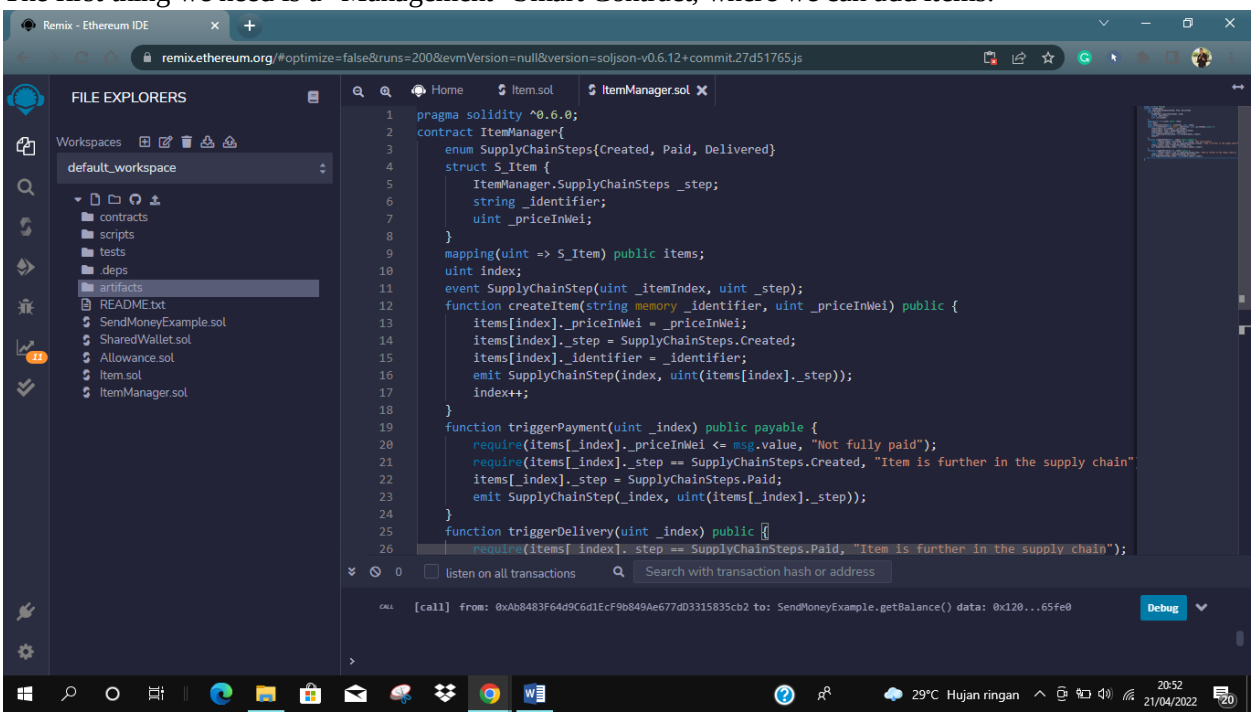
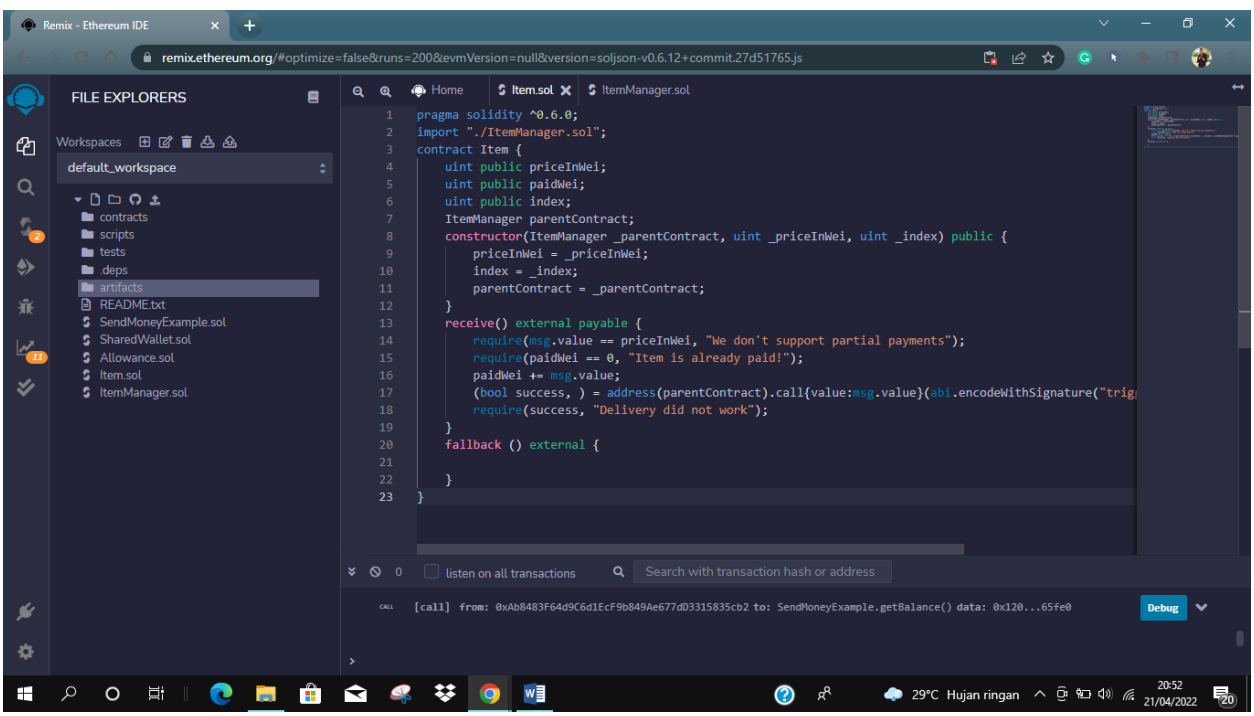
The Item Manager Smart Contract

The first thing we need is a "Management" Smart Contract, where we can add items.

