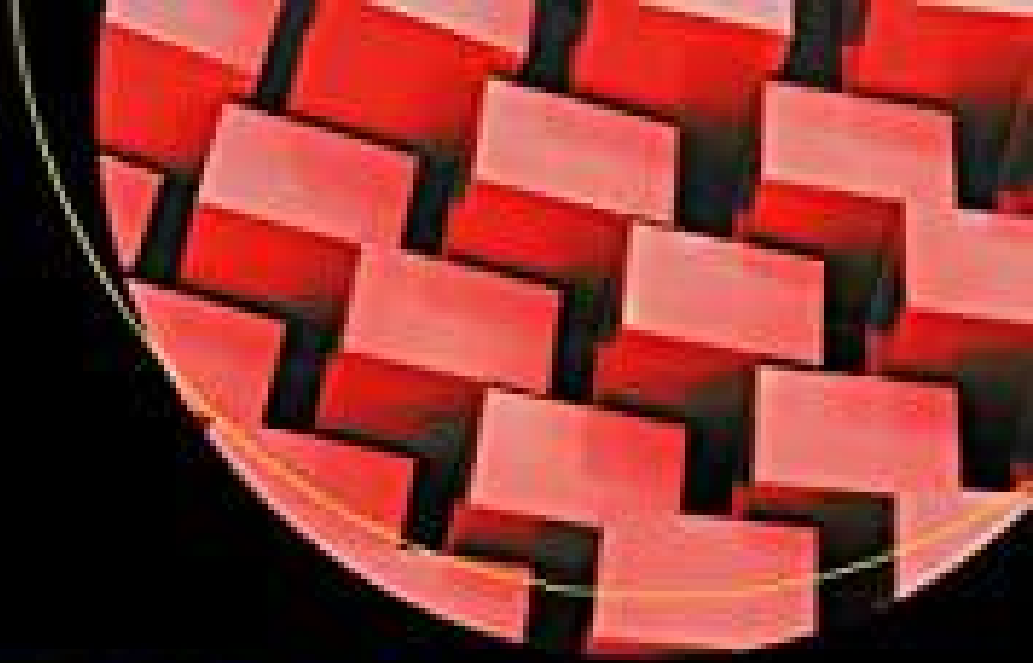

Начало работы с Ethereum

Пошаговое руководство, как стать разработчиком блокчейна

—

Дави Педро Бауэр

Apress®

Начало работы с Ethereum

Пошаговое руководство, как стать разработчиком блокчейна

Дави Педро Бауэр

Apress®

Начало работы с Ethereum: пошаговое руководство, как стать разработчиком блокчейна

Дави Педро Бауэр
Campo Bom, Rio Grande do Sul, Brazil

ISBN-13: 978-1-4842-8044-7 ISBN-13: 978-1-4842-8045-4
https://doi.org/10.1007/978-1-4842-8045-4

Управляющий директор, Apress Media LLC: Велмоэд Шпар
Редактор отдела приобретений: Спандана Чаттерджи
Координирующий редактор: Марк Пауэрс
Редактор текстов: Ким Уимпсетт

Cover designed by eStudioCalamar

Изображение на обложке Десмонда Маршалла на Unsplash (www.unsplash.com)

Распространяется в книжной торговле по всему миру компанией Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Телефон 1-800-SPRINGER, факс (201) 348-4505, эл. ком или посетите www.springeronline.com. Apress Media, LLC является калифорнийским ООО, а единственным участником (владельцем) является Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc — корпорация, зарегистрированная в Делавэре.

За информацией о переводах обращайтесь по электронной почте booktranslations@springernature.com; для перепечатки, мягкой обложки или прав на аудио, пожалуйста, по электронной почте bookpermissions@springernature.com.

Названия Apress можно приобрести оптом для академического, корпоративного или рекламного использования. Версии электронных книг и лицензии также доступны для большинства названий. Для получения дополнительной информации посетите нашу веб-страницу массовых продаж печатных и электронных книг по адресу www.apress.com/bulk-sales

Любой исходный код или другие дополнительные материалы, на которые ссылается автор в этой книге, доступны читателям на GitHub (https://github.com/Apress). Для получения более подробной информации посетите сайт www.apress.com/source-code

Напечатано на бескислотной бумаге

Содержание

Запуск блокчейна

Настройка сети блокчейн

Развертывание контракта

Добавление Ganache в сети MetaMask

Добавление токена в кошелек

Создайте токен ERC-20 с фиксированным предложением

Создание проекта

Написание контракта с фиксированным предложением

Компиляция контракта

Запуск блокчейна разработки Ganache

Миграция контракта в Ganache

Конфигурирование MetaMask

Добавление токена в MetaMask

Перенос токенов между аккаунтами

Разверните токен ERC-20 в тестовой сети с помощью Infura

Установка необходимых компонентов

Конфигурирование вашего проекта Infura

Конфигурирование смарт-контракта

Конфигурирование закрытого ключа

Развертывание смарт-контракта

Проверка баланса вашего кошелька

Проверка смарт-контракта на Etherscan

Разверните токен ERC-20 в тестовой сети Polygon (уровень 2)

Настройка узла

Тестирование узла

Изучение вашего узла IPFS

Добавьте файлы в IPFS

Добавление файла

Просмотр содержимого файла на консоли

Проверка файла в веб-интерфейсе

Просмотр содержимого файла в браузере

Настройка расширения браузера IPFS

Установка расширения браузера

Настройка типа узла

Запуск внешнего узла

Импорт файла

Закрепление и открепление файлов IPFS на локальном узле

Запуск вашего локального узла

Добавление файла в ваш узел

Проверка того, что ваш файл был добавлен

Проверка того, что ваш файл был закреплен

Открепление вашего файла

Закрепление вашего файла вручную

Закрепление и открепление файлов на удаленном узле с помощью Pinata

Настройка кнопок API в Pinata

Настройка Pinata в качестве удаленной службы на вашем терминале

Об авторе

Дэви Педро Бауэр имеет более чем 20-летний опыт работы в ИТ-секторе с опытом анализа и разработки систем. Он работает с методами agile с 2009 года, где он участвовал в программах внедрения agile в междисциплинарных командах, поддерживая внедрение процессов и практик, таких как Scrum и Kanban, а также запуск новых цифровых продуктов для веб- и мобильных платформ. С 2016 года он изучает темы, связанные с блокчейном, такие как криптовалюты, токенизация активов, смарт-контракты и децентрализованные приложения (DApps), а с 2019 года работает с DevSecOps, начиная с кода и заканчивая инфраструктурой.

О техническом рецензенте

Прасант Саху — идейный лидер, адъюнкт-профессор, технический спикер и штатный специалист в области блокчейна, DevOps, облачных вычислений и agile, работающий в PDI Software. Он был удостоен награды «Блокчейн и облачный эксперт 2019 года» от TCS Global Community за обмен знаниями в рамках академических услуг для сообщества. Он увлечен инициативами в области цифровых технологий посредством коучинга и наставничества. У Прасанта есть патент на его имя, и на сегодняшний день его аудитория состоит более чем из 50 000 специалистов, в основном в технической сфере. Он является членом рабочей группы в Совете по блокчейну, Консорциуме по сертификации криптовалют, Scrum Alliance, Scrum Organization и Международном институте бизнес-анализа.

Введение

Эта книга представляет собой пошаговое руководство для всех, кто хочет начать работу в качестве разработчика Ethereum. Она была написана для тех, кто никогда ничего не программировал в блокчейне, но хочет начать.

Я расскажу обо всем, от основных требований по установке до написания, тестирования и развертывания смарт-контрактов. Я также затрону такие темы, как IPFS, Filecoin, ENS, Chainlink, Truffle, Ganache, OpenZeppelin, Pinata, Fleek, Infura, MetaMask, OpenSea и другие.

В главе 1 я рассмотрю все необходимые требования для начала действий, описанных в этой книге. Она охватывает программное обеспечение и инструментарий, такой как Docker, Truffle, Ganache, MetaMask и Infura.

В главе 2 вы узнаете, как создать базовый проект Solidity с помощью расширения VS Code, а затем скомпилировать и развернуть смарт-контракт в локальной цепочке блоков.

В главе 3 вы узнаете, как кодировать смарт-контракты, чтобы создать собственную монету и развернуть ее в локальной цепочке блоков. Взаимозаменяемые токены взаимозаменяемы, поэтому они идеально подходят для решения таких проблем, как двойные списания. Вы также сможете добавить этот токен в свой кошелек и отправить его разным кошелькам, а также отправить другие монеты, которые у вас *уже есть*.

В главе 4 вы узнаете, как создать файл модульного теста для смарт-контракта, а также написать тестовые утверждения, *запустить* модульные тесты и *проверить результаты* модульного теста.

В главе 5 вы сможете создавать смарт-контракты для токенов бейджей. Вы можете использовать токены бейджей, также известные как NFT, для представления реальных вещей в виртуальном мире, таких как цифровые предметы коллекционирования, предметы из игр, цифровое искусство и т. д. Каждый токен NFT уникален и может иметь уникальное значение. В этой главе вы узнаете, как написать смарт-контракт с помощью библиотеки OpenZeppelin.

Вы также создадите токен и добавите его в узел IPFS. После этого вы научитесь закреплять его так, чтобы он был доступен для всех и везде. Далее вы узнаете, как перенести контракт в различные среды, такие как локальная цепочка блоков с использованием Ganache и тестовые сети с использованием Infura. Наконец, вы узнаете, как продавать собственные NFT на OpenSea.

В главе 6 мы рассмотрим различные способы пополнения вашего кошелька с помощью агрегаторов. Эта часть важна, потому что вам понадобится немного эфира в вашем кошельке, чтобы оплатить транзакцию. Большинство примеров будут развернуты в тестовых сетях, поэтому вам не потребуются реальные деньги для *их выполнения*.

В главе 7 вы узнаете, как создавать и сохранять файлы в децентрализованной файловой системе. Я также расскажу о некоторых инструментах, таких как расширение для браузера, которое поможет вам управлять узлом, а также Pinata, которая поможет вам закреплять файлы удаленно, а не хранить их локально. Кроме того, вы сможете разместить свой собственный сайт на IPFS с помощью Fleek.

В главе 8 я расскажу о способах сохранения файлов на локальном узле. Идея, лежащая в основе Filecoin, такая же, как у IPFS, с той лишь разницей, что Filecoin имеет механизм поощрения и узлы поощрения, чтобы сохранялись файлов. Filecoin был построен поверх IPFS.

В главе 9 вы узнаете, как зарегистрировать собственный домен в системе имен Ethereum. Вы можете использовать его для размещения сайта под этим доменным именем или даже в качестве домена для вашего кошелька для получения криптовалют, токенов или NFT.

В главе 10 я расскажу о случаях, когда вам нужно извлекать данные из офчейна с помощью оракулов. Вы узнаете, как использовать ценовые потоки, а затем крипто-цены внутри смарт-контрактов.

В главе 11 вы узнаете, как создать простой проект для подключения к Web3 с помощью платформы .NET и как извлекать данные из блокчейна для отображения баланса кошелька.

В главе 12 завершает книгу.

ГЛАВА 1

Начнем

В этой главе я объясню, что такое Ethereum, и проведу вас через процесс установки, который вам необходимо выполнить, прежде чем вы сможете начать его использовать.

Что такое Ethereum?

Ethereum решает проблемы, выходящие за рамки Биткойна. При разработке децентрализованных приложений нам нужна платформа, на которой мы можем кодировать не только монеты, но и самые разные решения.

Ethereum — это платформа, позволяющая создавать смарт-контракты на языке Solidity. Используя его, вы можете скомпилировать свой код в байт-код для интерпретации виртуальной машиной Ethereum (EVM).

Эта виртуальная машина будет интерпретировать инструкции, содержащиеся в байт-коде вашего смарт-контракта, и создаст новое состояние на основе правил, описанных в контракте. Как будто у вас в руках конечный автомат, где с каждым новым обновлением состояния в блокчейне обновляется новая запись.

Виртуальная машина, такая как EVM, потребляет ресурсы, поэтому вам нужен механизм, стимулирующий большее количество людей быть узлами сети, а также предотвращающий спам-атаки. Из-за этого для выполнения действий требуется элемент, называемый газом.

Чтобы обезопасить газ, вы должны для начала обладать *эфиром*, который является валютой сети Ethereum. Вы используете газ для оплаты вычислений, так что думайте об этом как о стоимости использования системы. Вам понадобится газ для выполнения большинства действий, описанных в этой книге.

Платформа Ethereum позволяет создавать децентрализованные приложения, то есть приложения, исходный код которых неизменен, а данные невозможно изменить после записи. Это открывает ряд новых решений, таких как голосование, цепочка поставок и децентрализованные финансы, среди прочего.

Лучший способ научиться — программировать, так что давайте начнем.

Установка кода Visual Studio

Прежде чем вы сможете начать использовать Ethereum, вам необходимо установить некоторое программное обеспечение. Во-первых, это Visual Studio Code (https://code.visualstudio.com), редактор кода с открытым исходным кодом, включающий функции отладки, выполнения задач и управления версиями. Он предоставляет разработчикам только те инструменты, которые им необходимы для быстрого цикла сборки-отладки кода, оставляя более сложные процессы полнофункциональным IDE, таким как Visual Studio IDE.

Вы можете скачать его бесплатно для разных платформ, таких как Windows, Linux и Mac. Все упражнения в книге опираются на использовании этого инструмента.

Установка Docker

Docker (https://docs.docker.com/get-docker/) — это бесплатная и открытая платформа для создания, доставки и выполнения

приложений. Docker позволяет отделить ваши приложения от вашей инфраструктуры, что позволяет быстрее развертывать программное обеспечение. Docker позволяет вам управлять своей инфраструктурой так же, как вы управляете своими приложениями. Используя методы Docker для быстрой доставки, тестирования и развертывания кода, вы можете существенно сократить время между разработкой кода и его выполнением в рабочей среде. Вам нужно будет запустить Docker перед компиляцией с использованием Truffle.

Установка расширения Blockchain Dev Kit на VS Code

Комплект разработчика блокчейна для Ethereum (https://marketplace.visualstudio.com/items?itemName=AzBlockchain.azure-blockchain) был создан как для новых пользователей Ethereum, так и для тех, кто уже знаком с этим процессом. Одна из основных целей — помочь пользователям в создании структуры проекта для этих смарт-контрактов; он также помогает пользователям компилировать и создавать активы, развертывать активы в конечных точках блокчейна и выполнять отладку контрактов (https://www.youtube.com/watch?v=WIUppKQhtKk).

Установка расширения

Перейдите в раздел «Extensions» и найдите *комплект для разработки блокчейна для Ethereum* (Blockchain Development Kit for Ethereum). Щелкните на расширении, созданном Microsoft; обычно оно идет первым (Рис. 1-1).

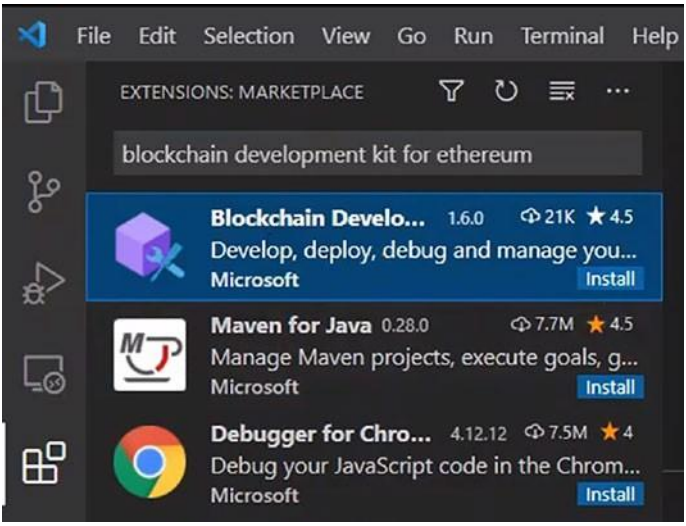

Рис. 1-1. *Установка Blockchain Development Kit для Ethereum*

Нажмите «Install» и дождитесь окончания процесса установки. Установка выполнена!

Установка Truffle

Truffle (https://www.trufflesuite.com/truffle) — это среда разработки Ethereum, среда тестирования и конвейер активов, цель которых — облегчить жизнь разработчиков Ethereum. Мы будем использовать этот инструмент на протяжении всей книги (https://www.youtube.com/watch?v=mE3_3wL5-jI).

Установка Truffle

Перейдите в окно терминала и установите пакет Truffle.

```
$ npm install -g truffle
```

Проверка установки Truffle

Теперь вы можете проверить, успешно ли завершилась установка. Если вы видите результат, показанный на Рис. 1-2, установка прошла успешно.

```
$ truffle
```

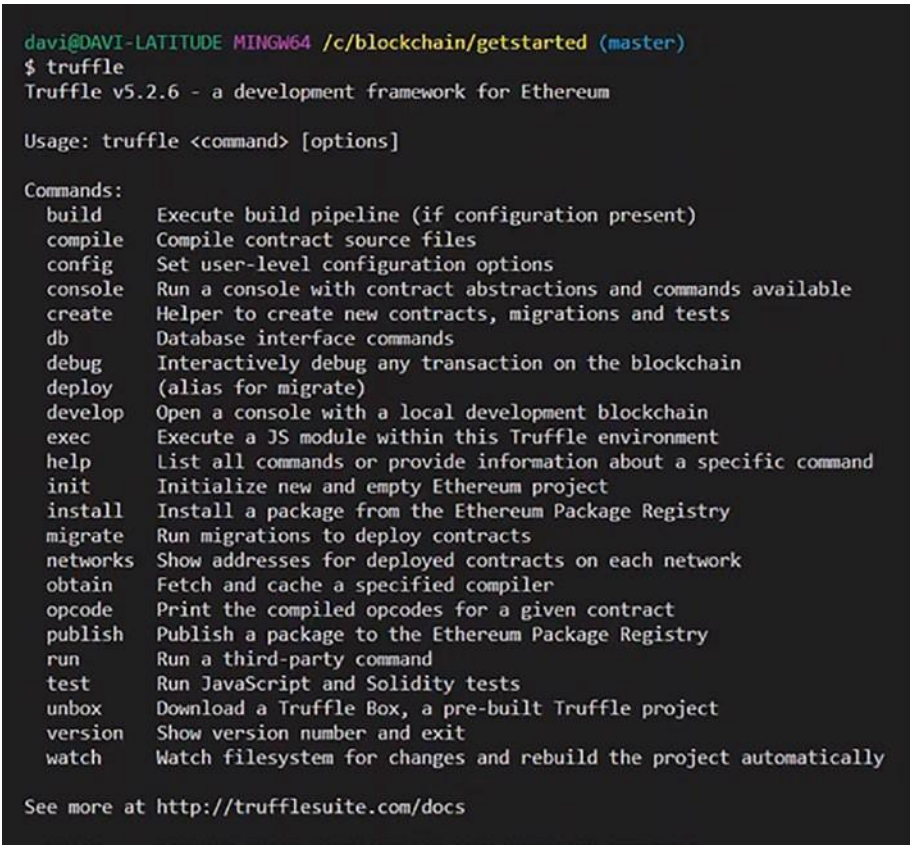

Рис. 1-2. *Результат вывода команды Truffle*

Установка интерфейса командной строки Ganache

Ganache (https://www.trufflesuite.com/ganache) — персональный блокчейн, позволяющий быстро разрабатывать распределенные приложения Ethereum и Corda. Ganache можно использовать на протяжении всего цикла разработки, что позволяет разрабатывать, развертывать и тестировать ваши DApp в безопасной и детерминированной среде (https://www.youtube.com/watch?v=fbQH6pzfXig).

Установка Ganache

Перейдите в окно терминала и установите командную строку Ganache.

```
npm install -g ganache-cli
```

Запускаем Ganache локально

Запустите интерфейс командной строки Ganache на 127.0.0.1:8545, используя следующую команду:

```
ganache-cli
```

Используя эту команду, в дополнение к локальному запуску Ganache, создается десять учетных записей с соответствующими открытыми и закрытыми ключами, чтобы вы могли использовать их в тестовых целях (Рис. 1-3).

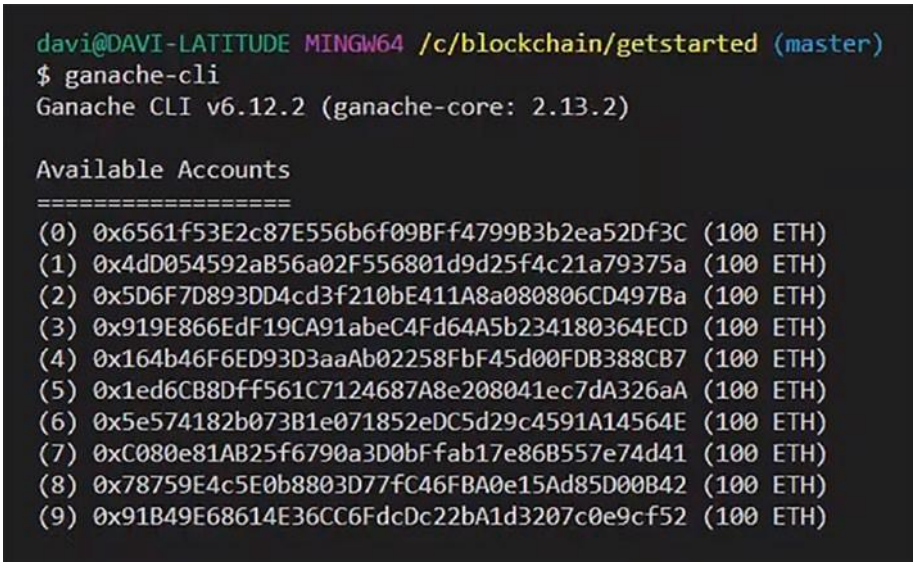

Рис. 1-3. *Аккаунты, созданные Ganache*

Установка и настройка кошелька MetaMask

MetaMask (https://metamask.io) — это расширение для браузера, которое позволяет вам получать доступ к распределенным приложениям с поддержкой Ethereum или DApps. Надстройка внедряет API Ethereum Web3 (https://web3js.readthedocs.io) в контекст JavaScript каждого веб-сайта, позволяя DApps читать из блокчейна (https://youtu.be/Bj6IozLyVxw).

