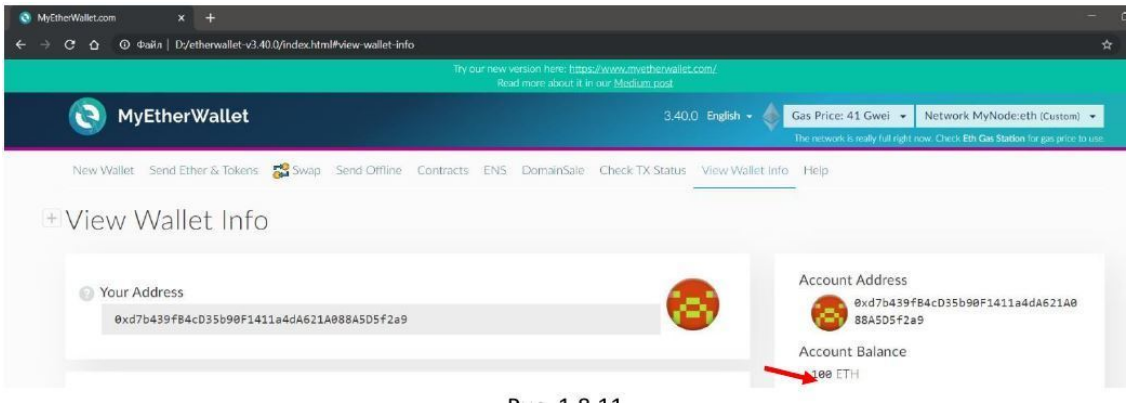

Рис. 1.8.11

Итак, мы подключили криптокошелек MyEtherWallet к эмулятору Ganache и проверили его работу. Теперь мы можем использовать MyEtherWallet для тестирования смарт-контрактов.

Заключение

На этом мы заканчиваем первую неделю нашего курса. В рамках недели мы создали рабочее окружение – «песочницу» – для создания и тестирования смарт-контрактов в блокчейн-сети Ethereum. В следующем модуле мы рассмотрим технологии создания простейших смарт-контрактов с помощью языка программирования Solidity.

> **Замечание.** Электронная версия данного учебного курса размещена на учебном портале Stepik по адресу https://stepik.org/60331. В конце каждого урока электронной версии добавлен небольшой аттестационный тест, а в конце каждой недели – практические задания для самостоятельного выполнения. Тем, кто сдаст все тесты и выполнит все практические задания, выдается сертификат по разработке смарт-контрактов и распределенных приложений (DApps) для блокчейн-сети Ethereum в операционной системе Windows.

Неделя № 2. Создание и тестирование простейших смарт-контрактов

Введение

В этой неделе мы рассмотрим состав проекта языка программирования смарт-контрактов Solidity [8], [9], создание и управление проектом, ознакомимся с основами синтаксиса языка Solidity и структурой смарт-контракта, а также разберем создание и запуск простейших смарт-контрактов.

Урок 1. Структура проекта Solidity в VS Code

Аннотация. В данном уроке мы рассмотрим файловую структуру проекта языка программирования смарт-контрактов Solidity. Будут рассмотрены все папки и файлы, входящие в проект, и описано их назначение.

Для начала рассмотрим более подробно файловую структуру проекта на языке программирования смарт-контрактов Solidity.

В языке программирования Solidity главная программная единица – это смарт-контракт. Смарт-контракт – это аналог программы в обычных языках программирования, именно смарт-контракт компилируется, публикуется и выполняется в блокчейн-сети Ethereum.

Перед тем как создавать смарт-контракты, нам необходимо создать проект в Truffle. Проект – это набор файлов и папок, необходимый для создания, публикации и выполнения смарт-контрактов. Давайте рассмотрим структуру проекта MetaCoin, который мы создали в предыдущих уроках.

Запустите VS Code и откройте проект MetaCoin. Для этого в окне VS Code на панели Explorer нажмите кнопку Open Folder (рис. 2.1.1).

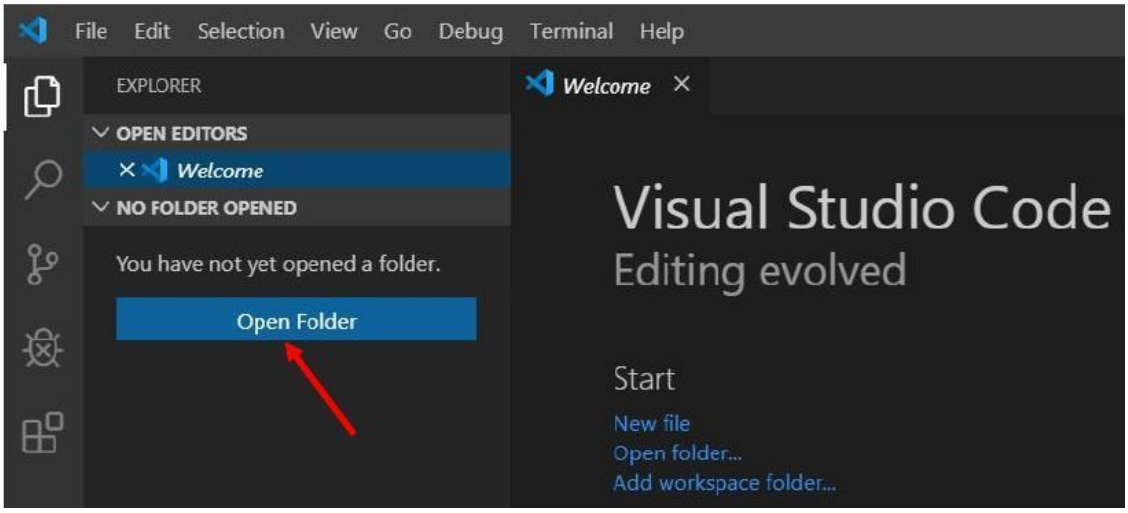

Рис. 2.1.1

Появится окно выбора папки с проектом (рис. 2.1.2).

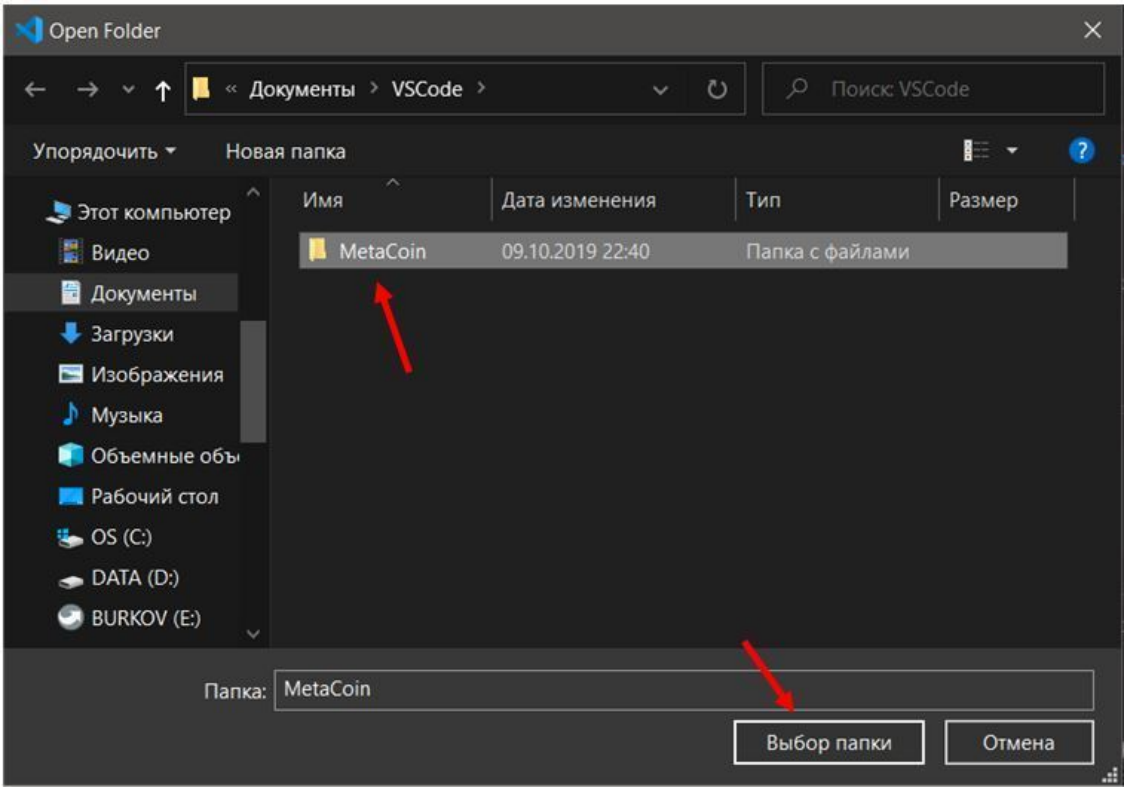

Рис. 2.1.2

Выберите папку «MetaCoin» и нажмите кнопку «Выбор папки».

Замечание. По умолчанию VS Code создает проекты в папке «Документы / VS Code».

После открытия проекта MetaCoin окно VS Code примет вид как на рис. 2.1.3.

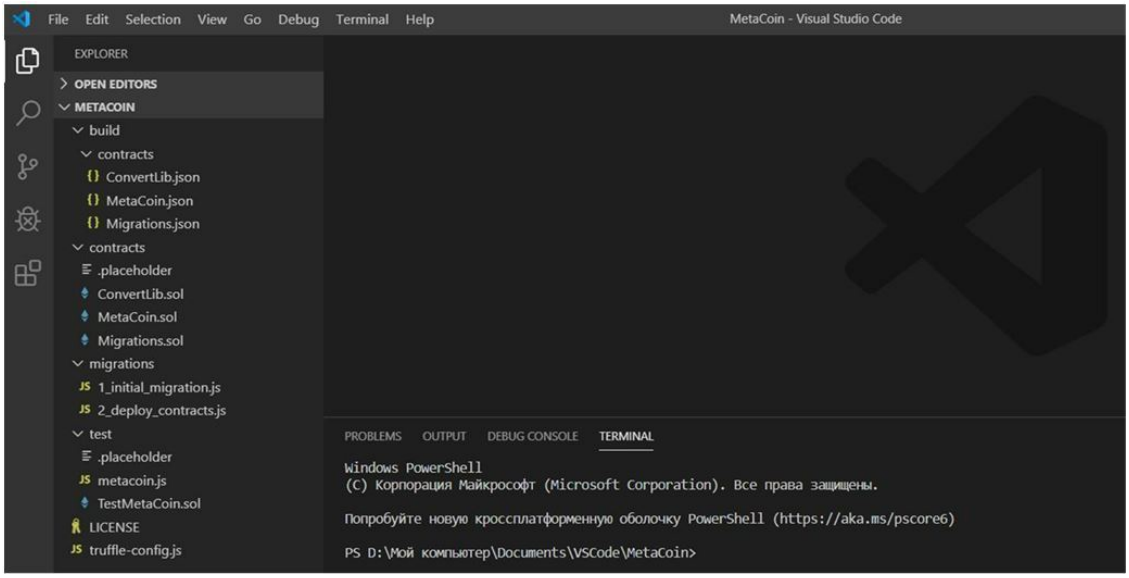

Рис. 2.1.3

На панели EXPLORER мы видим файловую структуру проекта MetaCoin. Давайте рассмотрим структуру подробнее. Проект содержит четыре папки: build, contracts, migrations и test. Рассмотрим назначение этих папок.

