Week 4 – Smart Contract Testing

Minfeng Qi

17 Sep 2023

Acknowledgement of Country

We respectfully acknowledge the Wurundjeri People of the Kulin Nation, who are the Traditional Owners of the land on which Swinburne's Australian campuses are located in Melbourne's east and outer-east, and pay our respect to their Elders past, present and emerging.

We are honoured to recognise our connection to Wurundjeri Country, history, culture, and spirituality through these locations, and strive to ensure that we operate in a manner that respects and honours the Elders and Ancestors of these lands.

We also respectfully acknowledge Swinburne's Aboriginal and Torres Strait Islander staff, students, alumni, partners and visitors.

We also acknowledge and respect the Traditional Owners of lands across Australia, their Elders, Ancestors, cultures, and heritage, and recognise the continuing sovereignties of all Aboriginal and Torres Strait Islander Nations.

Introduction to Solidity Testing

Testing in Solidity involves checking the correctness, security, and performance of smart contracts. Tests validate that a contract behaves as expected.

- **Unit Tests:** Testing individual functions or components of a contract.
- **Integration Tests:** Checking how multiple contracts interact with each other.

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

contract MyContract {
    uint256 public myNumber;
    uint256 public mySecondNumber;

    function setMyNumber(uint256 _myNumber) public {
        myNumber = _myNumber;
    }

    function setMySecondNumber(uint256 _mySecondNumber) public {
        mySecondNumber = _mySecondNumber;
    }

    function getNumbersSum() public view returns (uint256) {
        return myNumber + mySecondNumber;
    }
}
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

Unit Tests

- Definition: Unit testing involves testing individual components or functions of a smart contract.
- Importance: Unit tests help in identifying and fixing bugs at an early stage, ensuring each function of the contract works as intended.

For MyContract, a unit test could be written to test the setMyNumber() function:

```
1  it("should set myNumber to the correct value", async () => {
2      await myContract.setMyNumber(10, { from: accounts[0] });
3      const myNumber = await myContract.myNumber();
4      assert.equal(myNumber.toString(), "10");
5  });
```

Integration Tests

- **Definition:** Integration testing involves testing the interaction between multiple smart contracts.
- **Importance:** These tests ensure that contracts can work together seamlessly, identifying and addressing any interoperability issues.

In this case, an integration test could verify that setMyNumber() and setMySecondNumber() work together correctly with getNumbersSum():

```
it("should correctly calculate the sum after setting numbers", async () => {
    await myContract.setMyNumber(10, { from: accounts[0] });
    await myContract.setMySecondNumber(20, { from: accounts[0] });
    const sum = await myContract.getNumbersSum();
    assert.equal(sum.toString(), "30");
});
```

Overview of Solidity Testing Frameworks

- **Introduction:** Testing frameworks provide a set of tools and practices designed to help developers test their contracts more effectively.
- **Need:** Testing frameworks simplify the process of writing and running tests, provide useful features like test suites, and help organize and structure tests.
- Some of the most widely used testing frameworks for Solidity are **Truffle, Hardhat, Waffle, and Remix**. Each of them has its own features, advantages, and disadvantages

Introduction to Truffle Framework

- **Overview:** Truffle is a popular development framework for Ethereum that provides built-in smart contract compilation, linking, deployment, and binary management, as well as automated contract testing.
- **Benefits:** Truffle simplifies the process of testing Ethereum-based applications, supports automated testing, and includes helpful debugging features.

Testing with Truffle

- **Writing tests:** In Truffle, tests can be written in JavaScript, Solidity, or TypeScript. They should be placed in the 'test' directory.
- **Running tests:** Running tests in Truffle is as simple as running the command 'truffle test' in the terminal.

Step 1 : You can download Ganache from the official website and install Truffle via npm:

npm install -g truffle

Step 2: Open your terminal and create a new Truffle project using the following command:

truffle init

Hands-on Example: Writing a Truffle Test

Let's write our first Truffle test using JavaScript for a simple smart contract.