Когда DApp хочет совершить транзакцию и опубликовать ее в блокчейне, MetaMask предоставляет пользователю безопасный интерфейс для оценки транзакции, прежде чем одобрить или отклонить ее с помощью закрытых ключей, локальных клиентских кошельков и аппаратных кошельков, таких как Trezor (https://trezor.io).

Установка кошелька

Перейдите на https://metamask.io и нажмите «Install MetaMask». Нажмите «Add to Brave» или имя вашего браузера, а затем нажмите «Add extension». Наконец, нажмите «Get started».

Конфигрирование кошелька

Нажмите «Create a wallet», а затем нажмите «No thanks» (если хотите, нажмите вместо этого «Я согласен»). Задайте пароль, который вы будете использовать для открытия своего кошелька, а затем подтвердите пароль. Согласитесь с условиями использования. Наконец, нажмите «Create». Теперь вы можете сделать резервную копию вашей секретной фразы (это можно сделать и позже). А пока нажмите «Напомнить мне позже». Ваш кошелек готов!

Доступ к вашему кошельку

Нажмите «Extensions» и закрепите MetaMask на вашей панели. Теперь щелкните на значке MetaMask, и вы увидите свой кошелек.

Обнаружение адреса вашего кошелька

Нажмите на три точки в правом верхнем углу, а затем нажмите «Account details». Обратите внимание, что вы можете видеть адрес своего кошелька в формате хэша и в формате QR-кода. Вы также можете скопировать адрес своего кошелька, щелкнув на имени учетной записи. Вот и все!

Создание учетной записи на Инфуре

Infura (https://infura.io) предоставляет инструментарий и инфраструктуру, которые позволяют разработчикам быстро перевести свое блокчейн-приложение с тестирования на масштабное развертывание, сохраняя при этом простой и надежный доступ к Ethereum и IPFS. Хорошо известным вариантом использования Infura в качестве интерфейса данных является Uniswap (https://uniswap.org, https://www.youtube.com/watch?v=NcKMBgNsBuw).

Создание новой учетной записи

Перейдите на Infura (https://infura.io) и нажмите «Get started for free». Введите адрес электронной почты и пароль, а затем нажмите «Sign up». На ваш адрес будет отправлено письмо с подтверждением (Рис. 1-4).

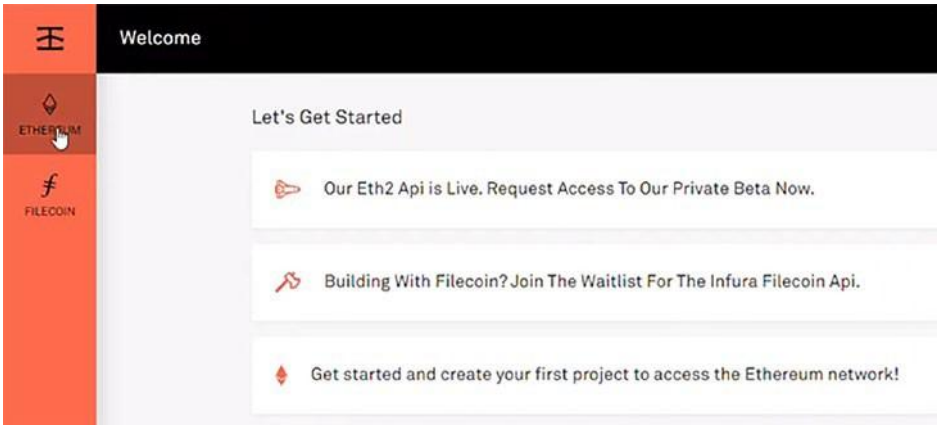

Рис. 1-4. *Страница приветствия Infura, которую вы увидите после входа в систему*

Проверьте свою электронную почту и подтвердите свою учетную запись, нажав на ссылку подтверждения. После этого вы будете

перенаправлены на вашу панель управления. Нажмите на вкладку Ethereum в левом меню.

Теперь, когда ваша учетная запись создана, вы можете приступить к настройке нового проекта(Рис. 1-5).

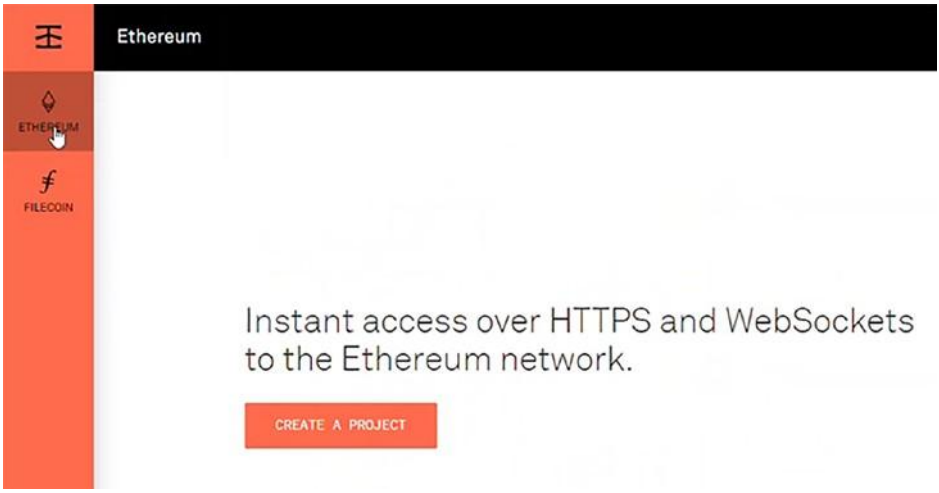

Рис. 1-5. *Нажав на Ethereum, вы попадете на домашнюю страницу проекта*

Конфигурирование вашего проекта Infura

Войдите в свою панель инструментов и нажмите Ethereum. Затем нажмите «Create a project» и укажите наименование проекта. Обратите внимание, что вы можете подключаться к разным тестовым сетям, а также к основной сети. Сохраните изменения.

После создания проекта предоставляется информация об идентификаторе, конфиденциальности и конечной точке подключения (Рис. 1-6).

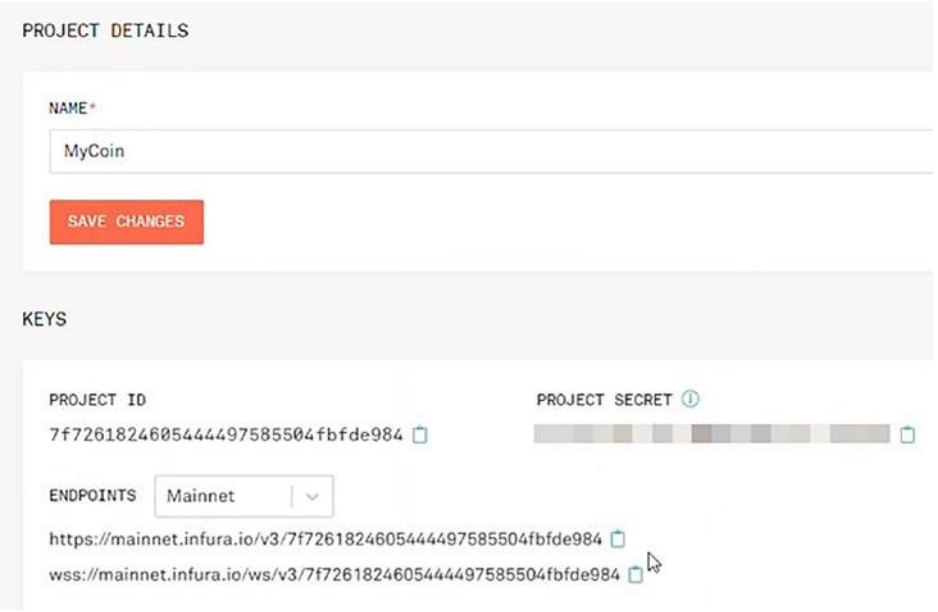

Рис. 1-6. *Страница сведений о проекте Infura*

Резюме

В этой главе вы узнали, что такое Ethereum и как установить необходимое программное обеспечение, чтобы начать разработку смарт-контрактов.

В следующей главе вы познакомитесь с Solidity и узнаете, как настроить свой первый проект на этом языке.

ГЛАВА 2

Solidity

Solidity — это объектно-ориентированный язык программирования высокого уровня, который используется для создания смарт-контрактов, позволяющих автоматизировать транзакции блокчейна. Язык был предложен в 2014 году Гэвином Вудом и разработан участниками проекта Ethereum. На Solidity повлияли C++, Python и JavaScript, поэтому вы найдете языковые структуры, характерные для этих языков. Данный язык в основном используется для создания смарт-контрактов на блокчейне Ethereum, но его также можно использовать для создания смарт-контрактов на других блокчейнах.

Solidity, будучи языком высокого уровня, избавляет нас от необходимости вводить код с нулями и единицами. Людям намного проще создавать программы в понятной им форме, комбинируя буквы и цифры.

Поскольку Solidity имеет статический тип, каждая переменная должна быть описана пользователем. Типы данных позволяют компилятору проверять использование переменных. Типы данных Solidity часто делятся на две категории: типы значений и ссылочные типы.

Экосистема Ethereum отличается тем, что может использоваться широким спектром криптовалют и децентрализованных приложений. В Ethereum смарт-контракты позволяют создавать решения для всех типов предприятий и организаций.

В конце этой главы вы сможете сделать следующее:

- Создать базовый проект Solidity, используя расширение VS Code
- Компилировать контракт
- Развернуть контракт на локальном блокчейне

Начало работы с проектом Solidity на VS Code

Ethereum — наиболее часто используемая платформа для смарт-контрактов. Ethereum — первый в мире программируемый блокчейн. Это позволяет создавать смарт-контракты, чтобы помочь в передаче цифровых активов, таких как эфир.

Solidity (https://soliditylang.org) — это язык, который вы будете использовать для заключения контрактов; он является полным по Тьюрингу, что означает, что он позволяет создавать сложные контракты четко определенным и закодированным образом.

Создание нового проекта

Выберите «View ➤ Command Palette», а затем нажмите «Blockchain: New SolidityProject» (Рис. 2-1). Наконец, нажмите «Create basic project» (Рис. 2-2).

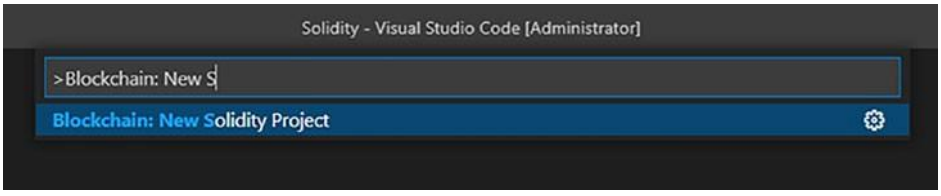

Рис. 2-1. *Новый проект Solidity*

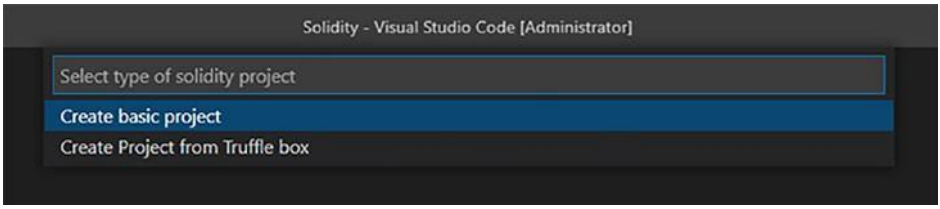

Рис. 2-2. *Создание базового проекта*

Выберите папку, в которой будет создан шаблон проекта, и дождитесь создания проекта. Убедитесь, что структура проекта создана, как показано на Рис. 2-3.

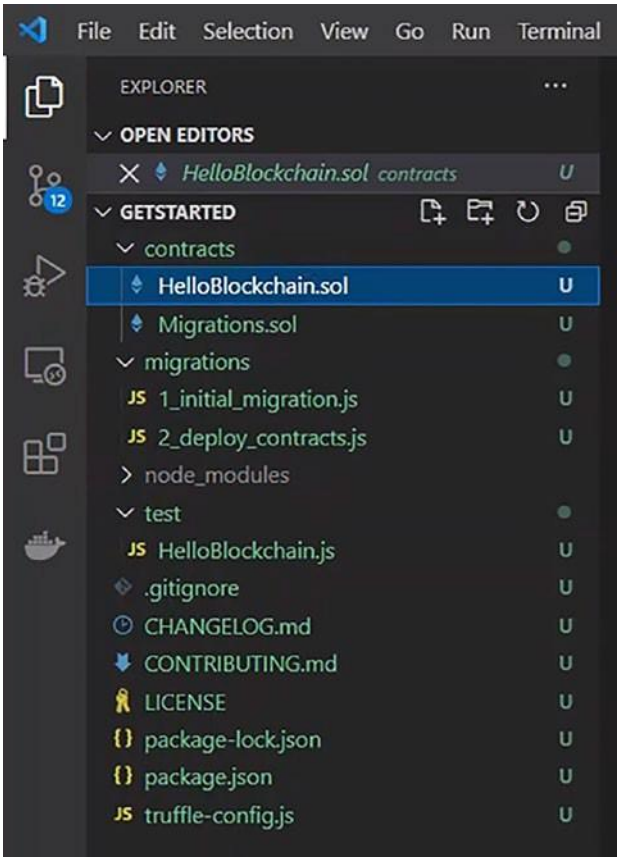

Рис. 2-3. *Структура проекта Solidity создана*

Компиляция проекта

Щелкните правой кнопкой мыши на файле HelloBlockchain.sol, выберите «Build contracts» и подождите, пока будут сформированы контракты.

Развертывание в блокчейне разработки

Щелкните правой кнопкой мыши на файле HelloBlockchain.sol, выберите «Deploy contracts», а затем выберите «Development 127.0.0.1:8545». Подождите, пока контракты будут развернуты в сети разработки блокчейна. Вот и все!

Резюме

В этой главе вы узнали, что такое Solidity, создали, скомпилировали и развернули свой первый смарт-контракт.

В следующей главе вы изучите стандарт токенов ERC-20 и узнаете, как их создавать и развертывать в средах разработки, тестирования и рабочей среде.

ГЛАВА 3

ERC-20: Взаимозаменяемые токены

Взаимозаменяемые токены — это токены, каждая единица которых имеет одинаковую стоимость, как и фиатная валюта. Это означает, что вы можете равноценно обменять одну единицу валюты на другую единицу этой валюты. Думая о том, чтобы воспроизвести это свойство в блокчейне, Фабиан Фогельстеллер и Виталик Бутерин предложили в ноябре 2015 года создать ERC-20, «Ethereum Request for Comments 20», чтобы создать простой формат для токенов на основе Ethereum. Эти токены работают в блокчейне Ethereum и могут взаимодействовать с другими криптовалютами в сети. В этой главе вы создадите простые контракты в стандарте ERC-20 и узнаете, как их развертывать в тестовых и рабочих сетях.

В конце этой главы вы сможете сделать следующее:

- Написать простой контракт в стандарте ERC-20.
- Написать фиксированный договор поставки.
- Наследовать ключевые реализации с OpenZeppelin.
- Скомпилировать контракт с помощью Truffle.
- Запустить локальный блокчейн с помощью Ganache.
- Развернуть существующий контракт в Ganache.
- Сконфигурировать MetaMask для подключения к Ganache.

- Добавить развернутый контракт токена в свой кошелек MetaMask.
- Перенести контракт в Ganache.
- Перенсти токены между аккаунтами.
- Добавить Polygon Mumbai в сети MetaMask.
- Активировать надстройку Polygon на Infura.
- Сконфигрировать закрытый ключ для подписания контракта.
- Развернуть смарт-контракт на Polygon Mumbai.
- Добавить основную сеть Polygon в сети MetaMask.
- Настроить сеть для использования основной сети Polygon.
- Развернуть смарт-контракт в основной сети Polygon.
- Проверить смарт-контракт в основной сети Polygon.

Написание простого токена ERC-20 с помощью OpenZeppelin

Давайте используем Truffle для разработки простого смарт-контракта ERC-20 Ethereum (https://ethereum.org/en/developers/docs/standards/tokens/erc-20/), а затем импортируем библиотеку контрактов OpenZeppelin.

OpenZeppelin — это открытая и поддающаяся аудиту библиотека, которая позволяет вам повторно использовать код из более распространенных реализаций, тем самым выступая в качестве исходной базы кода, которая всегда остается неизменной. Использование OpenZeppelin позволит вам больше сосредоточиться на кодировании бизнес-потребностей, а не на повторении ненужного кода.

В этом примере и в последующих главах данной книги мы будем использовать библиотеку OpenZeppelin .

Токены могут представлять в Ethereum практически все, например:

- Очки репутации на онлайн-платформе
- Навыки персонажа в игре
- Лотерейные билеты
- Финансовые активы, такие как доля в компании
- Фиатная валюта, такая как доллары США
- Унция золота

Подготовка среды

Инициализируйте Truffle, используя следующую команду:

```
$ truffle init
```

Теперь инициализируйте папку проекта.

```
$ npm init
```

Наконец, установите пакет контрактов OpenZeppelin.

```
$ npm install @openzeppelin/contracts
```

Написание контракта

Создайте новый файл в папке контрактов с именем ERC20MinerReward.sol. Добавьте директиву лицензии, укажите минимальную версию Solidity и импортируйте библиотеку контрактов OpenZeppelin ERC-20. Наконец, определите класс контракта, конструктор контракта, имя контракта и символ контракта.

```
// SPDX-License-Identifier:
MITpragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ERC20MinerReward is ERC20 {
  constructor() ERC20("MinerReward", "MRW"){}
}
```

Установка версии компилятора Solidity

Скопируйте версию Solidity, используемую в этом контракте, а затем откройте truffle-config.js. Раскомментируйте блок solc и установите версию Solidity, вставив скопированное значение.

```
compilers: {
    solc: {
        version: "0.8.0",
        docker: true,
        settings: {
            optimizer: {
                enabled: false,
                runs: 200
         },
         evmVersion: "byzantium"
       }
    }
 },
```

Компиляция контракта

Теперь пришло время скомпилировать контракт.

```
  $ truffle compile
```

Контракт скомпилирован успешно!

Проверка результата

Обратите внимание на то, что была создана новая папка build/contract (Рис. 3-1). Новый контракт находится здесь!

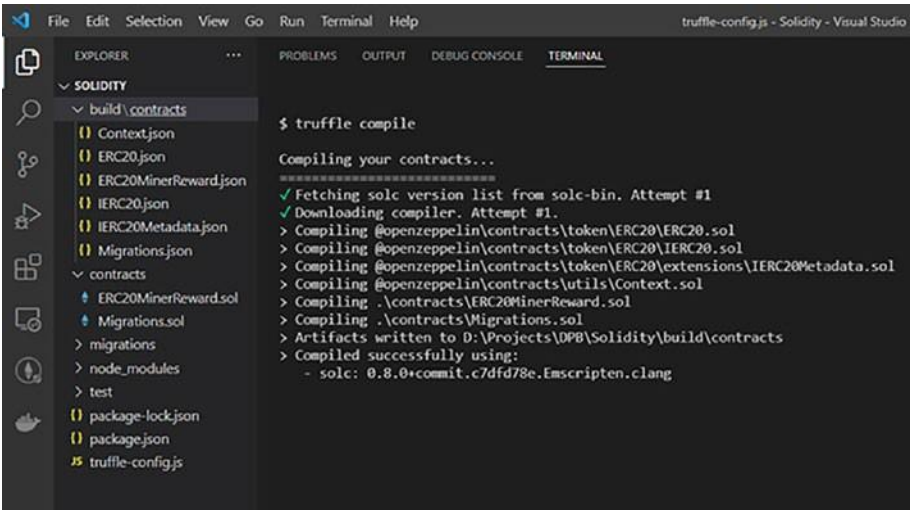

Рис. 3-1. *Результаты компиляции Truffle*

Разворачивание токен ERC-20 в блокчейне разработки Ganache

Ethereum Ganache — это локальный блокчейн в памяти, предназначенный для разработки и тестирования. Он имитирует характеристики реальной сети Ethereum, в том числе наличие ряда учетных записей, финансируемых с помощью тестового эфира.

Это хороший способ развертывания контрактов перед их перемещением в рабочую сеть. Используя блокчейн разработки, вы можете сосредоточиться на реализации, не беспокоясь о том, что вы тратите реальные деньги на развертывание контрактов.

Подготовка к миграции

Создайте в папке миграции новый файл миграции с наимнованием 2_deploy_contracts.js. В первой строке добавьте ссылку на смарт-контракт и добавьте функцию экспорта для развертывания смарт-контракта (Рис. 3-2).

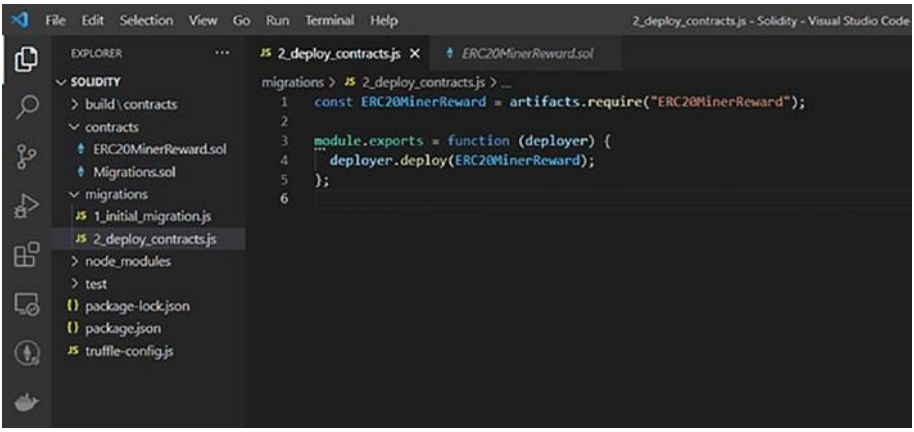

***Рис. 3-2.** Новый файл миграции*

Запуск блокчейна

Откройте новое окно терминала и запустите блокчейн Ganache.

```
$ ganache-cli
```

Новый блокчейн Ganache прослушивается на 127.0.0.1:8545.

Настройка сети блокчейн

Откройте файл truffle-config.js и раскомментируйте блок разработки в сетях. Убедитесь, что хост и порт указаны правильно (Рис. 3-3).

