

LAB 3 Supply Chain Project

3.1 Project Supply Chain

3.1.1 Penggunaan Dunia Nyata Dari Project ini

Dapat menjadi bagian solusi dari supply chain

Pengiriman otomatis pada pembayaran

Pengumpulan payment tanpa orang tengah

3.1.2 Development Goal

Showcase Event-Triggers

Memahami fungsi tingkat rendah `address.call.value()`

Pahami Alur Kerja dengan Truffle

Memahami Pengujian Unit dengan Truffle

Memahami Events dalam HTML

3.2 Smart Contract ItemManager

Pertama-tama kita membutuhkan sebuah smartcontract yang bernama "Management", kita dapat menggunakan code dibawah ini

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "../Ownable.sol";
import "../Item.sol";
contract ItemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
    mapping(uint => S_Item) public items;
    uint index;
    enum SupplyChainSteps {Created, Paid, Delivered}
    event SupplyChainStep(uint _itemIndex, uint _step, address _address);
    function createItem(string memory _identifier, uint _priceInWei) public onlyOwner {
        Item item = new Item(this, _priceInWei, index);
        items[index]._item = item;
        items[index]._step = SupplyChainSteps.Created;
        items[index]._identifier = _identifier;
        emit SupplyChainStep(index, uint(items[index]._step), address(item));
        index++;
    }
    function triggerPayment(uint _index) public payable {
        Item item = items[_index]._item;
        require(address(item) == msg.sender, "Only items are allowed to update themselves");
        require(item.priceInWei() == msg.value, "Not fully paid yet");
        require(items[_index]._step == SupplyChainSteps.Created, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Paid;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(item));
    }
}
```

Dengan ini memungkinkan untuk menambahkan barang dan membayarnya, dengan Bergeraknya maju didalam supply chain dan mentrigger pengiriman. Tetapi ada sebuah yang kita tidak sukai karena idealnya kita akan memberikan user dengan address yang mudah untuk pengiriman uang.

3.3 Smart Contract Item

Kita akan membuat satu lagi smart contract dengan nama “Item” untuk dari itu kit dapat menggunakan code dibawah ini

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./ItemManager.sol";
contract Item {
    uint public priceInWei;
    uint public paidWei;
    uint public index;
    ItemManager parentContract;
    constructor(ItemManager _parentContract, uint _priceInWei, uint _index) {
        priceInWei = _priceInWei;
        index = _index;
        parentContract = _parentContract;
    }
    receive() external payable {
        require(msg.value == priceInWei, "We don't support partial payments");
        require(paidWei == 0, "Item is already paid!");
        paidWei += msg.value;
        (bool success, ) = address(parentContract).call{value:msg.value}(abi.encodeWithSignature("triggerPayment(uint256)", index));
        require(success, "Delivery did not work");
    }
    fallback () external {
    }
}
```

Perlu kita ketahui pada solidity diatas 6.4 adanya penggantian

“call.value(msg.value)(abi.encodeWithSignature("triggerPayment(uint256)", index))”

menjadi “call{value:msg.value}(abi.encodeWithSignature("triggerPayment(uint256)", index))”

Lalu kita rubah juga pada smartcontract “Item Manager” kita agar dapat berinteraksi dengan smart contract “item” kita

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./Ownable.sol";
import "./Item.sol";
contract ItemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
    mapping(uint => S_Item) public items;
    uint index;
    enum SupplyChainSteps {Created, Paid, Delivered}
    event SupplyChainStep(uint _itemIndex, uint _step, address _address);
    function createItem(string memory _identifier, uint _priceInWei) public onlyOwner {
        Item item = new Item(this, _priceInWei, index);
        items[index]._item = item;
        items[index]._step = SupplyChainSteps.Created;
        items[index]._identifier = _identifier;
        emit SupplyChainStep(index, uint(items[index]._step), address(item));
        index++;
    }
    function triggerPayment(uint _index) public payable {
        Item item = items[_index]._item;
        require(address(item) == msg.sender, "Only items are allowed to update themselves");
        require(item.priceInWei() == msg.value, "Not fully paid yet");
        require(items[_index]._step == SupplyChainSteps.Created, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Paid;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(item));
    }
}
```

Dengan ini kita dapat hanya memberikan alamat ke customer dan mereka dapat langsung membayar dengan mengirim sekian Wei ke smartcontract. Tetapi smart contract kita masih belum aman. Kita membutuhkan sebuah fungsi kepemilikan.

3.4 Fungsi Kepemilikan

Normalnya kita akan menambahkan smartcontract OpenZeppelin dengan fungsi kepemilikannya. Tetapi Ketika penulisan ini, document belum di perbarui (masih di solidity 0.6.0). maka dari itu kita akan menambahkan fungsi kepemilikan oleh kita sendiri dan mirip dengan code OpenZeppelin. Kalian dapat membuat kode smart contract Bernama "Ownable.sol"

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;

contract Ownable {
    address public _owner;
    constructor () {
        _owner = msg.sender;
    }
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return (msg.sender == _owner);
    }
}
```

