Hardhat Introduction

Requirements

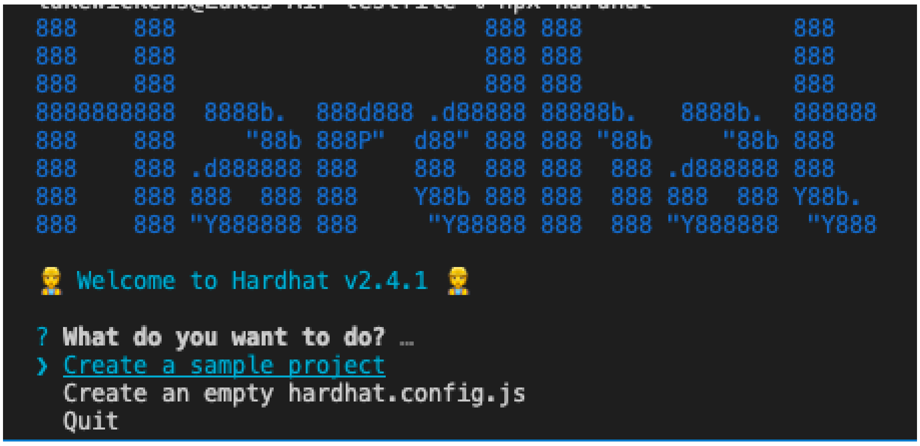
Node.js version 12 and above.

Installation

Steps for installing and using Hardhat:

1. `$ npm install -D hardhat`
2. `$ npx hardhat`

You should then see this in the terminal:

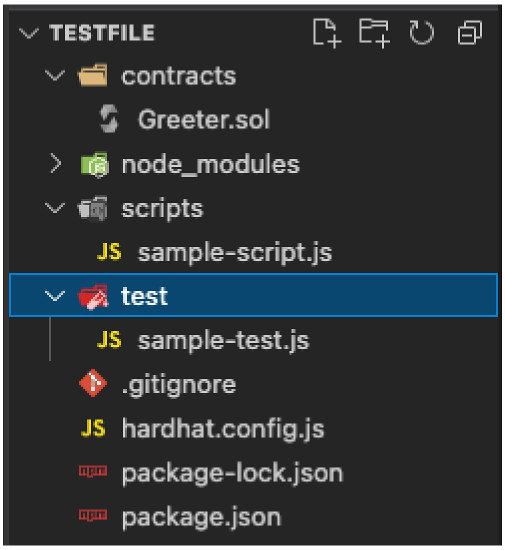


```
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

👷 Welcome to Hardhat v2.4.1 👷

? What do you want to do? …
❯ Create a sample project
  Create an empty hardhat.config.js
  Quit
```

(https://imgur.com/oqkL1rt)

Choosing *Create a sample project* will:

- Create a contracts folder with a dummy contract *‘Greeter.sol’*
- Create a test folder with a sample test for the dummy contract.
- Create a scripts folder that contains the deployment script for the dummy contract.
- Installs node modules:
- **@nomiclabs/hardhat-ethers**
- **@nomiclabs/hardhat-waffle**
- And other necessary packages



(https://imgur.com/OmR3o8v)

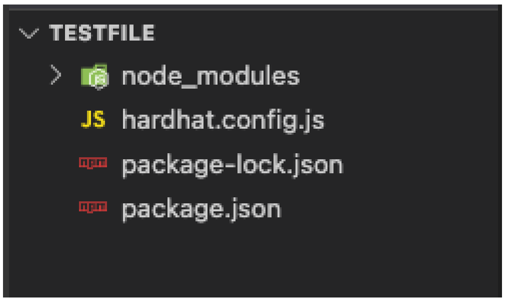
Choosing *Create and empty hardhat.config.js* will set up a new hardhat configuration file but without all of the dummy contracts, script files and tests. This method will not install any plugin (**ethers.js / Waffle**). If you want to use the **Web3.js / Truffle** plugin, this would be the options you would choose. If either of these sets of plugins are installed, you have to either put

```
require(‘@nomiclabs/hardhat-waffle’)
```

or

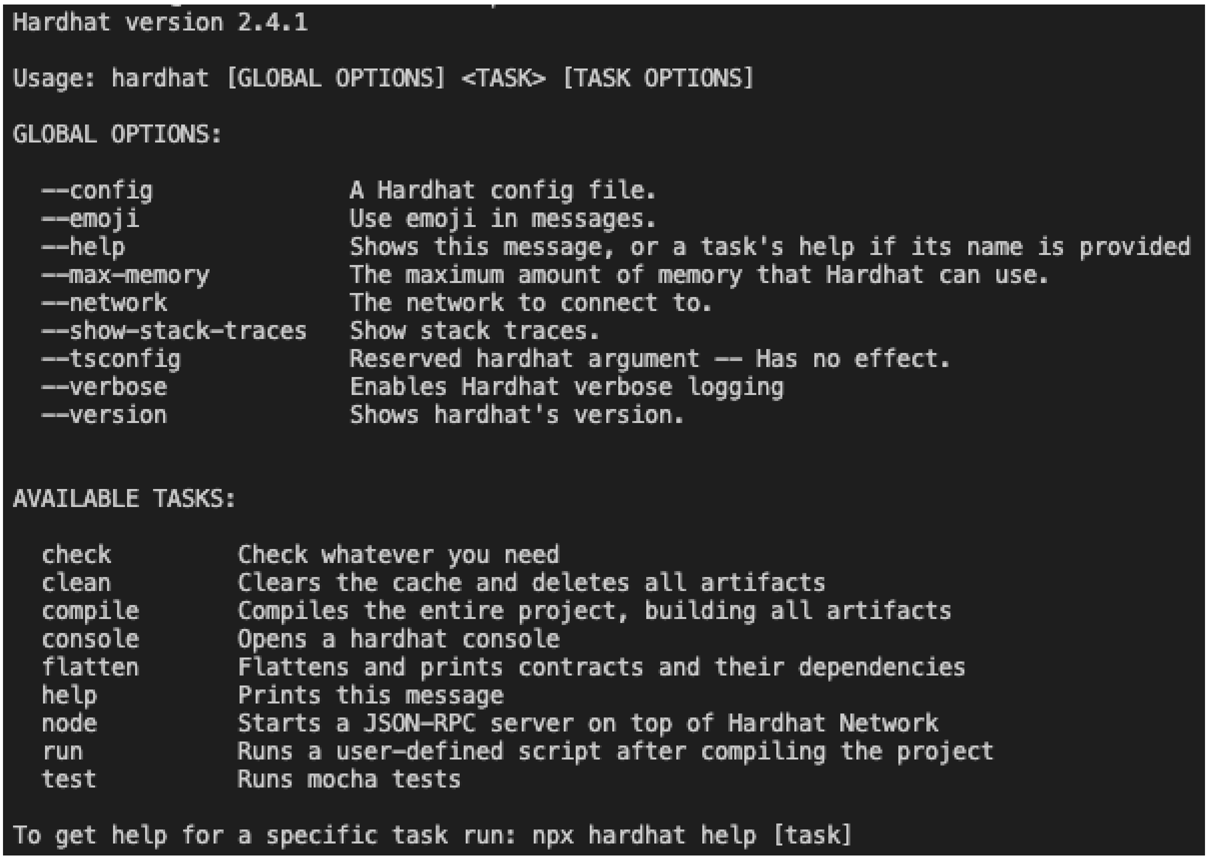
```
require(‘@nomiclabs/hardhat-truffle5’)
```

depending on the one that you are using.