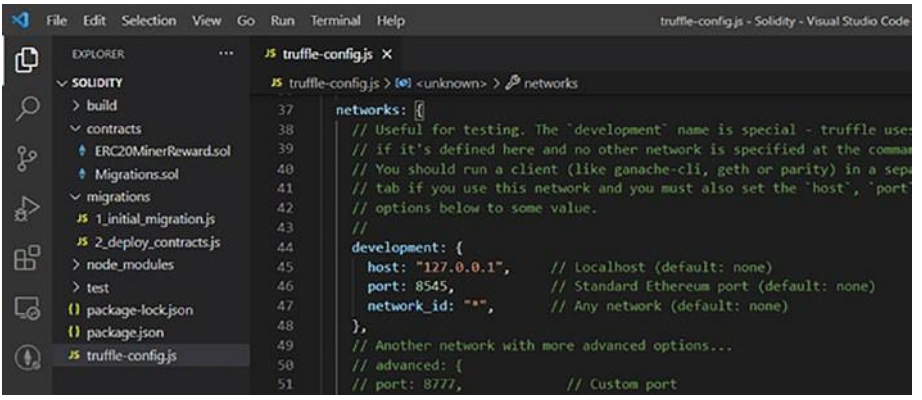

Рис. 3-3. *Сеть разработки*

Развертывание контракта

Скомпилируйте контракт с помощью следующей команды:

```
$ truffle compile
```

Перенесите контракт с помощью следующей команды:

```
$ truffle migrate
```

Контракт был развернут в блокчейне Ganache, и был создан адрес контракта (Рис. 3-4).

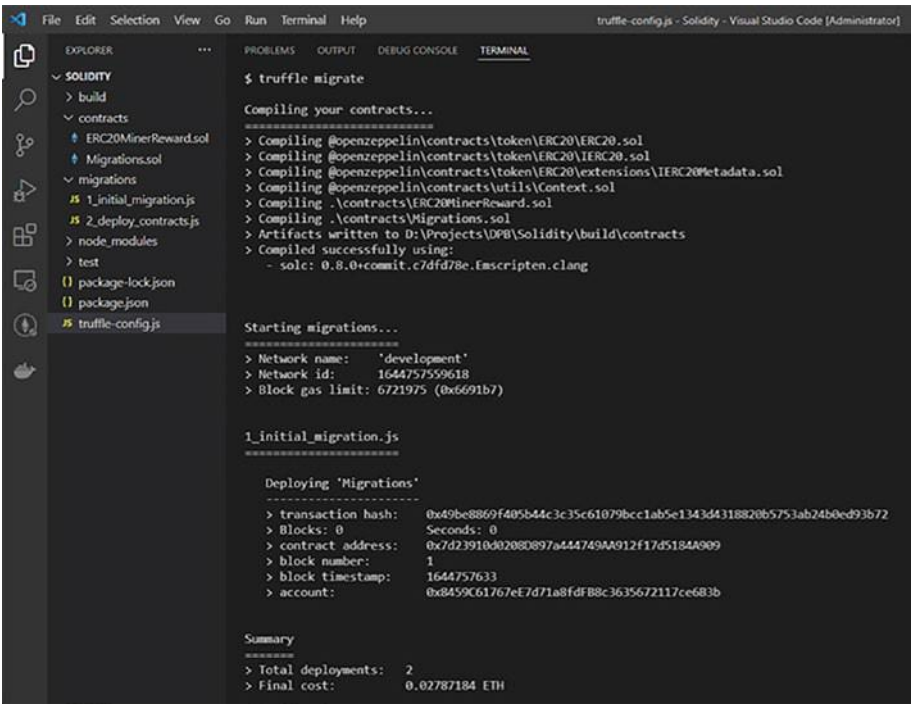

Рис. 3-4. *Контракт миграции в Truffle*

Добавление Ganache в сети MetaMask

Откройте расширение MetaMask и щелкните раскрывающийся список «Сеть». Выберите опцию Custom RPC и задайте следующие поля, как показано на Рис. 3-5:

- Введите наименование сети **Localhost 8545**.
- Укажите RPC URL как **http://localhost:8545**.
- Задайте идентификатор цепочки ID как **1337**.
- Укажите символ валюты как **ETH**.

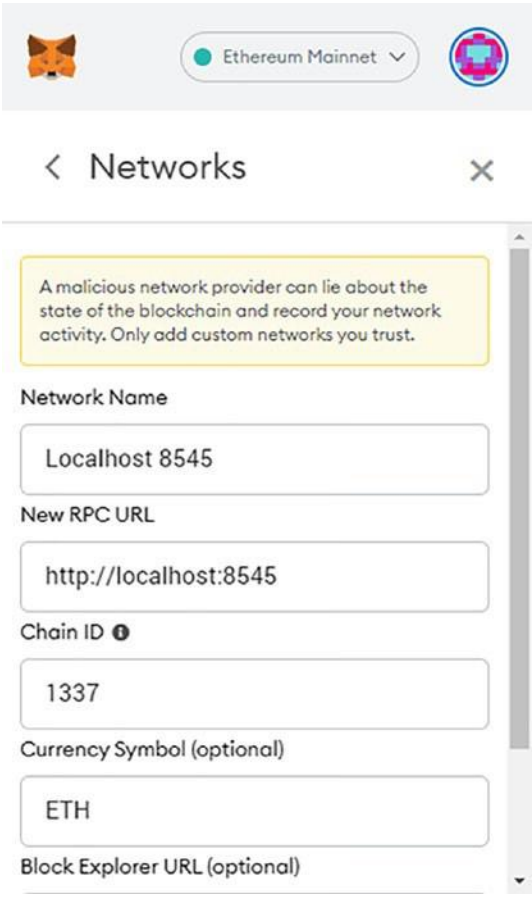

Рис. 3-5. *Конфигурация сети MetaMask*

Добавление токена в кошелек

Перейдите в браузер Brave (или любой браузер, совместимый с MetaMask) и выберите сеть «Localhost 8585».

Нажмите «Add token» и нажмите «Custom token». Скопируйте адрес контракта. Вставьте его в поле «Token contract address».

Поля «Token symbol» и «Decimals of precision» заполняются автоматически. Нажмите «Next» и нажмите «Add token». Токен был добавлен в кошелек MetaMask. Вот он, токен!

Создайте токен ERC-20 с фиксированным предложением

Общее количество токенов, разрешенных в смарт-контракте, определяется токенами с фиксированным предложением ERC-20. Вы не сможете обновить контракт после его развертывания в блокчейне. Это означает, что ваша монета будет иметь данную фиксированную стоимость после развертывания, и вы не сможете пополнить ее другими монетами позже.

Создание проекта

Инициализировать новый и пустой проект Ethereum.

```
$ truffle init
```

Создайте файл package.json для своего проекта.

```
$ npm init
```

Установите пакет контрактов OpenZeppelin. Он содержит многоразовые смарт-контракты, написанные на Solidity.

```
$ npm install @openzeppelin/contracts
```

Компилирование контракта с фиксированным предложением

Создайте новый файл Solidity и выполните следующие действия:

1. Включите лицензионное соглашение (это обязательно).
2. Определите минимальную версию Solidity.
3. Импортируйте библиотеку контрактов OpenZeppelin ERC-20.
4. Определите класс контракта с фиксированным предложением,

унаследованный от ERC-20.

5. Вызовите конструктор, передав наименование и символ.
6. Присвойте суммарное предложение адресу отправителя (кто создал контракт).
7. Переопределите функцию знаков посл десятичной запятой.
8. Установите количество знаков после десятичной запятой, которое будет иметь этот токен.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ERC20FixedSupply is ERC20 {
    constructor() ERC20("Fixed", "FIX"){
        _mint(msg.sender, 1000);
    }

    function decimals() public view virtual override returns
    (uint8){
        return 0;
    }
}
```

Перейдите к truffle-config.js и раскомментируйте блок solc (Ctrl+;). Теперь обновите номер версии Solidity.

```
compilers: {
    solc: {
        version: "0.8.0",
        docker: true,
        settings: {
            optimizer: {
                enabled: false,
```

```
                runs: 200
            },

        evmVersion: "byzantium"
      }
    }
  },
```

В папке миграции создайте новый файл. Присвойте ему наименование 2_deploy_contracts.sol.

```
$ touch migrations/2_deploy_contracts.sol
```

В этом файле установите требуемый метод для вашего файла контракта и экспортируйте функцию для развертывания контракта.

```
var ERC20FixedSupply = artifacts.require("./ERC20FixedSupply.sol");

module.exports = function(deployer){
    deployer.deploy(ERC20FixedSupply);
}
```

Компиляция контракта

Теперь пришло время компилировать контракт.

```
$ truffle compile
```

Договор был успешно компилирован!

Запуск блокчейна разработки Ganache

Разделите окно терминала. Теперь запустите блокчейн разработки Ganache на 127.0.0.1:8545.

```
$ ganache-cli
```

Перейдите к truffle-config.js и в разделе сетей раскомментируйте блок разработки.

```
networks: {
    development: {
        host: "127.0.0.1",
        port: 8545,
        network_id: "*"
    },
}
```

Миграция контракта в Ganache

Выполните команду migrate, чтобы развернуть контракты, как показано на Рис. 3-6.

```
$ truffle migrate
```

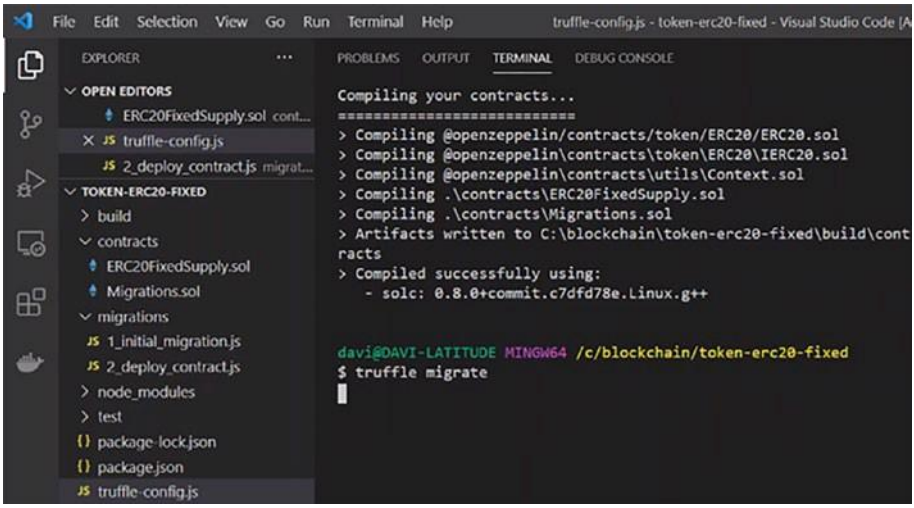

Рис. 3-6. *VS Code: перенос проекта с помощью командной строки truffle*

Прежде чем перейти к следующему разделу, скопируйте закрытый

ключ учетной записи, которая развернула токен.

Конфигурирование MetaMask

Откройте MetaMask. Выберите свою учетную запись, а затем нажмите «Import account». На этом шаге вставьте закрытый ключ учетной записи. Нажмите «Import», как показано на Рис. 3-7.

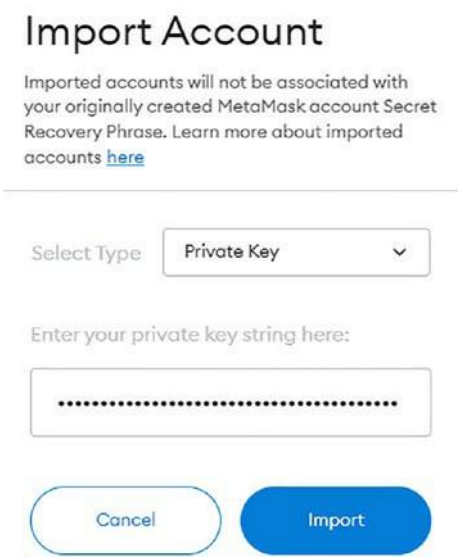

Рис. 3-7. *MetaMask: импорт существующего кошелька с использованием начальной фазы*

Щелкните на Networks list и выберите Localhost:8545. Использование локальной сети означает, что вы будете ориентировать свой кошелек на блокчейн локальный разработки, как это показано на Рис. 3-8.

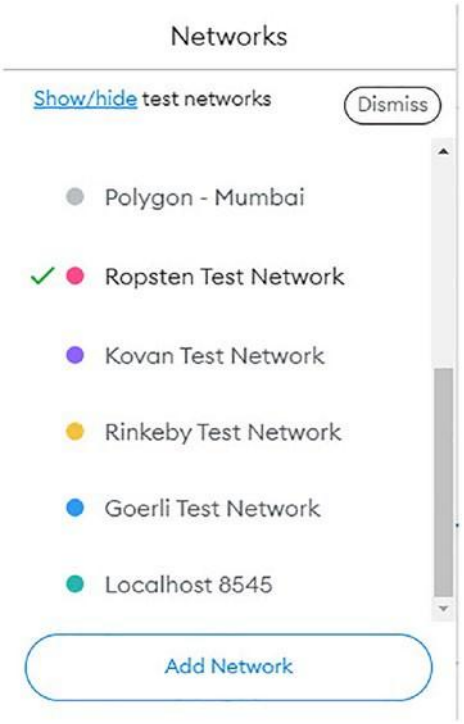

Рис. 3-8. *MetaMask: список выбора сети*

Добавление токена в MetaMask

Нажмите «Add token», а затем выберите «Custom token». Вставьте адрес контракта токена и нажмите «Next».

Добавление токена — это вопрос добавления публичного адреса контракта созданного токена. После этого MetaMask автоматически прочитает символ и количество знаков после десятичной запятой. Убедитесь, что вы получили тот же результат, что и на Рис. 3-9.

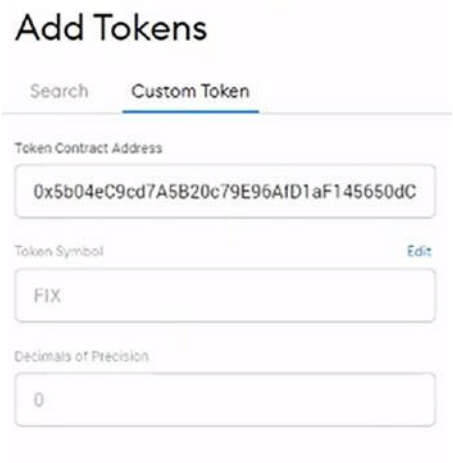

Рис. 3-9. *MetaMask: добавление пользовательского токена*

Нажмите «Add tokens». На экране будет отображаться символ токена, а также ваш баланс (Рис. 3-10).

Рис. 3-10. *MetaMask: обзор нового пользовательского токена*

Теперь вернитесь в VS Code (см. Рис. 3-6, где представлен вид терминала ganache-cli) и скопируйте закрытый ключ другой учетной записи. Вернитесь в MetaMask и повторите шаги, которые вы сделали для первой учетной записи, включая добавление токена.

Перенос токенов между аккаунтами

Теперь переключитесь на первую импортированную учетную запись (ту, в которой есть все токены). Нажмите «Send», а затем нажмите «Transfer between my accounts». Выберите вторую созданную учетную запись. Введите **115 FIX** в качестве суммы для перевода и нажмите «Next». Наконец, нажмите «Confirm».

Транзакция отправлена, но находится в состоянии ожидания. Подождите немного, пока транзакция не будет подтверждена. Как только это произойдет, общее количество токенов будет обновлено. Выберите вторую импортированную учетную запись; теперь на этом аккаунте 115 FIX!

Разверните токен ERC-20 в тестовой сети с помощью Infura

Infura можно использовать для развертывания смарт-контрактов в тестовых сетях, таких как Ropsten, Kovan, Rinkeby, Goerli, а также в основной рабочей сети. Для тестовой сети вам потребуется создать новый проект в Infura и получить доступ к закрытому ключу кошелька, который вы будете использовать для развертывания контрактов. Для выполнения транзакции создания контракта на этом кошельке также должен быть баланс эфира.

Установка необходимых компонентов

Откройте новое окно терминала и установите пакет fs. Установка этого пакета обеспечивает полезный функционал для доступа и взаимодействия с файловой системой.

```
$ npm install fs
```

Теперь установите пакет поставщика кошелька hdwallet. Он

используется для подписи транзакций для адресов, полученных из мнемоники 12 или 24 слов.

```
$ npm install @truffle/hdwallet-provider@1.2.3
```

Конфигурирование вашего проекта Infura

Перейдите на http://infura.io и получите доступ к панели инструментов. Нажмите «Ethereum», а затем нажмите «Create a Project». Введите наименование проекта. Обратите внимание, что вы можете подключаться к разным тестовым сетям, а также к основной сети. Скопируйте идентификатор проекта и сохраните изменения.

Конфигурирование смарт-контракта

Перейдите в Visual Studio Code и откройте truffle-config.js. Раскомментируйте четыре константы: hdwalletprovider, infurakey, fs и mnemonic. Вставьте идентификатор проекта в качестве значения константы Infurakey. Раскомментируйте блок ropsten. Убедитесь в том, что вы используете правильный идентификатор проекта в конечной точке ropsten.

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const infuraKey = "fj4jll3k. ";

const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();
```

Конфигурирование закрытого ключа

Перейдите в браузер и откройте свой кошелек MetaMask, подключенный к сети Infura. Нажмите «your account», затем нажмите «settings» и, наконец, нажмите «security & privacy» (Рис. 3-11).

У вас есть возможность просмотреть свою сид-фразу, но имейте в виду, что эта информация является конфиденциальной, и если кто-то

получит к ней доступ, он сможет восстановить ваш кошелек и использовать ваши средства.

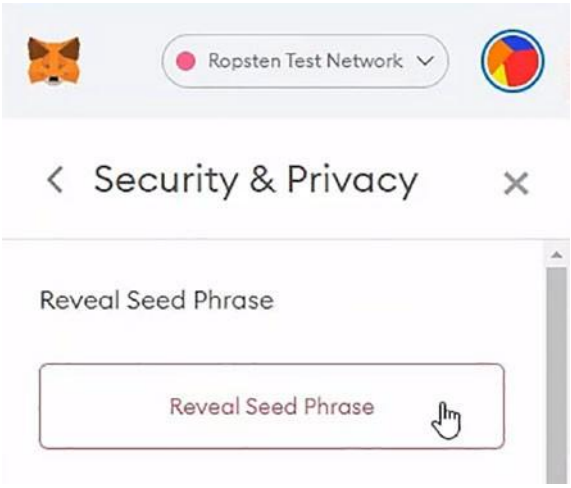

Рис. 3-11. *MetaMask: раскрываем сид-фразу*

Нажмите «Reveal Seed Phrase» и введите пароль своего кошелька, чтобы продолжить; затем скопируйте закрытый ключ.

Вернитесь в Visual Studio Code (Рис. 3-6) и создайте новый файл с наименованием `.secret`. Вставьте закрытый ключ в этот файл. Вы также можете создать данный файл с помощью командной строки (Рис. 3-12).

```
$ touch .secret
```

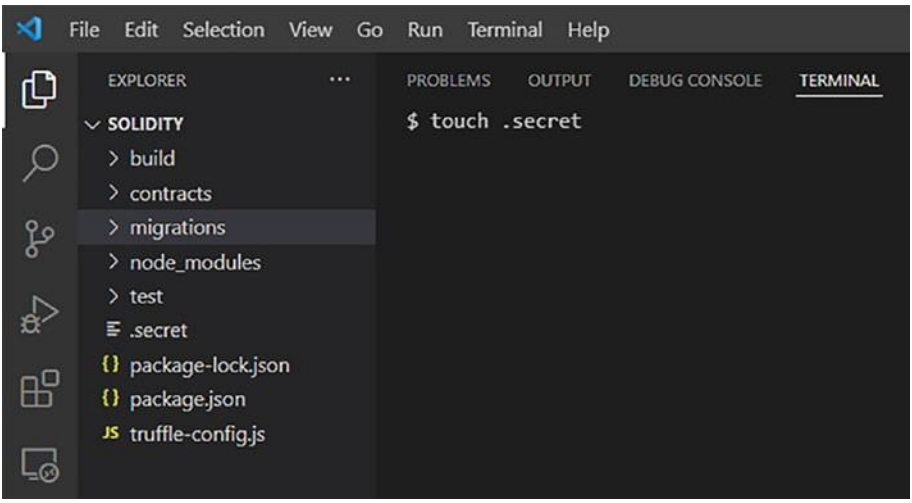

Рис. 3-12. *Секретный файл*

Развертывание смарт-контракта

Откройте окно терминала и запустите команду migrate, чтобы развернуть контракты в сети Ropsten.

```
$ truffle migrate --network ropsten
```

Проверка баланса вашего кошелька

Снова зайдите в свой кошелек MetaMask и обратите внимание на то, что ваш баланс уменьшился.

Проверка смарт-контракта на Etherscan

Откройте новое окно и скопируйте адрес контракта, созданный на этапе развертывания. Перейдите на https://ropsten.etherscan.io и вставьте адрес контракта в поле поиска. Нажмите кнопку «Find». Вот он, смарт-контракт!

Токены были созданы и переведены в кошелек, создавший контракт. Щелкните ссылку на токен Fixed (FIX). Здесь вы можете увидеть обзор вашего вновь созданного токена.

Разверните токен ERC-20 в тестовой сети Polygon (уровень 2)

Polygon — это протокол и фреймворк для построения и подключения Ethereum-совместимых блокчейн-сетей. Вы можете агрегировать масштабируемые решения на Ethereum для поддержки многоцепочечной экосистемы Ethereum.

MATIC, собственный токен Polygon, представляет собой токен ERC-20, работающий на блокчейне Ethereum. Токены используются для платежных услуг на Polygon и в качестве расчетной валюты между пользователями, работающими в экосистеме Polygon.

Установка необходимых компонентов

Откройте новое окно терминала и установите пакет fs, если вы еще не сделали этого. Этот пакет предоставляет множество полезных функций для доступа и взаимодействия с файловой системой.

```
$ npm install fs
```

Теперь установите пакет hdwallet поставщика кошелька, если вы этого еще не сделали. Он используется для подписи транзакций для адресов, полученных из мнемоники 12 или 24 слов.

```
$ npm install @truffle/hdwallet-provider@1.4.0
```

Добавление Polygon Mumbai в сети MetaMask

Откройте расширение MetaMask и щелкните раскрывающийся список «Network». Затем выберите параметр Custom RPC. Введите следующие значения, как показано на Рис. 3-13:

- Ввдите наимнование сети **Matic Testnet**.
- Введит URL-адрес RPC как **https://rpc-mumbai.matic-vigil.com**.
- Введите идентификатор цепочки Chain ID как **80001**.
- Ввдите символ валюты **MATIC**.
- Укажите URL-адрес обозревателя блоков как **https://explore-mumbai.maticvigil.com**.