Для начала рассмотрим папку contracts. Это самая главная папка проекта, в ней находятся файлы с кодом наших смарт-контрактов. Это файлы с расширением sol. В нашем случае в проекте MetaCoin в данной папке мы видим три файла: ConvertLib.sol, MetaCoin.sol и Migrations.sol. Хотелось бы отметить файл Migrations.sol (рис. 2.1.4).

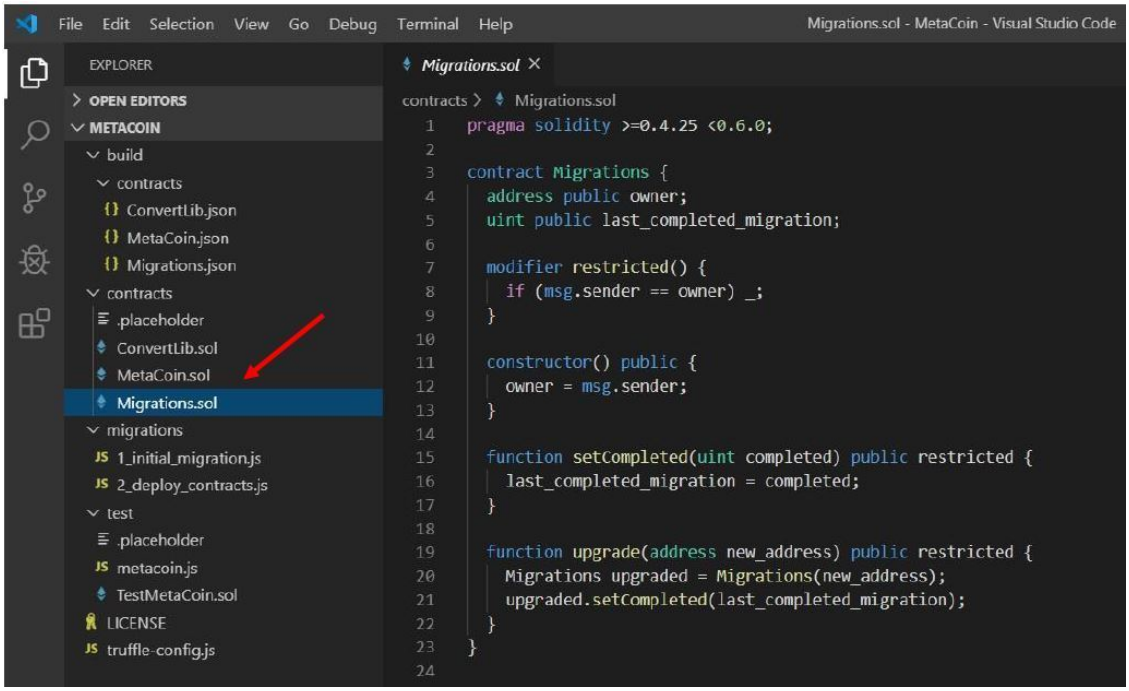

Рис. 2.1.4

Этот файл предназначен для публикации других смарт-контрактов проекта в сети Ethereum. Изменять файл Migrations.sol не рекомендуется. Остальные два файла реализуют логику проекта.

Теперь перейдем к папке build. В папке build располагаются файлы с расширением json – это откомпилированные в формат json смарт-контракты. Их компиляция была рассмотрена в предыдущих уроках. Они содержат бинарный код смарт-контракта, который понимает сеть Ethereum, а также входные и выходные параметры, необходимые для запуска смарт-контракта (рис. 2.1.5).

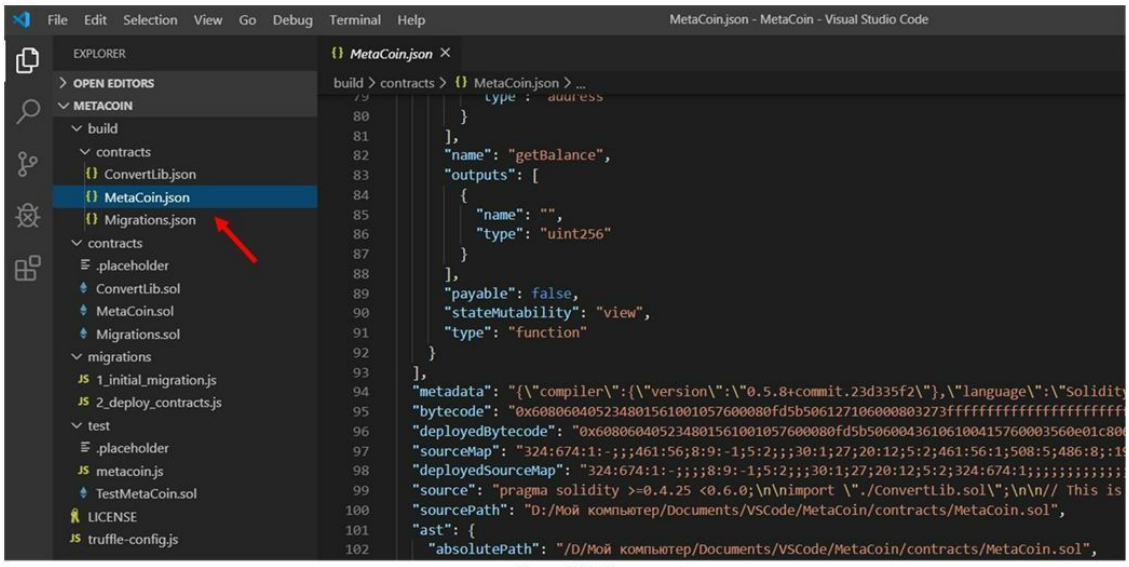

Рис. 2.1.5

Папка migrations содержит js-файлы (файлы в формате JavaScript). Они предназначены для публикации смарт-контрактов проекта в сеть Ethereum. Все файлы в папке migrations должны начинаться с цифры. Первым файлом должен быть файл 1_initial_migrations.js (рис. 2.1.6).

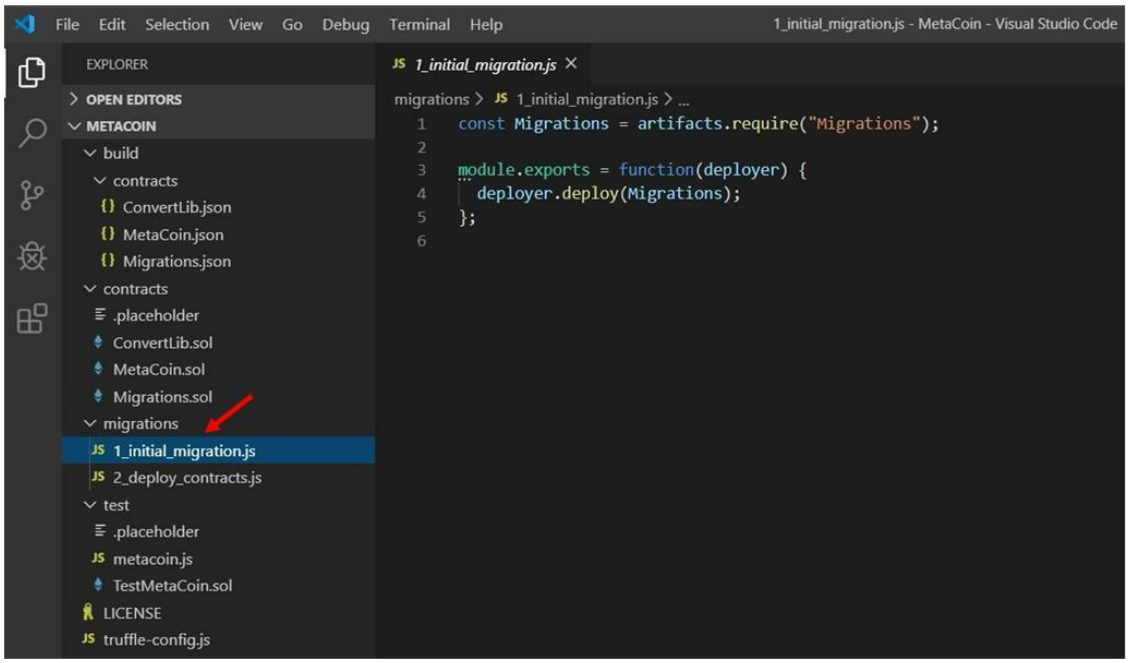

Рис. 2.1.6

Этот файл публикует в сети файл Migration.json, который инициирует публикацию других смарт-контрактов проекта. Второй js-файл предназначен для публикации остальных двух смарт-контрактов (рис. 2.1.7).

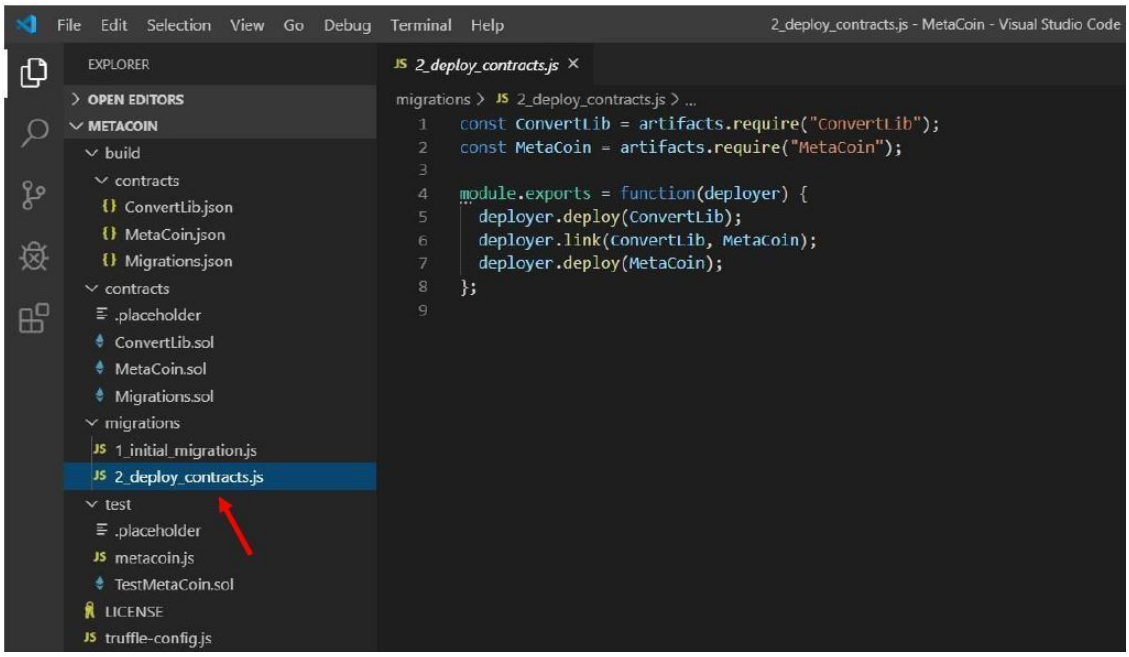

Рис. 2.1.7.