(https://imgur.com/a7ktiBW)

After initializing hardhat , using `npx hardhat` again shows a list of commands:

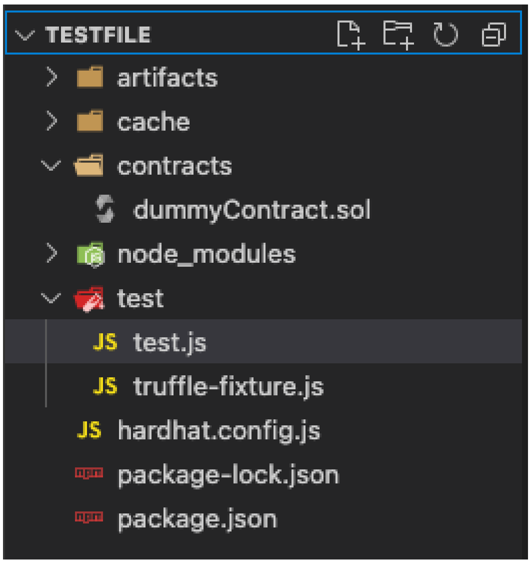


```
Hardhat version 2.4.1

Usage: hardhat [GLOBAL OPTIONS] <TASK> [TASK OPTIONS]

GLOBAL OPTIONS:

  --config              A Hardhat config file.
  --emoji               Use emoji in messages.
  --help                Shows this message, or a task's help if its name is provided
  --max-memory          The maximum amount of memory that Hardhat can use.
  --network             The network to connect to.
  --show-stack-traces   Show stack traces.
  --tsconfig            Reserved hardhat argument -- Has no effect.
  --verbose             Enables Hardhat verbose logging
  --version             Shows hardhat's version.


AVAILABLE TASKS:

  check         Check whatever you need
  clean         Clears the cache and deletes all artifacts
  compile       Compiles the entire project, building all artifacts
  console       Opens a hardhat console
  flatten       Flattens and prints contracts and their dependencies
  help          Prints this message
  node          Starts a JSON-RPC server on top of Hardhat Network
  run           Runs a user-defined script after compiling the project
  test          Runs mocha tests

To get help for a specific task run: npx hardhat help [task]
```

(https://imgur.com/l6yZLGF)

Example: Truffle & Web3

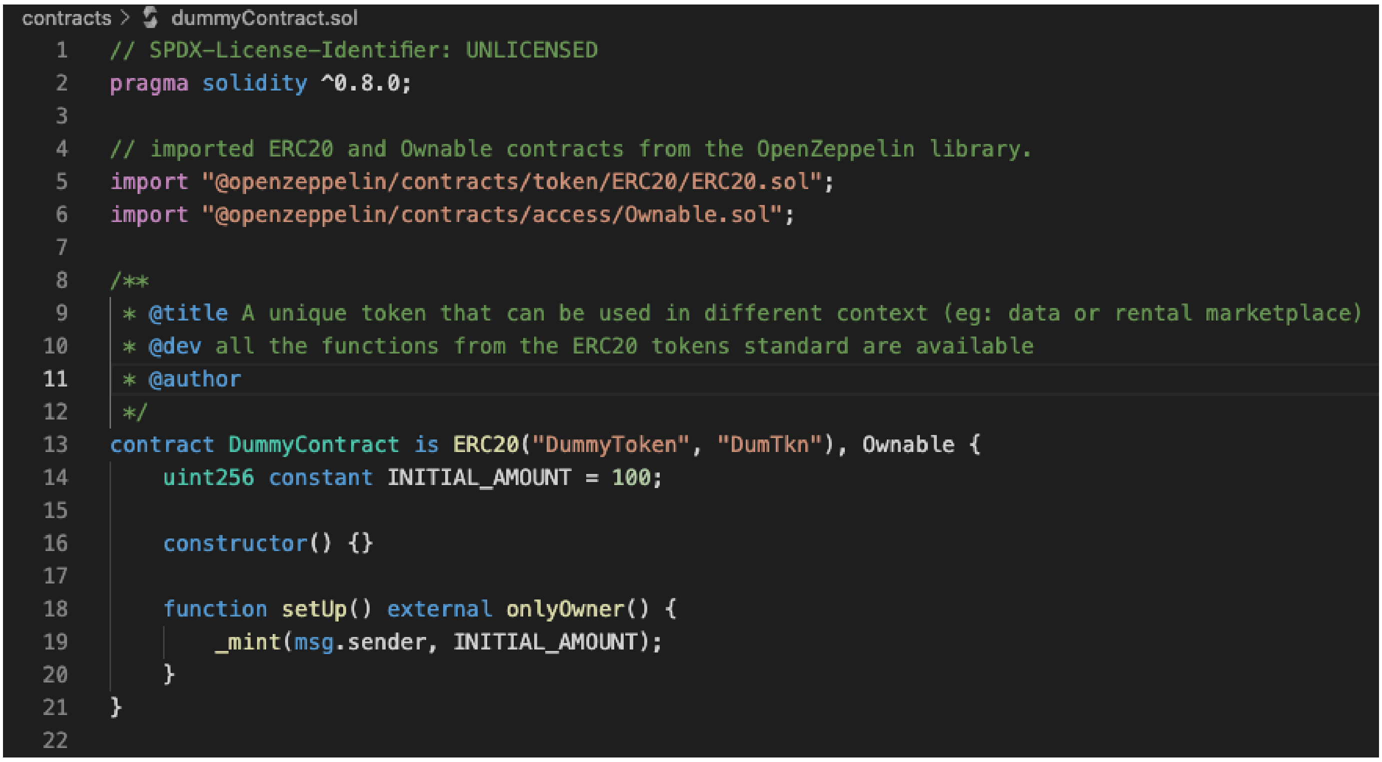
Set up a new project in which we have a contract (*dummyContract.sol*), a test file (*test.js*) and a deployment script (*truffle-fixture.js*). The file structure can be seen below (artifacts and cache folders are created by Hardhat when you test the contract so these will not be there until you run `$ npx hardhat test` for the first time).



```
TESTFILE
  artifacts
  cache
  contracts
    dummyContract.sol
  node_modules
  test
    test.js
    truffle-fixture.js
  hardhat.config.js
  package-lock.json
  package.json
```

(https://imgur.com/p5Chwqo)

dummyContract.sol

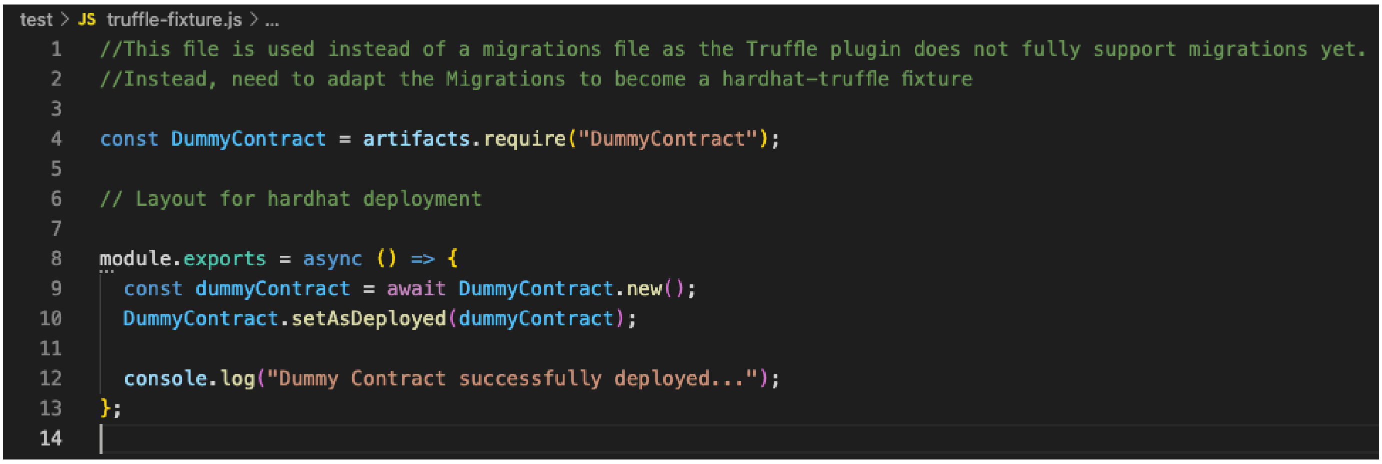


```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

// imported ERC20 and Ownable contracts from the OpenZeppelin library.
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/**
 * @title A unique token that can be used in different context (eg: data or rental marketplace)
 * @dev all the functions from the ERC20 tokens standard are available
 * @author
 */
contract DummyContract is ERC20("DummyToken", "DumTkn"), Ownable {
    uint256 constant INITIAL_AMOUNT = 100;

    constructor() {}

    function setUp() external onlyOwner() {
        _mint(msg.sender, INITIAL_AMOUNT);
    }
}