Рис. 3-13. *MetaMask: страница конфигурации сети*

Активация надстройки Polygon на Infura

Перейдите на https://infura. io/upgrade и нажмите «Select Addon» в Polygon PoS в разделе «Network Add-ons», как показано на Рис. 3-14. Polygon PoS в настоящее время находится на стадии бета-версии в Infura, и вам необходимо ее активировать.

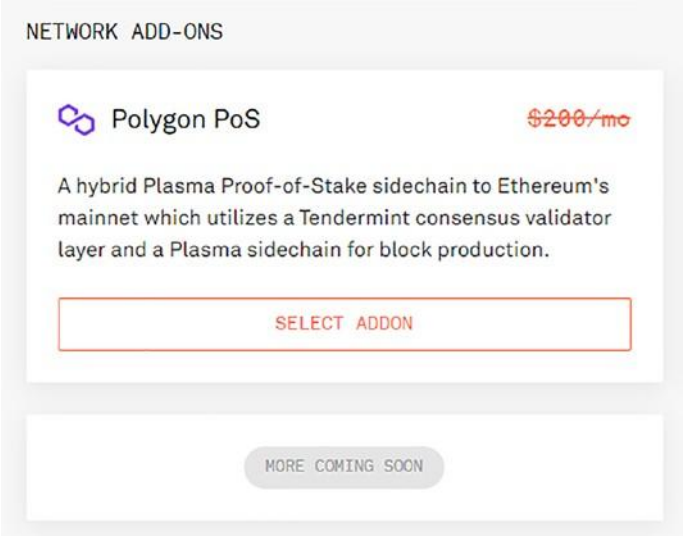

Рис. 3-14. *Infura: страница активации Polygon PoS*

После активации вы будете перенаправлены на страницу сводки. Бесплатный уровнь ограничен 100 000 запросов в день. Вам будет предложено предоставить кредитную карту для подтверждения; так как общая стоимость равна нулю, с вас не будет взиматься плата. Если вы согласны, нажмите «Get Started Now». Вы должны увидеть страницу, похожую на показанную на Рис. 3-15.

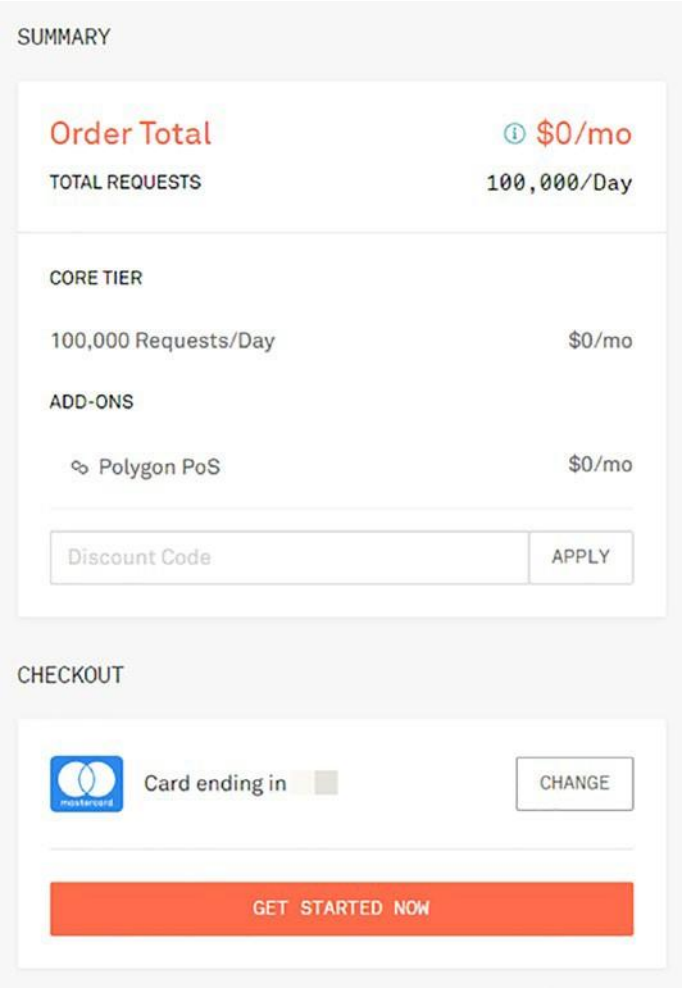

Рис. 3-15. *Infura: итоговый порядок страниц после добавления плагина Polygon PoS*

Конфигрирование вашего проекта Infura

Убедитесь в том, что у вас уже есть проект в Infura. Если вы еще этого не сделали, выполните действия, описанные в Главе 1.

Конфигрирование смарт-контракта

Перейдите в Visual Studio Code и откройте `truffle-config.js`. Раскомментируйте четыре константы: hdwalletprovider, infurakey, fs и mnemonic и вставьте идентификатор проекта в качестве значения константы infurakey.

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const infuraKey = "fj4jll3k.... ";

const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();
```

Конфигрирование сети (с использованием конечной точки Matic)

Первый способ подключения к сети Polygon — использование сети Matic. Теперь создайте конфигурацию `matic_testnet` в `networks` в файле `truffle-config.js` и введите следующие значения:

- URL-адрес кошелька `https://rpc-mumbai.matic-vigil.com`.
- Set `network_id` to 80001.

```
matic_testnet: {
  provider: () => new HDWalletProvider(mnemonic, `https://rpc-
  mumbai.maticvigil.com`),
  network_id: 80001,
  confirmations: 2,
  timeoutBlocks: 200,
  skipDryRun: true
},
```

Настройка сети (с использованием конечной точки Infura)

Другой способ подключения к сети Polygon — использование конечной точки Infura. Создайте конфигурацию matic_testnet в networks и установите следующие значения:

- Укажите URL-адрес кошелька https://polygon-mumbai.infura.io/v3/${infuraKey}.
- Укажите network_id как 80001.

```
matic_testnet: {
  provider: () => new HDWalletProvider(mnemonic,
  `https://polygon-mumbai.infura.io/v3/${infuraKey}`),
  network_id: 80001,
  confirmations: 2,
  timeoutBlocks:
  200, skipDryRun:
  true, chainId:
  80001,
  networkCheckTimeout: 1000000
},
```

Чтобы использовать сеть Polygon, вам необходимо активировать надстройку сети.

Конфигурирование закрытого ключа

Перейдите в браузер и откройте свой кошелек MetaMask, подключенный к сети Infura. Нажмите "*your account*", а затем нажмите "settings." Наконец, нажмите "security & privacy", как показано на Рис. 3-16.

У вас есть возможность просмотреть свою сид-фразу, но имейте в

виду, что эта информация является конфиденциальной, и если кто-то получит к ней доступ, он сможет восстановить ваш кошелек и использовать ваши средства.

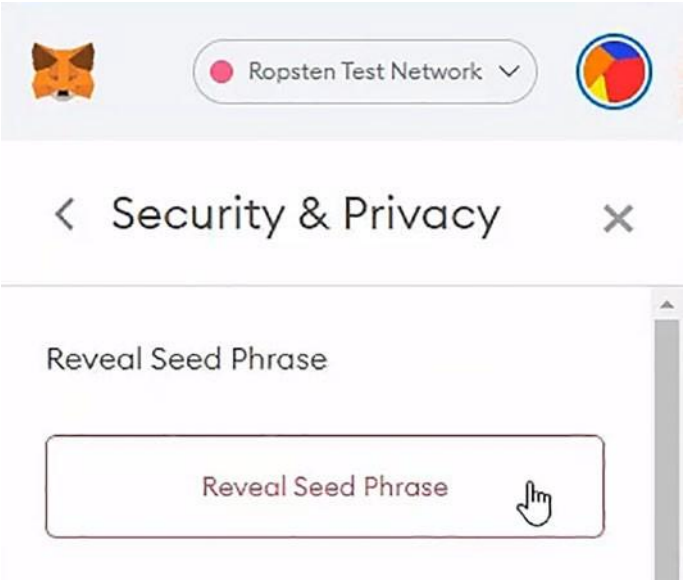

Рис. 3-16. *MetaMask: раскрываем сид-фразу*

Нажмите Reveal Seed Phrase и введите пароль кошелька, чтобы продолжить. Скопируйте закрытый ключ.

Вернитесь в VS Code (в окне терминала `ganache-cli`)и создайте новый файл с наименованием `.secret`. Вставьте секретную фразу восстановления в этот файл.

Разворачивание смарт-контракта

Запустите команду `migrate`, чтобы развернуть контракты в сети `matic_testnet`.

```
$ truffle migrate --network matic_testnet
```

Если терминал возвращает вам ошибку, то вам нужно будет сначала получить тестовый MATIC от Faucet.

```
1_initial_migration.js
======================

   Deploying 'Migrations'
---------------------

Error:   *** Deployment Failed ***

"Migrations" -- insufficient funds for gas * price + value.
```

Проверка баланса кошелька

Снова зайдите в свой кошелек MetaMask и обратите внимание на то, что ваш баланс уменьшился. Это происходит потому, что вам нужно платить за каждое развертывание контракта. Он имеет эквивалентную стоимость в газе, и эта стоимость рассчитывается в соответствии с инструкциями, которые вы используете в смарт-контракте. Это означает, что чем больше машинной обработки вам нужно, тем выше стоимость газа для вас, чтобы выполнить данный контракт. Вы можете найти более подробное объяснение того, как это рассчитывается, в желтой книге Ethereum.

Проверка смарт-контракта на PolygonScan

Скопируйте адрес контракта, который был создан при развертывании (этот адрес будет отображаться в консоли после завершения миграции `truffle`) и перейдите на `https://mumbai.polygonscan.com`. Вставьте адрес контракта в поле поиска и нажмите кнопку «Find». Смарт-контракт найден!

Токены были созданы и переведены в кошелек, создавший контракт. Теперь нажмите на ссылку Fixed (FIX) token, и здесь вы увидите обзор только что созданного токена!

Развертывание токена ERC-20 в рабочей сети Polygon (уровень 2)

Рабочая сеть используется для реальных транзакций, а тестовые сети используются для тестирования смарт-контрактов и децентрализованных приложений (DApps). Polygon используется в качестве второго уровня, он приобрел популярность из-за стоимости транзакций, которая ниже, чем в основной сети.

Добавление основной сети Polygon в сети MetaMask

Откройте расширение MetaMask, щелкните раскрывающийся список «Network» и выберите параметр «Custom RPC». Установите следующие значения, как показано на Рис. 3-17:

- Ввдите наимнование сети как **Matic Mainnet**.
- Укажите RPC URL как **https://rpc-mainnet.matic-vigil.com**.
- Введите идентификатор цепочки Chain ID как **137**.
- Введите символ валюты **MATIC**.
- Введите URL-адрес проводника блоков как **https://explore-mainnet.maticvigil.com**.

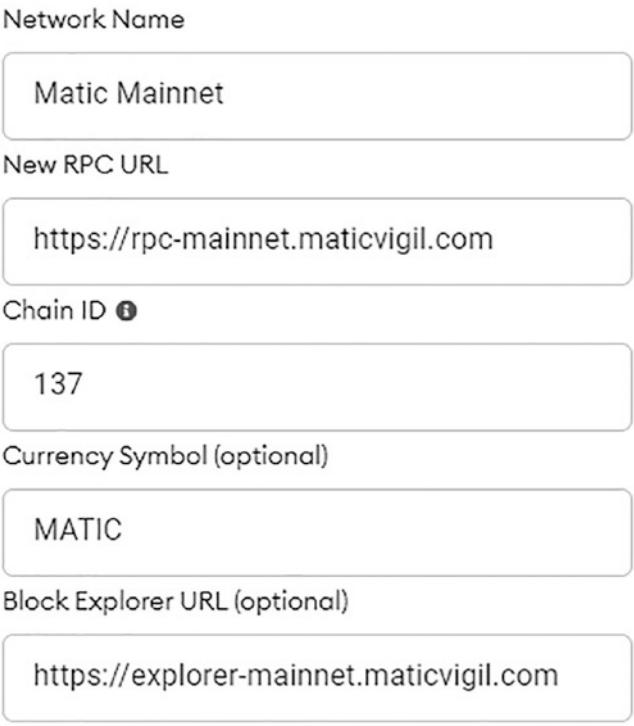

Рис. 3-17. *MetaMask: страница конфигурации сети*

Конфигурирование сети (с использованием конечной точки Infura)

Другой способ подключения к сети Polygon — использование конечной точки Infura. Создайте конфигурацию `matic_mainnet` в сетях и установите следующие значения:

- Установите URL-адрес кошелька как `https://polygon-mainnet.infura.io/v3/${infuraKey}`.
- Set `network_id` установите в 137.

```
matic_mainnet: {
  provider: () => new HDWalletProvider(mnemonic, `https://
  polygon-mainnet.infura.io/v3/${infuraKey}`),
  network_id: 137,
  gasPrice: 100000000,
  confirmations: 2,
  timeoutBlocks: 200,
  skipDryRun: true,
  chainId: 137,
  networkCheckTimeout: 1000000
},
```

Развертывание смарт-контракта

Запустите команду `migrate`, чтобы развернуть контракты в сети `matic_mainnet`.

```
$ truffle migrate --network matic_mainnet
```

Проверка баланса кошелька

Снова зайдите в свой кошелек MetaMask и обратите внимание, что ваш баланс уменьшился.

Проверка смарт-контракта на PolygonScan

Скопируйте адрес контракта, созданный при развертывании, и перейдите на сайт PolygonScan (https://polygonscan.com). Вставьте адрес контракта в поле поиска и нажмите кнопку «Find». Смарт-контракт подтвержден!

Токены были созданы и переведены в кошелек, создавший контракт. Теперь нажмите на ссылку Fixed (FIX) token, и здесь вы увидите обзор только что созданного токена.

Резюме

этой главе вы узнали, что такое стандарт токенов ERC-20, и узнали, как создавать и развертывать взаимозаменяемые токены для Ganache в тестовой и основной сетях на блокчейнах Ethereum и Polygon.

В следующей главе вы изучите модульные тесты смарт-контрактов и узнаете, как написать свой первый модульный тест.

ГЛАВА 4

Модульные тесты для смарт-контрактов

Модульные тесты используются для тестирования сценариев выполнения кода и обеспечения их ожидаемого поведения. Важно отметить, что модульные тесты позволяют повторно раскладывать код на составляющие и вносить новые изменения, не нарушая его существующее функционирование.

В смарт-контрактах модульные тесты еще более важны, так как после развертывания контракта его уже невозможно исправить, если вы только не развернете новый контракт. Из-за этого крайне важно, чтобы вы включали модульные тесты во все смарт-контракты, которые вы пишете, ища максимально возможное покрытие.

В этой главе вы узнаете, как писать модульные тесты, используя Mocha в качестве средства запуска тестов и Chai в качестве библиотеки утверждений.

В конце этой главы вы сможете сделать следующее:

- Создайть файл модульного теста
- Написать модульные тесты для смарт-контракта
- Написать тестовые утверждения
- Запустить модульные тесты
- Проверьте результаты модульного теста

Написание модульных тестов для смарт-контрактов ERC-20

Truffle по умолчанию имеет автоматизированную среду тестирования, что значительно упрощает тестирование ваших контрактов. Данная структура позволяет создавать базовые и управляемые тесты различными способами. Давайте начнем кодировать наш первый модульный тест.

Создание нового файла модульного теста

Откройте новое окно терминала и выполните следующую команду:

```
$ truffle create test erc20FixedSupply
```

Написание теста для контракта полной ставки

В файле ERC20FixedSupply.js напишите новый тест, подтверждающий, что контракт был создан с фиксированной ставкой в 1000 монет.

```
const erc20FixedSupply = artifacts.require("erc20FixedSupply");

contract("erc20FixedSupply", function () {
    it("should assert true", async function()
    {
        let token = await
        erc20FixedSupply.deployed();let name =
        await token.name(); assert.equal(name,
        'Fixed');
    });

    it("should return total supply of 1000", async
        function() {const instance = await
        erc20FixedSupply.deployed(); const totalSupply =
```

```
        await instance.totalSupply();

        assert.equal(totalSupply, 1000);
    });
});
```

Протестируйте с помощью следующей команды:

```
$ truffle test --network development
```

Убедитесь, что тест прошел.

Написание тестовых утверждений для балансового контракта

В файле ERC20FixedSupply.js добавьте еще один тест, подтверждающий правильность баланса после нового перевода между двумя учетными записями.

```
it("should transfer 150 FIX", async function(){
    const instance = await erc20FixedSupply.deployed();
    await instance.transfer(account[1], 150);

    const balanceAccount0 = await instance.balanceOf
    (accounts[0]);
    const balanceAccount1 = await instance.balanceOf
    (accounts[1]);

    assert.equal(balanceAccount0.toNumber(), 850);
    assert.equal(balanceAccount1.toNumber(), 150);
});
```

Запуск модульных тестов

Теперь снова выполните тест.

```
$ truffle test --network development
```

Еще раз убедитесь в том, что все тесты пройдены. Если модульный тест пройден, появится зеленая галочка; в противном случае появится красный символ x.

Проверка результатов модульного теста

Убедитесь, что вы получаете тот же результат, что и на рис. 4-1.

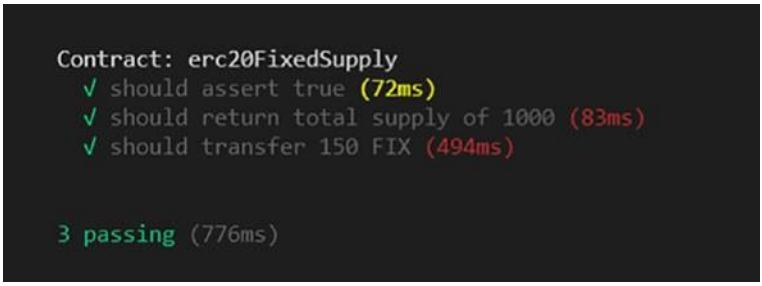

Рис. 4-1. *Код VS: результаты успешного модульного теста*

Попробуйте изменить некоторые значения, такие как баланс счета, и посмотрите, как меняются результаты от прохождения до отказа, как показано на рис. 4-2.

```
it("should transfer 150 FIX", async function(){
    const instance = await erc20FixedSupply.deployed();
    await instance.transfer("account[1]", 150);

    const balanceAccount0 = await instance.balanceOf
    (accounts[0]);
    const balanceAccount1 = await instance.balanceOf
    (accounts[1]);

    assert.equal(balanceAccount0.toNumber(), 750);
```

```
    assert.equal(balanceAccount1.toNumber(), 250);
});
```

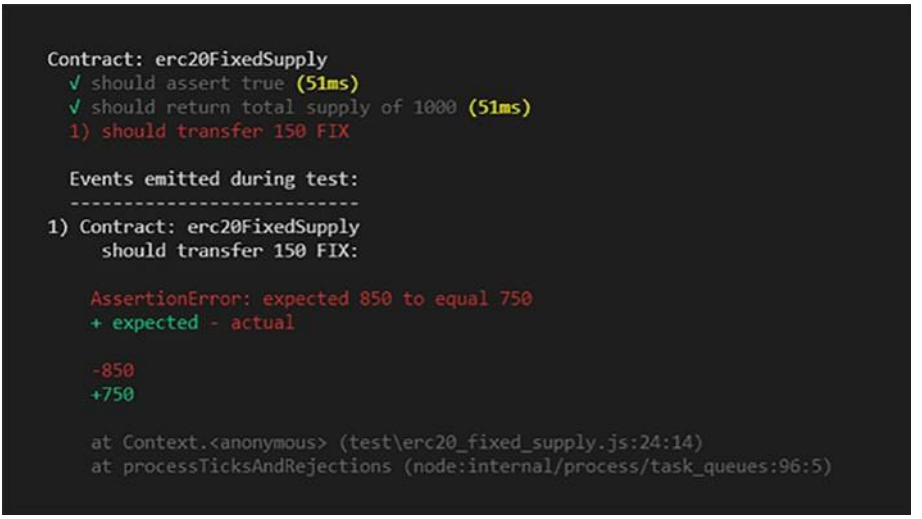

Рис. 4-2. *Код VS: отрицательный результаты модульного теста*

Резюме

В этой главе вы узнали о важности написания модульных тестов для смарт-контрактов и написали свой первый модульный тест.

В следующей главе мы рассмотрим стандарт токенов ERC-721 и его отличия от ERC-20. Кроме того, вы научитесь создавать и развертывать контракты в этом стандарте.

ГЛАВА 5

Невзаимозаменяемые токены ERC-721

Незаменимый токен (NFT) — это цифровой актив, представляющий физические объекты, такие как предметы искусства, музыка, содули игр и видео. В этой главе я покажу вам, как создать NFT ERC-721 и развернуть его в тестовой сети Ethereum, а также как добавить его в ваш мобильный кошелек MetaMask.

В конце этой главы вы сможете сделать следующее:

- Создать новый проект NFT
- Сконфигурировать сеть для развертывания на Ganache
- Сконфигурировать закрытый ключ
- Создать изображение бейджа
- Добавить бейдж в локальную IPFS
- Закрепить бейдж в удаленной IPFS
- Создать метаданные бейджа
- Развернуть смарт-контракт
- Присвоить бейдж своему кошельку
- Проверить бейдж на Etherscan
- Добавить бейдж в свой мобильный кошелек

Создайте свой NFT предмета искусства, используя Ganache и OpenZeppelin

Давайте настроим ваш первый NFT с помощью библиотеки OpenZeppelin и создадим метаданные, в которых будет храниться информация о токенах, а затем развернем их в тестовой сети. В конце вы увидите токен в своем кошельке.

Создание проекта

Создайте новый проект с помощью Truffle.

```
$ truffle init
```

Установите контракты OpenZeppelin.

```
$ npm install @openzeppelin/contracts
```

Создайте новый смарт-контракт Solidity.