Замечание. В представленном на рис. 2.1.6 файле осуществляется публикация сразу двух смарт-контрактов. Однако для упрощения понимания проекта рекомендуется создавать отдельный публикационный js-файл для каждого смарт-контракта. Поэтому так мы будем поступать в наших проектах позже.

Наконец, рассмотрим папку test. В папке test располагаются смарт-контракты для отладки нашего проекта. Важным отличием смарт-контрактов от обычных программ является тот факт, что их невозможно изменить после публикации в сети блокчейн. Поэтому столь важное внимание уделяется отладке смарт-контрактов. В рассматриваемой папке мы видим два файла – sol- и js-файлы. Смарт-контракт с расширением sol содержит код на языке Solidity, предназначенный для тестирования проекта (рис. 2.1.8).

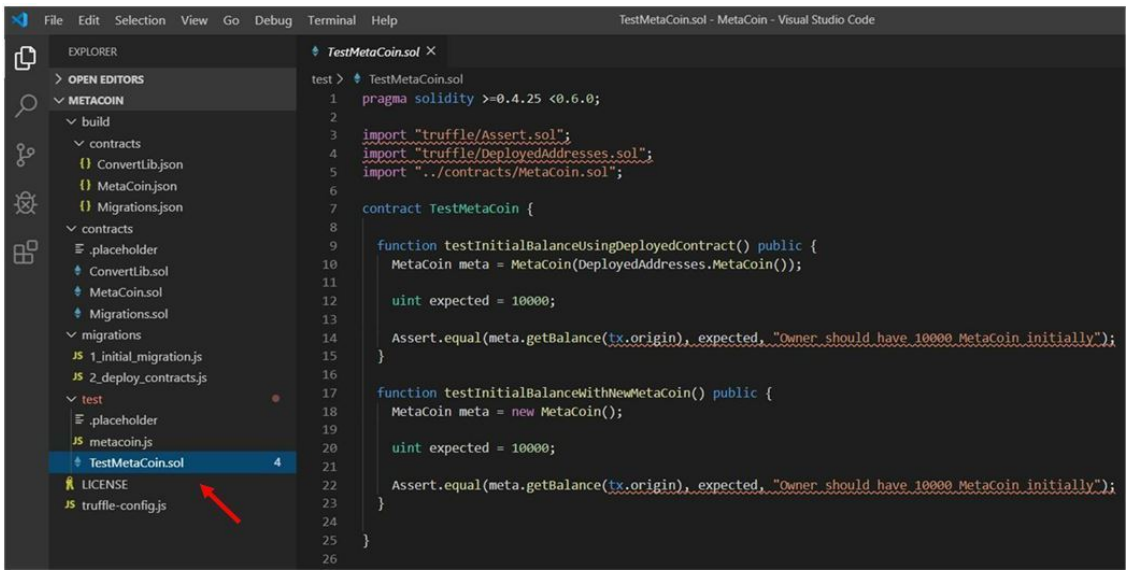

Рис. 2.1.8

Файл с расширением js – это файл, который содержит код на языке JavaScript. Он предназначен для выполнения тестового смарт-контракта фреймворком Truffle (рис. 2.1.9).

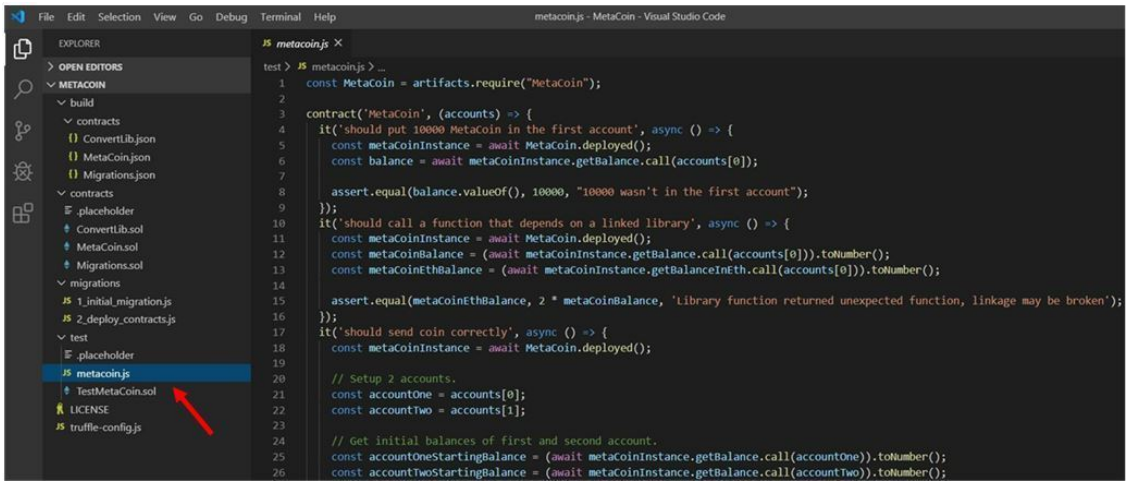

Рис. 2.1.9

В заключение урока кроме четырех рассмотренных выше папок в проекте можно заметить еще два файла – truffle-config.js и LICENSE. Файл truffle-config.js мы рассматривали в предыдущих уроках. Он предназначен для подключения проекта к серверу сети Ethereum или к эмулятору сети, например Ganache. Файл LICENSE – это обычный текстовый файл с лицензионным соглашением фреймворка Truffle (рис. 2.1.10).

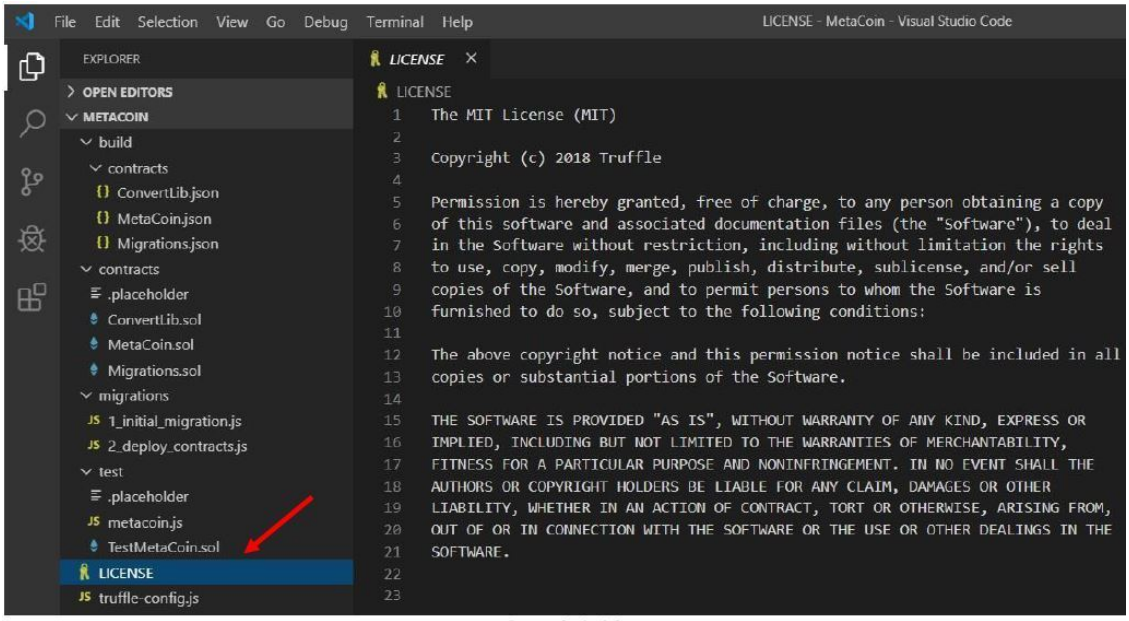

Рис. 2.1.10

Замечание. В проекте также можно заметить файлы с именем .placeholder. Это служебные файлы для системы управления версиями Git. Эти файлы можно удалить.

На этом мы заканчиваем рассмотрение файловой структуры проекта Solidity и переходим к созданию нашего проекта для создания простых смарт-контрактов. Проект MetaCoin можно закрыть, выбрав в оконном меню VS Code пункт File / Close Folder.

Урок 2. Создание нового проекта Solidity

Аннотация. В уроке рассматривается создание нового проекта на языке программирования Solidity с помощью фреймворка Truffle.

Создадим новый проект на языке программирования Solidity с помощью фреймворка Truffle. Для начала нам необходимо создать папку, где будет располагаться наш проект. Советуем создать папку проекта в папке «Документы / VS Code». Давайте назовем наш проект SimpleContracts, поскольку он будет содержать простейшие смарт-контракты. Поэтому создайте одноименную папку (рис. 2.2.1).

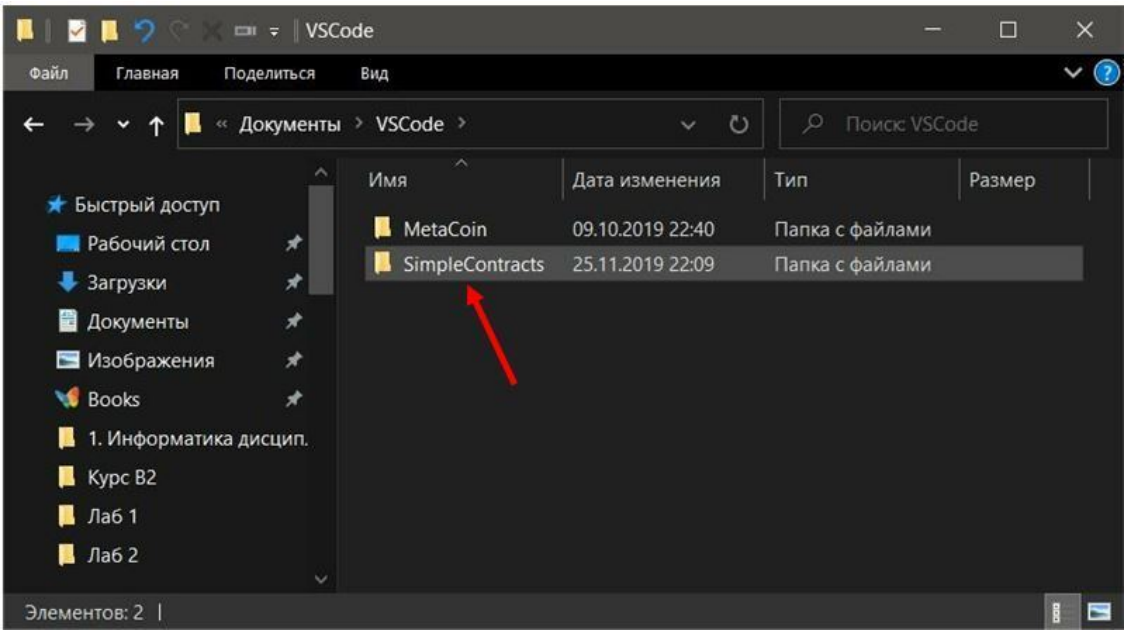

Рис. 2.2.1

Откройте созданную папку SimpleContracts, как открывали папку MetaCoin в предыдущем уроке. Мы видим, что папка пуста (рис. 2.2.2).