```

(https://imgur.com/BEVSWMR)

Its a very basic contract that creates a new ERC20 coin (DummyToken). The only function it has is a setUp function that calls the mint function in the *ERC20 contract* that has been imported from the *OpenZeppelin library*, and mint the amount specified in the **INITIAL_AMOUNT** variable to the user that called the function. As we are utilising the *Ownable contract* for this function it will only allow the owner of the contract to call this function and therefore will only mint the initial amount to the owner's wallet address.

truffle-fixture.js

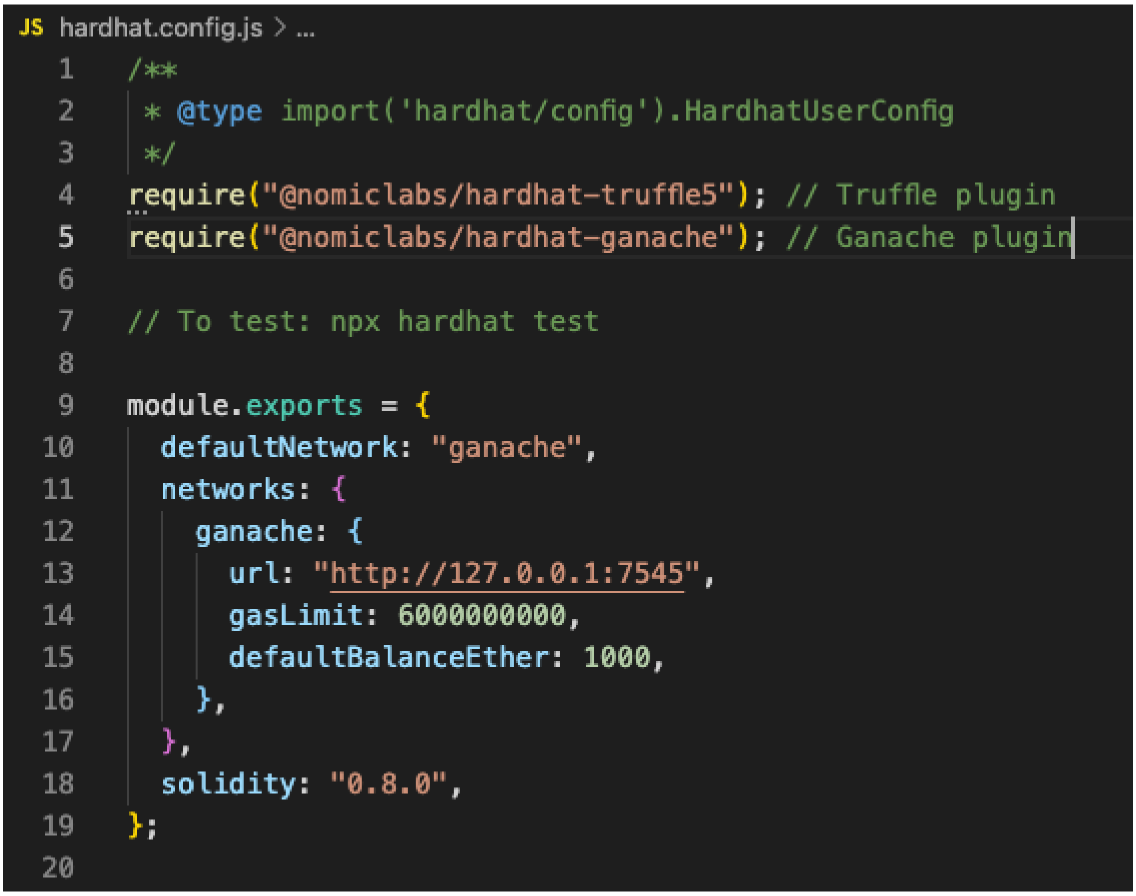


```
//This file is used instead of a migrations file as the Truffle plugin does not fully support migrations yet.
//Instead, need to adapt the Migrations to become a hardhat-truffle fixture

const DummyContract = artifacts.require("DummyContract");

// Layout for hardhat deployment

module.exports = async () => {
  const dummyContract = await DummyContract.new();
  DummyContract.setAsDeployed(dummyContract);

  console.log("Dummy Contract successfully deployed...");
};

```

(https://imgur.com/Q7qabV3)

As can be seen from the comments at the top of this file, using Hardhat is slightly different than Truffle when it comes to deployment. Normally there is a migrations file that would contain the deployment script for the contracts. However, **this feature has not yet been implemented in Hardhat**, so a truffle-fixture file has to be used instead. This requires in the contract(s) that you need to deploy, then creates a new version of that contract(s) and then sets it as deployed.

hardhat-config.js

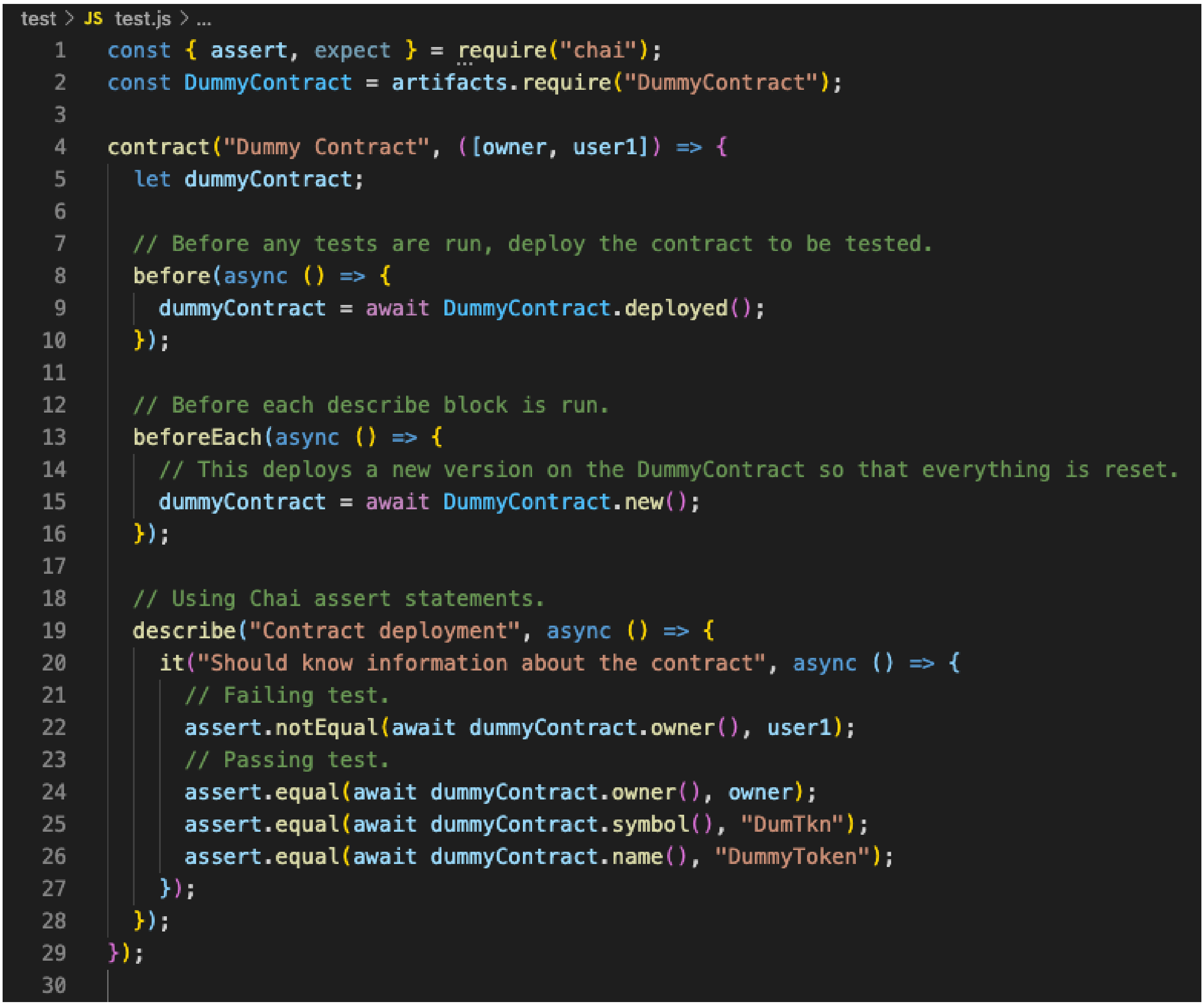


```js
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
require("@nomiclabs/hardhat-truffle5"); // Truffle plugin
require("@nomiclabs/hardhat-ganache"); // Ganache plugin

// To test: npx hardhat test

module.exports = {
  defaultNetwork: "ganache",
  networks: {
    ganache: {
      url: "http://127.0.0.1:7545",
      gasLimit: 6000000000,
      defaultBalanceEther: 1000,
    },
  },
  solidity: "0.8.0",
};
```