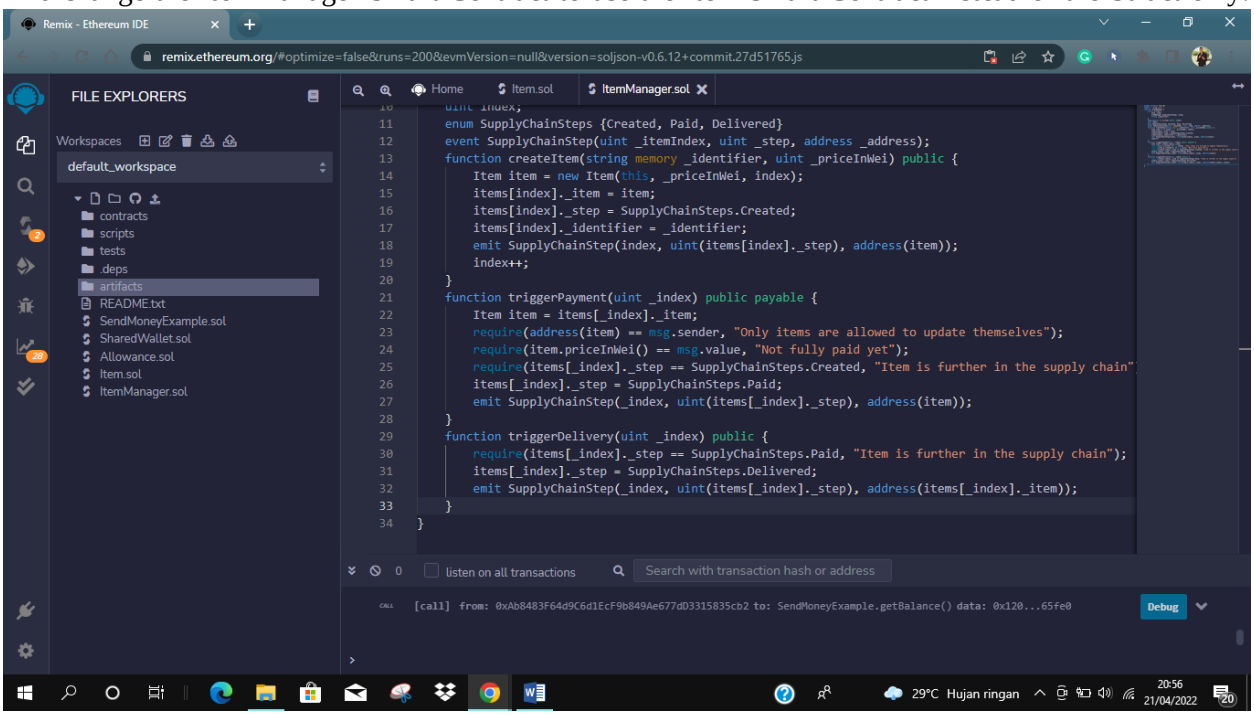


Item Smart Contract

Let's add another smart contract:

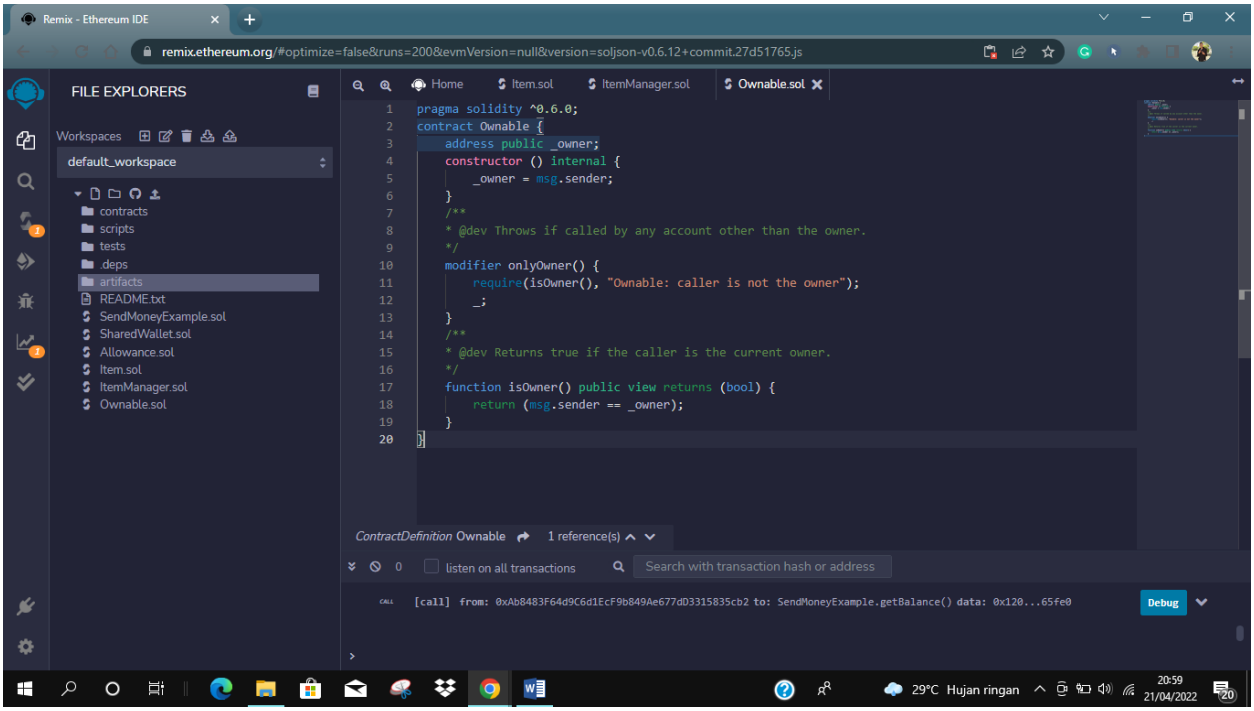


And change the ItemManager Smart Contract to use the Item Smart Contract instead of the Struct only:

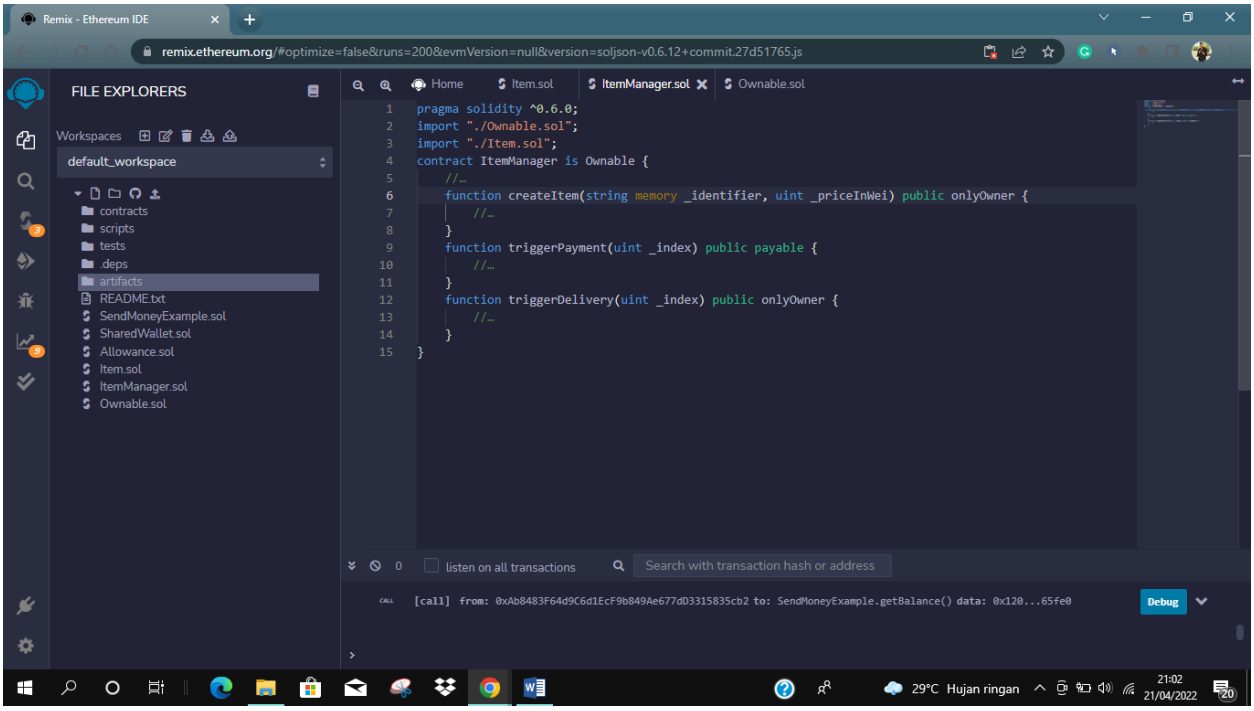


Ownable Functionality

Normally we would add the OpenZeppelin Smart Contracts with the Ownable Functionality. But at the time of writing this document they are not updated to solidity 0.6 yet. So, instead we will add our own Ownable functionality very much like the one from OpenZeppelin:



Then modify the ItemManager so that all functions, that should be executable by the "owner only" have the correct modifier:



Install Truffle

To install truffle open a terminal (Mac/Linux) or a PowerShell (Windows 10)

```
Windows PowerShell
PS C:\Users\User> npm install -g truffle
npm WARN              har-validator@5.1.5: this library is no longer supported
npm WARN              mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports promises now, plea
se switch to that.
npm WARN              request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN              multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN              cids@0.7.5: This module has been superseded by the multiformats module
npm WARN          Failed to remove some directories [
npm WARN            [
npm WARN              'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle\\node_modules',
npm WARN              [Error: EPERM: operation not permitted, unlink 'C:\Users\User\AppData\Roaming\npm\node_modules\truffle\node_modul
es\mime\CHANGELOG.md'] {
npm WARN                errno: -4048,
npm WARN                code: 'EPERM',
npm WARN                syscall: 'unlink',
npm WARN                path: 'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle\\node_modules\\mime\\CHANGELOG.md'
npm WARN              }
npm WARN            ],
npm WARN            [
npm WARN              'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle\\node_modules\\@ethersproject',
npm WARN              [Error: EPERM: operation not permitted, unlink 'C:\Users\User\AppData\Roaming\npm\node_modules\truffle\node_modul
es\@ethersproject\strings\package.json'] {
npm WARN                errno: -4048,
npm WARN                code: 'EPERM',
npm WARN                syscall: 'unlink',
npm WARN                path: 'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle\\node_modules\\@ethersproject\\strings\\pa
ckage.json'
npm WARN              }
npm WARN            ],
npm WARN            [
npm WARN              'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle',
npm WARN              [Error: EPERM: operation not permitted, unlink 'C:\Users\User\AppData\Roaming\npm\node_modules\truffle\node_modul
es\mime\CHANGELOG.md'] {
npm WARN                errno: -4048,
npm WARN                code: 'EPERM',
npm WARN                syscall: 'unlink',
npm WARN                path: 'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle\\node_modules\\mime\\CHANGELOG.md'
npm WARN              }
npm WARN            ],
npm WARN            [
npm WARN              'C:\\Users\\User\\AppData\\Roaming\\npm\\node_modules\\truffle',
npm WARN              [Error: EPERM: operation not permitted, unlink 'C:\Users\User\AppData\Roaming\npm\node_modules\truffle\node_modul
es\mime\CHANGELOG.md'] {
npm WARN                errno: -4048,
```