```
$ touch contracts/UniqueAsset.sol
```

Откройте файл `UniqueAsset.sol` и импортируйте расширение ERC721URIStorage.sol и утилиту Counters.sol. Создайте новый класс, расширяющий `ERC721URIStorage`. Объявите переменную counters и объявите конструктор, передающий имя монеты и код.

Создайте новый метод с наименованием `awardItem`. Внутри нового метода увеличьте идентификатор токена. Получите новый номер токена, используя `_tokenIds.current()`.

Выпустите новый предмет, используя метод `_mint`. Наконец, установите URI токена, передав метаданные, используя метод `_setTokenURI`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

```
import "@openzeppelin/contracts/token/ERC721/extensions/
ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
    contract UniqueAsset is ERC721URIStorage {
        using Counters for Counters.Counter;
        Counters.Counter private _tokenIds;

        constructor() ERC721("UniqueAsset", "UNA") {}

        function awardItem(address recipient, string memory metadata)
        public
        returns (uint256)
        {
            _tokenIds.increment();
            uint256 newItemId = _tokenIds.current();
            _mint(recipient, newItemId);
            _setTokenURI(newItemId, metadata);
            return newItemId;
        }
    }
```

Создайте новый файл миграции с помощью команды touch. Эта команда создает новый файл в папке миграции.

```
$ touch migrations/2_deploy_contracts.sol
```

Внутри файла 2_deploy_contracts.js экспортируйте смарт-контракт в файл миграции.

```
const UniqueAsset = artifacts.require("UniqueAsset");

module.exports = function (deployer) {
    deployer.deploy(UniqueAsset);
}
```

Конфигурирование кошелька для подписи транзакций

Установите пакет файловой системы fs.

```
$ npm install fs
```

Install the wallet provider hdwallet package.

```
$ npm install @truffle/hdwallet-provider@1.2.3
```

Откройте файл truffle-config.js file и раскомментируйте раздел кода HDWalletProvider.

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const infuraKey = '<your_infura_key>';

const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();
```

Вставьте идентификатор вашего проекта Infura в качестве значения переменной infuraKey.

Конфигурирование сети

Внутри файла truffle-config.js file раскомментируйте раздел ropsten network и внесите следующие изменения:

- Смените ropsten на rinkeby.
- Измените URL-адрес Ropsten Infura на rinkeby.
- Смените YOU-PROJECT-ID на ${infuraKey}.
- Смените network_id на 42.

```
rinkeby: {
    provider: () => new HDWalletProvider(mnemonic, `https://
    rinkeby.infura.io/v3/${infuraKey}`),
```

```
 network_id: 42,
 gas: 5500000,
    confirmations: 2,
    timeoutBlocks: 200,
    skipDryRun: true
},
```

Конфигурирование компилятора Solidity

Кроме того, внутри файла truffle-config.js file раскомментируйте раздел компиляторов и измените версию на 0.8.0.

```
compilers: {
    solc: {
        version: "0.8.0",
        docker: true,
        settings: {
            optimizer: {
                enabled: false,
                runs: 200
            },
            evmVersion: "byzantium"
        }
    }
},
```

Конфигурирование закрытого ключа

Перейдите в браузер и откройте кошелек MetaMask, подключенный к сети Infura. Нажмите "*your account*", а затем нажмите "settings." Наконец, нажмите"security & privacy" (Рис. 5-1).

У вас есть возможность просмотреть свою сид-фразу, но имейте в виду, что эта информация является конфиденциальной, и если кто-то

получит к ней доступ, он сможет восстановить ваш кошелек и использовать ваши средства.

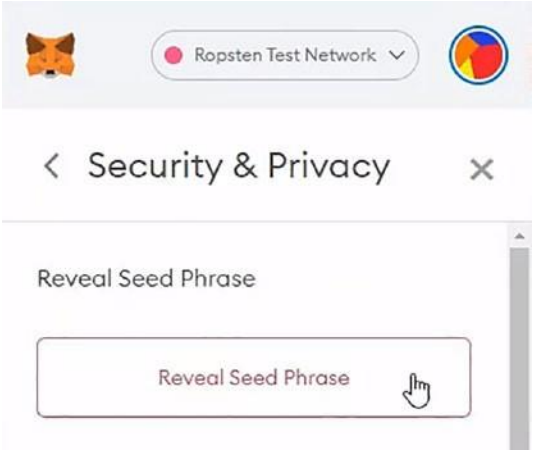

Рис. 5-1. *MetaMask: раскрываем сид-фразу*

Нажмите Reveal Seed Phrase и введите пароль кошелька, чтобы продолжить. Скопируйте закрытый ключ.

Вернитесь в Visual Studio Code и создайте новый файл с наименованием `.secret`. Вставьте секретную фразу восстановления в этот файл.

Создание изображения бейджа

Создайте папку бейджа.

```
$ mkdir badge
```

Теперь перейдите в корневую папку бейджа.

```
$ cd badge
```

Загрузите изображение, которое вы будете использовать в качестве бейджа, из Интернета. Вы также можете скопировать и вставить в эту папку существующее изображение. Команда curl используется для

передачи данных посредством синтаксиса URL.

```
$ curl https://planouhost.z15.web.core.windows.net/badge.png > badge-image.png
```

Добавление бейджа в вашу локальную IPFS

Инициализируйте локальный узел IPFS. Эта команда запустит локальный сервер IPFS на 127.0.0.1:5001.

```
$ ipfs daemon
```

Добавьте изображение бейджа в IPFS.

```
$ ipfs add badge-image.png
```

Running this command, you will receive a hash. This hash is yourimage address in IPFS. Make sure that you see the output shown in Figure 5-2.

```
$ ipfs add badge-image.png
 20.68 KiB / 20.68 KiB [=============================================
added QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4 badge-image.png
 20.68 KiB / 20.68 KiB [=============================================
```

Рис. 5-2. *Вывод IPFS после добавления файла*

Закрепление бейджа на удаленном узле IPFS

Закрепите свой бейдж, используя Pinata в качестве удаленной службы IPFS.

```
ipfs pin remote add --service=pinata --name=badge-image.png
QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4
```

Вы получите ответ о том, что файл был успешно закреплен.

```
CID: QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4
Name: badge-image.png
Status: pinned
```

Создание метаданных бейджа

Создайте JSON-файл метаданных бейджа.

```
touch badge-metadata.json
```

Откройте файл badge-metadata.json и задайте наименование бейджа, описание и адрес изображения. Для последнего вы можете использовать шлюз IPFS, чтобы изображение отображалось в любом кошельке, который поддерживает этот тип бейджа; в противном случае вы будете зависеть от поддержки целевого кошелька, отображающего изображения из хэшей IPFS напрямую.

```
{
    "name": "My badge",
    "description": "My badge description",
    "image":
    "https://ipfs.io/ipfs/QmZPxKJWqJTdudyaZUyf6uBzww
    AT41QQyxhTHmMZWB9yx4"
}
```

Добавьте метаданные своего бейджа в IPFS.

```
$ ipfs add badge-metadata.json
```

Закрепите метаданные бейджа с помощью удаленной службы IPFS.

```
$ ipfs pin remote add --service=pinata --name=badge-
metadata.json
QmRzcwAtLWbeYqyaZUyf6uBzwwAT41QQyxhTHmMZWBfUTa
```

Составление смарт-контракта

Скомпилируйте контракт с помощью Truffle.

```
$ truffle compile
```

Миграция смарт-контракта

Перенесите контракт в сеть Rinkeby с помощью Truffle.

```
$ truffle migrate --network rinkeby
```

Создание экземпляра смарт-контракта

Создайте экземпляр контракта с помощью консоли Truffle.

```
$ truffle console --network rinkeby
```

Получить экземпляр развернутого контракта.

```
truffle(rinkeby) let instance = await UniqueAsset.deployed()
```

Присвоение бейджа кошельку

Вызовите метод awardItem и передайте адрес Ethereum в качестве первого параметра, а также адрес IPFS для метаданных бейджа. Убедитесь, что адрес IPFS соответствует метаданным вашего бейджа.

```
truffle(rinkeby) let result = await instance.awardItem
("0x62761466bB3A3Da83B408B5F5fE00ac7b2a5A996","https://ipfs.io/
ipfs/QmRzcwAtLWBeYqUx3ba1BkYKubSDLNTHCuiUB7WAmdfUTa")
```

Проверка бейджа на Etherscan

Как только ваш контракт будет развернут, вы сможете увидеть публичный адрес вашего контракта. Найдите в терминале адрес созданного контракта и скопируйте его.

Перейдите на https://rinkeby.etherscan.io и вставьте адрес контракта в строку поиска (Рис. 5-3). Вы можете использовать инструмент Rinkeby Testnet Explorer для просмотра сведений о

созданном смарт-контракте.

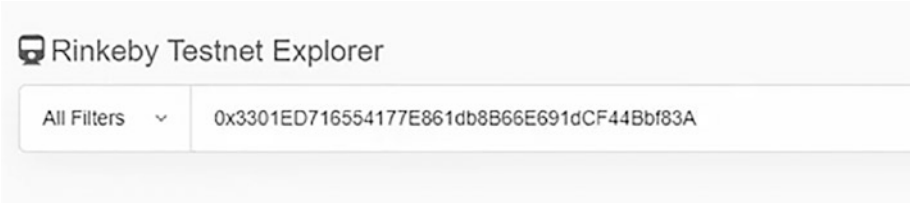

Рис. 5-3. *Обозреваель Rinkeby Testnet: поиск смарт-контракта*

Щелкните значок поиска. Теперь вы можете видеть, что контракт был успешно развернут (Рис. 5-4). На этой странице сведений вы можете просматривать такие данные, как транзакции.

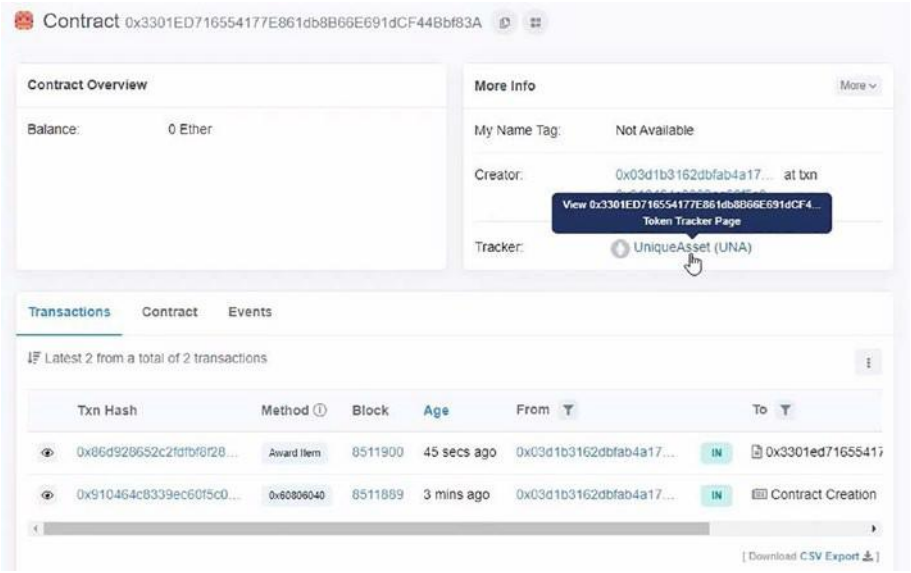

Рис. 5-4. *Обозреваель Rinkeby Testnet: просмотр транзакций смарт-контрактов*

Вы также можете выяснить, что последняя транзакция была сделана для присвоения нового предмета.

Добавление токена NFT в ваш кошелек

Откройте кошелек MetaMask на мобильном телефоне и нажмите Collectibles (Рис. 5-5). Обратите внимание, что предметы коллекционирования доступны только в мобильной версии.

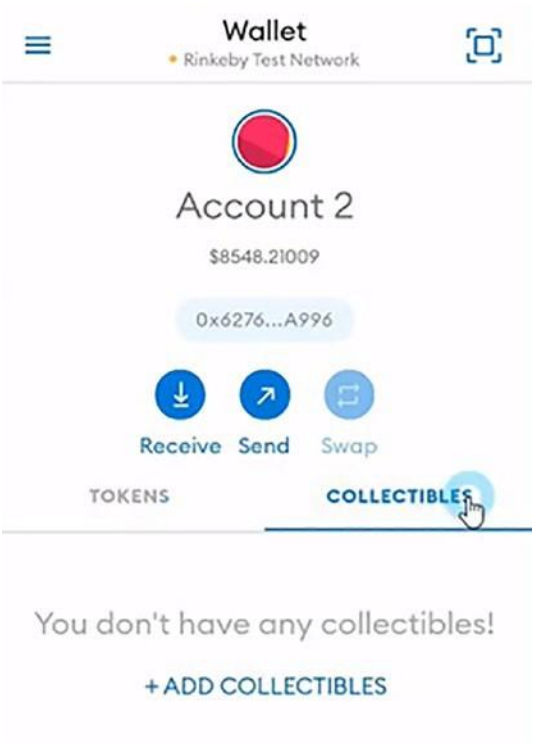

Рис. 5-5. *MetaMask: вкладка Collectibles*

Нажмите Add Collectibles. Вставьте сюда адрес контракта токена (тот же, что вы скопировали в предыдущем разделе) и введите идентификатор токена (поскольку это первый токен, вы должны ввести здесь 1).

Нажмите Add и подождите несколько секунд (Рис. 5-6). Токен NFT был добавлен !

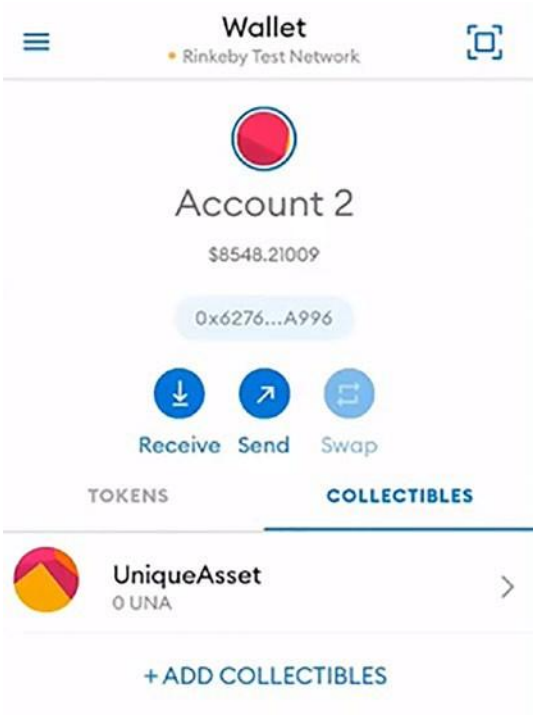

Рис. 5-6. *MetaMask: после добавления смарт-контракта он появится здесь*

Щелкните Уникальный актив. Теперь вы сможете увидеть все значки, которые вы заработаете (Рис. 5-7). У вас может быть несколько токенов, происходящих из одного и того же смарт-контракта, каждый из которых будет отличаться уникальным идентификатором.

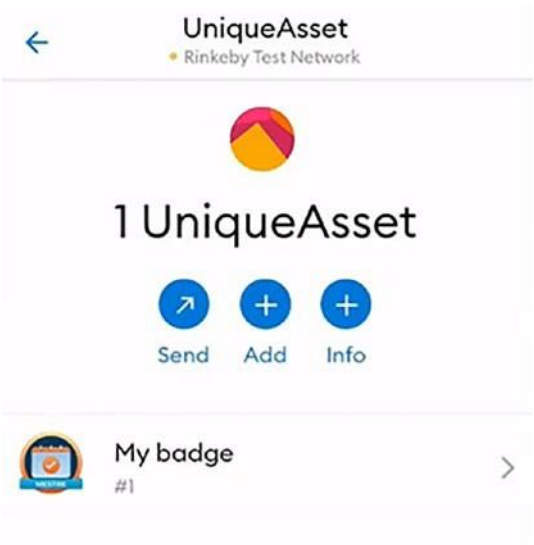

Рис. 5-7. *MetaMask: список бейджей*

Нажмите “My badge.” Теперь вы можете увидеть детали бейджа! Кроме того, у вас есть кнопка Send button, чтобы вы могли отправить значок в другой кошелек (Рис. 5-8).

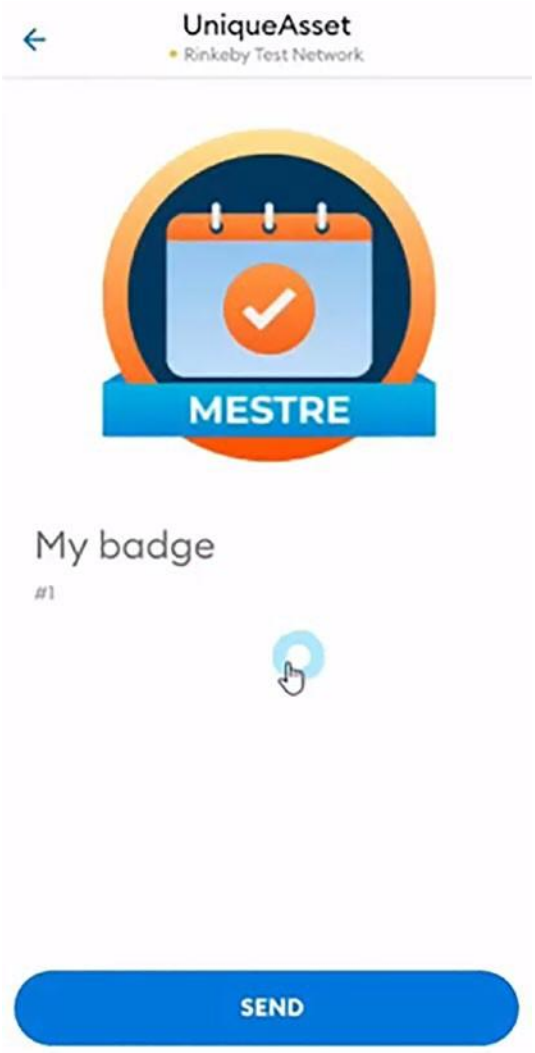

Рис. 5-8. *MetaMask: отображение бейджа*

Вот и все! Вы только что создали свой первый токен NFT!

Продайте свое произведение искусства NFT на OpenSea

OpenSea — это рынок цифровых товаров, таких как предметы коллекционирования, модули игр, цифровое искусство и другие цифровые активы, поддерживаемые блокчейном, таким как Ethereum. Вы можете покупать, продавать и обменивать любые из этих вещей с кем угодно в мире на OpenSea.

Подключение к OpenSea

Перейдите в OpenSea (https://testnets.opensea.io) и убедитесь в том, что вы подключены к кошельку, содержащему NFT, и что вы используете тестовую сеть Rinkeby.

Просмотр вашего бейджа

Перейдите в Мой профиль. Нажмите Activity, а затем нажмите на название бейджа. Это данные вашего бейджа. На странице сведений вы можете просмотреть различную информацию, касающуюся продажи вашего бейджа.

Выставление бейджа на продажу

Нажмите Sell, а затем нажмите Set Price. В поле Price установите цену, по которой вы хотите продать NFT. На этой странице вы можете установить метод ценообразования для бейджа, а также запланировать его отображение в будущем (Рис. 5-9).

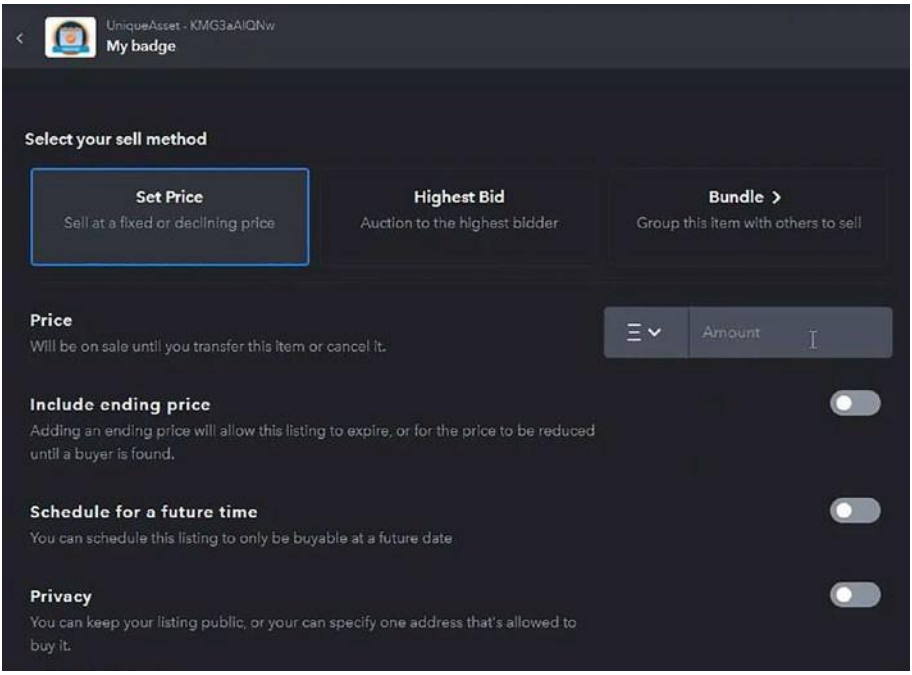

Рис. 5-9. *OpenSea: страница с ценами на бейдж*

Нажмите Post Your Listing. Вы будете перенаправлены на страницу сводки (Рис. 5-10). На этой странице вы можете увидеть общую сумму сборов, которые будут вычтены при продаже вашего бейджа.

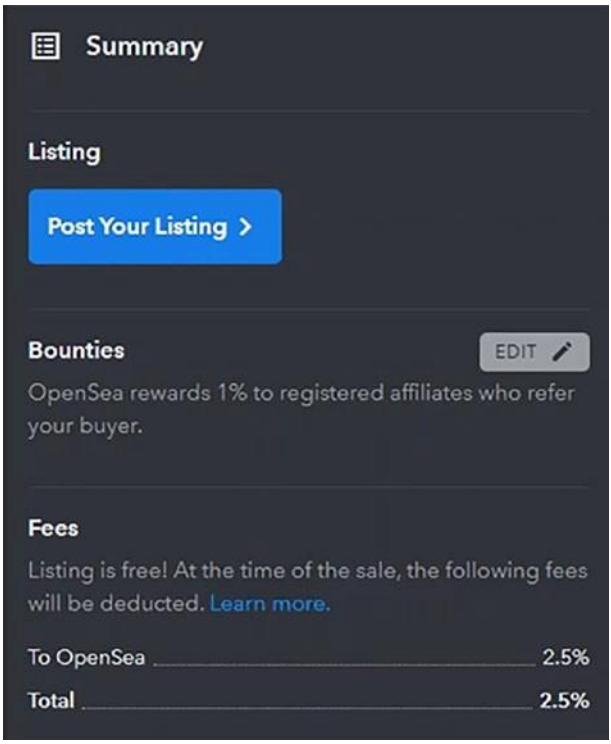

Рис. 5-10. *OpenSea: Страница сводной информации о бейдже*

MetaMask будет открыт для проверки транзакции (Рис. 5-11). Нажмите Confirm. На этом этапе вам необходимо одобрить транзакцию, которая подтвердит выставление вашего бейджа на продажу на платформе.