(https://imgur.com/IC1JBF4)

The hardhat-config.js file is where you require the **Truffle/Waffle** plugins. **There is no need to require Web3/Ethers as Truffle/Waffle already do this**. In the example above, the Ganache plugin has also been required as this will spin up a local blockchain on the specified URL. You can also specify other parameters such as the default ETH balance of each user. This plugin automatically starts Ganache before running your tests and stops it after.
You can also achieve the same thing by starting Ganache manually and running the command `$ npx hardhat --network localhost test`

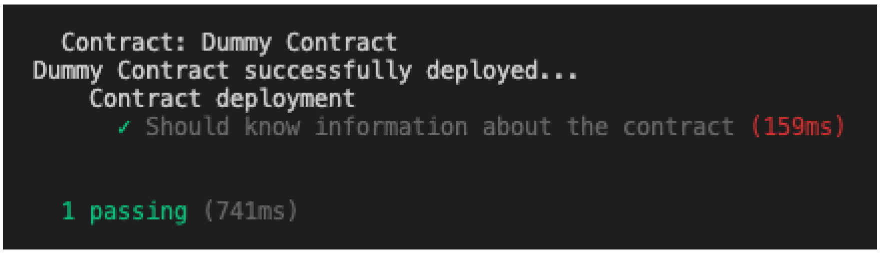
test.js



```
const { assert, expect } = require("chai");
const DummyContract = artifacts.require("DummyContract");

contract("Dummy Contract", ([owner, user1]) => {
  let dummyContract;

  // Before any tests are run, deploy the contract to be tested.
  before(async () => {
    dummyContract = await DummyContract.deployed();
  });

  // Before each describe block is run.
  beforeEach(async () => {
    // This deploys a new version on the DummyContract so that everything is reset.
    dummyContract = await DummyContract.new();
  });

  // Using Chai assert statements.
  describe("Contract deployment", async () => {
    it("Should know information about the contract", async () => {
      // Failing test.
      assert.notEqual(await dummyContract.owner(), user1);
      // Passing test.
      assert.equal(await dummyContract.owner(), owner);
      assert.equal(await dummyContract.symbol(), "DumTkn");
      assert.equal(await dummyContract.name(), "DummyToken");
    });
  });
});

```

(https://imgur.com/GaCR1VZ)

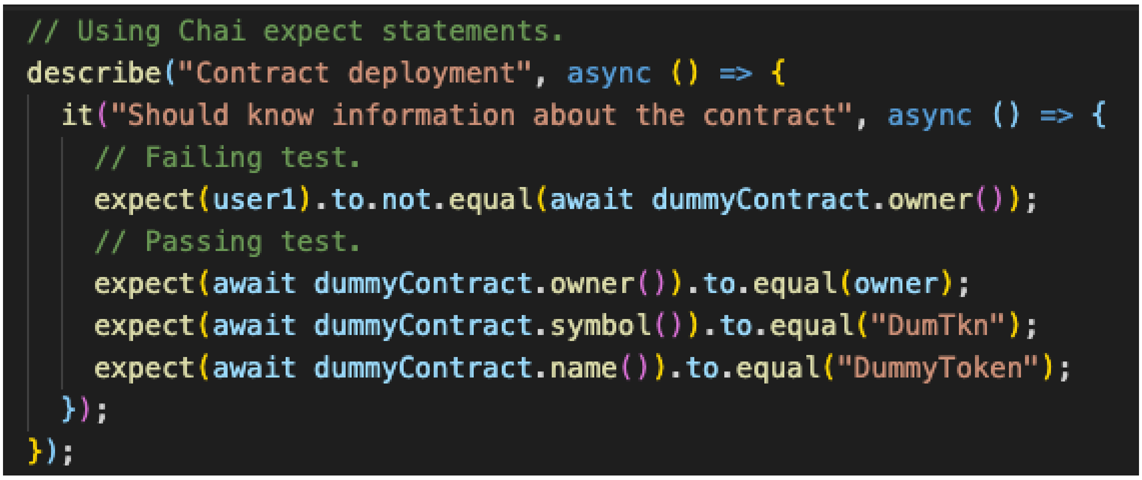
This version of the tests are using Chai assert. The way you write tests in Hardhat is almost identical to Truffle.
To run the tests use command `$ npx hardhat test`
If you want to run a specific test file, specify the name after the above command (including the path) `$ npx hardhat test test/test.js` .



```
  Contract: Dummy Contract
Dummy Contract successfully deployed...
    Contract deployment
      ✓ Should know information about the contract (159ms)


  1 passing (741ms)
```

(https://imgur.com/Zd15eAh)

Using Chai expect the tests would be written like this:

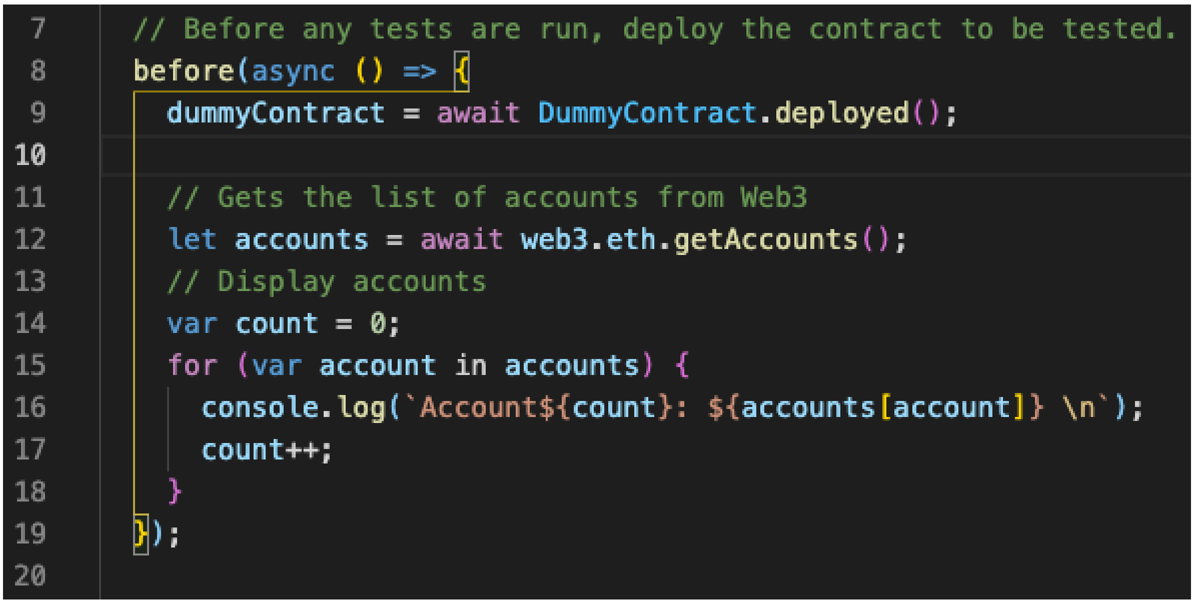


```javascript
// Using Chai expect statements.
describe("Contract deployment", async () => {
  it("Should know information about the contract", async () => {
    // Failing test.
    expect(user1).to.not.equal(await dummyContract.owner());
    // Passing test.
    expect(await dummyContract.owner()).to.equal(owner);
    expect(await dummyContract.symbol()).to.equal("DumTkn");
    expect(await dummyContract.name()).to.equal("DummyToken");
  });
});
```

(https://imgur.com/Ho6Sg7X)

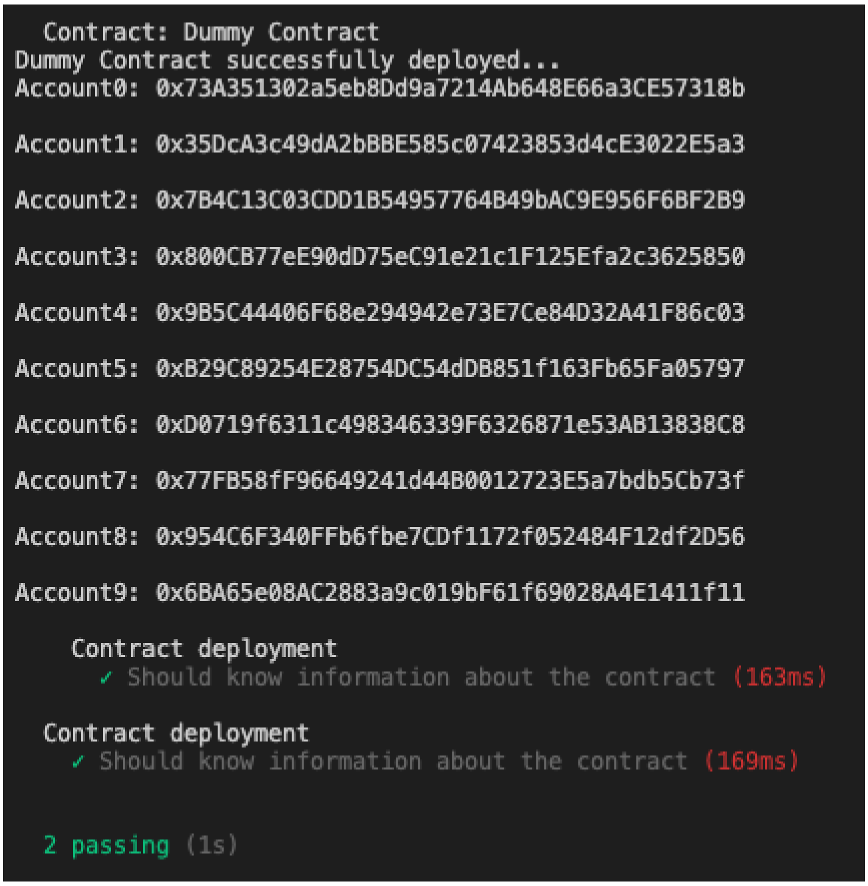
Web3.js

Web3 is automatically used when the Truffle plugin has been required in the hardhat-config.js file. Web3.js is available in the global scope. In the below example, I have added extra code into the before block that gets a list of accounts from web3 and then displays them in the console. These accounts will be different every time as a new blockchain instance is used on each test run.



```javascript
// Before any tests are run, deploy the contract to be tested.
before(async () => {
  dummyContract = await DummyContract.deployed();

  // Gets the list of accounts from Web3
  let accounts = await web3.eth.getAccounts();
  // Display accounts
  var count = 0;
  for (var account in accounts) {
    console.log(`Account${count}: ${accounts[account]} \n`);
    count++;
  }
});

```

(https://imgur.com/N6943X5)

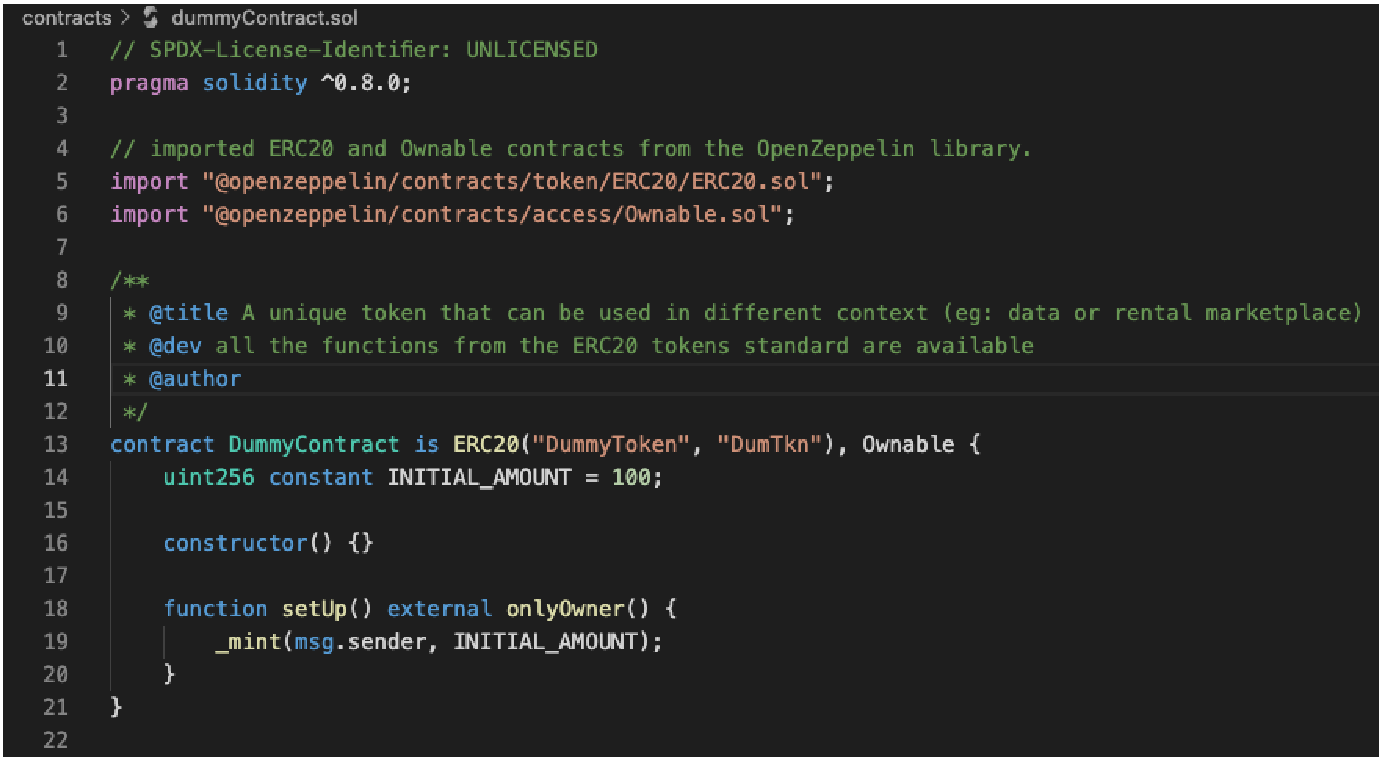


```
  Contract: Dummy Contract
Dummy Contract successfully deployed...
Account0: 0x73A351302a5eb8Dd9a7214Ab648E66a3CE57318b

Account1: 0x35DcA3c49dA2bBBE585c07423853d4cE3022E5a3

Account2: 0x7B4C13C03CDD1B54957764B49bAC9E956F6BF2B9

Account3: 0x800CB77eE90dD75eC91e21c1F125Efa2c3625850

Account4: 0x9B5C44406F68e294942e73E7Ce84D32A41F86c03

Account5: 0xB29C89254E28754DC54dDB851f163Fb65Fa05797

Account6: 0xD0719f6311c498346339F6326871e53AB13838C8

Account7: 0x77FB58fF96649241d44B0012723E5a7bdb5Cb73f

Account8: 0x954C6F340FFb6fbe7CDf1172f052484F12df2D56

Account9: 0x6BA65e08AC2883a9c019bF61f69028A4E1411f11

    Contract deployment
      ✓ Should know information about the contract (163ms)

  Contract deployment
    ✓ Should know information about the contract (169ms)


  2 passing (1s)
```