Inside the contracts folder of your project, create a new file named **SimpleStorage.sol** with the following content:

contracts > SimpleStorage.sol > ...

report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml | draw.io

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.0;
3
```

UnitTest stub | dependencies | uml | draw.io

```
4   contract SimpleStorage {
5       uint256 public storedData;
6
```

ftrace | funcSig

```
7       function set(uint256 x) public {
8           storedData = x;
9       }
10
```

ftrace | funcSig

```
11      function get() public view returns (uint256) {
12          return storedData;
13      }
14  }
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

Create Migration File

In your migrations folder, create a file named 2_deploy_simplestorage.js and add the following code:

migrations > JS 2_deploy_simplestorage.js > ...

```
1  const SimpleStorage = artifacts.require("SimpleStorage");
2
3  module.exports = function (deployer) {
4    deployer.deploy(SimpleStorage);
5  };
```

The 2_deploy_simplestorage.js file in a Truffle project is a migration script that is responsible for deploying your smart contract onto the Ethereum blockchain or a simulated blockchain environment like Ganache for development and testing purposes.

- The artifacts.require function is a part of Truffle's contract abstraction which helps to fetch the compiled contract artifacts.
- module.exports is a node.js syntax that exports a function to be used in other files.
- The function takes a parameter called deployer which is a Truffle object that assists in deploying smart contracts onto the network.

Writing the Test

Inside the test folder, create a new JavaScript file named simplestorage_test.js and add the following code

test > JS simplestorage_test.js > ...

```
1  const SimpleStorage = artifacts.require('SimpleStorage');
2
3  contract('SimpleStorage', (accounts) => {
4    it('should store the value 89', async () => {
5      const simpleStorageInstance = await SimpleStorage.deployed();
6      await simpleStorageInstance.set(89, { from: accounts[0] });
7      const storedData = await simpleStorageInstance.get();
8      assert.equal(storedData, 89, 'The value 89 was not stored.');
9    });
10 });
```

- The **contract function** is provided by Truffle and it is used to group together a suite of tests pertaining to the SimpleStorage contract.
- The **accounts parameter** is an array of account addresses available during testing, provided by the Ethereum client (like Ganache).

Running Test

To run your tests, use the truffle test command:

truffle develop

```
Truffle Develop started at http://127.0.0.1:9545/

Accounts:
(0) 0x1096dea993b7e401004af1f80d5c93f9ff9a0bd2
(1) 0xec7e11f7ceb8312bb96d893483c84331386467e2
(2) 0x461b7db9f6e72c0111c0b7a6817d4f240fdeb97e
(3) 0xc29918b5fb92fdf8b298dddd344d93cb9132d9e2
(4) 0x3bfc6f7bf96c680160b33d875ad31b831b9a7977
(5) 0x538853947d20a339c79da902cff85a1f08d2442a
(6) 0xba51744957d12523982a2d85da9cc51ca44c1ffc
(7) 0xbd2bee4de95e2bc4dbe28b29026f5c20acb6ff31
(8) 0xe440315872e01916060e4a6b3fd694fe82ef2f6c
(9) 0x30640b7805c243b1d5e4709dc31cb2293a34d9ad

Private Keys:
(0) 68a6892f92e55a93548822683d3319e98fb9e6e9a386ae6d6241027a7b1737d8
(1) 4991acb8ec89217ca2dd865e43e41687f9aecce7a285f369ac08793fa0e7979c
(2) bd74db92c9b801737fd4c241951b3207db4a81799301744ff1b86791d2c27f10
(3) ddc97becb0adf6cf6c1c92e62445e2f671600aab17cb3087e3c729345a336545
(4) 3a8333c5e67d3a8301cead7e86d96edfd1a0aba35069cabc3848d4ef6ba4140f
(5) 8fee7703a395ecd0a4d1f294d0d701fa07948b0bba19c2ffc1a707dd2838502b
(6) 38ad1e1d3644cccc9f36b507432e5ae236886c7fcadd9e5db30dfc959dab770d
(7) d7af71439ca4fdcebd79f19887ba741a9b9c33fd881c8ffdc9536876357a82a8
(8) 53be96e106702a23f731099640782a106a48a914b0a755ad2ff2e754398d5c65
(9) d0c68dd57efcb360a29d486485d1772d728bb668c05faf50dea299e6c786df9b
```