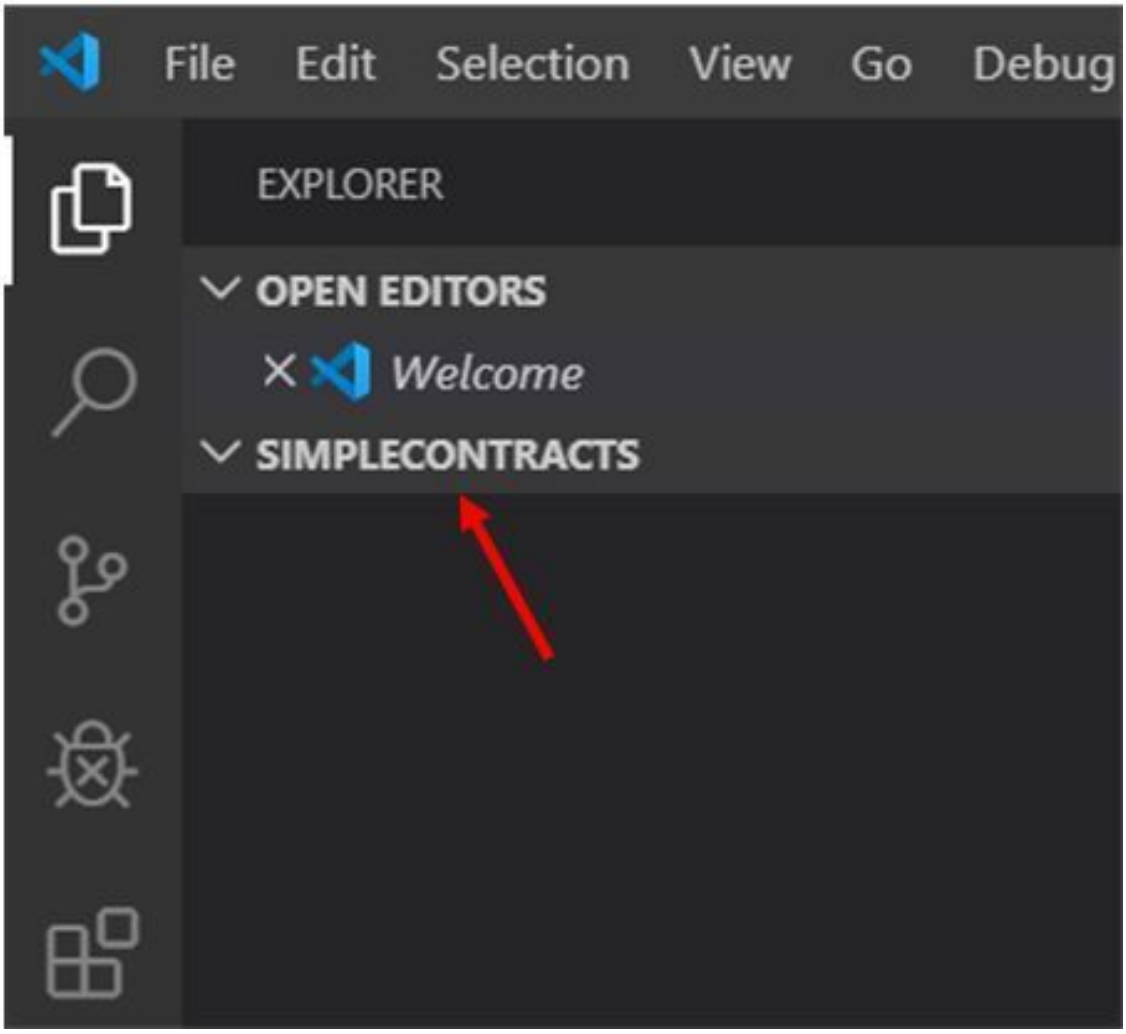

Рис. 2.2.2

Теперь, используя фреймворк Truffle, создадим в этой папке файловую структуру нового проекта Solidity. Откройте терминал, выбрав в оконном меню пункт Terminal / New Terminal. Для создания нового проекта в терминале наберите команду «truffle init» и нажмите клавишу Enter (рис. 2.2.3).

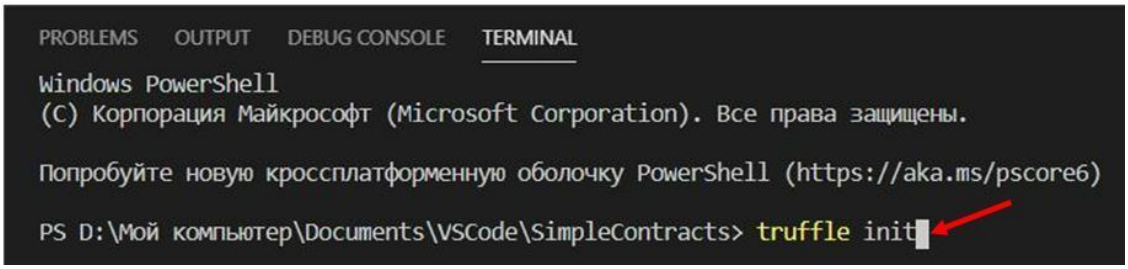

Рис. 2.2.3

Терминал примет вид как на рис. 2.2.4.

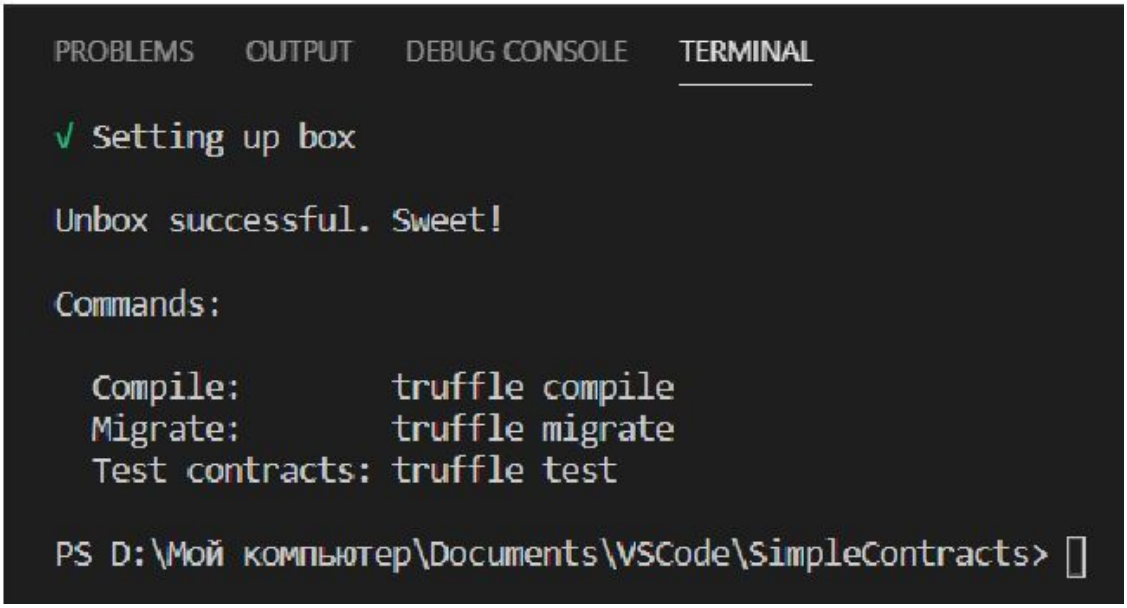

Рис. 2.2.4

На панели EXPLORER появилась рассмотренная в предыдущем уроке файловая структура проекта (рис. 2.2.5).

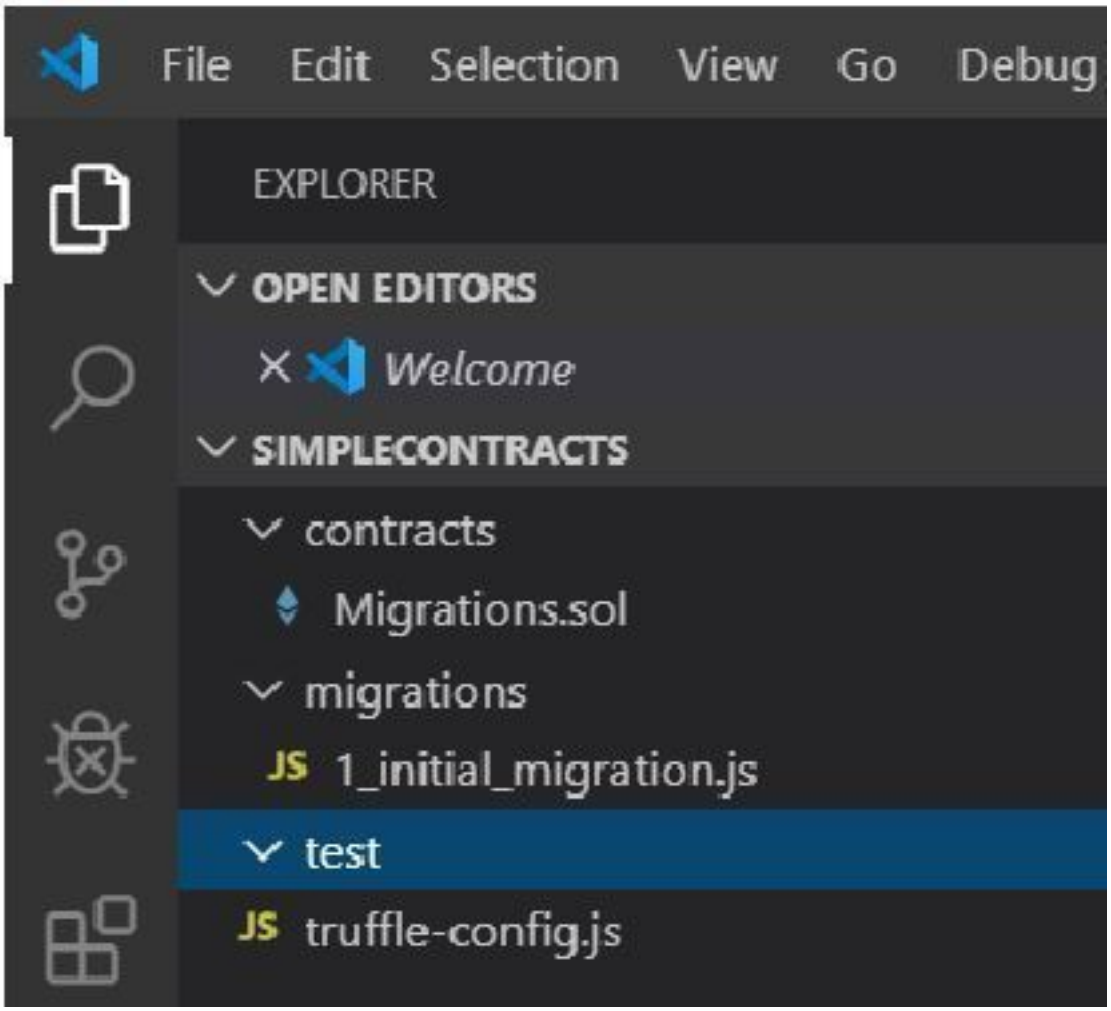

Рис. 2.2.5

Обратите внимание на то, что новый проект состоит только из трех файлов: Migrations.sol, 1_initial_migration.js и truffle-config.js. Файлы Migrations.sol, 1_initial_migration.js полностью самодостаточны, нам их изменять не нужно. Файл truffle-config.js будет необходим при запуске нашего смарт-контракта. Поэтому его настройку мы рассмотрим позднее, в уроке, посвященному запуску смарт-контрактов.