(https://imgur.com/57QDQYV)

Example: Waffle & Ethers

dummyContract.sol



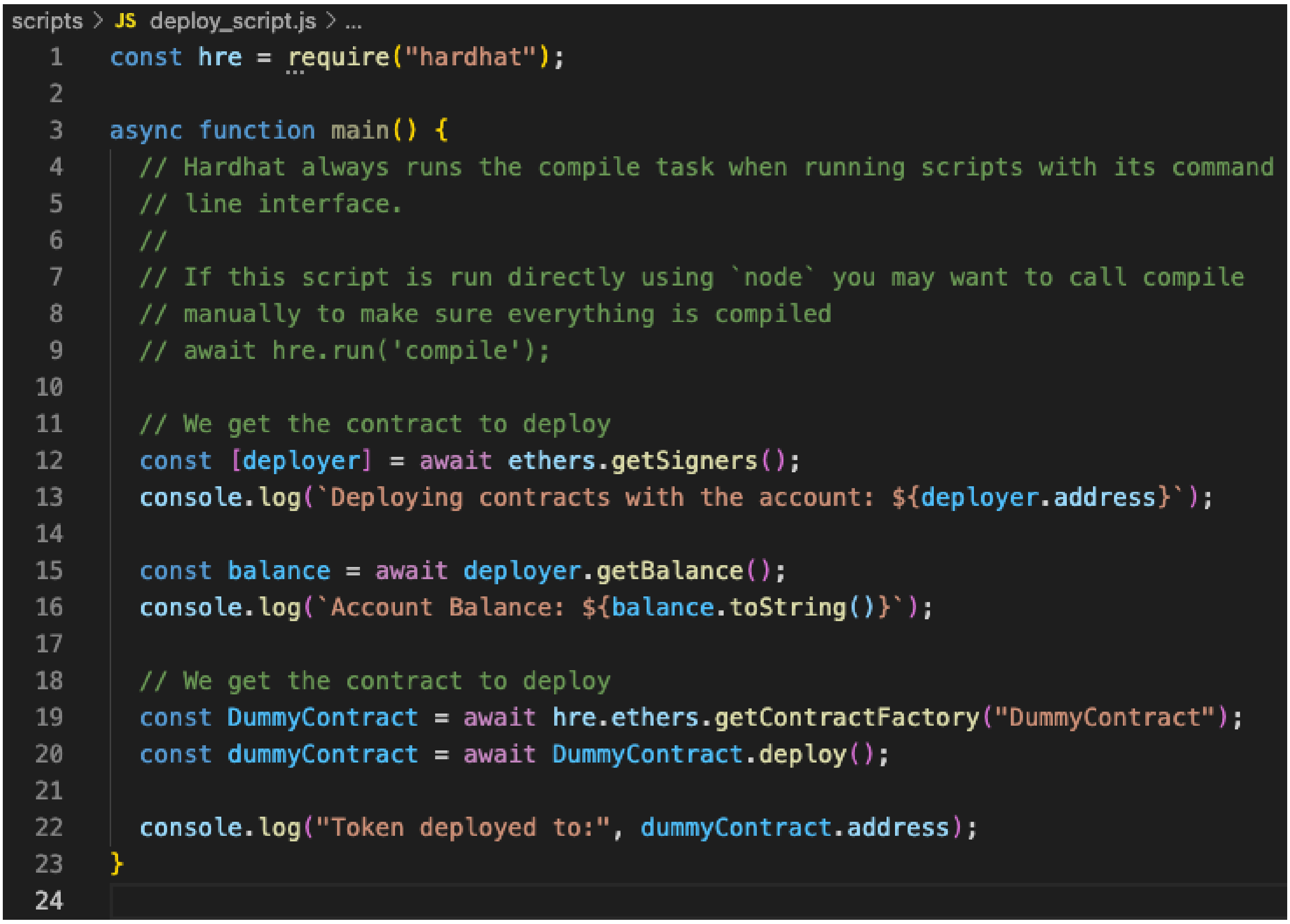
contracts > dummyContract.sol

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

// imported ERC20 and Ownable contracts from the OpenZeppelin library.
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/**
 * @title A unique token that can be used in different context (eg: data or rental marketplace)
 * @dev all the functions from the ERC20 tokens standard are available
 * @author
 */
contract DummyContract is ERC20("DummyToken", "DumTkn"), Ownable {
    uint256 constant INITIAL_AMOUNT = 100;