Running Test

Inside the Truffle develop prompt, run:

migrate

```
Starting migrations...
======================
> Network name:    'develop'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


2_deploy_simplestorage.js
=========================

   Deploying 'SimpleStorage'
   -------------------------
   > transaction hash:    0x797639797ef0c3ae2fdd577cde2e6c058be26a632ff6bad0c07d6fe44e95075b
   > Blocks: 0            Seconds: 0
   > contract address:    0x2665a408d3206742303d742cde1eac686A940176
   > block number:        1
   > block timestamp:     1694928633
   > account:             0x1096deA993b7E401004Af1f80D5c93f9FF9a0bD2
   > balance:             99.999541637875
   > gas used:            135811 (0x21283)
   > gas price:           3.375 gwei
   > value sent:          0 ETH
   > total cost:          0.000458362125 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:      0.000458362125 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.000458362125 ETH
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

Running Test

Inside the Truffle develop prompt, run:

test

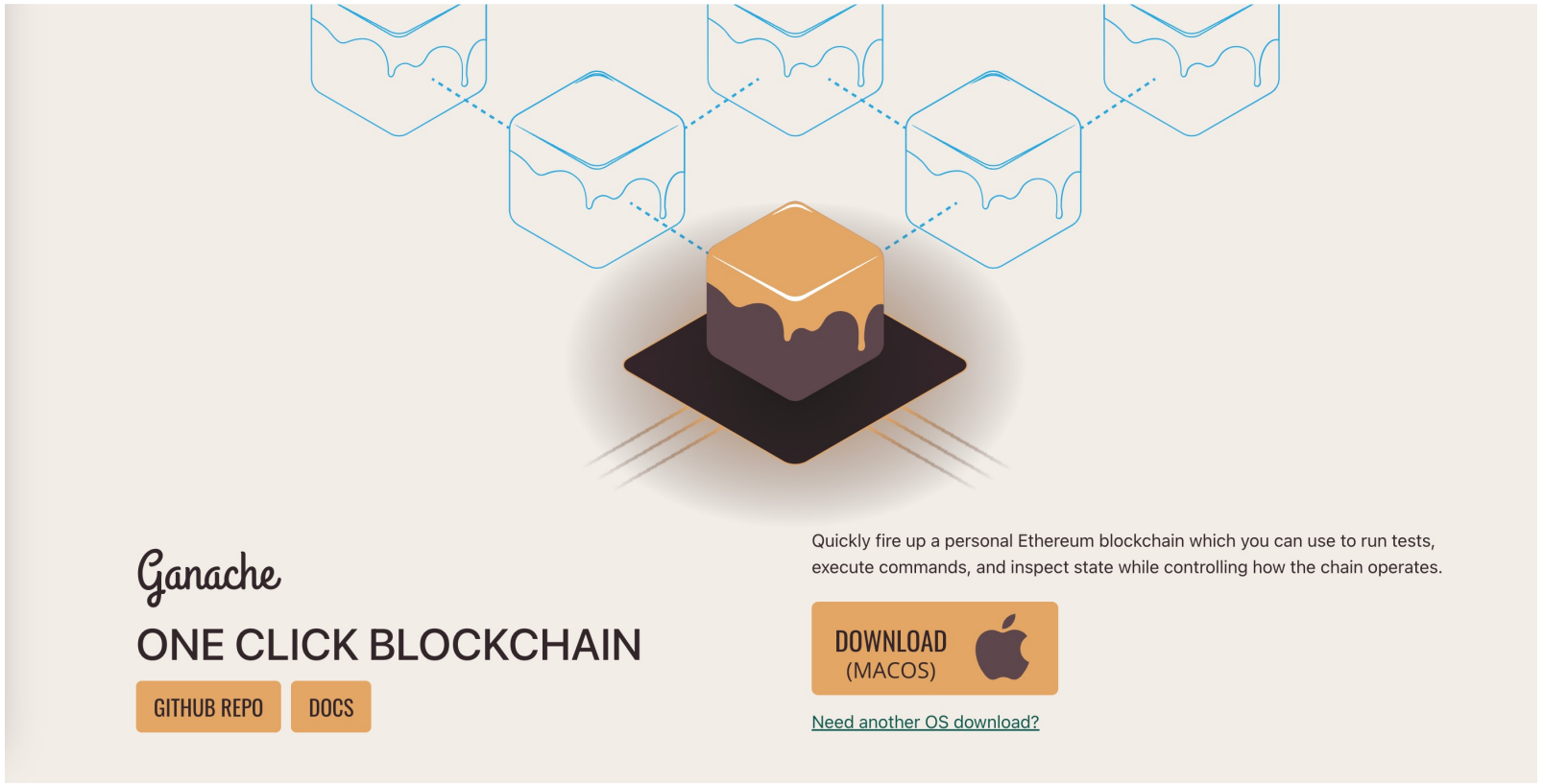
```
truffle(develop)> test
Using network 'develop'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