Папка test пуста. На данном этапе нам пока рано заниматься глубокой отладкой наших простых смарт-контрактов. Поэтому пока оставим эту папку пустой.

Мы видим, что в нашем проекте отсутствует папка с откомпилированными контрактами build. Давайте создадим эту папку. Для этого нам необходимо просто откомпилировать все смарт-контракты проекта. В нашем случае это смарт-контракт Migrations.sol. Для компиляции всех смарт-контрактов проекта необходимо в терминале выполнить команду «truffle compile» (рис. 2.2.6).

Рис. 2.2.6

Терминал примет вид как на рис. 2.2.7.

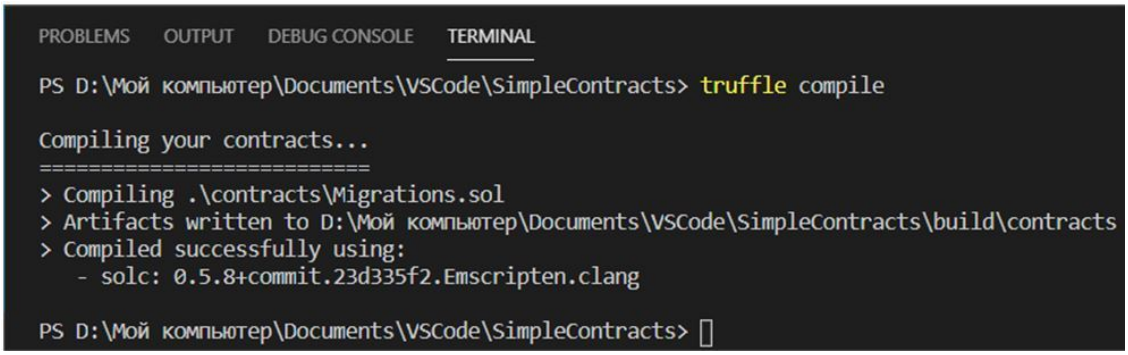

Рис. 2.2.7

После выполнения в терминале команды «truffle compile» на панели EXPLORER появится папка build с откомпилированным файлом Migrations.json (рис. 2.2.8).

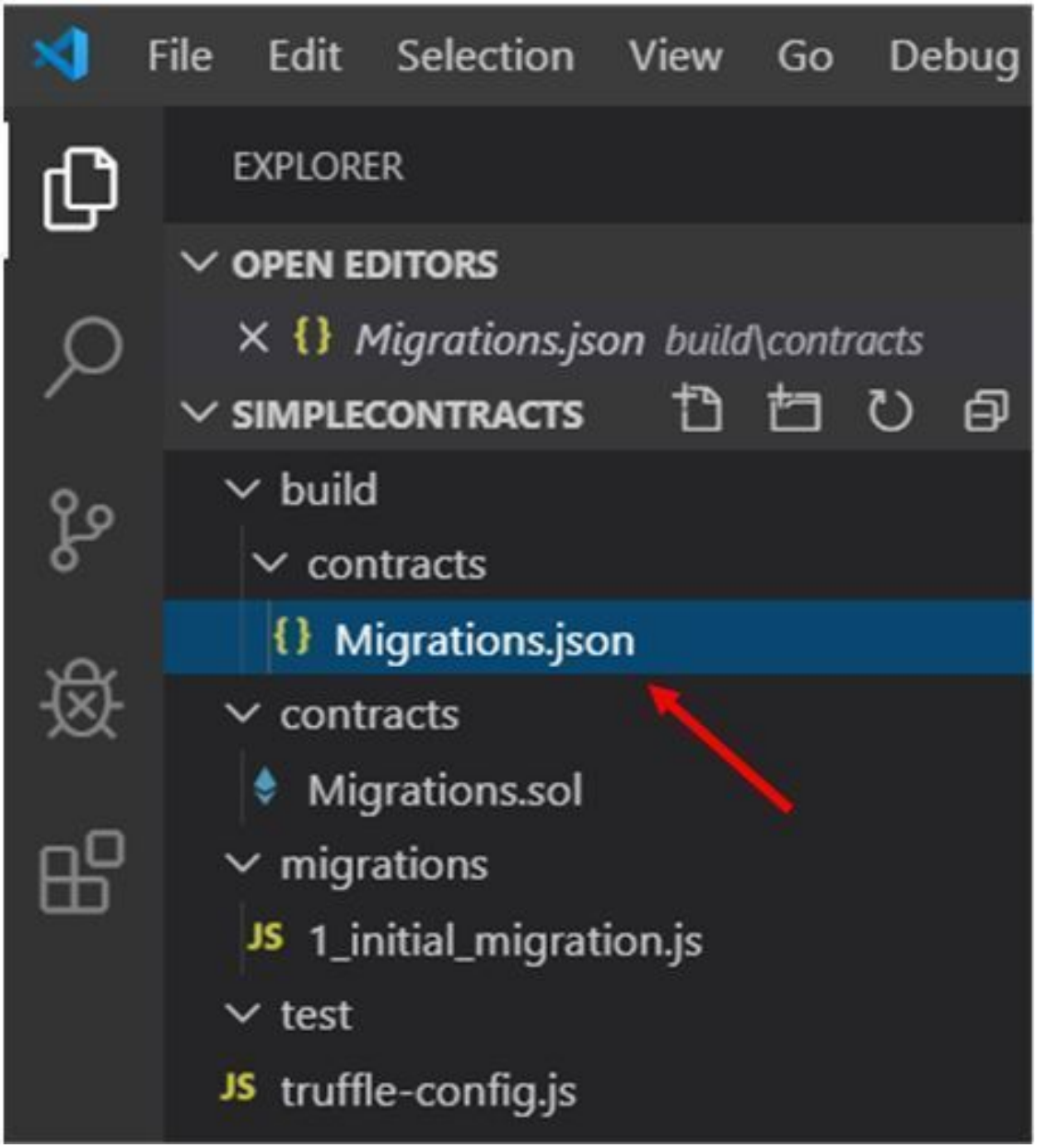

Рис. 2.2.8

На этом мы заканчиваем урок, посвященный созданию проекта на языке программирования смарт-контрактов Solidity. В следующем уроке мы напишем наш первый смарт-контракт на языке Solidity.

Урок 3. Создаем наш первый смарт-контракт Hello World

Аннотация. Теперь давайте создадим наш первый смарт-контракт. По традиции пусть это будет «Hello World!!!», т. е. смарт-контракт будет выводить сообщение «Hello World!!!» [11].

Для начала мы создадим пустой смарт-контракт, а затем поместим в него необходимый код.

Для создания нового смарт-контракта HelloWorld в терминале наберите команду «truffle create contract “HelloWorld”» и нажмите клавишу Enter (рис. 2.3.1).

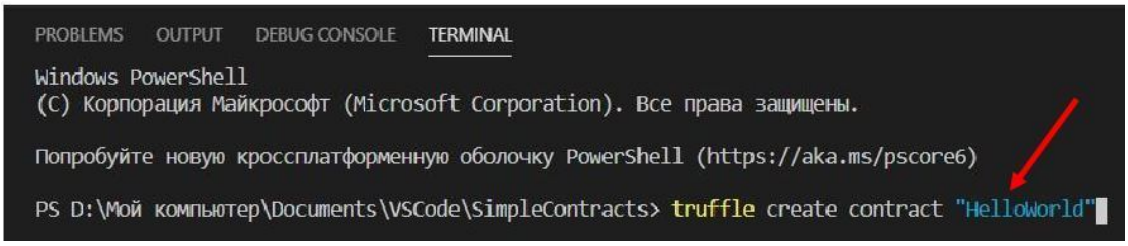

Рис. 2.3.1

После выполнения команды в нашем проекте появится новый файл смарт-контракта HelloWorld.sol. Давайте рассмотрим его содержимое. Откройте смарт-контракт, щелкнув по нему на панели EXPLORER. Окно VS Code примет вид как на рис. 2.3.2.

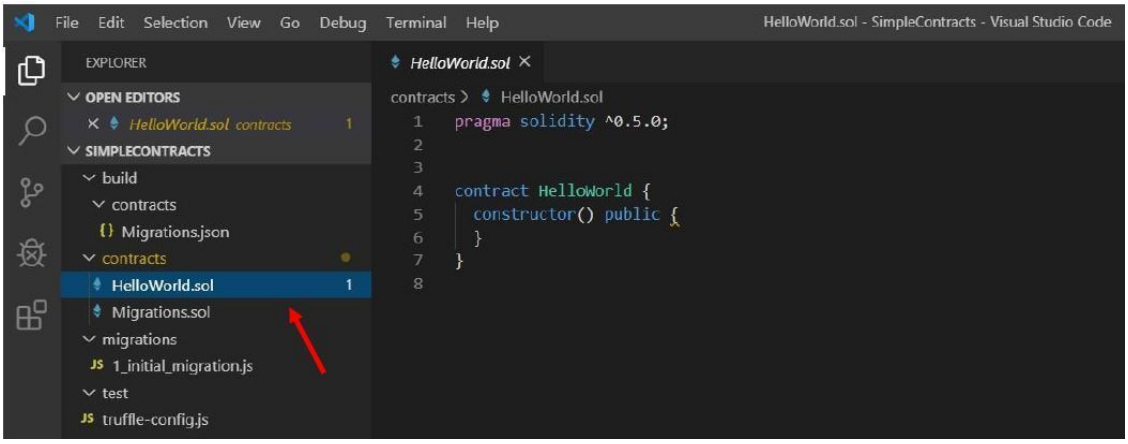

Рис. 2.3.2

Здесь мы видим шаблон нового пустого смарт-контракта. Рассмотрим данный шаблон подробнее.

> **Замечание.** Синтаксис языка Solidity очень похож на синтаксис таких языков программирования, как JavaScript и Java. Поэтому если вы знаете эти языки программирования, то вы без труда поймете код смарт-контрактов.

