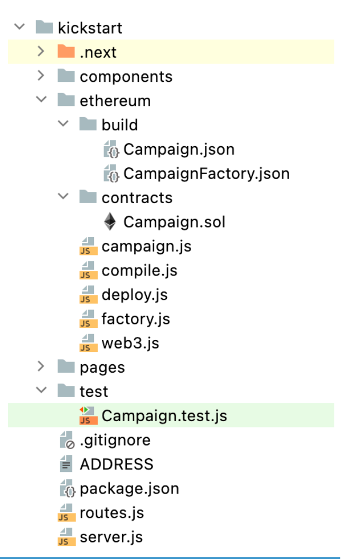
Ethereum and Solidity - The Complete Developer's Guide

Section 6. Ethereum Project Infrastructure

Structure of the project



```
kickstart
  .next
  components
  ethereum
    build
      Campaign.json
      CampaignFactory.json
    contracts
      Campaign.sol
    campaign.js
    compile.js
    deploy.js
    factory.js
    web3.js
  pages
  test
    Campaign.test.js
  .gitignore
  ADDRESS
  package.json
  routes.js
  server.js
```

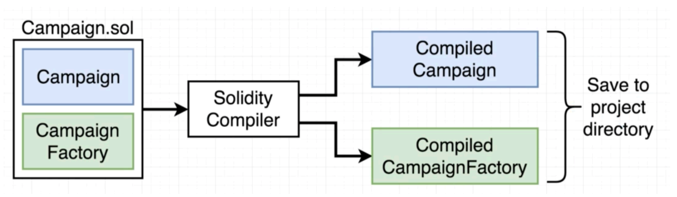
Contracts are in "Ethereum" folder

Contracts (in are "contracts" folder)

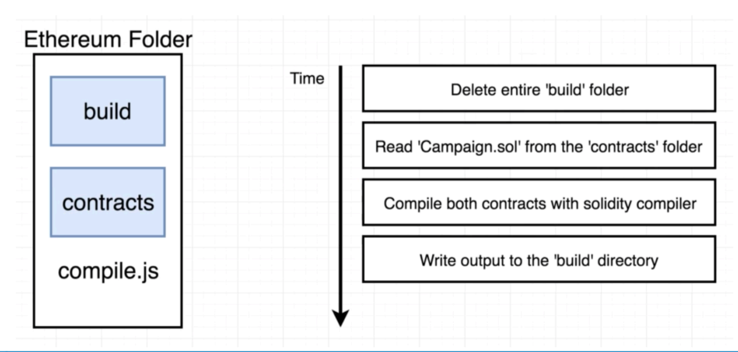
see Section #5

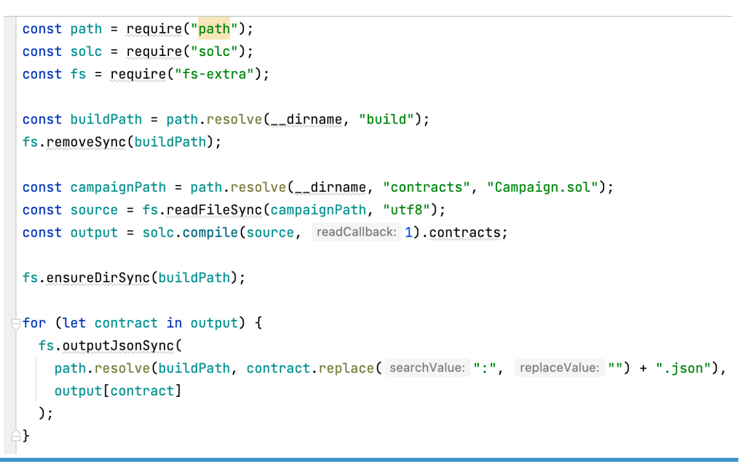
Compiling contracts ("compile.js" file creates results in "build" folder)

- Compile them every time



- Compile them when they were changed (*node compile.js*). And put results into another project folder