  Contract: SimpleStorage
    ✔ should store the value 89 (173ms)


  1 passing (224ms)
```

Your test should now run, and if everything is set up correctly, it should pass.

Introduction to Ganache

Ganache is an Ethereum blockchain network suitable for development scenarios making the development of Ethereum applications faster, easier, and more secure.

(It can be understood as an Ethereum blockchain "simulator", "simulating" a production environment during the development process)



SWINBURNE UNIVERSITY OF TECHNOLOGY

Why using Ganache

Sometimes when you want to implement contract testing or some contract development, you need to claim ETH on the test network. Sometimes the faucet does not have ETH to claim, at this time, you can choose **to fork a network locally**, and you can generate as much ETH as you want directly. The principle is to fork a main network to run locally (or it can be called running on a local chain).

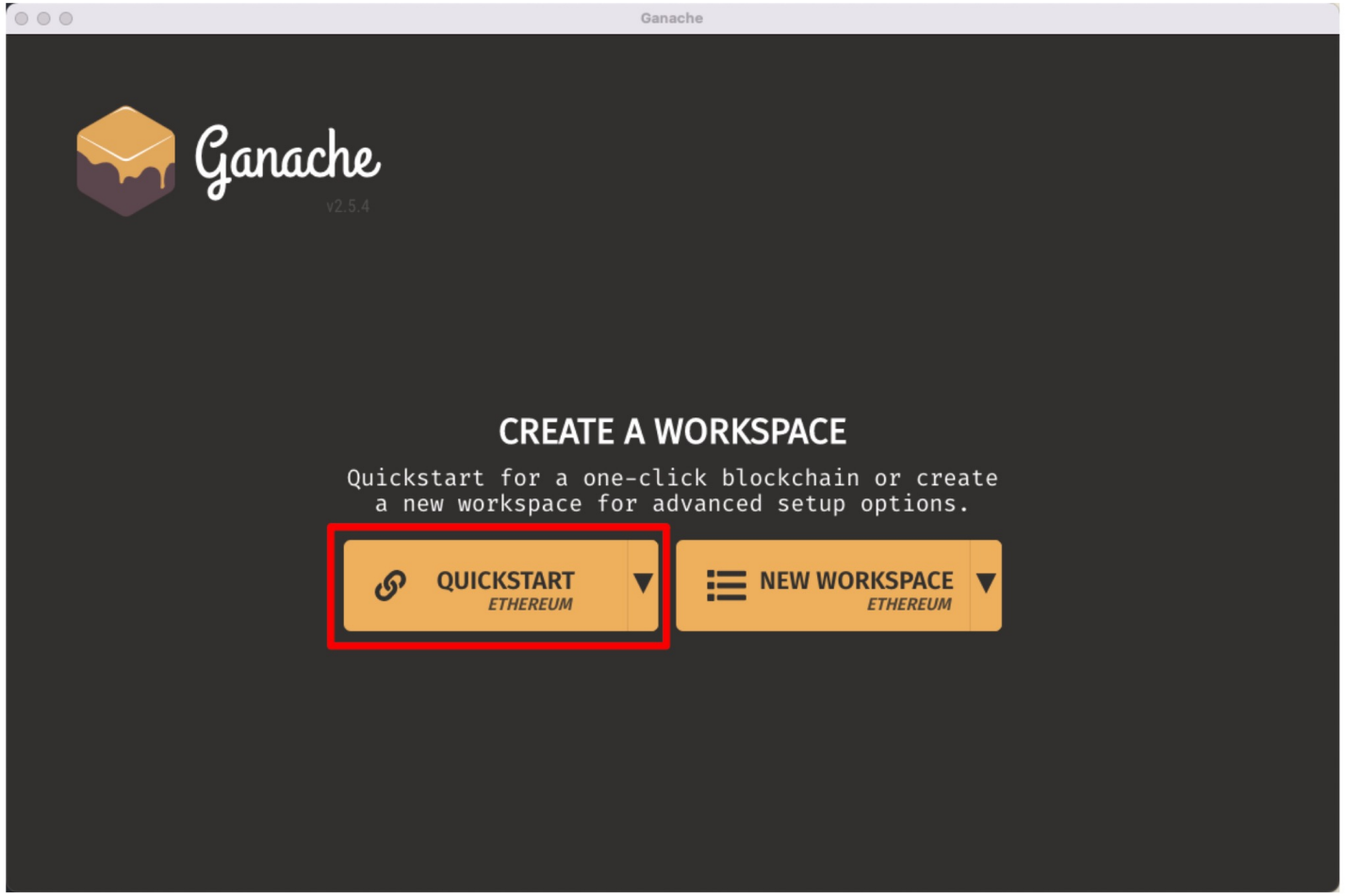
Advantages:

- **Ready to use**, quickly start an EVM blockchain network (you can set up miners, block generation time)
- **Conveniently fork** (branch) an existing blockchain network (without waiting for block synchronization)