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\User> npm install -g truffle@5.1.8
npm WARN              mkdirp@0.5.1: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that the API s
urface has changed to use Promises in 1.x.)

changed 27 packages, and audited 28 packages in 15s

3 critical severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\Users\User> mkdir s06-eventtrigger


    Directory: C:\Users\User


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----         4/21/2022   9:28 PM                s06-eventtrigger


PS C:\Users\User> cd s06-eventtrigger
PS C:\Users\User\s06-eventtrigger> ls
PS C:\Users\User\s06-eventtrigger>
```

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\User> cd ./s06-eventtrigger/
PS C:\Users\User\s06-eventtrigger> truffle unbox react
truffle : The term 'truffle' is not recognized as the name of a cmdlet, function, script file, or operable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ truffle unbox react
+ ~~~~~~~
    + CategoryInfo          : ObjectNotFound: (truffle:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\User\s06-eventtrigger>
```

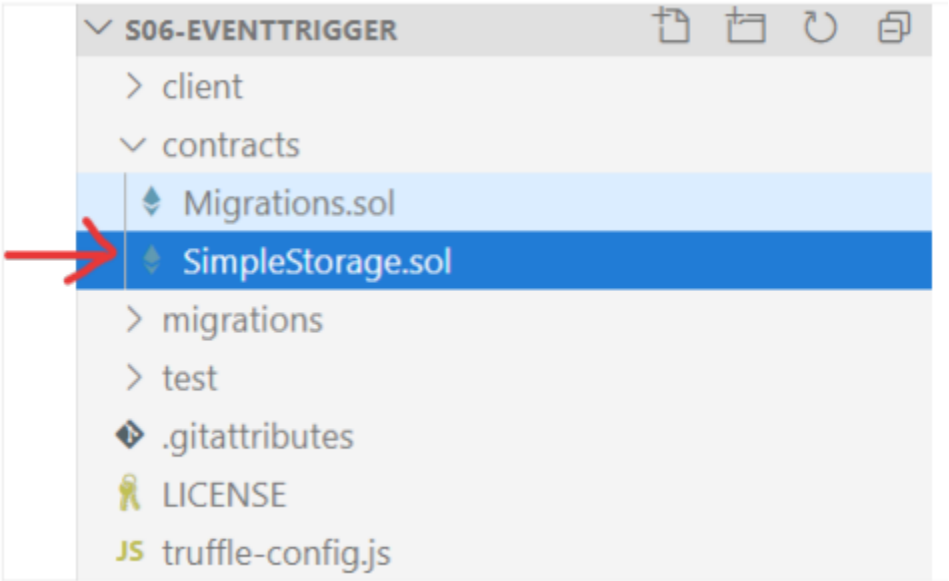
```
ebd> mkdir s06-eventtrigger


    Directory: C:\101Tmp\ebd


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        1/11/2020  10:39 AM                s06-eventtrigger


ebd> cd .\s06-eventtrigger\
s06-eventtrigger> ls
s06-eventtrigger>
```

And unbox the react box: truffle unbox react this should download a repository and install all dependencies in the current folder:

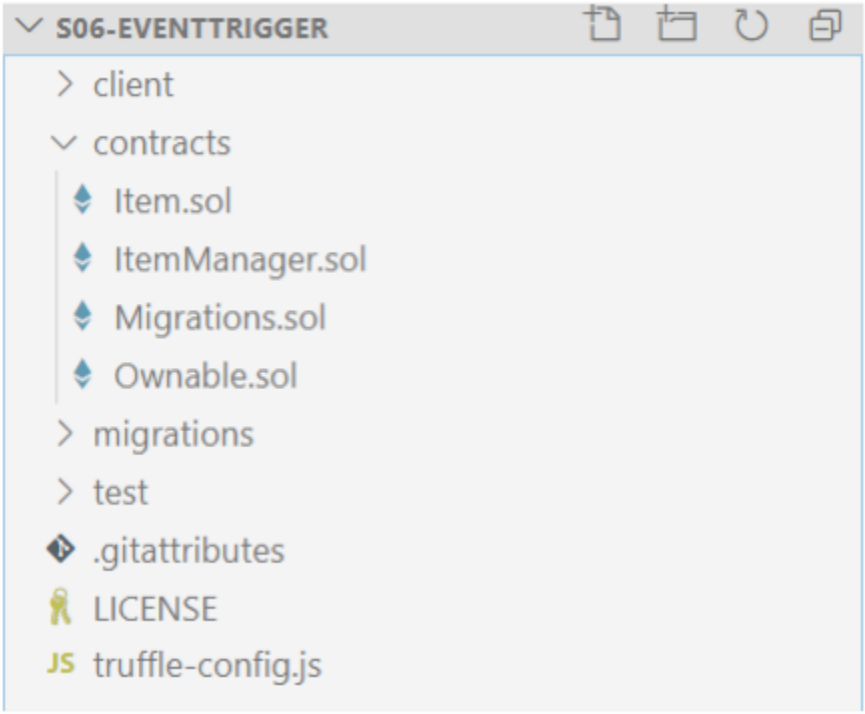
truffle unbox react