Из каких же частей состоит новый смарт-контракт? Первая строка «pragma solidity ^0.5.0;» задает версию языка программирования Solidity, которую мы используем. В нашем случае это версия 0.5.0. Так как язык программирования Solidity постоянно развивается, то со временем эта версия будет обновляться в сторону увеличения.

Далее мы видим строку «contract HelloWorld {» – это начало нашего смарт-контракта HelloWorld (рис. 2.3.3).

Замечание. Символ «{» показывает начало блока кода, а символ «}» в строке номер 7 показывает окончание блока кода, то есть начало и конец смарт-контракта HelloWorld. В языке Solidity блок кода группирует команды, и они рассматриваются как одна команда. В строках номер 4 и 6 данные символы обозначают начало и конец функции.

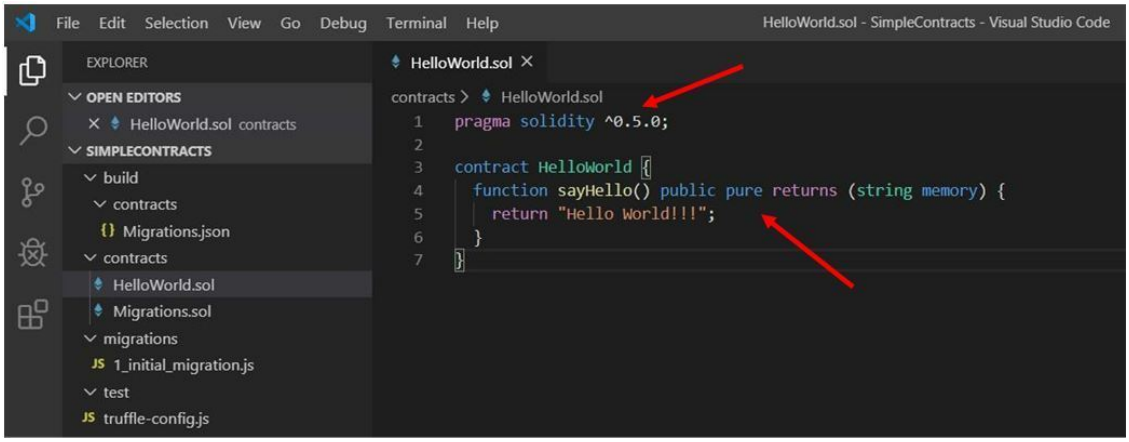

Рис. 2.3.3

Наш смарт-контракт содержит функцию sayHello. Она определяется командой «function sayHello() public pure returns (string memory)».

Команда function определяет функцию и имеет следующее строение:

• sayHello() – имя функции, наша функция не имеет входных параметров, поэтому область определения параметров пуста (). Позже мы рассмотрим функции с параметрами;

• public – это область видимости функции, означает, что мы можем использовать эту функцию в любом месте кода смарт-контракта;

• pure – определяет, что для выполнения нашего смарт-контракта не требуется эфир;

• returns (string memory) – показывает, что наша функция возвращает строку (HelloWorld – это string). Запись результата происходит в память, а не в блок, поэтому указываем параметр «memory».

Ну и наконец, рассмотрим содержимое нашей функции sayHello. Она состоит только из одной команды «return “Hello World!!!”;». Команда return выводит результат работы функции. В нашем случае это строка “Hello World!!!”. Строка заключается в кавычки. В языке Solidity любая команда заканчивается знаком «;» (рис. 2.3.3).

Замечание. Как мы можем видеть, язык программирования смарт-контрактов Solidity по синтаксису очень похож на языки программирования JavaScript и Java.

Теперь давайте откомпилируем наш смарт-контракт. Мы видим, что в коде нашего контракта отсутствуют подчеркнутые команды (рис. 2.3.3). Это говорит о том, что в коде отсутствуют ошибки и его можно компилировать. Для компиляции всего проекта в терминале выполните команду «truffle compile» (рис. 2.3.4).

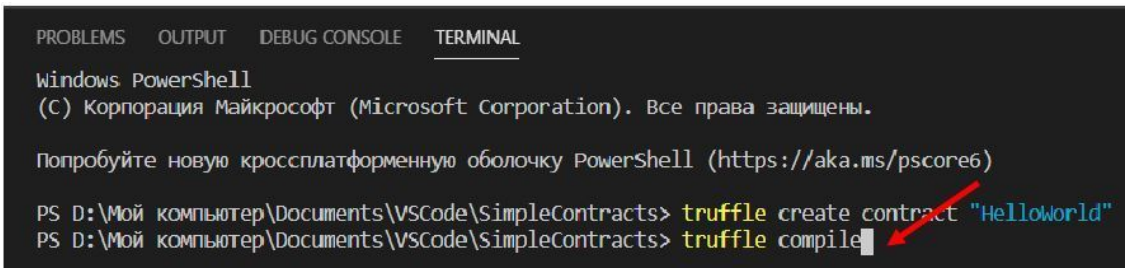

Рис. 2.3.4

После компиляции проекта мы видим, что на панели EXPLORER в папке build появился откомпилированный смарт-контракт HelloWorld.json (рис. 2.3.5).

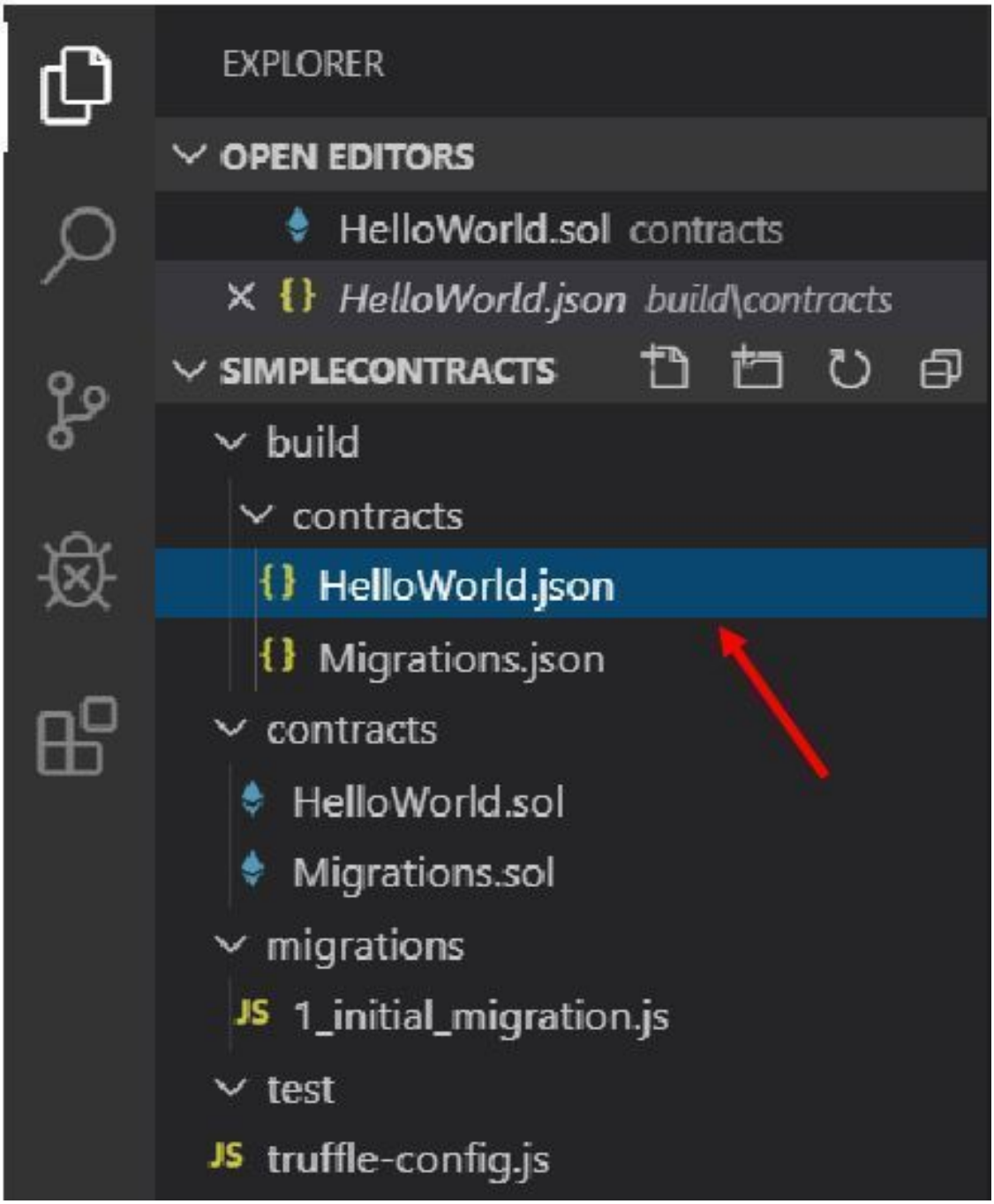

Рис. 2.3.5

На этом мы заканчиваем урок по содержимому нашего первого смарт-контракта и переходим к описанию его публикации в эмуляторе.

Урок 4. Публикация смарт-контракта HelloWorld в эмуляторе блокчейн-сети Ganache

Аннотация. В данном уроке мы рассмотрим публикацию простейшего смарт-контракта в эмуляторе блокчейн-сети Ganache.

В предыдущей неделе мы рассматривали публикацию смарт-контракта MetaCoin в эмуляторе блокчейн-сети Ganache. Однако смарт-контракт MetaCoin был тестовым контрактом, он был развернут по заданному в truffle шаблону и не требовал создания js-файлов для публикации (они уже были созданы).

В нашем новом проекте SimpleContracts смарт-контракта HelloWorld был сгенерирован по шаблону нового пустого контракта и при этом не создавались js-файлы для его публикации. По умолчанию в новом проекте присутствует js-файл 1_initial_migration.js, который публикует смарт-контракт Migrations.sol, развертывающий все остальные контракты проекта. Рассмотрим файл 1_initial_migration.js более подробно (рис. 2.4.1).