- **Simulate any account** (you can simulate the use of any user's token in the environment without a private key).

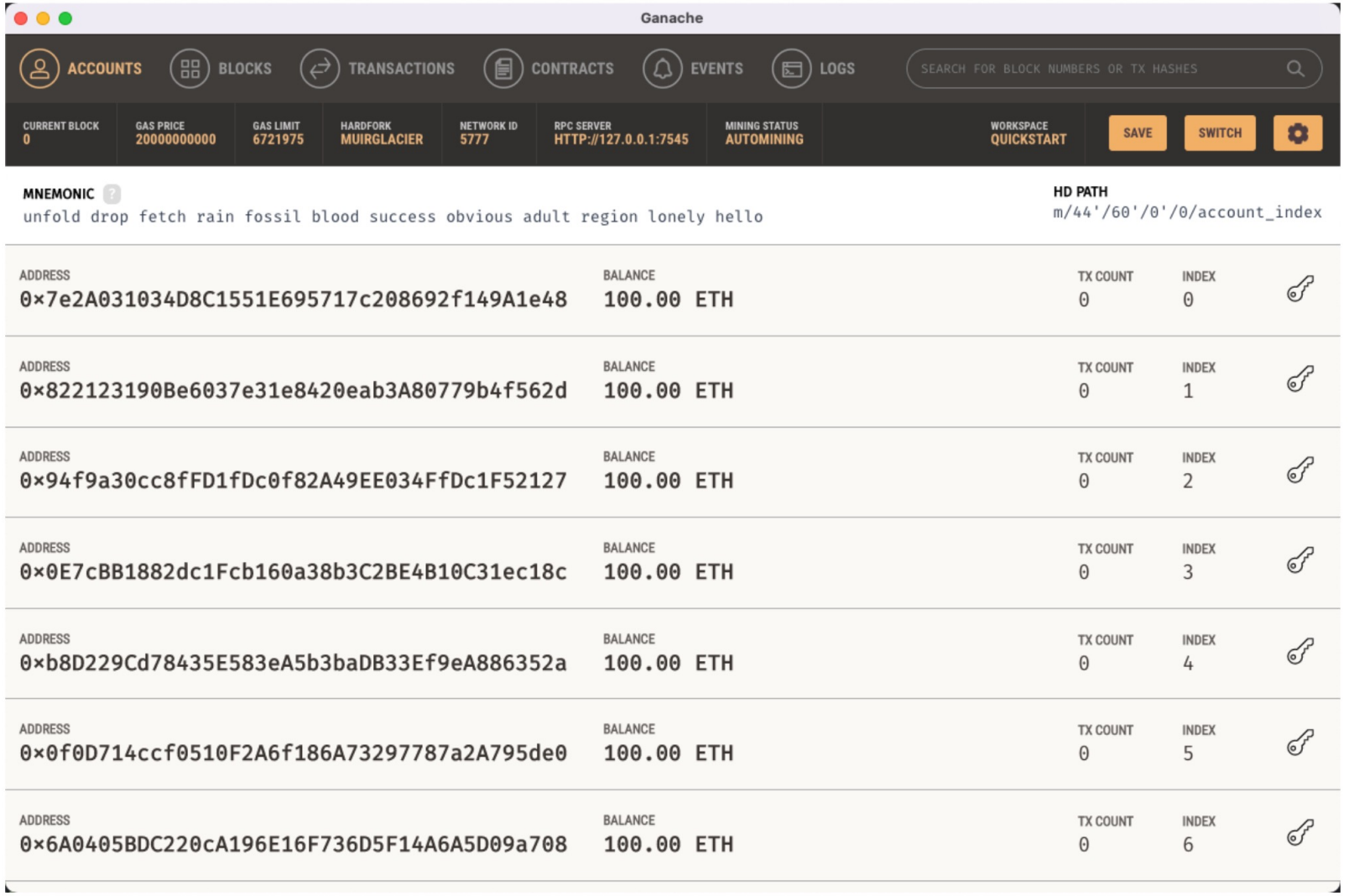
Using Ganache Through GUI

Official website: https://trufflesuite.com/ganache/

Click on quickstart to start quickly, but the default network port for quick start is 7545. You need to modify it when linking with Remix. After starting, it looks as shown below:

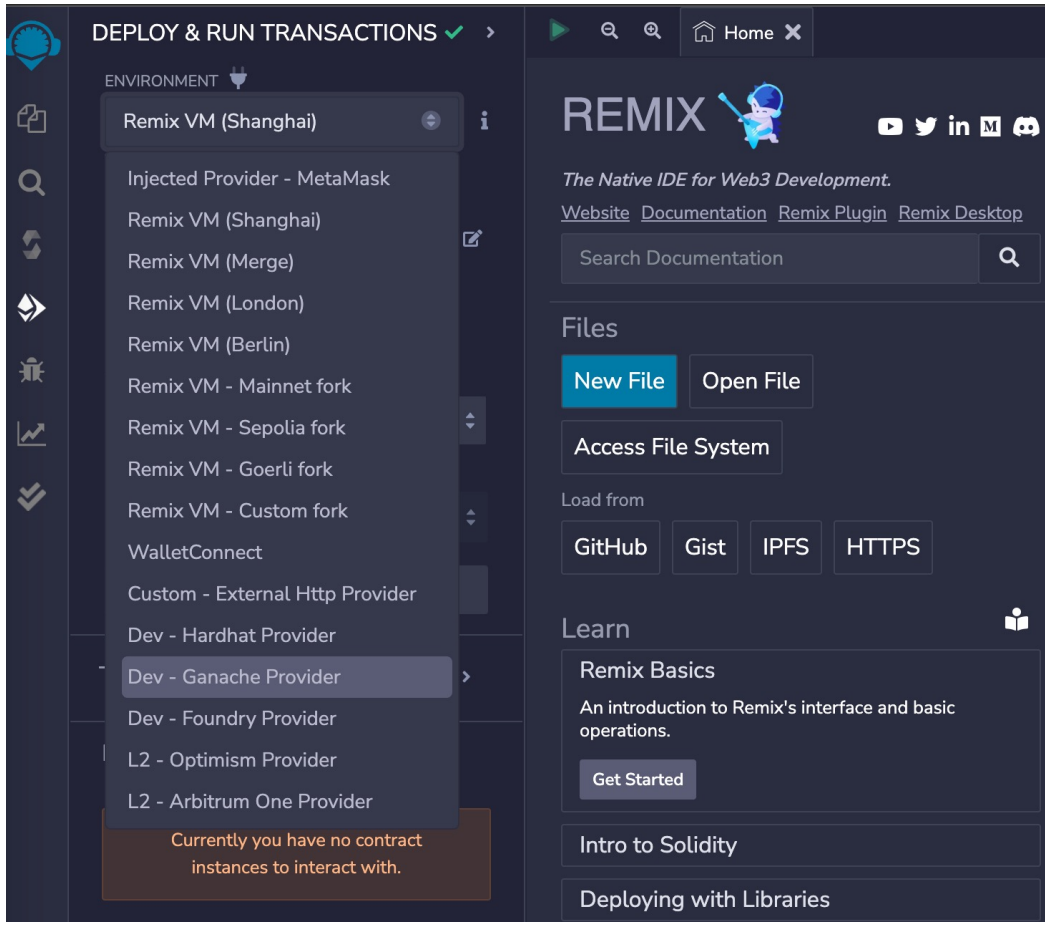


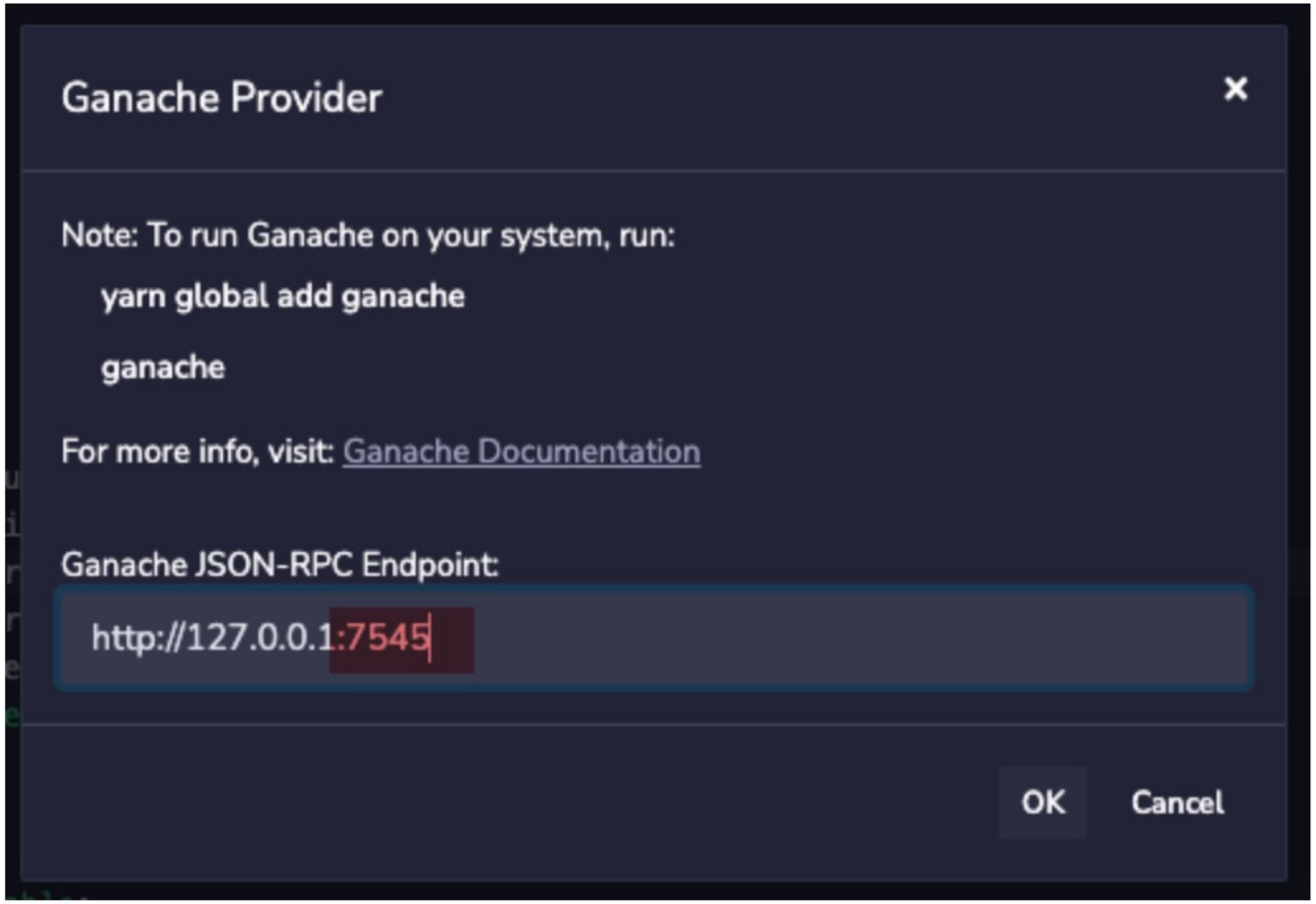
Using Ganache Through GUI



SWINBURNE UNIVERSITY OF TECHNOLOGY

Config Ganache with Remix





Config Ganache with Truffle

Open the truffle-config.js file and configure it to connect to your Ganache instance.