```
s06-eventtrigger> truffle unbox react
√ Preparing to download box
√ Downloading
√ cleaning up temporary files
√ Setting up box
s06-eventtrigger> ls


    Directory: C:\101Tmp\ebd\s06-eventtrigger


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        1/11/2020  10:41 AM                client
d-----        1/11/2020  10:41 AM                contracts
d-----        1/11/2020  10:41 AM                migrations
d-----        1/11/2020  10:41 AM                test
-a----        1/11/2020  10:41 AM             33 .gitattributes
-a----        1/11/2020  10:41 AM           1075 LICENSE
-a----        1/11/2020  10:41 AM            297 truffle-config.js


s06-eventtrigger>
```

Remove the existing SimpleStorage Smart Contract but leave the "Migrations.sol" file:



Add in our Files:



Then modify the "migration" file in the migrations/ folder:

migrations/2_deploy_contracts.js

```
var ItemManager = artifacts.require("./ItemManager.sol");

module.exports = function(deployer) {
    deployer.deploy(ItemManager);
};
```

Modify the truffle-config.js file to lock in a specific compiler version:

truffle-config.js

```
const path = require("path");

module.exports = {
    // See <http://truffleframework.com/docs/advanced/configuration>
    // to customize your Truffle configuration!
    contracts_build_directory: path.join(__dirname, "client/src/contracts"),
    networks: {
        develop: {
        port: 8545
        }
    },
    compilers: {
        solc: {
        version: "^0.6.0"
        }
    }
};
```

```
truffle(develop)> migrate

Compiling your contracts...
===========================
> Compiling .\contracts\Item.sol
> Compiling .\contracts\ItemManager.sol
> Compiling .\contracts\Ownable.sol
> Artifacts written to C:\101Tmp\ebd\s06-eventtrigger\client\src\co
> Compiled successfully using:
   - solc: 0.6.1+commit.e6f7d5a4.Emscripten.clang



Starting migrations...
======================
> Network name:    'develop'
```

Modify HTML

Now it's time that we modify our HTML so we can actually interact with the Smart Contract from the Browser.

Open "client/App.js" and modify a few things inside the file:

```
import React, { Component } from "react";
import ItemManager from "./contracts/ItemManager.json";
import Item from "./contracts/Item.json";
import getWeb3 from "./getWeb3";
import "./App.css";

class App extends Component {
  state = {cost: 0, itemName: "exampleItem1", loaded:false};

  componentDidMount = async () => {
    try {
      // Get network provider and web3 instance.
      this.web3 = await getWeb3();

      // Use web3 to get the user's accounts.
      this.accounts = await this.web3.eth.getAccounts();

      // Get the contract instance.
      const networkId = await this.web3.eth.net.getId();

      this.itemManager = new this.web3.eth.Contract(
        ItemManager.abi,
        ItemManager.networks[networkId] && ItemManager.networks[networkId].address,
      );
      this.item = new this.web3.eth.Contract(
        Item.abi,
        Item.networks[networkId] && Item.networks[networkId].address,
      );

      this.setState({loaded:true});

    } catch (error) {
      // Catch any errors for any of the above operations.
      alert(
        `Failed to load web3, accounts, or contract. Check console for details.`,
      );
      console.error(error);
    }
  };
//.. more code here ...
```

Then add in a form to the HTML part on the lower end of the App.js file, in the "render" function:

```
  render() {
    if (!this.state.loaded) {
      return <div>Loading Web3, accounts, and contract...</div>;
    }
    return (
      <div className="App">
        <h1>Simply Payment/Supply Chain Example!</h1>
        <h2>Items</h2>