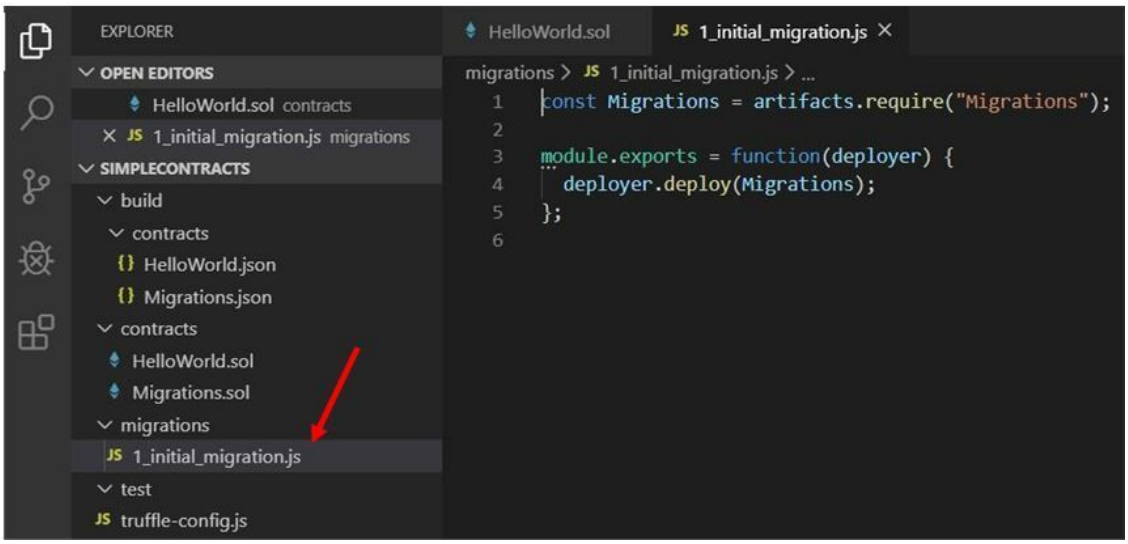

Рис. 2.4.1

Выбрав файл 1_initial_migration.js на панели EXPLORER, мы увидим его содержимое (рис. 2.4.1). В первой строке командой const Migrations = artifacts.require(“Migrations”); создается константа, привязанная к публикуемому смарт-контракту. Такие константы называются артефактами. В нашем случае мы создали артефакт Migrations (код голубого цвета), привязанный к смарт-контракту Migrations, указанный без расширения sol (код коричневого цвета). Далее создается функция, публикующая наш артефакт, команда module.exports = function(deployer) {. Как вы поняли из предыдущих уроков, функция закрывается символом }; в строке 5. Внутри функции расположена команда deployer.deploy(Migrations);. Данная команда публикует наш артефакт Migrations и привязанный к нему смарт-контракт Migrations.sol.

Создадим js-файл для публикации нашего смарт-контракта. Чтобы не набирать код js-файла, просто скопируем файл 1_initial_migration.js. Для этого щелкните по файлу 1_initial_migration.js правой кнопкой мыши и в появившемся меню выберите Copy (рис. 2.4.2).

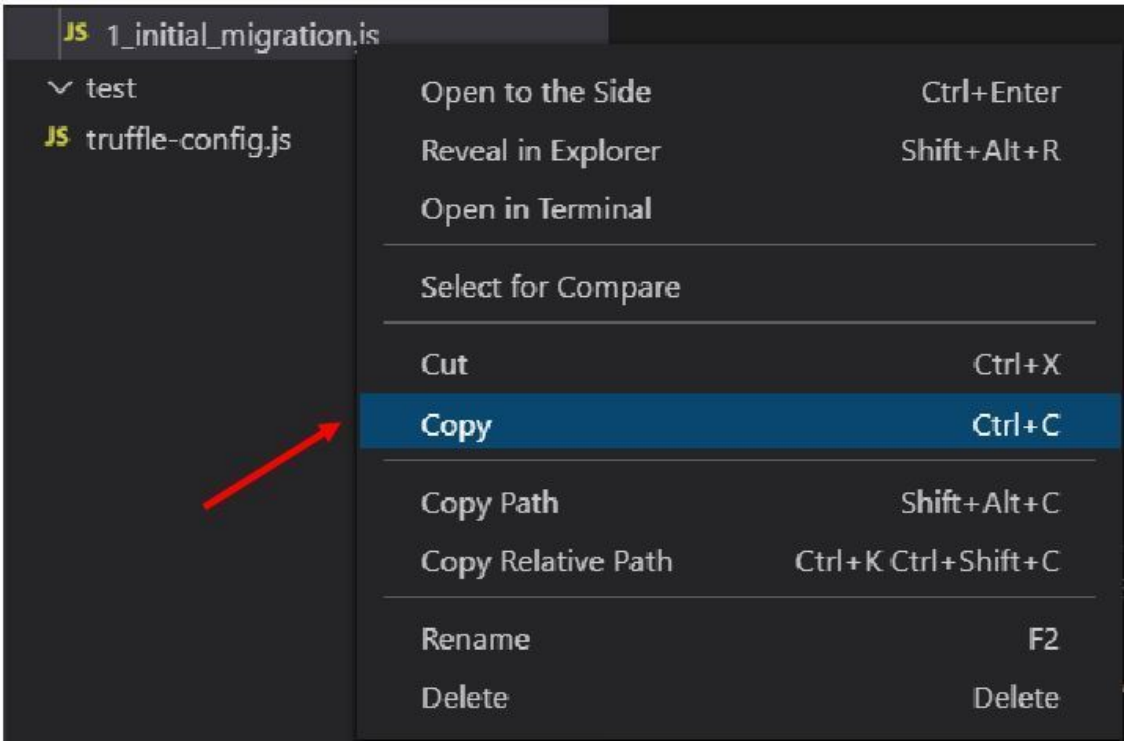

Рис. 2.4.2

Затем щелкните по папке migrations правой кнопкой мыши и в появившемся меню выберите Paste (рис. 2.4.3).

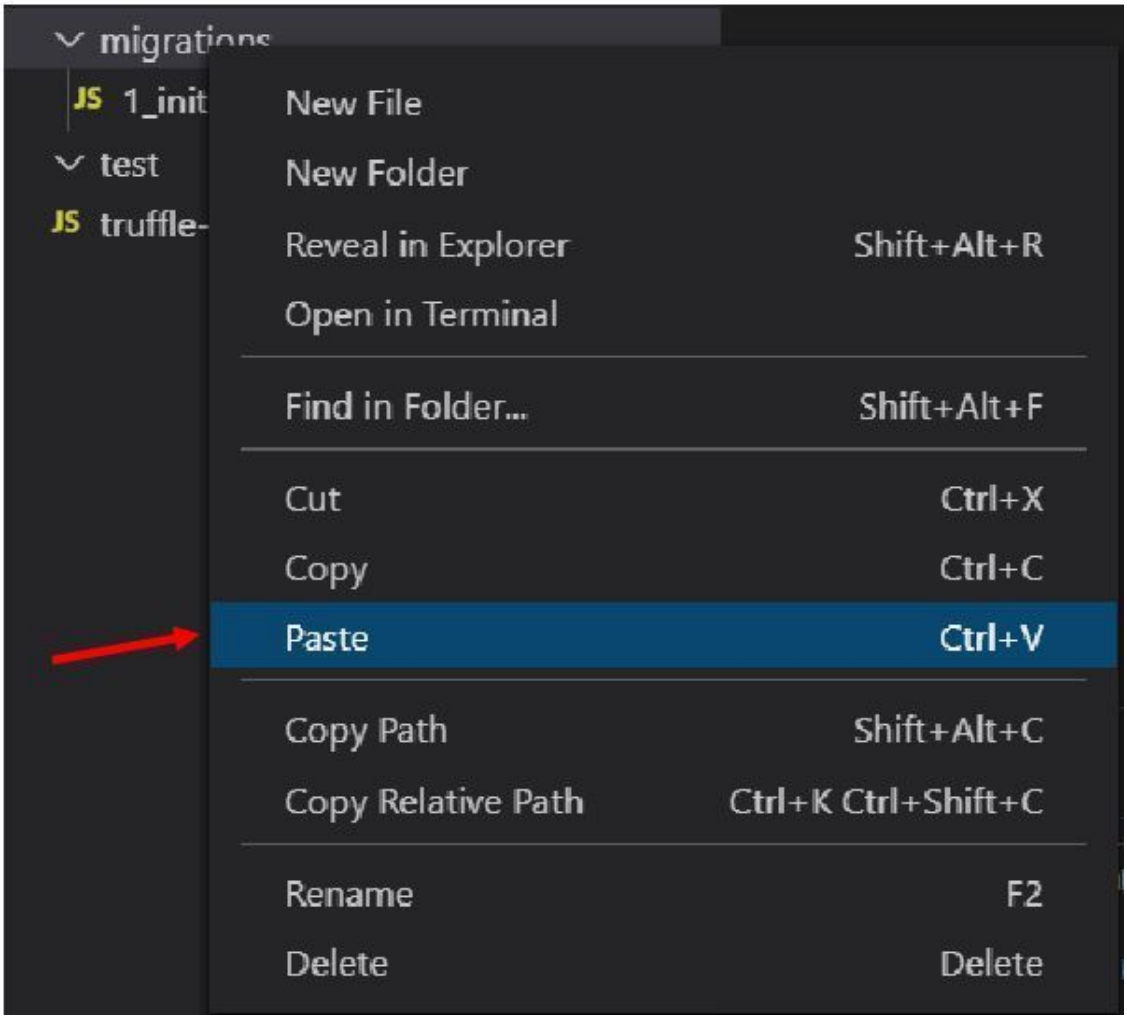

Рис. 2.4.3

В папке migrations появится файл 1_initial_migration copy.js (рис. 2.4.4).