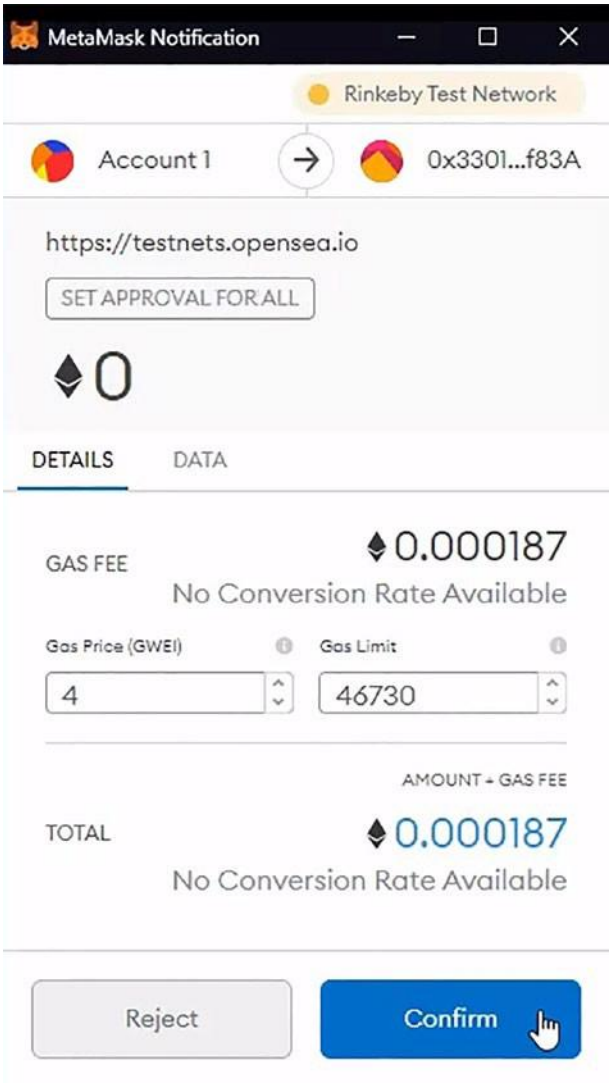

Рис. 5-11. *MetaMask: подтверждение транзакции на OpenSea*

Теперь вам нужно будет предоставить дополнительную информацию о себе, например, адрес электронной почты и псевдоним. После одобрения транзакции вас попросят предоставить дополнительную информацию (Рис. 5-12).

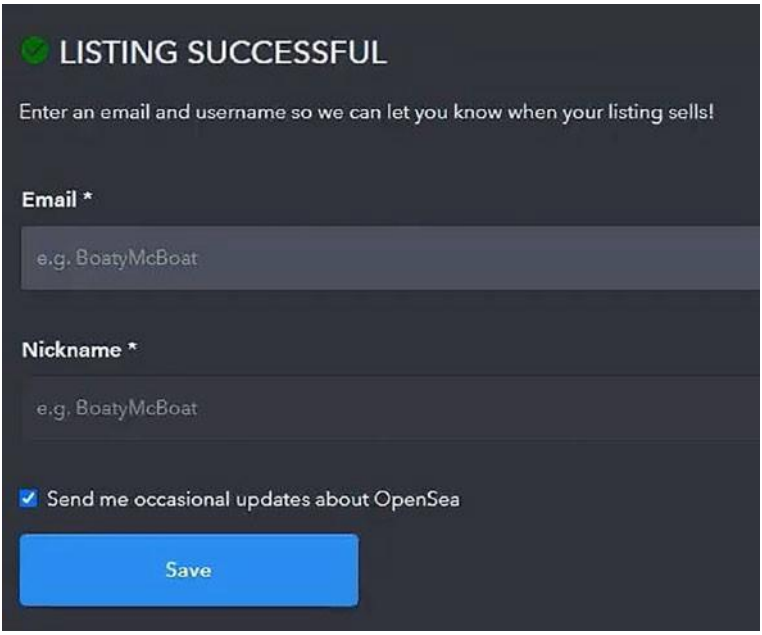

Рис. 5-12. *OpenSea: дополнительная информация*

Щелкните Save. Now, Теперь OpenSea перечислит ваши NFT для вас!

Изучение подробной информации перечня

Прокрутите вниз окно до раздела Trading History (Рис. 5-13); как вы можете видеть, было создано новое событие под названием List и с ценой, установленной в 10 долларов США. На этой странице вы можете увидеть всю историю торговли бейджами на платформе.

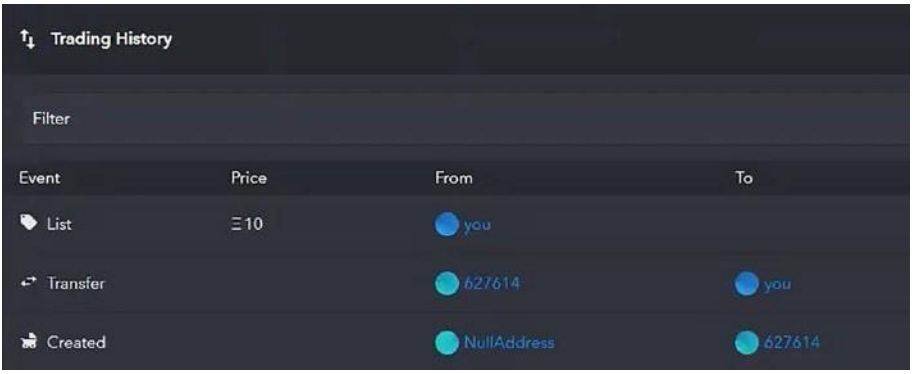

Рис. 5-13. *OpenSea: история торговли*

Щелкните на иконке Share. Вы можете скопировать ссылку или поделиться ею в своих социальных сетях.

Резюме

В этой главе вы узнали, как создавать токены стандарта ERC-721, закреплять образ в IPFS, импортировать его в OpenSea и выставлять на продажу.

В следующей главе мы узнаем, как использовать агрегаторы и почему они важны в тестовых сетях.

ГЛАВА 6

Агрегаторы

Вы можете использовать агрегаторы для тестирования смарт-контрактов в тестовой сети без необходимости использования для этой цели реального эфира (поскольку эфир из основной сети недействителен в тестовых сетях, и наоборот). Эфир в тестовой сети не имеет реальной ценности, кроме целей тестирования при разработке смарт-контрактов.

В конце этой главы вы сможете сделать следующее:

- Получить доступ к агрегатору в сети Ropsten
- Получить доступ к агрегатору в сети Rinkeby
- Получить доступ к агрегатору в тестовой сети Polygon Mumbai
- Получить доступ к агрегатору в основной сети Polygon
- Отправить тестовый эфир в свой кошелек
- Отправить тестовый MATIC в свой кошелек
- Проверить обновленный баланс вашего кошелька

Получение тестового эфира от агрегатора в сети Ropsten

Этот тестовый агрегатор Ethereum (https://faucet.ropsten.be) сбрасывает 1 эфир каждые пять секунд. Чтобы избежать образование сетевого спама, для вас существует лимит запросов 1 eth каждые 24 часа, .

Доступ к агрегатору

Перейдите к агрегатору Ropsten Ethereum (rETH) (https://faucet.dimensions.network/) и скопируйте адрес своего кошелька (убедитесь, что выбрана сеть Ropsten). Вставьте адрес вашего контракта в поле формы. Нажмите «Send Ropsten ETH», как показано на Рис. 6-1.

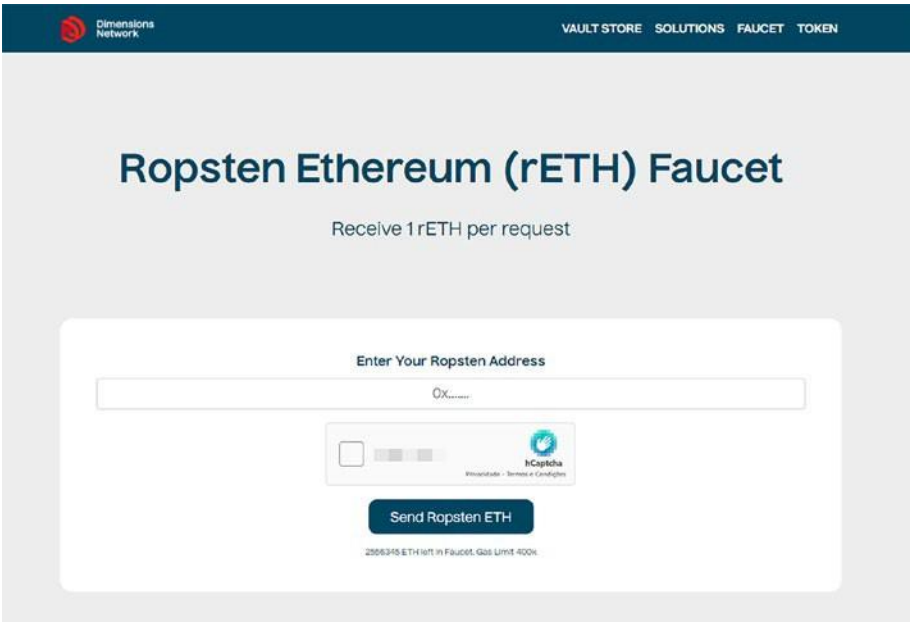

Рис. 6-1. *Домашняя страница агрегатора Ropsten*

Ожидание транзакции

Нажмите на хэш транзакции (который откроет новое окно) и дождитесь завершения транзакции. Как только транзакция будет успешно завершена, перейдите в свой кошелек MetaMask, и вы увидите, что теперь у вас есть 1 эфир!

Получение тестового эфира от агрегатора в тестовой сети Rinkeby Testnet

Этот агрегатор эфира подключен к сети Rinkeby. Запросы привязаны к общим учетным записям сторонних социальных сетей, чтобы злоумышленники не могли исчерпать все доступные активы или накопить достаточно эфира для организации длительных спам-атак. Любой, у кого есть учетная запись Twitter или Facebook, может запросить средства в пределах разрешенных лимитов.

Подготовка к финансированию

Откройте свой кошелек MetaMask и скопируйте адрес своего кошелька в буфер обмена. Перейдите в свою учетную запись Twitter и вставьте адрес своего кошелька. Щелкните свой твит и скопируйте адрес твита (URL-адрес в адресной строке).

Пополнение вашего кошелька

Перейдите на https://faucet.rinkeby.io и вставьте свой адрес твита в текстовое поле, как показано на Рис. 6-2. Нажмите “Give me Ether” и выберите один из доступных вариантов (например, 3 эфира/8 часов). Заявка будет профинансирована в течение нескольких секунд.

Важно отметить, что существуют разновидности агрегаторов. Периодически какие-то из них могут пустовать, либо со временем они могут выйти из строя. Или могут быть даже созданы новые агрегаторы. Если какие-то из них станут недоступными, вы можете поискать новые в сети Интернет.

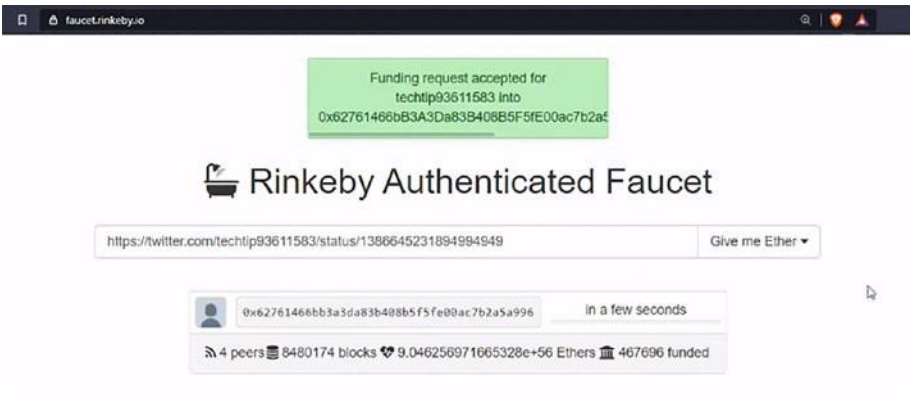

Рис. 6-2. *Домашняя страница агрегатора Ринкеби*

Проверка вашего кошелька

Подождите несколько минут и проверьте свой кошелек MetaMask. Вы получите в свой кошелек 3 эфира!

Получение Test MATIC от агрегатора в тестовой сети Mumbai Testnet

Этот агрегатор передает TestToken/MATIC-ETH в тестовые сети Matic и соответствующую родительскую цепочку.

Подготовка к финансированию

Откройте свой кошелек MetaMask и скопируйте адрес своего кошелька в буфер обмена.

Пополнение вашего кошелька

Перейдите на https://faucet.matic.network. Для токена выберите MATIC Token, а для сети выберите Mumbai. Вставьте свой адрес твита в текстовое поле и нажмите Submit (Рис. 6-3). Вам будет рдставлна детальная информация подтверждения. Нажмите Confirm. Заявка будет профинансирована в течение нескольких секунд.

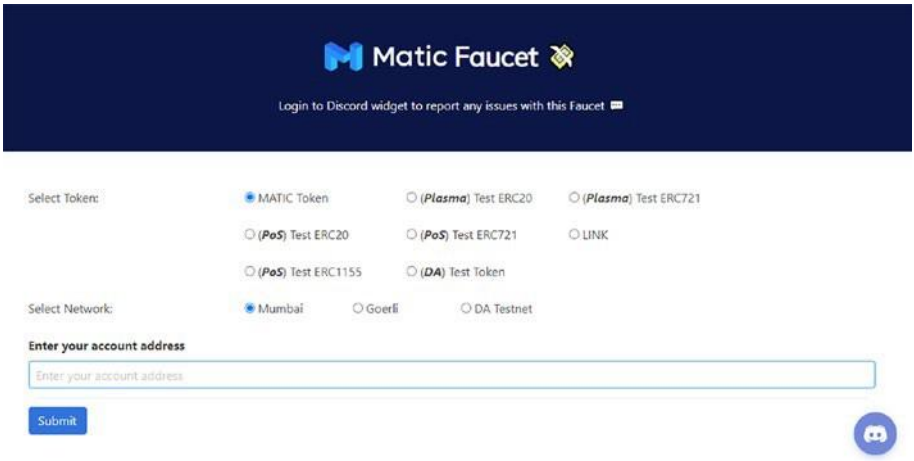

Рис. 6-3. *Домашняя страница агрегатора Matic*

Проверка вашего кошелька

Подождите несколько минут и проверьте свой кошелек MetaMask. Вы получите 0.1 MATIC в свой кошелек!

Получение тестового MATIC от агрегатора в основной сети

Этот агрегатор передает TestToken/MATIC-ETH в тестовые сети Polygon и соответствующую родительскую цепочку.

Подготовка к финансированию

Откройте свой кошелек MetaMask и скопируйте адрес своего кошелька в буфер обмена.

Пополнение вашего кошелька

Перейдите на https://matic.supply и нажмите “Connect”, чтобы подключиться к вашему кошельку MetaMask (Рис. 6-4). Убедитесь, что ваш кошелек MetaMask подключен к основной сети Matic. Нажмите “I am human” и решите капчу.

Рис. 6-4. *Домашняя страница агрегатора Polygon*

Проверка вашего кошелька

Подождите несколько минут и проверьте свой кошелек MetaMask. Вы только что получили 0,001 MATIC в свой кошелек!

Резюме

В этой главе вы узнали, как получить тестовый эфир от агрегаторов в сети Интернет. Это будет очень полезно при развертывании контрактов в тестовых сетях, чтобы вам не пришлось тратить реальный эфир.

ГЛАВА 7

Файловая система InterPlanetary

Фаловая систма InterPlanetary (https://ipfs.io) (IPFS) представляет собой протокол и одноранговую сеть, которая позволяет хранить и совместно использовать данные в распределенной файловой системе. IPFS использует адресацию контента, чтобы различать каждый файл в глобальном пространстве имен, которое объединяет все вычислительные устройства.

В конце этой главы вы сможете сделать следующее:

- Установить пакет узла IPFS и инициализировать узел
- Просматривать одноранговые узлы IPFS
- Протестировать и изучить узел IPFS
- Добавлять файлы в IPFS
- Просмотривать содержимое файла на консоли и проверять файл в веб-интерфейсе
- Просматривать содержимое файла прямо в браузере
- Установить расширение для браузера IPFS
- Сконфигрировать тип узла IPFS и запустить узел
- Импортировать файл в узел IPFS
- Запустить локальный узел IPFS и добавить к нему файлы

- Проверить добавленные файлы и проверить, был ли файл закреплен
- Закрепить файлы вручную
- Сконфигурировать API-ключи в Pinata
- Настроить Pinata как удаленную службу
- Закрепить файл на удаленном узле IPFS, а также открепить его
- Войти в Fleek
- Клонировать существующий репозиторий
- Установить и инициализировать пакет Fleek
- Развернуть сайт на Fleek

Создайте свой узел IPFS

Теперь давайте создадим узел IPFS с помощью командной строки и загрузим ваш первый файл.

Установка узла

Установите IPFS с помощью менеджера пакетов Choco (https://chocolatey.org). Пакет go-ipfs — это реализация IPFS в Go.

```
$ choco install go-ipfs
```

Настройка узла

Запустите локальный репозиторий IPFS.

```
$ ipfs init
```

Запустите локальный сервер IPFS. Команда демона запускает

локальный сервер IPFS на 127.0.0.1:5001.

```
$ ipfs daemon
```

Тестирование узла

Вы можете протестировать узел IPFS, показав пиры, которые напрямую подключены к вашему узлу.

```
$ ipfs swarm peers
```

Вы также можете просмотреть некоторое содержимое файла IPFS, используя команду cat и передав хэш в качестве параметра (Рис. 7-1).

```
$ ipfs cat <hash>
```

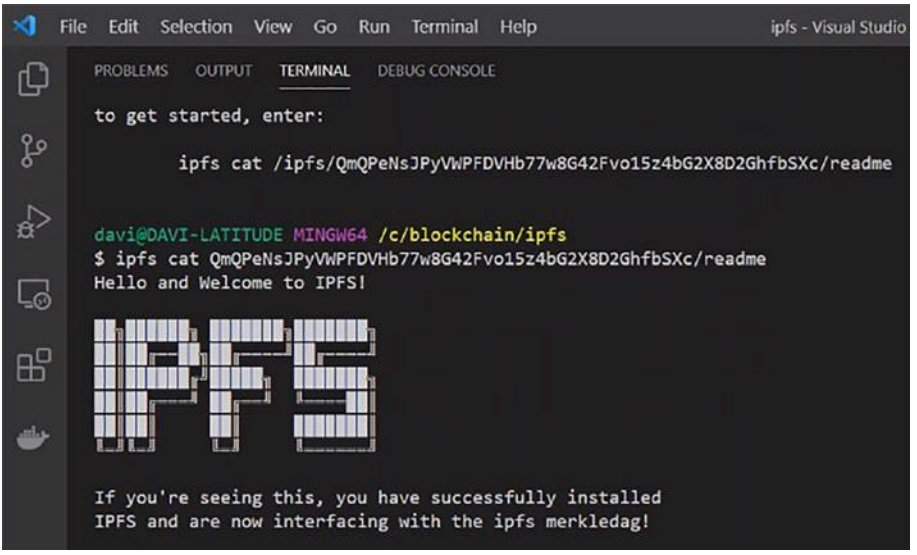

Рис. 7-1. *Вывод команды IPFS cat*

Изучение вашего узла IPFS

Перейдите в браузер и войдите в веб-интерфейс по адресу http://127.0.0.1:5002/. Ваш узел теперь подключен к IPFS! Теперь нажмите Files и обратите внимание, что здесь еще нет файлов (Рис. 7-2).

Нажмите «Explore», а затем нажмите «Peers». Это пиры, к которым вы подключены. Наконец, нажмите «Settings». Здесь вы можете увидеть настройки вашего узла.

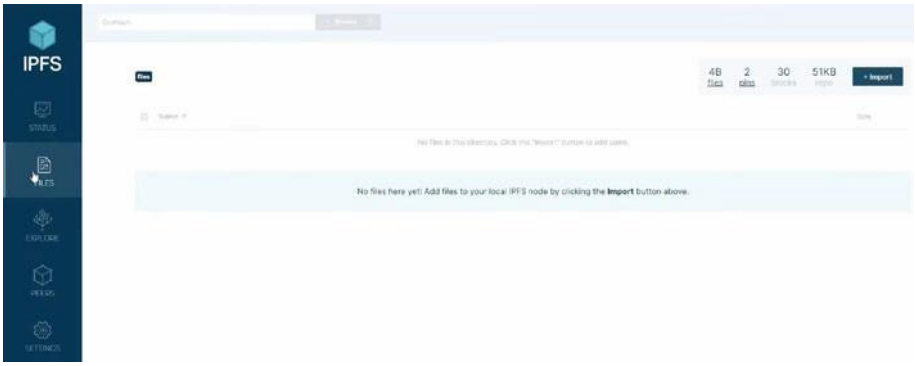

Рис. 7-2. *Веб-интерфейс IPFS*

Добавьте файлы в IPFS

Компьютер, на котором запущена IPFS, может запросить у всех одноранговых узлов, к которым он подключен, есть ли у них файл с определенным хэшем, и если один из них обладает таким файлом, то этот одноранговый узел отправит обратно весь файл. Это было бы невозможно без короткого уникального идентификатора, такого как криптографический хэш.