        <h2>Add Element</h2>
        Cost: <input type="text" name="cost" value={this.state.cost} onChange={this.handleInputChange} />
        Item Name: <input type="text" name="itemName" value={this.state.itemName} onChange={this.handleInputChange} />
        <button type="button" onClick={this.handleSubmit}>Create new Item</button>
      </div>
    );
  }
```

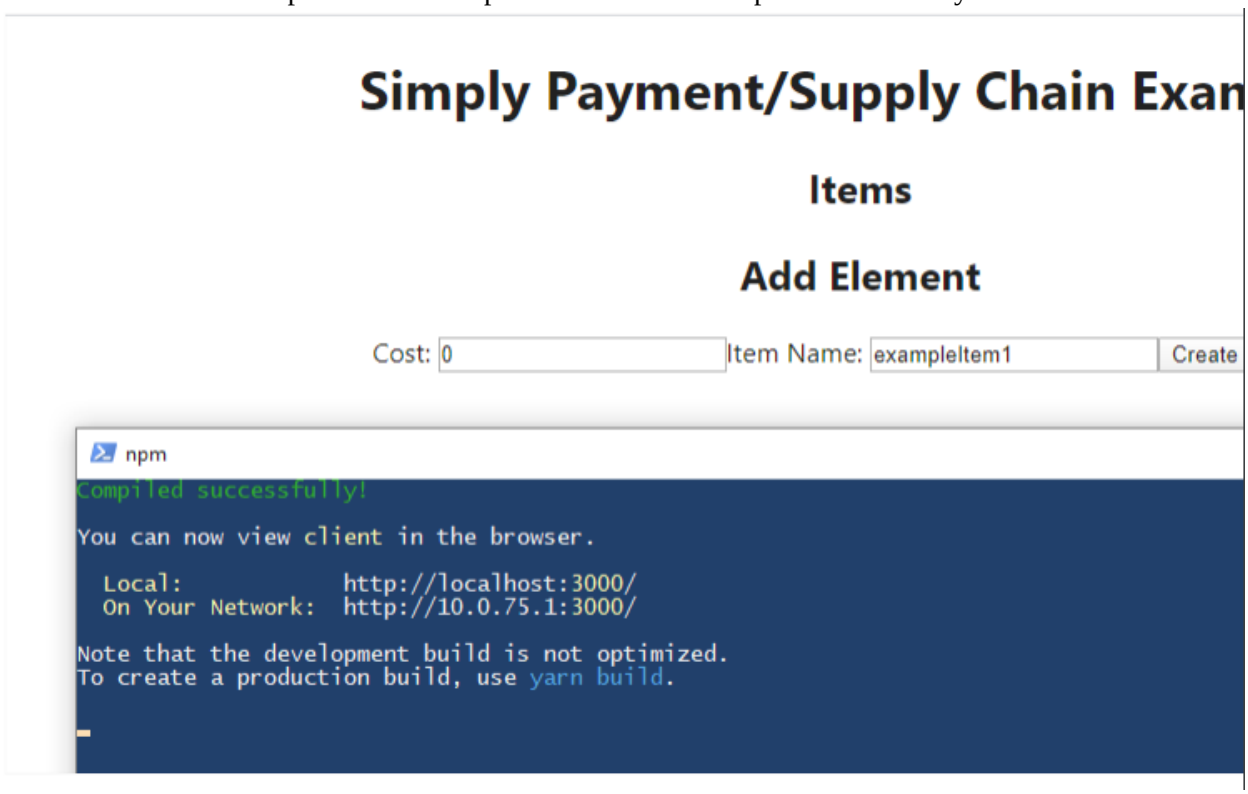
And add two functions, one for handleInputChange, so that all input variables are set correctly. And one for sending the actual transaction off to the network:

```
handleSubmit = async () => {
    const { cost, itemName } = this.state;
    console.log(itemName, cost, this.itemManager);
    let result = await this.itemManager.methods.createItem(itemName, cost).send({ from: this.accounts[0] });
    console.log(result);
    alert("Send "+cost+" Wei to "+result.events.SupplyChainStep.returnValues._address);
  };

  handleInputChange = (event) => {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;
```

```
    this.setState({
      [name]: value
    });
  }
```

Open another terminal/powershell (leave the one running that you have already opened with truffle) and go to the client folder and run

This will start the development server on port 3000 and should open a new tab in your browser:

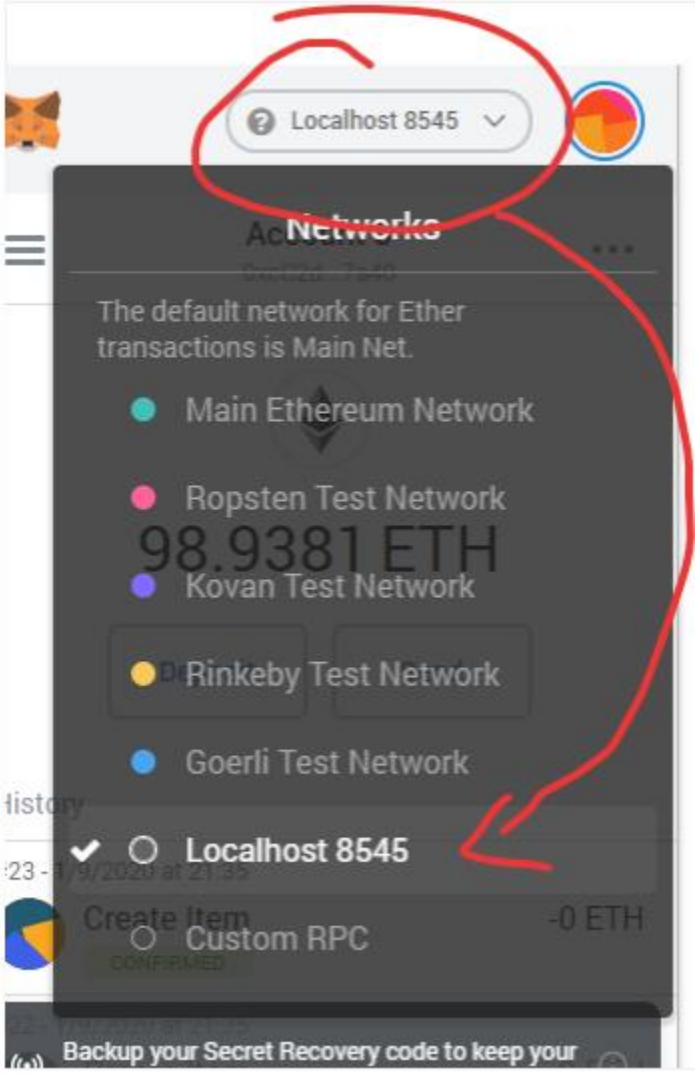


Connect with MetaMask

What We Do

In this section we want to connect our React App with MetaMask and use MetaMask as a Keystore to sign transactions. It will also be a proxy to the correct blockchain.

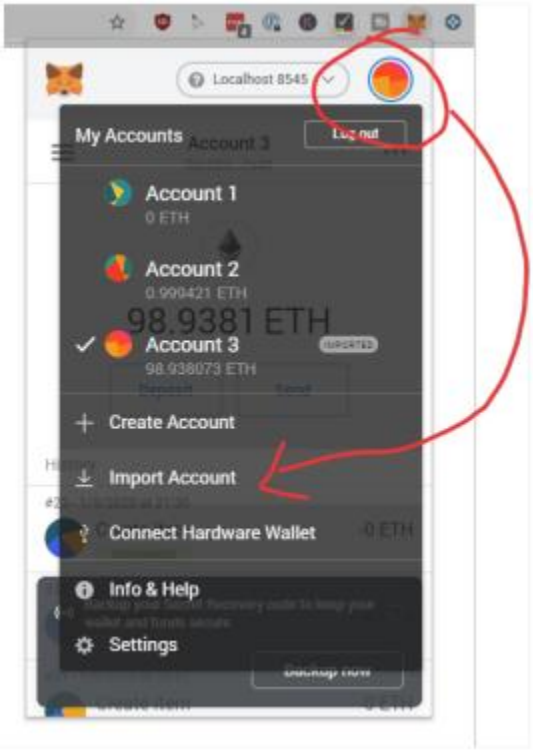
First, connect with MetaMask to the right network:

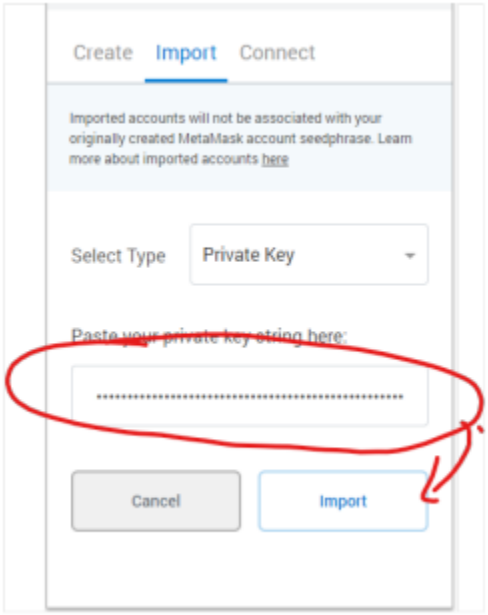


In the Terminal/Powershell where Truffle Developer Console is running scroll to the private keys on top:

```
Private Keys:
(0) 2a9ed36cdb66f81093a82443c2b9f237f3534ef75f4f044fa6ebd76d5d05f61
(1) f9c941a67e63fe4b84fe63ad652c29b2f225eb57562b246bf44bd3527b94b48
```

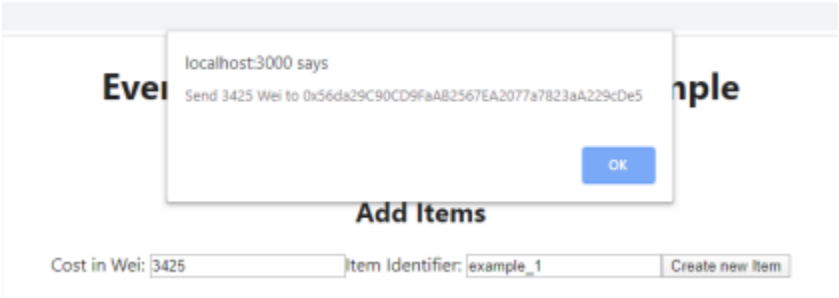
Copy the Private Key and add it into MetaMask:





Then your new Account should appear here with ~100 Ether in it.

Now let's add a new Item to our Smart Contract. You should be presented with the popup to send the message to an end-user.



Listen to Payments

Now that we know how much to pay to which address we need some sort of feedback. Obviously we don't want to wait until the customer tells us he paid, we want to know right on the spot if a payment happened.

There are multiple ways to solve this particular issue. For example you could poll the Item smart contract. You could watch the address on a low-level for incoming payments. But that's not what we want to do.

What we want is to wait for the event "SupplyChainStep" to trigger with _step == 1 (Paid).

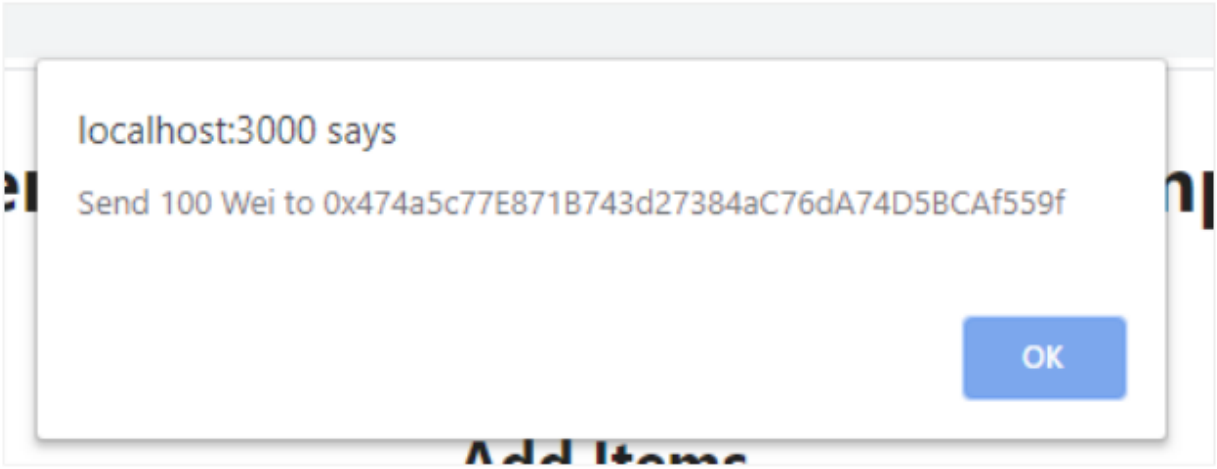
Let's add another function to the App.js file:

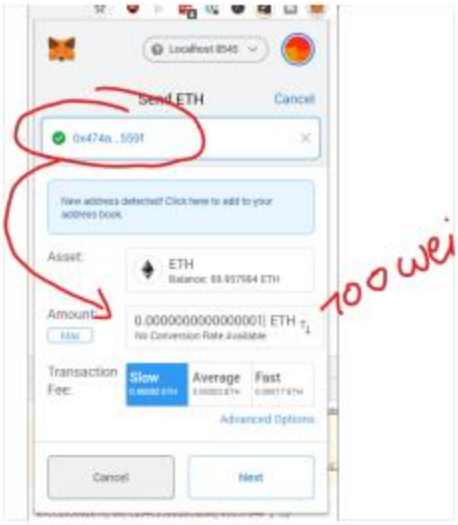
```
listenToPaymentEvent = () => {
    let self = this;
    this.itemManager.events.SupplyChainStep().on("data", async function(evt) {
      if(evt.returnValues._step == 1) {
        let item = await self.itemManager.methods.items(evt.returnValues._itemIndex).call();
        console.log(item);
        alert("Item " + item._identifier + " was paid, deliver it now!");
      };
      console.log(evt);
    });
  }
```

And call this function when we initialize the app in "componentDidMount":

