Published in Coding The Smart Way

Sebastian Follow

Mar 31 · 3 min read · · Listen

Learn Full Stack Ethereum Development — Part 2

Default Project Setup — What's included?

Welcome to the second part of the *Learn Full Stack Ethereum Development* series. This series will provide you with a beginner's tutorial to learn full stack Ethereum development from scratch. In last part we've been setting up the React project structure and added the Hardhat development environment to the project. In this part we're going to take a closer look at the default project setup and learn how a smart contract is implemented and deployed to Hardhat's local Ethereum blockchain. Let's get started

This is the fo iles. A default Smart Contract implementation is provided in file *Greeter.sol*. From the file extension (**.sol*) you can see that the programming language which is used for implementation is Solidity. If you take a look into the file you'll see the following default implementation:

```
1  //SPDX-License-Identifier: Unlicense
2  pragma solidity ^0.8.0;
3
4  import "hardhat/console.sol";
5
6  contract Greeter {
7      string private greeting;
8
9      constructor(string memory _greeting) {
10         console.log("Deploying a Greeter with greeting:", _greeting);
11         greeting = _greeting;
12     }
13
14     function greet() public view returns (string memory) {
15         return greeting;
16     }
17
18     function setGreeting(string memory _greeting) public {
19         console.log("Changing greeting from '%s' to '%s'", greeting, _greeting);
20         greeting = _greeting;
21     }
22 }
```

Greeter.sol hosted with ❤ by GitHub view raw

The first line of code is used to set the minimum Solidity version which must be used when compiling this Smart contract implementation by using the `pragma` keyword. In the example a minimum Solidity version of 0.8.0 is needed by the compiler.

In the next line you can find the import statement for package *hardhat/console.sol* which enables to output messages directly to the console.

- A constr
- The *greet* function
- The *setGreeting* function

This sample Smart Contract is used to store and manage a simple greeting string. The greeting message is set by the constructor to an initial value. The function *greet* can be used to retrieve the greeting message and the function *setGreeting* is used to set a new greeting message. We'll see the Smart Contract in action later on.

Default Hardhat Deployment Script

In the scripts folder you can find a sample deployment script which is used to deploy the Smart Contract to the blockchain. The file *sample-script.js* contains the following deployment code by default:

The deployment code for the *Greeter* Smart Contract can be found within the *main* function. Rename the file *sample-script.js* to *deploy.js*.

Hardhat Configuration File

The file *hardhat.config.js* in the root project folder contains the default configuration settings for Hardhat:


For this tutorial we'll be using the Hardhat local Ethereum blockchain, so we need to add the following *networks* entry to the *module.exports* section of the configuration file:

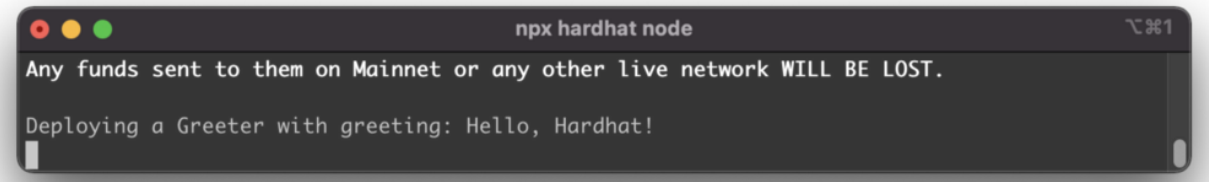
Deploying The Greeter Smart Contract To The Blockchain

Now it's time to use the deployment script (which we have renamed to deploy.js) and deploy the Greeter smart contract to the local Ethereum blockchain. Therefore we'll use the following command:

```
$ npx hardhat run scripts/deploy.js --network localhost
```


```
$ npx hardhat node
```

After successful deployment of the Greeter Smart Contract you should then be able to see the following output on the console which is directly coming from Greeter's constructor:



The output confirms that Greeter Smart Contract has been deployed successfully

What's Next?

In this part of the *Learn Full Stack Ethereum Development* tutorial series you've gained a deeper understanding of the default Hardhat project structure and you've learnt how the default Greeter Smart Contract implementation can be deployed to Hardhat's local blockchain.

In the next part we'll move on and learn how to install and use the Metamask wallet to connect to the Hardhat local blockchain and make use of some of the test accounts which are provided by Hardhat.