Exercise 9

This is an extra exercise sheet that goes beyond the lecture. Programming JavaScript is not relevant for the exam but is relevant if you want to do real-world blockchain engineering.

1. In this exercise, we create a simple data analysis program using NodeJS, Web3, and Infura.
   (a) What is Web3.js?
   (b) What is Infura? Create an account at `https://infura.io` and create a new project in order to obtain an API key.
   (c) Create a new NodeJS project and install Web3.
   (d) Write a JavaScript program to determine the number of the block that contains the very first contract-creation transaction, i.e. a transaction with recipient `null`. Use the endpoint of your Infura project from sub-task (b) as the Web3 provider.
   (e) What could be done to speed up the execution?

2. Truffle is a popular development environment and testing framework for Ethereum smart contracts. In this exercise, we use it to develop and test an ERC20 token.
   (a) Install Truffle (`https://trufflframework.com/`) and create a new project with `truffle init`. Which folders does Truffle create and what are they for?
   (b) The `truffle-config.js` file is used to configure Truffle. Set the deployment network to the pre-configured `development` network and set the compiler version to 0.5.1. Also, enable the optimizer by uncommenting the appropriate lines in the config file.
   (c) We want to implement a token for the BBSE lecture according to the ERC20 standard (`https://eips.ethereum.org/EIPS/eip-20`). Create a new interface `ERC20.sol` containing only the function and event definitions from the standard.
   (d) Create another contract `BBSECoin.sol`, which implements the interface. Use the directive `import "./ERC20.sol";` to import your other file. Add global variables for the token name and symbol. Also, set the decimals to 0 and the total supply to 1000. Then, add the following two global variables and explain what they are used for:

```
mapping (address => uint256) public balances;
mapping (address => mapping (address => uint256)) public allowed;
```

   (e) Now implement the token functions. See the standard for the exact function definitions and their purpose. Initially, all tokens should belong to the contract creator. Make sure to emit the proper events at the right places.
   Use the command `truffle compile` to compile your contract and check for compilation errors.
   (f) Write a migration script `2_bbse_coin.js` that deploys your `BBSECoin` contract to the blockchain. Refer to the other already existing migration script in your project to see what it should look like.
   (g) Now use `truffle develop` to start the local development environment. Type `migrate` to deploy your contract. Explain what Truffle does in these two steps.
   (h) Truffle supports the JavaScript test framework mocha.js. The following script tests whether the token name is correct.

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

contract("BBSECoin", async accounts => {
  it("should set the token name correctly", async () => {
    let bbseCoinInstance = await BBSECoin.deployed();
    assert.equal(await bbseCoinInstance.name(), "BBSECoin");
  });
});
```

See `https://truffleframework.com/docs/truffle/testing/writing-tests-in-javascript` for some more information on how to reference accounts and send transactions from specific accounts. Then add the following test cases to the script:

- The initial token balance of the creator account is equal to the total token supply
- Tokens can be transferred using the `transfer()` function
- The allowance can be set and read
- Accounts can transfer tokens on behalf of other accounts

Type `test` to run the test cases.

(i) In order to implement some advanced test cases, the npm library `truffle-assertions` is useful. It allows to check if events occurred, if a transaction reverted, and more. See the documentation here for more details: `https://www.npmjs.com/package/truffle-assertions`. Install the library, then implement and run the following test cases:

- An insufficient balance throws an error when trying to transfer tokens
- Transferring from an account that has not explicitly authorized the transfer should revert the transaction
- The `transfer()` and `transferFrom()` functions must fire the `Transfer` event (even for 0 value transfers)
- The `approve()` function must fire the `Approval` event

Note: Of course, this list of test cases is far from complete. Many more useful test cases can be thought of.