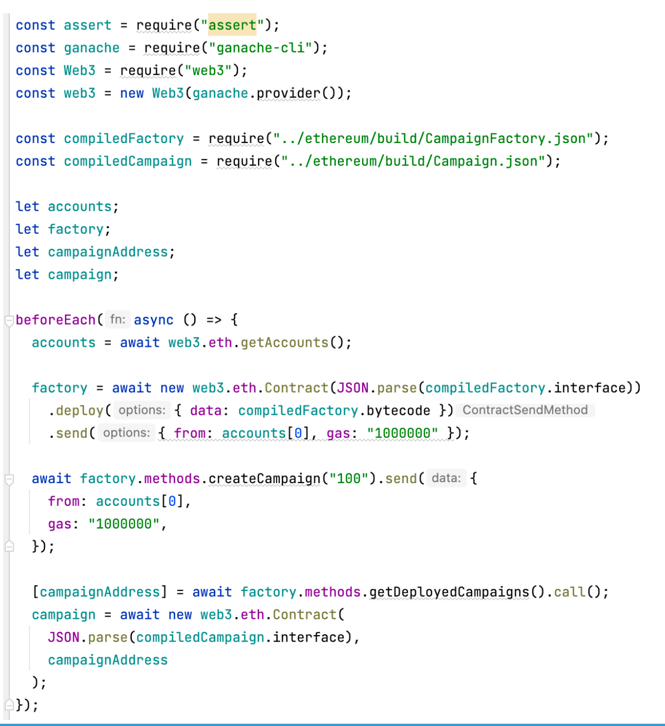

Ethereum Folder
build
contracts
compile.js
Time
Delete entire 'build' folder
Read 'Campaign.sol' from the 'contracts' folder
Compile both contracts with solidity compiler
Write output to the 'build' directory



```
const path = require("path");
const solc = require("solc");
const fs = require("fs-extra");

const buildPath = path.resolve(__dirname, "build");
fs.removeSync(buildPath);

const campaignPath = path.resolve(__dirname, "contracts", "Campaign.sol");
const source = fs.readFileSync(campaignPath, "utf8");
const output = solc.compile(source, readCallback: 1).contracts;

fs.ensureDirSync(buildPath);

for (let contract in output) {
  fs.outputJsonSync(
    path.resolve(buildPath, contract.replace( searchValue: ":",  replaceValue: "") + ".json"),
    output[contract]
  );
}
```

Tests are in "Test" folder ("Campaign.test.js" file contains tests for the contract). Tests are executed with *npm test*

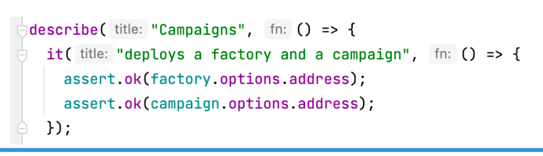
Warmup (before-each)



```
const assert = require("assert");
const ganache = require("ganache-cli");
const Web3 = require("web3");
const web3 = new Web3(ganache.provider());

const compiledFactory = require("../ethereum/build/CampaignFactory.json");
const compiledCampaign = require("../ethereum/build/Campaign.json");

let accounts;
let factory;
let campaignAddress;
let campaign;

beforeEach( fn: async () => {
  accounts = await web3.eth.getAccounts();

  factory = await new web3.eth.Contract(JSON.parse(compiledFactory.interface))
    .deploy( options: { data: compiledFactory.bytecode }) ContractSendMethod
    .send( options: { from: accounts[0], gas: "1000000" });

  await factory.methods.createCampaign("100").send( data: {
    from: accounts[0],
    gas: "1000000",
  });

  [campaignAddress] = await factory.methods.getDeployedCampaigns().call();
  campaign = await new web3.eth.Contract(
    JSON.parse(compiledCampaign.interface),
    campaignAddress
  );
});
```