    constructor() {}

    function setUp() external onlyOwner() {
        _mint(msg.sender, INITIAL_AMOUNT);
    }
}

```

(https://imgur.com/BEVSWMR)

We are going to use the same contract as used in the Truffle/Web3 example.

deploy_script.js



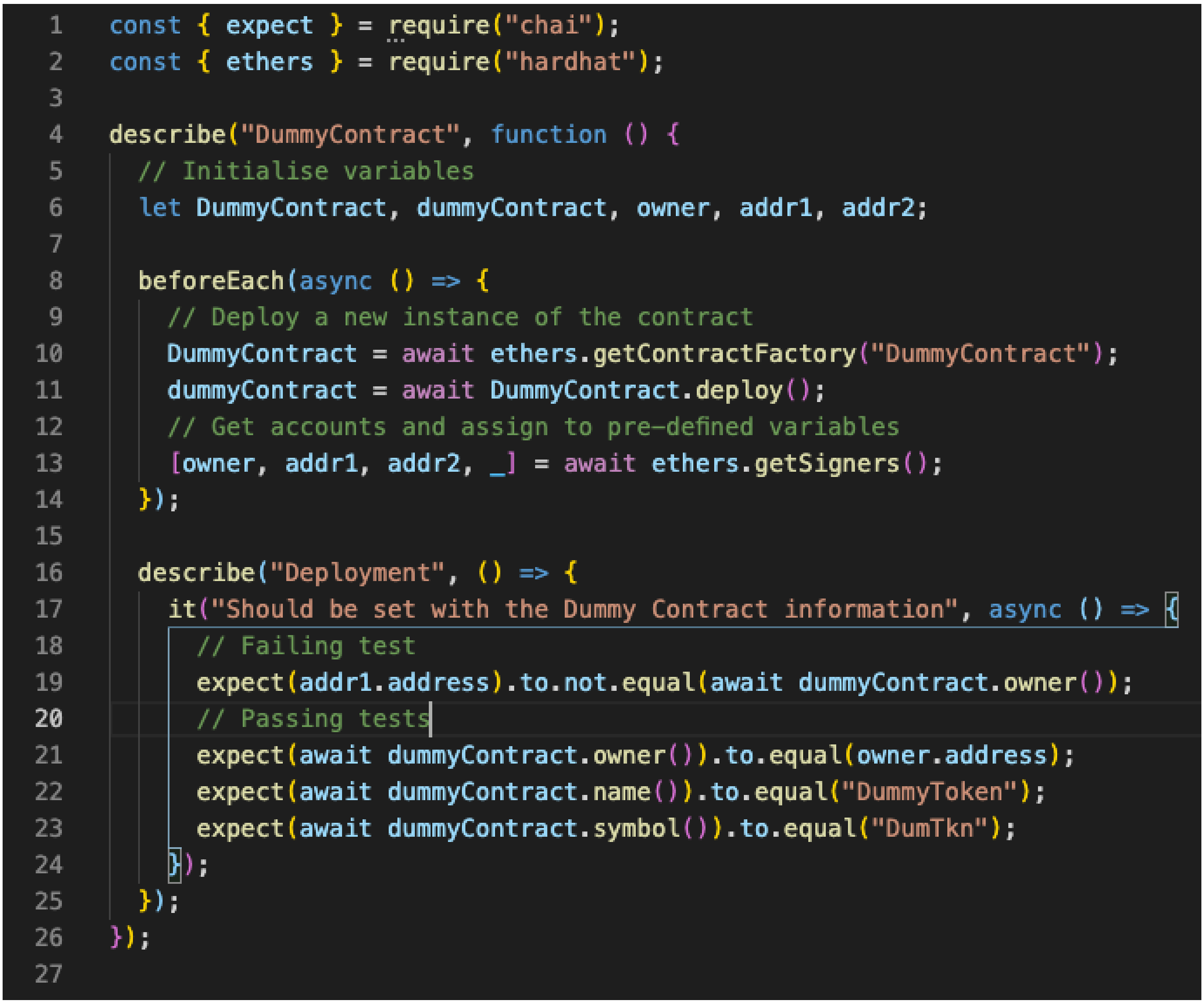
```
const hre = require("hardhat");

async function main() {
  // Hardhat always runs the compile task when running scripts with its command
  // line interface.
  //
  // If this script is run directly using `node` you may want to call compile
  // manually to make sure everything is compiled
  // await hre.run('compile');

  // We get the contract to deploy
  const [deployer] = await ethers.getSigners();
  console.log(`Deploying contracts with the account: ${deployer.address}`);

  const balance = await deployer.getBalance();
  console.log(`Account Balance: ${balance.toString()}`);

  // We get the contract to deploy
  const DummyContract = await hre.ethers.getContractFactory("DummyContract");
  const dummyContract = await DummyContract.deploy();

  console.log("Token deployed to:", dummyContract.address);
}

```

(https://imgur.com/FkjiXJf)

The file does the same thing as the *truffle_fixture.js* file in the Truffle/Web3 example. Essentially, it obtains the deployer information using **ethers.getSigners()** (which uses the first account in the list of generated accounts), then gets the balance of the deployer's account and returns this information in the terminal. The contract is then deployed and the address is returned in the terminal.

The function **getContractFactory()**, which is called on ethers is an abstraction used to deploy new smart contracts. It is a special type of transaction called an initcode transaction. **The contract bytecode is sent in the transaction, then It evaluates the code and allows you to create a new contract based upon that information**. So in the example above, the DummyContract variable that the contract information is assigned to is sent through as the initcode . This then allows you to deploy an instance of that contract.

test.js

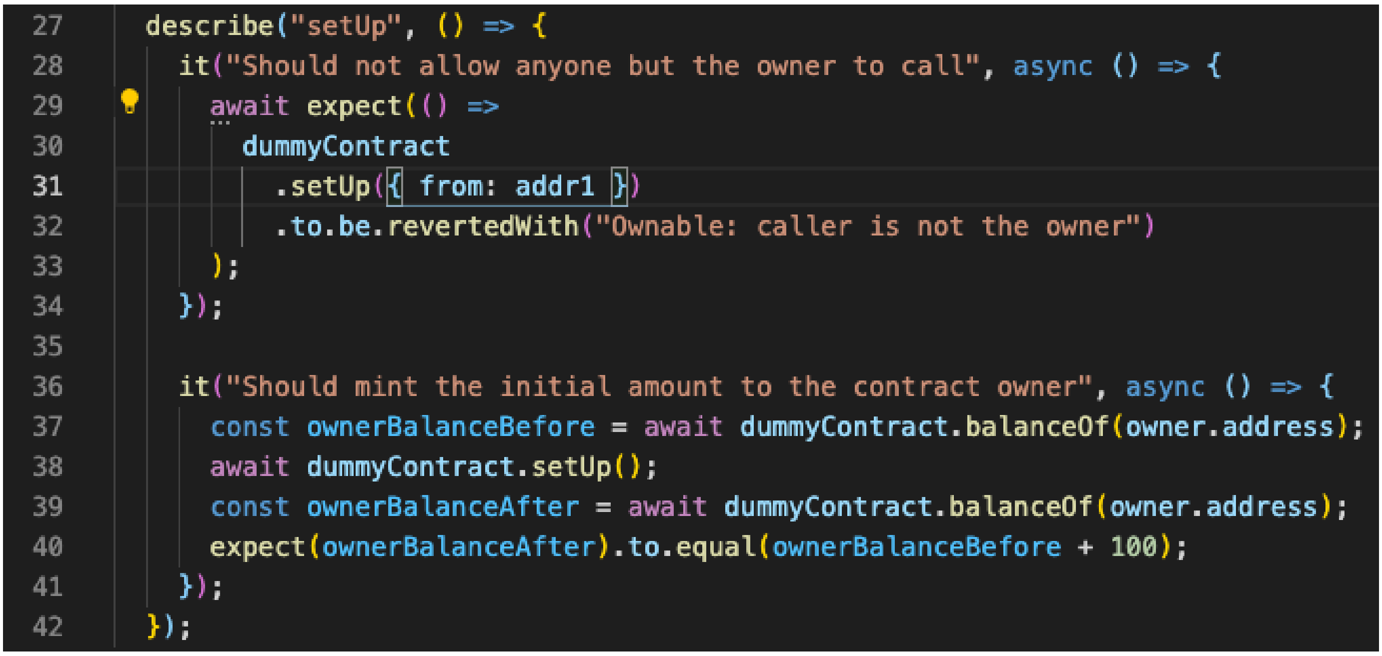