JS truffle-config.js > <unknown> > networks

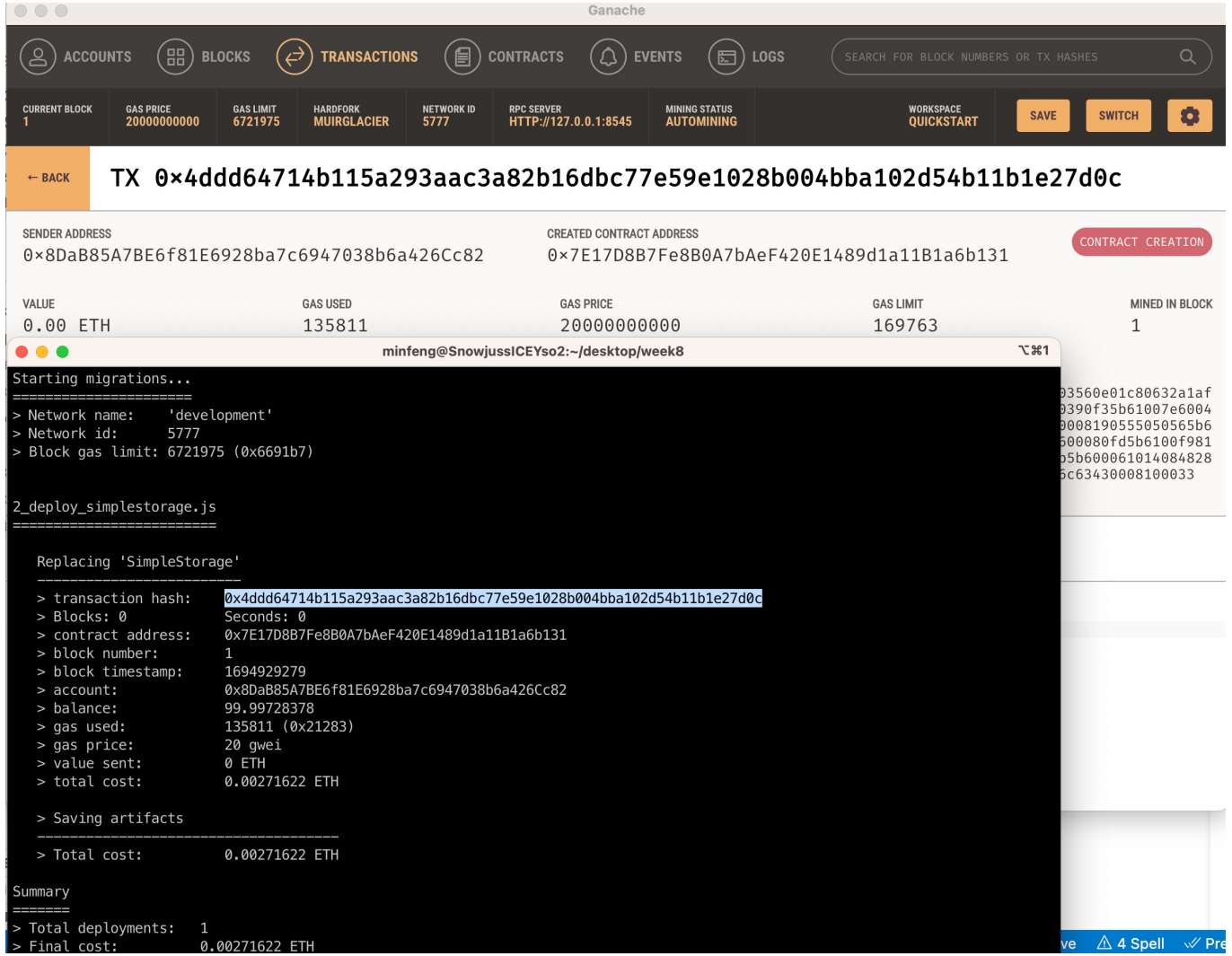
```
 * $ truffle test --network <network-name>
 */

networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache, geth, or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
  development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 7545,            // Standard Ethereum port (default: none)
   network_id: "*",       // Any network (default: none)
  },

// Set default mocha options here, use special reporters, etc.
mocha: {
  // timeout: 100000
},

// Configure your compilers
compilers: {
  solc: {
    version: "0.8.0",      // Fetch exact version from solc-bin (default: truffle's version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
    // settings: {          // See the solidity docs for advice about optimization and evmVersion
    //  optimizer: {
    //    enabled: false,
    //    runs: 200
    //  },
    //  evmVersion: "byzantium"
    // }
  }
},
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

Check result in Ganache

In the terminal, navigate to your project directory and run the following commands to deploy the contract and run the tests:

- truffle compile
- truffle migrate
- truffle test



Thank you