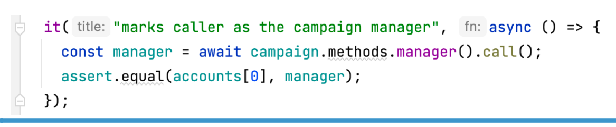
Tests

Deployment of Factory and Campaign



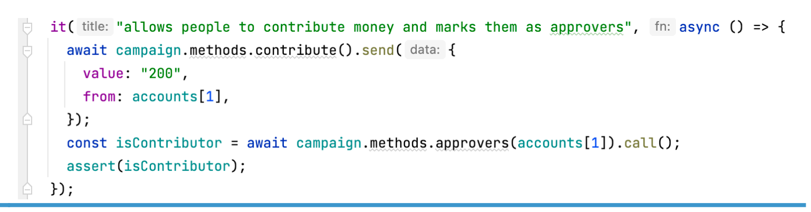
```
describe( title: "Campaigns",  fn: () => {
  it( title: "deploys a factory and a campaign",  fn: () => {
    assert.ok(factory.options.address);
    assert.ok(campaign.options.address);
  });
```

Mark caller as the Campaign Manager



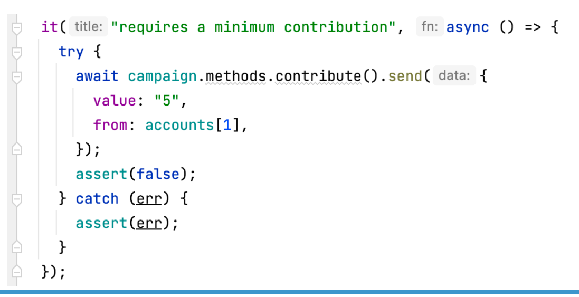
```
  it( title: "marks caller as the campaign manager",  fn: async () => {
    const manager = await campaign.methods.manager().call();
    assert.equal(accounts[0], manager);
  });
```

Contribute money become approver



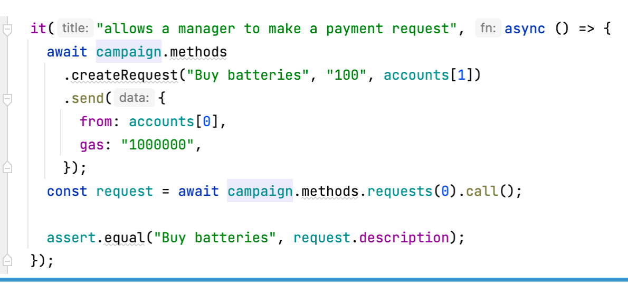
```
  it( title: "allows people to contribute money and marks them as approvers",  fn: async () => {
    await campaign.methods.contribute().send( data: {
      value: "200",
      from: accounts[1],
    });
    const isContributor = await campaign.methods.approvers(accounts[1]).call();
    assert(isContributor);
  });
```

Check minimum contribution



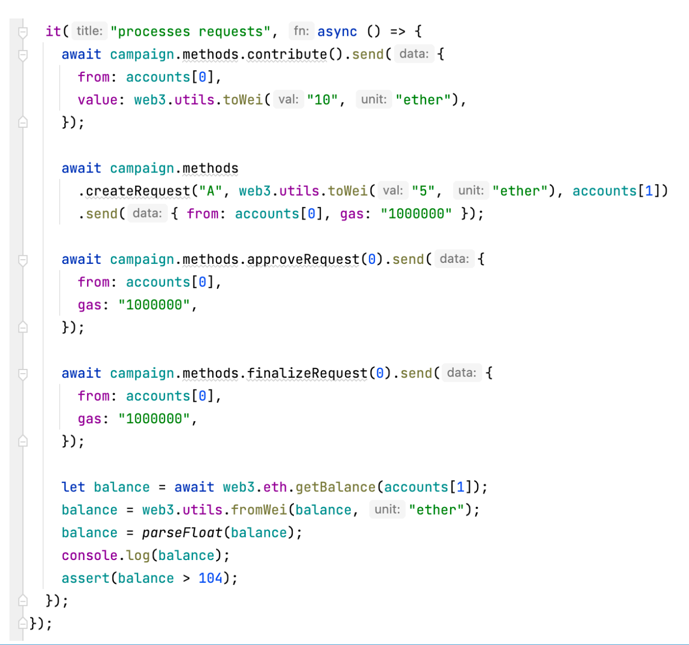
```
  it( title: "requires a minimum contribution",  fn: async () => {
    try {
      await campaign.methods.contribute().send( data: {
        value: "5",
        from: accounts[1],
      });
      assert(false);
    } catch (err) {
      assert(err);
    }
  });
```