```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("DummyContract", function () {
  // Initialise variables
  let DummyContract, dummyContract, owner, addr1, addr2;

  beforeEach(async () => {
    // Deploy a new instance of the contract
    DummyContract = await ethers.getContractFactory("DummyContract");
    dummyContract = await DummyContract.deploy();
    // Get accounts and assign to pre-defined variables
    [owner, addr1, addr2, _] = await ethers.getSigners();
  });

  describe("Deployment", () => {
    it("Should be set with the Dummy Contract information", async () => {
      // Failing test
      expect(addr1.address).to.not.equal(await dummyContract.owner());
      // Passing tests
      expect(await dummyContract.owner()).to.equal(owner.address);
      expect(await dummyContract.name()).to.equal("DummyToken");
      expect(await dummyContract.symbol()).to.equal("DumTkn");
    });
  });
});

```

(https://imgur.com/N0E9tXW)

As you can see there is not much different between this test.js and the one used in the Truffle/Web3 project. The main difference comes in the **beforeEach** block. As explained in the deploy-script.js file, the **getContractFactory()** function allows the contract to be deployed, which is an extra step compared with Truffle and then the contract is deployed and assigned to the dummyContract variable. Then **getSigners()** is called which attributes account information (*public key, private key*) to each variable (*owner, addr1, addr2*). So the first address will be assigned to the owner, the second address to addr1 and so on.

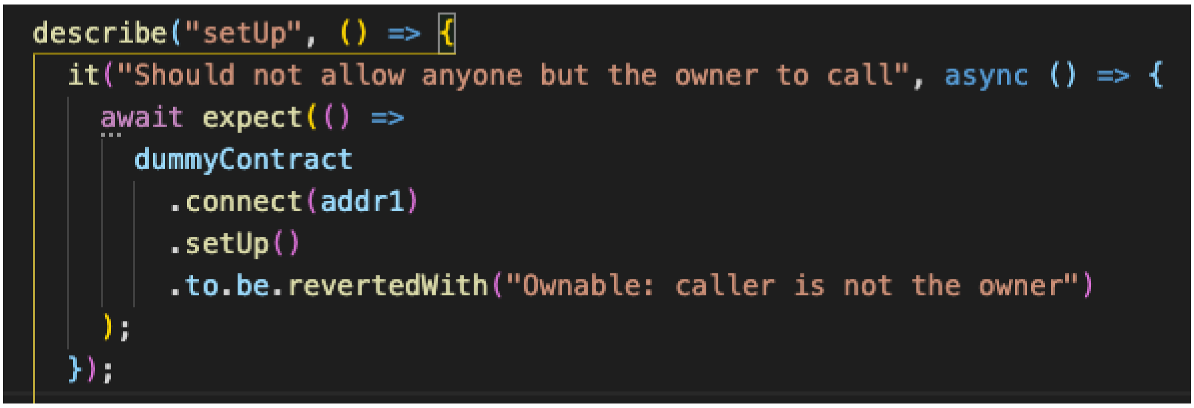
Specifying function caller and reverting



```
describe("setUp", () => {
  it("Should not allow anyone but the owner to call", async () => {
    await expect(() =>
      dummyContract
        .setUp({ from: addr1 })
        .to.be.revertedWith("Ownable: caller is not the owner")
    );
  });

  it("Should mint the initial amount to the contract owner", async () => {
    const ownerBalanceBefore = await dummyContract.balanceOf(owner.address);
    await dummyContract.setUp();
    const ownerBalanceAfter = await dummyContract.balanceOf(owner.address);
    expect(ownerBalanceAfter).to.equal(ownerBalanceBefore + 100);
  });
});
```

(https://imgur.com/T4fgiKs)

When testing the setUp() function in the contract, we need to make sure that only the owner can call this. Therefore, we need to test what will happen when a different account tries to call it. We can do this in Truffle by specifying from inside of the function call ({from: address}). This tells the test to call the function from the address specified. We then check that the test is reverted and give the message that we expect to receive back.
The method for doing the same thing using ethers.js if to use the connect method.
The above example would now change to look like this.



```
describe("setUp", () => {
  it("Should not allow anyone but the owner to call", async () => {
    await expect(() =>
      dummyContract
        .connect(addr1)
        .setUp()
        .to.be.revertedWith("Ownable: caller is not the owner")
    );
  });
```

(https://imgur.com/wGyxXyj)

Example Code

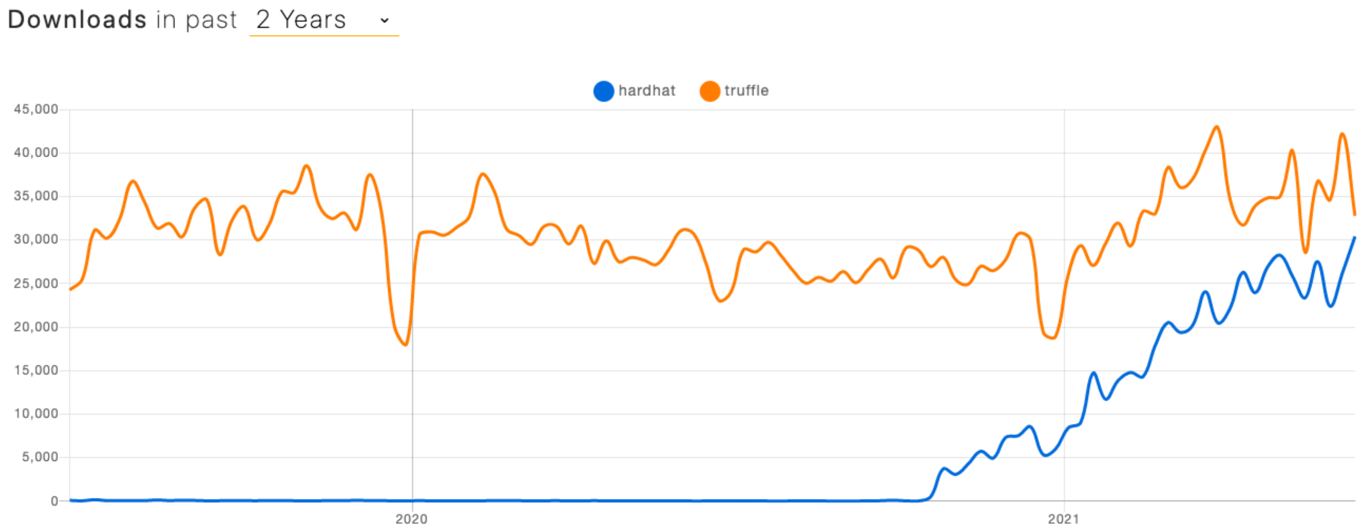
- ▶ dummyContract.sol
- ▶ truffle-fixture.js
- ▶ hardhat-config.js (Truffle/Web3)
- ▶ test.js (Truffle/Web3)
- ▶ hardhat-config.js (Waffle/Ethers)
- ▶ test.js (Waffle/Ethers)

Useful plugins

1.Solidity-coverage (https://hardhat.org/plugins/solidity-coverage.html (https://hardhat.org/plugins/solidity-coverage.html)): Test coverage for Solidity contracts.
2.Hardhat-etherscan (https://hardhat.org/plugins/nomiclabs-hardhat-etherscan.html (https://hardhat.org/plugins/nomiclabs-hardhat-etherscan.html)): Verifies contract deployment on etherscan.
3.Hardhat-gas-reported (https://hardhat.org/plugins/hardhat-gas-reporter.html (https://hardhat.org/plugins/hardhat-gas-reporter.html)): Shows gas usage per unit test and gives monetary cost (in specified currency).
4.Hardhat-tracer (https://hardhat.org/plugins/hardhat-tracer.html (https://hardhat.org/plugins/hardhat-tracer.html)): Shows emitted events when running tests.

Hardhat Advantages

- Ability console.log inside Solidity file to help with debugging.
- Provides smart contract stack traces to aid debugging.
- You can choose to use Truffle/Web3 or Waffle/Ethers, making it very versatile.
- Lots of useful plugins.
- Becoming more popular - see npm package downloads chart below.



(https://imgur.com/VlJXukM)

Quick Guide

Initialising a project from scratch:

- In the terminal `$ npm install --save-dev hardhat`
- Then run `$ npx hardhat`
- Then select `create and empty hardhat-config.js`
- If using Waffle/Ethers, select `y` when prompted to install those packages
- If using Truffle/Web3, select `n`

- Once the project has been set up, if using Truffle/Web3, open **hardhat-config.js** and require the **truffle-5 plugin**.
- If you are planning on using **Ganache**, also require the ganache plugin.
- You have to specify the default network to be **Ganache**.
- In the project main directory, create a test folder, scripts folder and a contracts folder.