Добавление файла

Запустите локальный сервер IPFS. Команда daemon запускает локальный сервер IPFS на 127.0.0.1:5001.

```
$ ipfs daemon
```

Теперь создайте новый файл с именем hello.txt, используя команду echo. Эта команда выводит данный текст в новый файл.

```
$ echo "test" hello.txt
```

Добавьте вновь созданный файл на локальный узел IPFS с помощью команды ipfs add.

```
$ ipfs add hello.txt
```

Файл был добавлен в IPFS, в результате чего появился хэш-идентификатор.

Просмотр содержимого файла на консоли

Вы можете увидеть содержимое этого недавно добавленного файла, просто используя команду ipfs cat, за которой следует хэш. Для этого замените <your_hash> вырезанным хеш-идентификатором, сгенерированным на предыдущем шаге командой "ipfs add hello.txt".

```
$ ipfs cat <your_hash>
```

После запуска этой команды содержимое файла будет отображаться в окне терминала.

Проверка файла в веб-интерфейсе

Перейдите на http://127.0.0.1:5001/webui и нажмите "Files". Теперь нажмите "Pins" и скопируйте хэш. Ищите этот хэш и вы увидите, что этот хеш существует.

Просмотр содержимого файла в браузере

Откройте новую вкладку и перейдите по адресу ipfs://<your_hash>. Теперь вы можете видеть содержимое вашего файла в браузере. Здесь снова замените <your_hash> вырезанным хэш-идентификатором, сгенерированным для вашего файла, командой “ipfs cat <your_hash>”.

Настройка расширения браузера IPFS

Дополнительное расширение IPFS (https://chrome.google.com/webstore/detail/ipfs-companion/nibjojkomfdiaoajekhjakgkdhaomnch) позволит вам запускать узел IPFS локально в предпочитаемом вами браузере, обеспечивая поддержку адресов ipfs://, автоматическую загрузку веб-сайтов и путей к файлам через шлюз IPFS, простой импорт и обмен файлами IPFS и многое другое.

Установка расширения браузера

Перейдите к дополнительному расширению IPFS (https://chrome.google.com/webstore/detail/ipfs-companion/nibjojkomfdiaoajekhjakgkdhaomnch?hl=en) и нажмите “Add to Brave” или название вашего браузера. Нажмите “Add extension”, а затем щелкните на иконке “Extensions”. Наконец, закрепите дополнительное расширение IPFS на панели расширений.

Настройка типа узла

Щелкните на иконке IPFS Companion, а затем щелкните на пиктограмме шестеренки. Для типа узла IPFS выберите External.

Запуск внешнего узла

Перейдите в Visual Studio Code и откройте новое окно терминала. Запустите новый локальный сервер IPFS.

```
$ ipfs daemon
```

Импорт файла

Щелкните на иконке IPFS Companion, а затем нажмите “Import”. Нажмите “Pick a file” и выберите файл на локальном диске. Файл будет храниться на вашем узле IPFS.

Закрепление и открепление файлов IPFS на локальном узле

Данные можно закрепить на одном или нескольких узлах IPFS, чтобы они оставались в IPFS и не удалялись во время удаления мусора. Закрепление позволяет управлять пространством для хранения и хранением данных. Поэтому вам следует закреплять любой контент, который вы хотите сохранить в IPFS на неопределенный срок. По умолчанию IPFS прикрепляет файлы к вашему локальному узлу IPFS.

Запуск вашего локального узла

Запустите локальный сервер IPFS. Команда daemon запускает локальный сервер IPFS на 127.0.0.1:5001.

```
$ ipfs daemon
```

Добавление файла в ваш узел

Теперь создайте новый файл с именем hello.txt, используя команду echo. Эта команда выводит данный текст в новый файл.

```
$ echo "world" hello.txt
```

Добавьте вновь созданный файл на локальный узел IPFS с помощью команды ipfs add.

```
$ ipfs add hello.txt
```

Файл был добавлен в IPFS, в результате чего появился хэш-идентификатор. Когда вы добавляете файл, он автоматически прикрепляется к вашему локальному узлу.

Проверка того, что ваш файл был добавлен

Чтобы убедиться, что ваш файл был добавлен, вы можете использовать команду ipfs cat для вывода содержимого файла в окно терминала.

```
$ ipfs cat your_file_hash
```

Проверка того, что ваш файл был закреплен

Перейдите по адресу http://127.0.0.1:5001/webui, нажмите Files, а затем нажмите Pins. Ваш файл там!

Открепление вашего файла

Вы можете открепить файл от вашего локального узла IPFS, просто используя следующую команду:

```
$ ipfs pin rm <your_file_hash>
```

Закрепление вашего файла вручную

Вы можете закрепить файлы вручную, используя следующую команду. Помните, что вам нужно скопировать хэш вашего файла, чтобы закрепить или открепить его.

```
$ ipfs pin add <your_file_hash>
```

Вы закончили; ваш файл снова закреплен!

Закрепление и открепление файлов на удаленном узле с помощью Pinata

Вы также можете прикрепить свои файлы к удаленной службе закрепления. Эти сторонние службы позволяют вам закреплять файлы на узлах, которыми они управляют, а не на вашем собственном локальном узле. Вам не нужно беспокоиться о дисковом пространстве или времени безотказной работы вашего собственного узла.

Хотя вы можете управлять файлами IPFS, прикрепленными к удаленной службе закрепления, с помощью ее собственного графического интерфейса пользователя, интерфейса командной строки или других инструментов разработки, вы также можете напрямую работать со службами закрепления, используя локальную установку IPFS, что устраняет необходимость изучения уникального API службы закрепления или другого инструментария.

Настройка кнопок API в Pinata

Войдите в свою учетную запись Pinata и перейдите к ключам API. Нажмите New Key, а затем установите флажок Admin. Введите **admin-cli** в качестве имени ключа и, наконец, нажмите Create Key. Для вас будет сгенерирован новый ключ. Скопируйте значение JWT из этого окна.

Настройка Pinata в качестве удаленной службы на вашем терминале

Добавьте Pinata в качестве закрепляемой удаленной службы. Вставьте свой JWT внутрь фрагмента <your_jwt_key>.

```
$ ipfs pin remote service add pinata
https://api.pinata.cloud/psa <your_jwt_key>
```

Перечислите все существующие удаленные сервисы и убедитесь, что там есть Pinata.

```
$ ipfs pin remote service ls
```

Добавление нового файла на локальный узел IPFS

Добавьте новый файл на локальный узел IPFS.

```
$ echo "world" > hello.txt
$ ipfs add hello.txt
```

Скопируйте хеш-идентификатор, сгенерированный после добавления файла на локальный узел.

Закрепление файла на удаленном узле IPFS

Закрепите файл с помощью следующей команды. Вставьте хэш файла внутри фрагмента <*your_file_hash*>.

```
$ ipfs pin remote add --service=pinata -name=hello.txt <your_
file_hash>
```

Вернитесь на веб-сайт Pinata и нажмите Pin Manager. Ваш файл появится на этой странице!

Открепление вашего файла от удаленного узла IPFS

Открепите файл с помощью следующей команды. Вставьте хэш файла внутрь вашего фрагмента <*your_file_hash*>.

```
$ ipfs pin remote rm --service=pinata -name=hello.txt <your_file_hash>
```

Вернитесь на веб-сайт Pinata и нажмите Pin Manager. Ваш файл больше не будет отображаться на этой странице; это означает, что ваш файл был откреплен.

Разместите свой сайт на IPFS с помощью Fleek

Fleek позволяет создавать архитектуру базового уровня на основе открытых веб-протоколов. Создавайте и размещайте свои сайты, приложения, DApps и другие сервисы на не требующих доверия, не требующих разрешений и открытых технологиях, которые ориентированы на обеспечение контролируемого пользователем, зашифрованного, частного, одноранговое взаимодействия.

Зайдите на Fleek

Перейдите на https://fleek.co и войдите в свою учетную запись; затем перейдите в редактор VS Code.

Клонирование существующего репозитория

Клонируйте существующий репозиторий с некоторым образцом кода.

```
$ git clone https://github.com/johnnymatthews/random-planet-facts
```

Установка Fleek

Установите командную строку Fleek.

```
$ npm install -g @fleekhq/fleek-cli
```

Войдите в свою учетную запись Fleek (вам будет предложено завершить процесс в браузере).

```
$ fleek login
```

Инициализация Fleek

Инициализируйте сайт Fleek в текущем каталоге.

```
$ fleek site:init
```

Вам будет предложено выбрать, какую команду вы хотите использовать (используйте клавиши со стрелками для выбора). Кроме того, выберите, какой сайт вы хотите использовать. Наконец, выберите публичный каталог для развертывания.

Развертывание вашего сайта

Разверните изменения в вашем публичном каталоге.

```
$ fleek site:deploy
```

Вернитесь на сайт Fleek и нажмите Hosting; затем нажмите Verify

on IPFS. Это ваш сайт, размещенный на IPFS. Теперь вы можете увидеть развернутый сайт онлайн.

Вернитесь в Hosting и щелкните `your-site.on.fleek.co`. Это дружественный адрес хоста вашего сайта (Рис. 7-3).

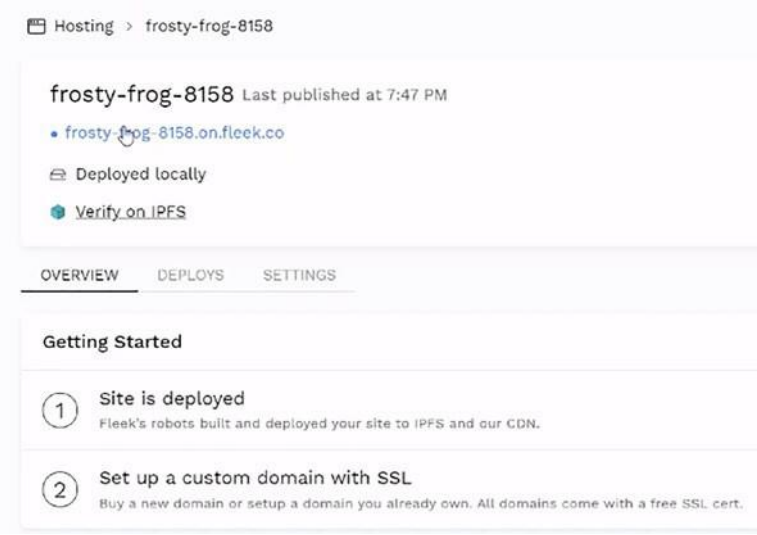

Рис. 7-3. *Обзор хостинга Fleek*

Резюме

В этой главе вы узнали, как установить и создать узел IPFS, а также как управлять файлами посредством этого протокола.

В следующей главе вы узнаете, как создать проект Filecoin.

ГЛАВА 8

Filecoin

Protocol Labs, та же компания, которая изобрела и поддерживает IPFS, создала Filecoin. Он расширяет концепцию IPFS за счет создания децентрализованной сети хранения данных с открытым исходным кодом и стимулирует пользователей сохранять данные, закрепленные в IPFS.

Filecoin хорош тем, что предлагает решение для децентрализованного хранения, которое не требует использования более крупных централизованных решений. В отличие от решений, полностью основанных на IPFS, узлы Filecoin имеют финансовые стимулы для того, чтобы продолжать оставаться активными.

В этом примере мы будем использовать реализацию Filecoin с наимнованием Lotus. Он проверяет сетевые транзакции, управляет кошельком FIL и может выполнять операции хранения и извлечения.

В конце этой главы вы сможете сделать следующее:

- Создать проект Filecoin
- Настроить Truffle для использования конечных точек Filecoin
- Запускать локальный сервер Filecoin
- Добавить изображения для сохранения на Filecoin

Как сохранить файлы на локальном узле Filecoin

Чтобы сохранить файлы в Filecoin, вы изучите, как создать проект с нуля и настроить Truffle так, чтобы он указывал на правильный адрес. Вы запустите локальные конечные точки и, наконец, добавите изображение в Filecoin с помощью командной строки.

Создание проекта

Перейдите в терминальный режим и нажмите New Terminal. Теперь создайте новую папку, чтобы начать свой проект.

```
$ mkdir create filecoin
```

Конфигурирование Truffle

Создайте файл `truffle-config.js`.

```
$ touch truffle-config.js
```

В этом файле настройте параметры файла для серверов lotus IPFS и Filecoin. Установите следующие значения:

- Установите адрес IPFS как `http://127.0.0.1:5001`.
- Установите адрес Filecoin address как http://localhost:7777/rpc/v0.

Позже вы будете использовать Ganache Filecoin для запуска этих конечных точек.

```
module.exports = {
  environments: {
    development: {
      ipfs: {
        address: "http://127.0.0.1:5001",
```

```
  },
    filecoin: {
      address: "http://localhost:7777/rpc/v0",
      storageDealOptions: {
        epochPrice: "2500",
        duration: 518400,
      }
    },
  }
 }
};
```

Добавление изображения для сохранения

Создайте папку бейджа.

```
$ mkdir badge
```

Перейдите в корневую папку бейджа.

```
$ cd badge
```

Загрузите изображение, которое вы будете использовать для сохранения на Filecoin. Вы также можете скопировать и вставить в эту папку существующее изображение. Команда curl используется для передачи данных с синтаксисом URL.

```
$ curl https://planouhost.z15.web.core.windows.net/badge.png > badge-image.png
```

Установка зависимостей

Установите базовый пакет Ganache с тегом Filecoin.

```
$ npm install ganache@filecoin
```

Установите пакет взаимозависимостей пиров Filecoin.

```
$ npm install @ganache/filecoin
```

Запустите локальные серверы Lotus и IPFS с помощью Ganache. Он также создает 10 учетных записей разработки со 100 Filecoins (FIL) у каждой.

Запуск локальных конечных точек

Запустите Ganache Filecoin локально.

```
$ npx ganache filecoin
```

Сохранение файлов в Filecoin

Откройте новое окно терминала и сохраните бейдж папки.

```
$ truffle preserve ./badge/ --filecoin
```

Если все прошло хорошо, будет выведена следующая информация:

```
Preserving target: ./badge/
===========================
Warning: multiple plugins found that output the label "ipfs-cid".

  ✓ Loading target...
    ✓ Reading directory ./badge/...
      ✓ Opening ./badge\badge-image.png...

  ✓ Preserving to IPFS...
    ✓ Connected to IPFS node at http://127.0.0.1:5001
    ✓ Uploading...
      Root CID: QmNjGGACAMpm718WjeM7oN3WJ5ntXKgwurH4mMivU6JXoS
        ./badge-image.png: QmemrBxhPpg8Hw9sPpAUTWRq3kNAFhCPKUD
        hMnc5U9KptS
```

```
✓ Preserving to Filecoin...
  ✓ Connected to Filecoin node at http://localhost:7777/rpc/v0
  ✓ Retrieving miners...
  ✓ Proposing storage deal...
    Deal CID: bafyreidrt2pvuh44umo7vzebnxnw46qzntneyojah6oym4
    ximokyzvwxgq
  ✓ Waiting for deal to finish...
    Deal State: StorageDealActive
```

Теперь у вас есть изображение, сохраненное на вашем локальном узле Filecoin!

Резюме

В этой главе вы узнали, как сохранять файлы с помощью Filecoin. В следующей главе вы узнаете, что такое ENS и как ее использовать.

ГЛАВА 9

Служба имен Ethereum

Служба имен Ethereum (ENS) позволяет пользователям отправлять и получать Ethereum, а также получать доступ к специальным веб-сайтам, используя простые имена, а не длинные, сложные последовательности букв и цифр.

В конце этой главы вы сможете сделать следующее:

- Поиск доменного имени в сети ENS
- Зарегистрировать свободный домен
- Управлять зарегестрированным в ENS именем
- Проверять разрешение имени ENS

Зарегистрируйте свой ENS, чтобы получать в свой кошелек криптовалюту, токены или NFT

Давайте найдем свободное доменное имя в ENS и зарегистрируем его. Далее вы будете управлять адресом через страницу регистрации. Наконец, вы проверите, правильно ли домен разрешен для сконфигурированного адреса.

Поиск вашего доменного имени

Перейдите на страницу доменов ENS (https://ens.domains) и нажмите Launch App. Найдите доменное имя, которое вы хотите зарегистрировать (например, planou.eth). Проверьте срок регистрации (например, минимум один год) и стоимость регистрации.

Регистрация вашего имени

Нажмите Request to Register. Откроется уведомление MetaMask для подтверждения транзакции. Нажмите Confirm и дождитесь подтверждения транзакции в блокчейне. После подтверждения нажмите Register. Появится новое уведомление MetaMask. Нажмите Confirm еще раз и дождитесь подтверждения транзакции в блокчейне.

Управление вашим регистрационным именем

Нажмите “Manage name” и прокрутите вниз до Addresses. Учтите, что адрес ETH установлен для кошелька, создавшего домен, в данном случае вашего кошелька(Рис. 9-1).

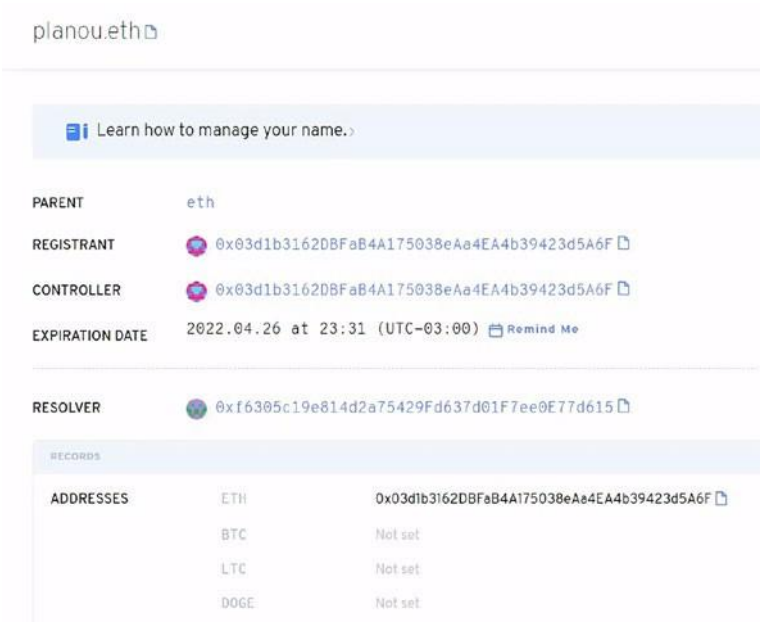

Рис. 9-1. *Страница регистрации домена ENS*

Проверка разрешения имени

Щелкните свой кошелек MetaMask и нажмите Send. Введите свое имя ENS (например, planou.eth). Обратите внимание на то, что имя преобразовано в адрес кошелька (Рис. 9-2).

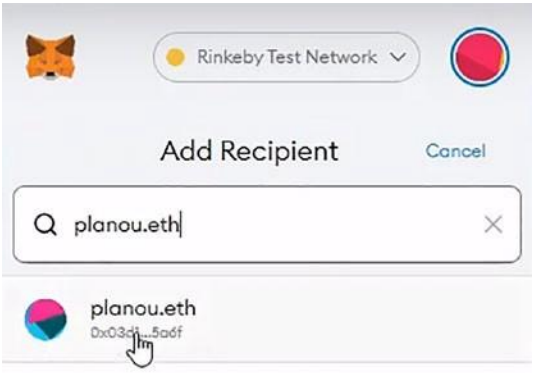

Рис. 9-2. *Разрешенное имя MetaMask ENS*

Теперь вы можете использовать для своих транзакций имя ENS в качестве получателя вместо хэш-адреса кошелька.

Резюме

В этой главе вы узнали, как просто можно настроить домен ENS, и узнали, как сразу начать его использовать.

В следующей главе вы начнете изучение того, как получать данные вне сети с помощью оракулов посредством Chainlink.

ГЛАВА 10

Chainlink

Chainlink (https://chain.link) — это децентрализованная сеть узлов, которая использует оракулы для передачи данных и информации из источников вне блокчейна в смарт-контракты на блокчейне. В этой главе вы узнаете, как использовать канал цен ETH/USD в тестовой сети Kovan для доступа к самой последней цене криптовалюты в смарт-контрактах.

В конце этой главы вы сможете сделать следующее:

- Создать простой смарт-контакт для получения цен
- Настроить проект Infura
- Сконфигурировать закрытый ключ для подписи транзакций
- Развернуть смарт-контракт в сети Kovan
- Получить информацию о ценах от смарт-контакта в сети Kovan

Получение цены на криптовалюту внутри смарт-контрактов, используя оракулы Chainlink

Давайте начнем с создания нового проекта, а затем установим пакет контрактов Chainlink. Вы будете использовать существующий адрес контракта, который сообщает вам цену пары ETH/USD, а затем вы сможете увидеть, что эта цена возвращается вашим смарт-контактом.

Создание проекта

Перейдите в меню Terminal, нажмите New Terminal и инициализируйте новый проект Truffle.

```
$ truffle init
```

Теперь инициализируйте папку проекта.

```
$ npm init
```

Наконец, установите пакет контрактов Chainlink (https://www.npmjs.com/package/@chainlink/contracts).

```
$ npm install @chainlink/contracts@0.1.9
```

Создание смарт-контракта

Создайте новый смарт-контракт для получения цены.

```
$ touch contracts/PriceConsumer.sol
```

Откроте файл PriceConsumer.sol (Рис. 10-1).