Manager makes Payment Request



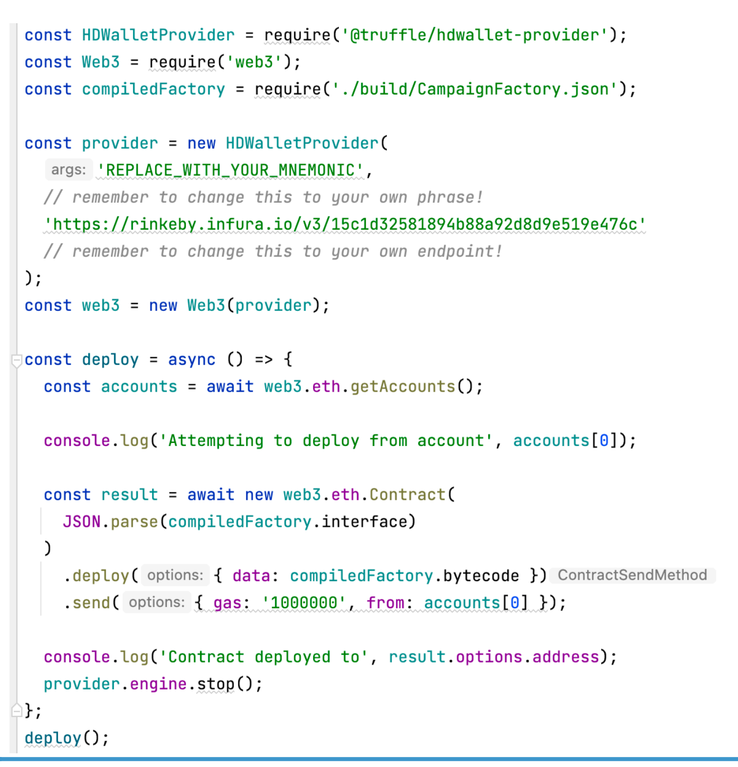
```
  it( title: "allows a manager to make a payment request",  fn: async () => {
    await campaign.methods
      .createRequest("Buy batteries", "100", accounts[1])
      .send( data: {
        from: accounts[0],
        gas: "1000000",
      });
    const request = await campaign.methods.requests(0).call();

    assert.equal("Buy batteries", request.description);
  });
```

Manager processes Payment Request



```
  it( title: "processes requests",  fn: async () => {
    await campaign.methods.contribute().send( data: {
      from: accounts[0],
      value: web3.utils.toWei( val: "10",  unit: "ether"),
    });

    await campaign.methods
      .createRequest("A", web3.utils.toWei( val: "5",  unit: "ether"), accounts[1])
      .send( data: { from: accounts[0], gas: "1000000" });

    await campaign.methods.approveRequest(0).send( data: {
      from: accounts[0],
      gas: "1000000",
    });

    await campaign.methods.finalizeRequest(0).send( data: {
      from: accounts[0],
      gas: "1000000",
    });

    let balance = await web3.eth.getBalance(accounts[1]);
    balance = web3.utils.fromWei(balance,  unit: "ether");
    balance = parseFloat(balance);
    console.log(balance);
    assert(balance > 104);
  });
});
```

Deployment script (using *node deploy.js*)



```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const Web3 = require('web3');
const compiledFactory = require('./build/CampaignFactory.json');

const provider = new HDWalletProvider(
  args: 'REPLACE_WITH_YOUR_MNEMONIC',
  // remember to change this to your own phrase!
  'https://rinkeby.infura.io/v3/15c1d32581894b88a92d8d9e519e476c'
  // remember to change this to your own endpoint!
);
const web3 = new Web3(provider);

const deploy = async () => {
  const accounts = await web3.eth.getAccounts();

  console.log('Attempting to deploy from account', accounts[0]);

  const result = await new web3.eth.Contract(
    JSON.parse(compiledFactory.interface)
  )
    .deploy( options: { data: compiledFactory.bytecode }) ContractSendMethod
    .send( options: { gas: '1000000', from: accounts[0] });

  console.log('Contract deployed to', result.options.address);
  provider.engine.stop();
};
deploy();
```

Write down address of the contract!