```
//…
this.item = new this.web3.eth.Contract(
        ItemContract.abi,
        ItemContract.networks[this.networkId] && ItemContract.networks[this.networkId].address,
      );

      // Set web3, accounts, and contract to the state, and then proceed with an
      // example of interacting with the contract's methods.
      this.listenToPaymentEvent();
      this.setState({ loaded:true });
    } catch (error) {
      // Catch any errors for any of the above operations.
      alert(
        `Failed to load web3, accounts, or contract. Check console for details.`,
      );
      console.error(error);
    }
//...
```

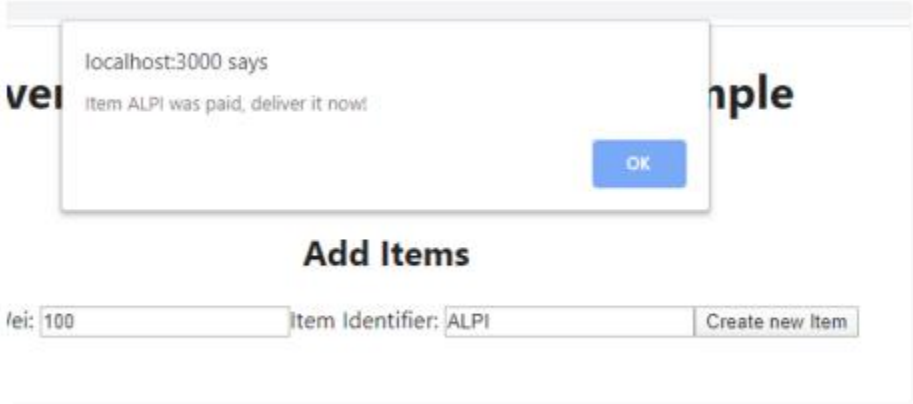
Whenever someone pays the item a new popup will appear telling you to deliver. You could also add this to a separate page, but for simplicity we just add it as an alert popup to showcase the trigger-functionality:





Take the address, give it to someone telling them to send 100 wei (0.0000000000000001 Ether) and a bit more gas to the specified address. You can do this either via MetaMask or via the truffle console:

Then a popup should appear on the website



Unit Test

Unit testing is important, that's out of the question. But how to write unit tests?

There is something special in Truffle about unit testing. The problem is that in the testing suite you get contract-abstractions using truffle-contract, while in the normal app you worked with web3-contract instances.

Let's implement a super simple unit test and see if we can test that items get created.

First of all, delete the tests in the "/test" folder. They are for the simplestorage smart contract which doesn't exist anymore.

Then add new tests:

test/ItemManager.test.js

```
const ItemManager = artifacts.require("./ItemManager.sol");

contract("ItemManager", accounts => {
it("... should let you create new Items.", async () => {
    const itemManagerInstance = await ItemManager.deployed();
    const itemName = "test1";
    const itemPrice = 500;

    const result = await itemManagerInstance.createItem(itemName, itemPrice, { from: accounts[0] });
    assert.equal(result.logs[0].args._itemIndex, 0, "There should be one item index in there")
    const item = await itemManagerInstance.items(0);
    assert.equal(item._identifier, itemName, "The item has a different identifier");
});
});
```

Truffle Contract vs Web3js

Mind the difference: In web3js you work with "instance.methods.createItem" while in truffle-contract you work with "instance.createItem". Also, the events are different. In web3js you work with result.events.returnValues and in trufflecontract you work with result.logs.args. The reason is that truffle-contract mostly took the API from web3js 0.20 and they did a major refactor for web3js 1.0.0.

Keep the truffle development console open and type in a new terminal/powershell window:

It should bring up a test like this:

```
Compiling your contracts...
===========================
> Compiling .\contracts\Item.sol
> Compiling .\contracts\ItemManager.sol
> Compiling .\contracts\Ownable.sol



  Contract: ItemManager
    √ ... should let you create new Items. (160ms)


  1 passing (216ms)

s06-eventtrigger>
```

This is how you add unit tests to your smart contracts.