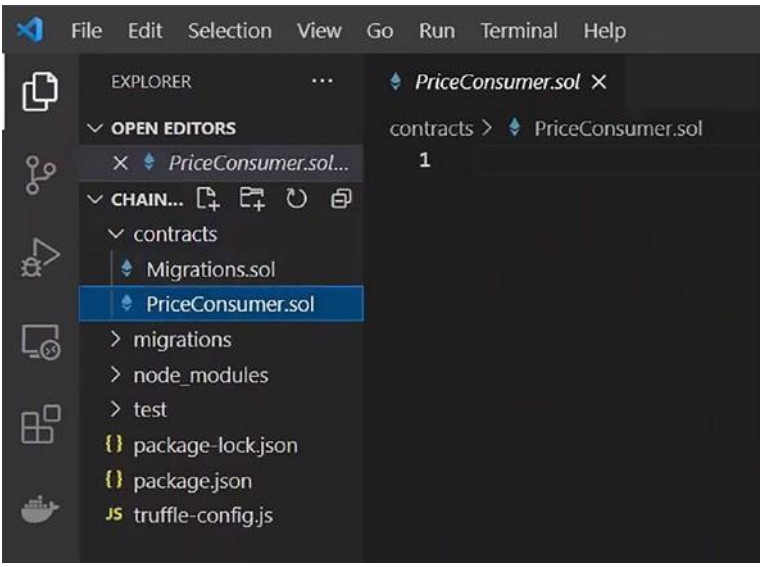

Рис. 10-1. *Папка контрактов VS Code*

В файле PriceConsumer.sol укажите версию Solidity, а затем импортируйте интерфейс контракта Chainlink. После этого определите наименование контракта и конструктор контракта.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@chainlink/contracts/src/v0.6/interfaces/
AggregatorV3Interface.sol";

contract PriceConsumer {

    AggregatorV3Interface internal priceFeed;

    constructor(){
        priceFeed = AggregatorV3Interface()
    }
}
```

Перейдите на https://docs.chain.link/docs/ethereum-addresses и прокрутите вниз до раздела Kovan. Скопируйте адрес прокси в строку "ETH/USD" (Рис. 10-2).

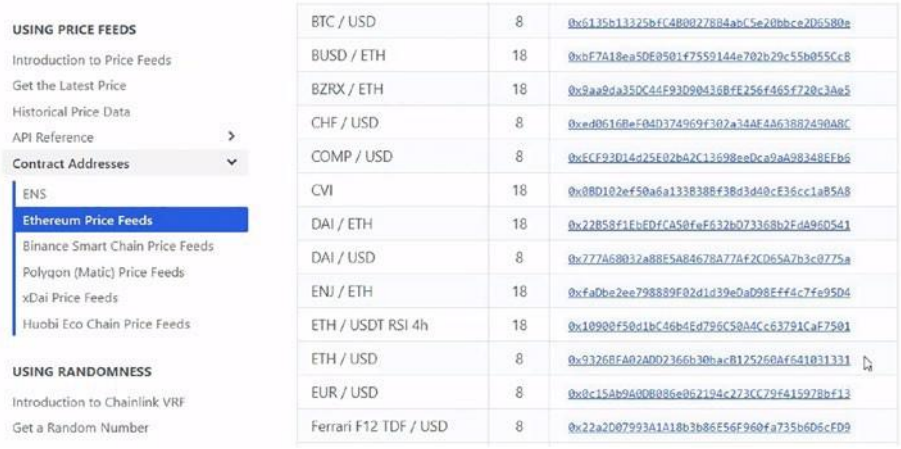

Рис. 10-2. *Каналы цен Chainlink*

Вставьте адрес в конструктор AggregatorV3Interface. После этого создайте функцию для получения цены.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@chainlink/contracts/src/v0.6/interfaces/
AggregatorV3Interface.sol";

contract PriceConsumer {

    AggregatorV3Interface internal priceFeed;
    constructor() {
        priceFeed = AggregatorV3Interface(0x9326BFA02ADD2366b30
                    bacB125260Af641031331);
    }
```

```
    function getThePrice() public view returns (int){
        (
            uint80 roundID,
            int price,
            uint startedAt,
            uint timeStamp,
            uint80 answeredInRound
        ) = priceFeed.latestRoundData();
        return price;
    }
}
```

Создание миграции

Создайте файл миграции в папке миграции.

```
$ touch migrations/2_deploy_contracts.sol
```

Напишите код для развертывания смарт-контракта PriceConsumer (Рис. 10-3).

```
const PriceConsumer = artifacts.require("PriceConsumer");

module.exports = function(deployer){
    deployer.deploy(PriceConsumer);
};
```

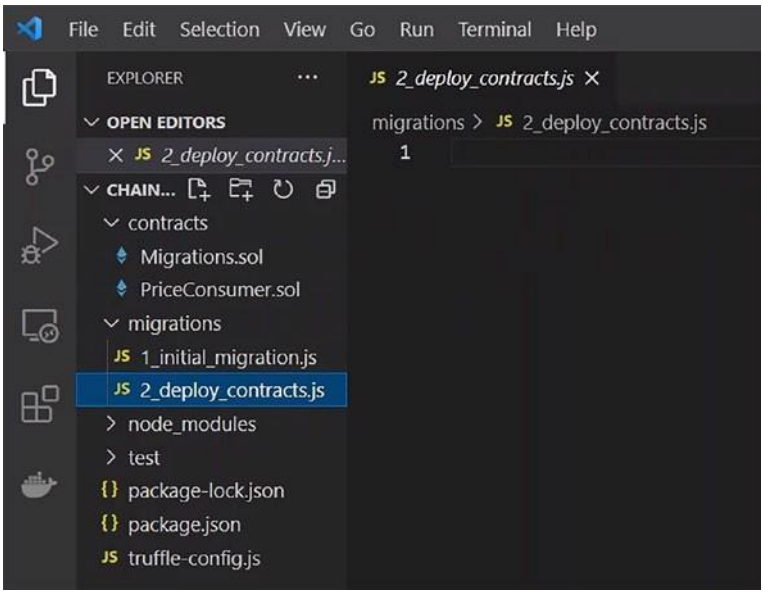

Рис. 10-3. *Папка миграции VS Code*

Настройка вашего проекта Infura

Перейдите на https://infura.io и получите доступ к своей панели инструментов. Нажмите Ethereum, а затем нажмите “Create a project”. Наконец, определите имя проекта и скопируйте идентификатор проекта (Рис. 10-4). Обратите внимание на то, что вы можете подключаться к различным тестовым сетям, а также к основной сети.

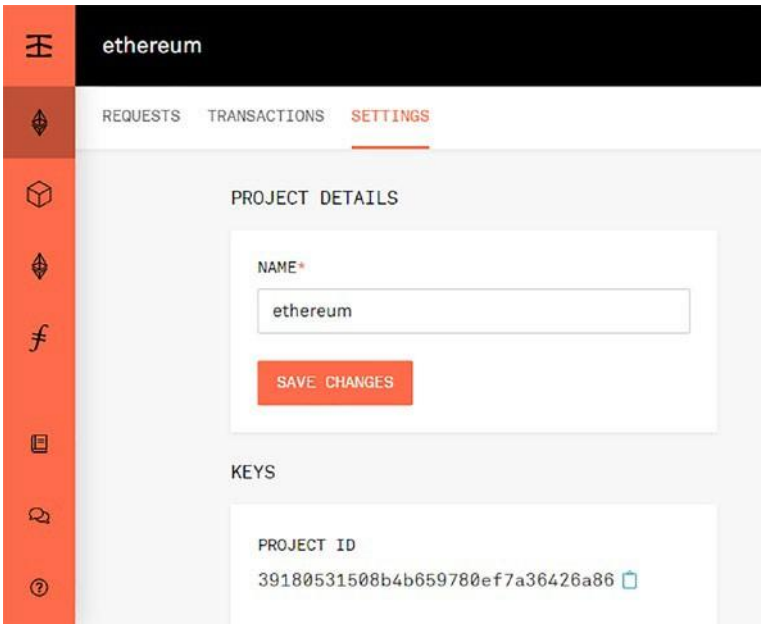

Рис. 10-4. *Настройки Infura*

Теперь нажмите Save Changes.

Настройка кошелька для подписи транзакций

Установите пакет файловой системы fs. Этот пакет предоставляет множество полезных функций для доступа и взаимодействия с файловой системой.

```
$ npm install fs
```

Установите пакет поставщика кошелька hdwallet. Это также ведет к установке провайдера Web3 с поддержкой HD-кошелька, который используется для подписи транзакций для адресов, полученных из мнемоники из 12 или 24 слов.

```
$ npm install @truffle/hdwallet-provider@1.2.3
```

Откройте файл truffle-config.js и раскомментируйте раздел кода HDWalletProvider.

```
const HDWalletProvider = require('@truffle/hdwallet-
provider');const infuraKey = '<your_infura_key>';

const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();
```

Вставьте идентификатор вашего проекта Infura в качестве значения переменной infuraKey.

Настройка сети

В truffle-config.js раскомментируйте раздел ropsten network и измените следующие значения:

- Измените ropsten на kovan.
- Измените Ropsten Infura URL на kovan.
- Изменит YOUR-PROJECT-ID на ${infuraKey}.
- Измените network_id на 42.

```
kovan: {
    provider: () => new
    HDWalletProvider(mnemonic, `
    https://kovan.infura.io/v3/${infuraKey}`),
    network_id: 42,
    gas: 5500000,
    confirmations: 2,
    timeoutBlocks:
    200, skipDryRun:
    true
},
```

Конфигурирование компилятора Solidity

Как прежде в truffle.config.js раскомментируйте раздел компиляторов и измените версию на 0.8.0.

```
compilers: {
    solc: {
        version: "0.8.0",
        docker: true,
        settings: {
            optimizer: {
                enabled: false,
                runs: 200
            },
            evmVersion: "byzantium"
        }
    }
},
```

Конфигурирование закрытого ключа

Создайте файл secret следующим образом:

```
$ touch .secret
```

Перейдите в браузер и откройте свой кошелек MetaMask, подключенный к сети Infura. Нажмите “Your Account”, а затем нажмите “Settings”. Наконец, нажмите “Security & Privacy” (Рис. 10-5).

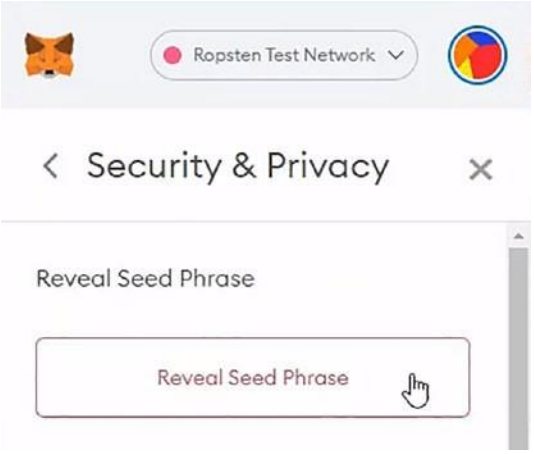

Рис. 10-5. *MetaMask: раскрываем сид-фразу*

У вас есть возможность просмотреть свою сид-фразу, но имейте в виду, что эта информация является конфиденциальной, и если кто-то получит к ней доступ, он сможет восстановить ваш кошелек и использовать ваши средства.

Нажмите Reveal Seed Phrase и введите пароль кошелька, чтобы продолжить. Скопируйте закрытый ключ. Вернитесь в Visual Studio Code и вставьте личную секретную фразу восстановления в файл .secret.

Составление смарт-контракта

Скомпилируйте контракт с помощью Truffle.

```
$ truffle compile
```

Развертывание смарт-контракта

Разверните контракт в сети Kovan с помощью Truffle. Команда migrate запускает миграцию для развертывания контрактов в сети Kovan.

```
$ truffle migrate --network kovan
```

Дождитесь развертывания контракта и подтверждения транзакций в блокчейне. Теперь проверьте адрес вашего контракта, который был создан (Рис. 10-6).

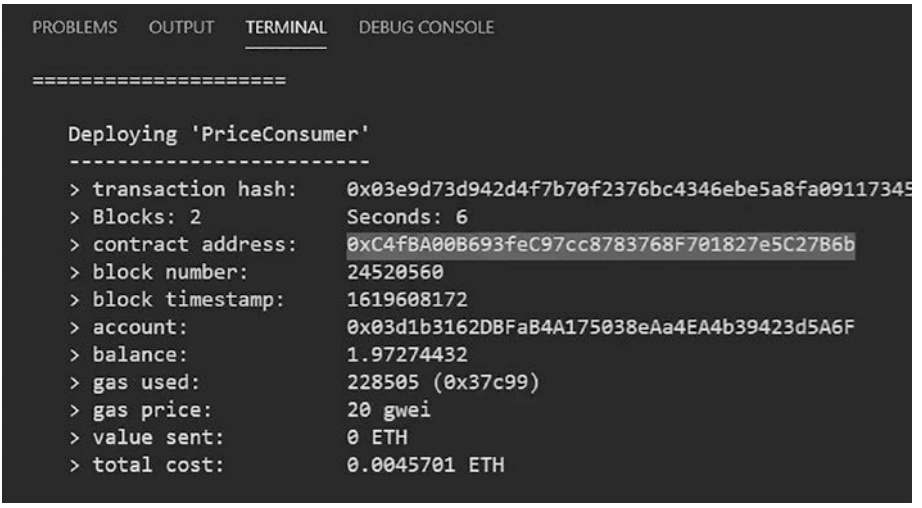

Рис. 10-6. *Вывод развернутого контракта VS Code*

Получение информации о цене из смарт-контракта

Создайте экземпляр контракта с помощью консоли Truffle. Эта консольная команда открывает базовую интерактивную консоль, которая подключается к клиенту Ethereum в сети Kovan:

```
$ truffle console --network kovan
```

Теперь используйте команду deployed, чтобы вернуть развернутый экземпляр контракта в сети Kovan, как показано здесь:

```
truffle(kovan) let instance = await PriceConsumer.deployed()
```

Вызовите метод getThePrice. Команда let сохраняет результат метода в переменной price, а команда await выполнит метод асинхронно.

```
truffle(kovan) let price = await instance.getThePrice()
```

Наконец, выведите результат в виде обычного числа. Метод toNumber() преобразует объекты больших чисел в обычные числа.

```
truffle(kovan)
price.toNumber()265499339990
```

Вот и все, вы только что создали смарт-контракт и воспользовались оракулом котировок Chainlink!

Резюме

В этой главе вы узнали, как создать простой смарт-контракт с использованием Chainlink для получения информации о ценах от оракула Chainlink.

В следующей главе вы узнаете о Nethereum, библиотеке .NET для Ethereum.

ГЛАВА 11

Nethereum

Nethereum (https://nethereum.com) — это библиотека интеграции .NET с открытым исходным кодом для Ethereum, которая упрощает обслуживание смарт-контрактов и взаимодействие с общедоступными и частными узлами Ethereum. Эта структура предоставляет класс Web3, где можно взаимодействовать с методами кошельков или смарт-контрактов. В примере, который вы увидите в этой главе, вы будете использовать метод GetBalance кошелька, чтобы узнать его баланс.

В конце этой главы вы сможете сделать следующее:

- Создать новый консольный проект с помощью dotnet
- Создать метод для получения баланса кошелька с помощью Nethereum
- Отобразить результат в консоли

Получение баланса эфира с помощью Nethereum

Начнем с создания нового консольного проекта и добавления пакета Nethereum Web3 в наше приложение. Затем вы создадите метод, который будет получать баланс с определенного адреса кошелька. Наконец, вы выведите эту информацию в консоли в wei и в эфире.

Создание проекта

Перейдите в окно терминала и нажмите New Terminal. Создайте новый проект консоли dotnet следующим образом. Эта команда создает новый проект, файл конфигурации или решение на основе указанного шаблона:

```
$ dotnet new console -o sample
```

Перейдите в корневой каталог проекта.

```
$ cd sample
```

Установка Web3

Установите пакет Nethereum Web3. Эта команда добавляет ссылку на пакет в файл проекта:

```
$ dotnet add package Nethereum.Web3
```

Восстановите все пакеты проекта. Эта команда восстанавливает зависимости и инструменты проекта:

```
$ dotnet restore
```

Создание метода

Откройте файл Program.cs и добавьте ссылку на потоки и Web3. Теперь добавьте новый метод для получения баланса учетной записи, а затем создайте экземпляр нового объекта Web3.

Перейдите в настройки вашего проекта Infura и выберите сеть Ropsten. Скопируйте конечную точку Ropsten https (Рис. 11-1).

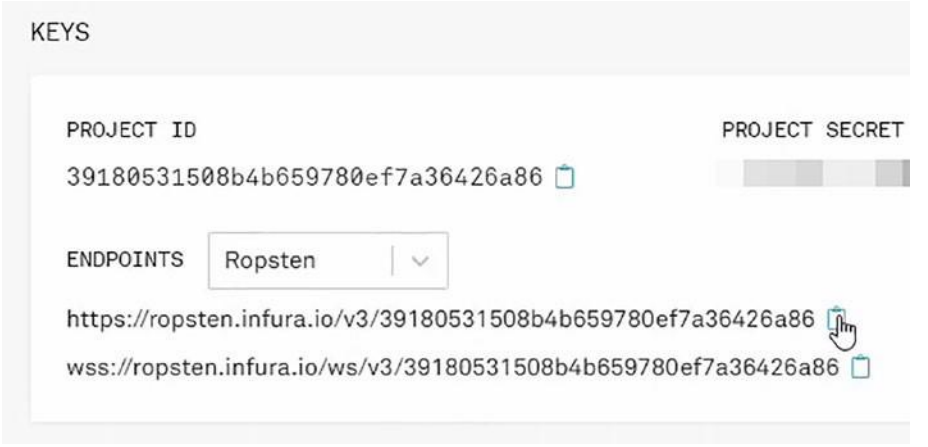

Рис. 11-1. *Ключи проекта Infura*

Используйте эту конечную точку в качестве параметра для конструктора объектов Web3. В файле Program.cs получите состояние баланса от Web3, используя публичный адрес вашего кошелька в качестве параметра. Напишите код для вывода баланса в wei, а затем конвертируйте баланс wei в эфир. Наконец, напишите код для вывода баланса в эфире. Теперь измените основной метод на вызов `GetAccountBalance()`.

```
using System;
using System.Threading.Tasks;
using Nethereum.Web3;

namespace NethereumSample
{
    class Program
    {
        static void Main(string[] args)
        {
            GetAccountBalance().Wait();
            Console.ReadLine();
        }
```

```
        static async Task GetAccountBalance()
        {
            var web3 = new Web3("https://ropsten.infura.io/
                       v3/39180531508b4b659780ef7a36426a86");
            var balance = await web3.Eth.GetBalance.SendRequest
                          Async("0x03d1b3162DBFaB4A175038eAa4EA
                          4b39423d5A6F");
            Console.WriteLine($"Balance in Wei: {balance.Value}");

            var etherAmount = Web3.Convert.FromWei(balance.Value);
            Console.WriteLine($"Balance in Ether: {etherAmount}");
        }
    }
}
```

Получение баланса

Создайте проект. Эта команда создаст проект и все его зависимости:

```
$ dotnet build
```

Запустите проект. Команда запускает исходный код без каких-либо явных команд компиляции или запуска:

```
$ dotnet run
```

Проверьте вывод в окне терминала и убедитесь, что вы получили результат, показанный на Рис. 11-2.

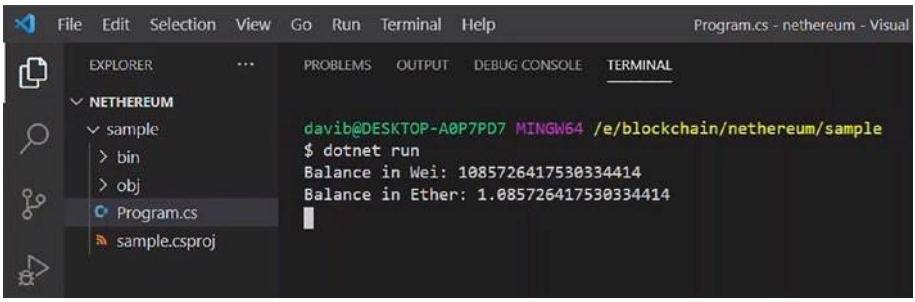

Рис. 11-2. *Баланс кода VS в wei и эфире*

Резюме

В этой главе вы узнали, как создать консольный проект, который получает баланс кошелька посредством Nethereum.

ГЛАВА 12

Заключение

В этой книге я в простой форме представил наиболее распространенные на сегодняшний день варианты использования блокчейна. Идея заключалась в том, чтобы не усложнять и сразу перейти к делу. Все примеры, представленные в этой книге, созданы для того, чтобы помочь вам быстро продвинуться по пути обучения. Как разработчик, я знаю, что лучший способ учиться — это программировать, поэтому я хотел поделиться с вами этим способом обучения.

Научиться быть разработчиком блокчейна сложно. Это область, которая постоянно меняется, и чтобы быть в курсе последних событий как профессионалу, вам потребуется много часов и много усилий. Не позволяйте этому подрывать вашу волю. Мы все начинаем с нуля и учимся новым навыкам маленькими шагами. Я надеюсь, что эта книга дала вам знания, необходимые для начала работы.

Обладая этими знаниями, вы теперь можете создать свою собственную криптовалюту или создать свой собственный NFT и выставить его на продажу в OpenSea, или, что еще лучше, вы можете разместить свой первый сайт в IPFS с собственным доменом в ENS. Вы также можете запрашивать данные вне сети с помощью оракулов Chainlink или использовать Filecoin для сохранения данных распределенным способом. Возможности безграничны.

Помните, что блокчейн набирает обороты, и он больше не ограничивается Биткойном и Ethereum. В будущем мы увидим, как много проектов развивается и как другие проекты теряют аудиторию. Но никогда не забывайте, что технологии существуют для решения реальных проблем. Чтобы применить то, что вы узнали из этой книги, подумайте сначала о том, какие проблемы вы пытаетесь решить и лучше ли их решать с помощью решений вне сети. В конце концов, блокчейн поможет вам решить проблемы, связанные с

доверием. Заслуживает ли доверия централизованный источник данных? Можно ли доверять централизованному приложению? Если нет, используйте блокчейн.

Я надеюсь, вам понравилось читать эту книгу; последнее, но не менее важное: никогда не прекращайте учиться. Надеюсь скоро увидеть вас в качестве разработчика блокчейна.

Вот несколько ссылок для дальнейшего чтения:

- OpenSea (https://opensea.io)
- OpenSea Testnets (https://testnets.opensea.io)
- Rinkeby Authenticated Faucet (https://faucet.rinkeby.io) (social validation)
- Rinkeby Ether Faucet (http://rinkeby-faucet.com) (no social validation)
- Matic Faucet (https://faucet.matic.network)
- Polygon Faucet (https://matic.supply)
- IPFS documentation (https://docs.ipfs.io)
- Filecoin documentation (https://docs.filecoin.io)
- Filecoin Lotus (https://docs.filecoin.io/get-started/lotus/)
- Truffle, Filecoin (https://www.trufflesuite.com/docs/filecoin/truffle/quickstart)
- Ethereum Name Service (https://ens.domains/)
- Chainlink documentation (https://docs.chain.link)
- Nethereum documentation (http://docs.nethereum.com/en/latest/)