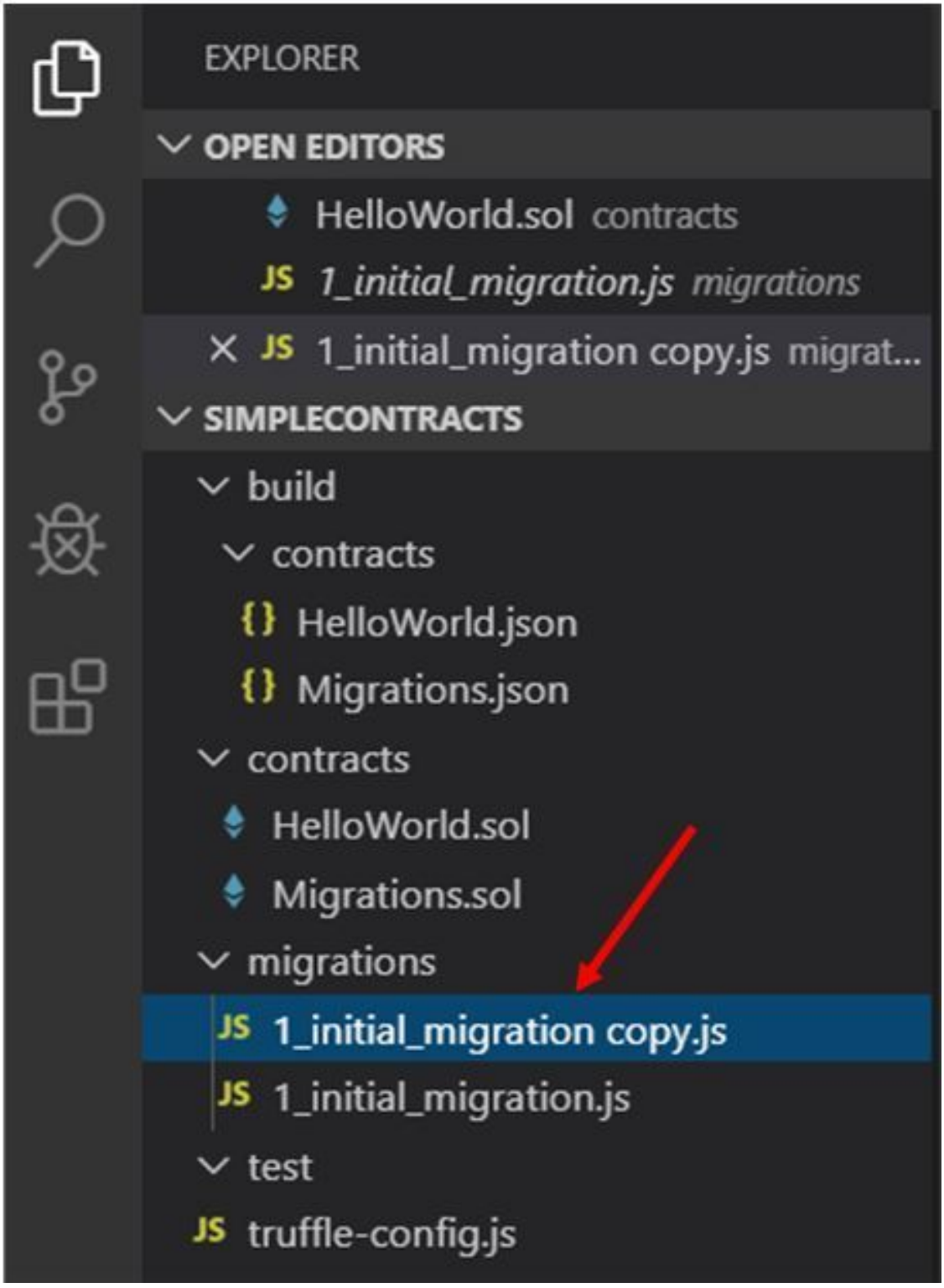

Рис. 2.4.4

Переименуем новый файл в js-файл с именем 2_HelloWorld_migration.js. Напоминаем, что js-файлы для публикации смарт-контрактов должны начинаться с цифры. Для переименования файла щелкните правой кнопкой мыши и в появившемся меню выберите Rename (рис. 2.4.5).

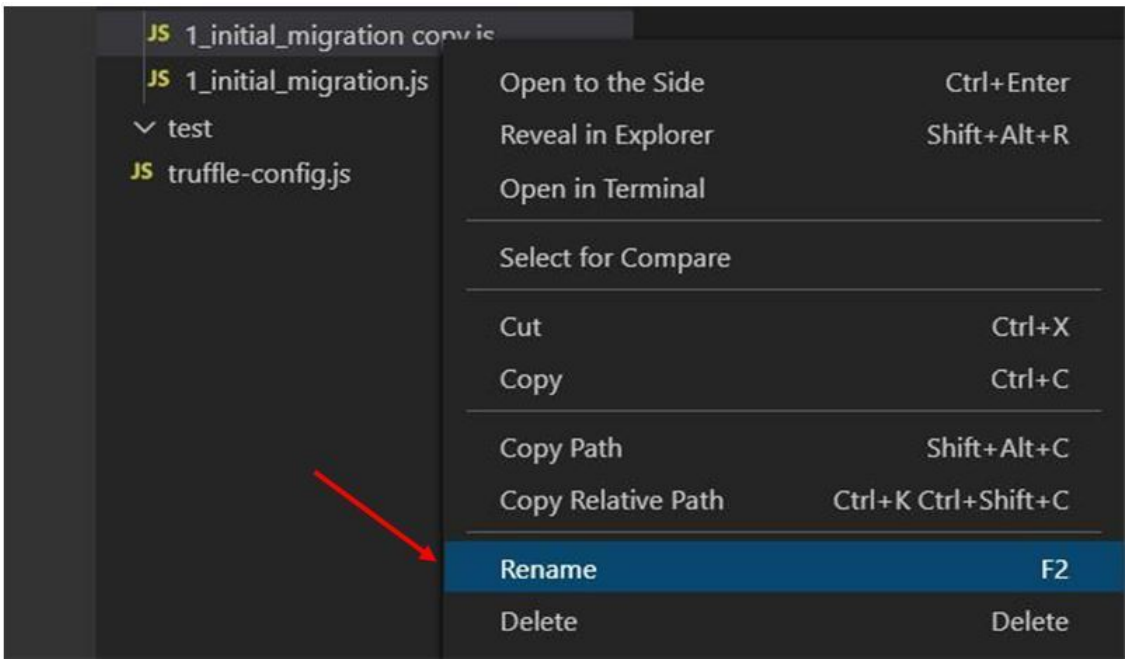

Рис. 2.4.5

Введите имя файла 2_HelloWorld_migration и нажмите клавишу Enter. Папка migrations примет вид как на рис. 2.4.6.

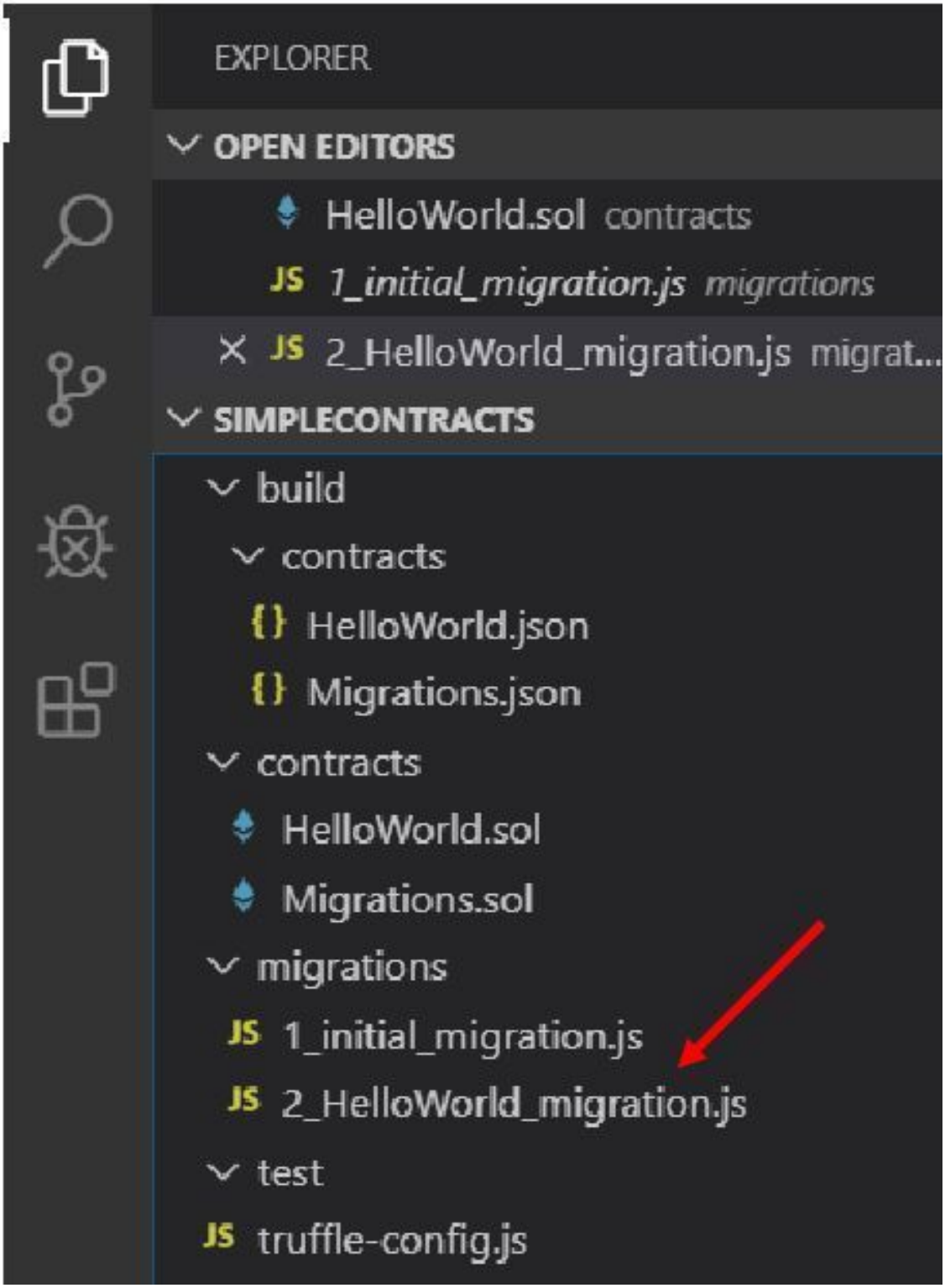

Рис. 2.4.6

Откройте js-файл, щелкнув по нему. Внутри файл будет аналогом скопированного js-файла 1_initial_migration.js. Замените все Migrations на HelloWorld (код голубого и коричневого цвета). Содержимое файла 2_HelloWorld_migration.js станет как на рис. 2.4.7.

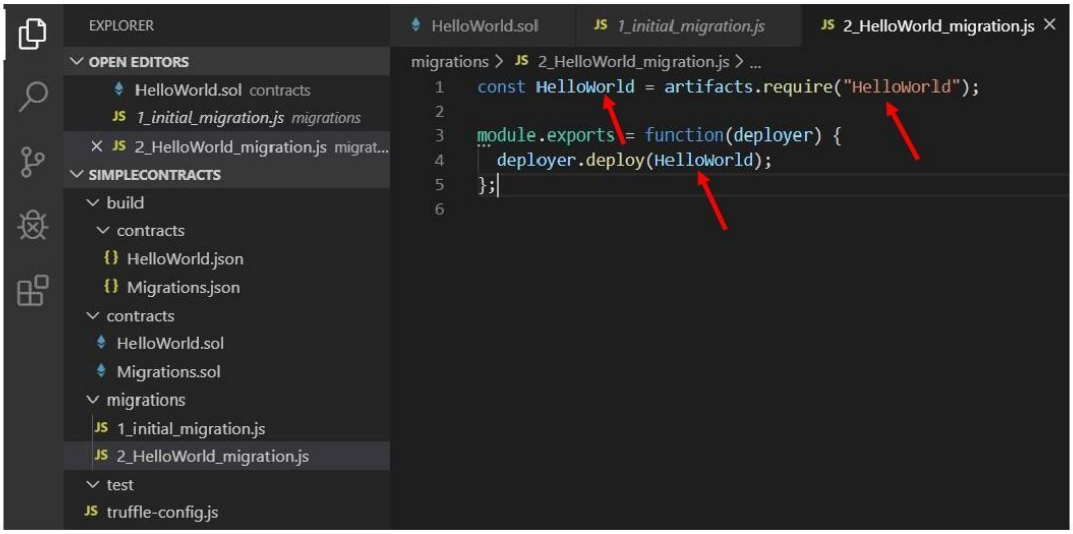

Рис. 2.4.7

Сохраните файл 2_HelloWorld_migration.js и запустите эмулятор блокчейн-сети Ganache. При запуске Ganache выберите вариант QUICKSTART.

Конец ознакомительного фрагмента.