Lalu kita rubah sedikit pada smartcontract "ItemManager" kita dan kita set untuk dapat di eksekusi oleh pemilik saja

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./Ownable.sol";
import "./Item.sol";
contract ItemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
    mapping(uint => S_Item) public items;
    uint index;
    enum SupplyChainSteps {Created, Paid, Delivered}
    event SupplyChainStep(uint _itemIndex, uint _step, address _address);
    function createItem(string memory _identifier, uint _priceInWei) public onlyOwner {
        Item item = new Item(this, _priceInWei, index);
        items[index]._item = item;
        items[index]._step = SupplyChainSteps.Created;
        items[index]._identifier = _identifier;
        emit SupplyChainStep(index, uint(items[index]._step), address(item));
        index++;
    }
    function triggerPayment(uint _index) public payable {
        Item item = items[_index]._item;
        require(address(item) == msg.sender, "Only items are allowed to update themselves");
        require(item.priceInWei() == msg.value, "Not fully paid yet");
        require(items[_index]._step == SupplyChainSteps.Created, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Paid;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(item));
    }
}
```

3.5 Install Truffle

Untuk instalasi truffle pada windows kita dapat menggunakan powershell, sedangkan yang berbasis UNIX bisa menggunakan terminal.

```
PS H:\> mkdir s06-eventtrigger

Directory: H:\

Mode                LastWriteTime         Length Name
----                -
d-----          21/04/2022   13:02             s06-eventtrigger

PS H:\> cd .\s06-eventtrigger\
PS H:\s06-eventtrigger> ls
PS H:\s06-eventtrigger>
```

```
Administrator: Windows PowerShell

PS H:\> cd .\s06-eventtrigger\
PS H:\s06-eventtrigger> ls
PS H:\s06-eventtrigger> truffle unbox react

Starting unbox...
=====
Preparing to download box
Installing...
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
npm WARN old lockfile so supplemental metadata must be fetched from the registry.
npm WARN old lockfile
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile
npm WARN deprecated mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports promises now, please switch to that.
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when used in a Node.js environment. It is recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/issues/797)
npm WARN deprecated request-promise-native@1.0.8: request-promise-native has been deprecated because it extends the now deprecated request package, see https://github.com/request/request/issues/3142
npm WARN deprecated multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN deprecated source-map-url@0.4.0: See https://github.com/lydell/source-map-url#deprecated
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiformats module
npm WARN deprecated sane@4.1.0: some dependency vulnerabilities fixed, support for node < 10 dropped, and newer ECMAScript syntax/features added
npm WARN deprecated uuid@3.3.2: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that the API surface has changed to use Promises in 1.x.)
npm WARN deprecated multicodec@0.5.7: This module has been superseded by the multiformats module
npm WARN deprecated uuid@3.3.3: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
```

Cek isi dari folder trufflenya

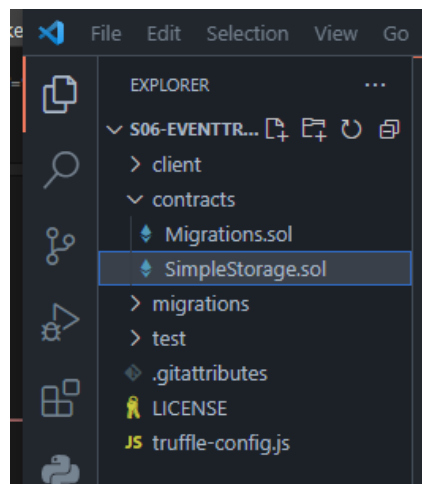
```
PS H:\s06-eventtrigger> ls

Directory: H:\s06-eventtrigger

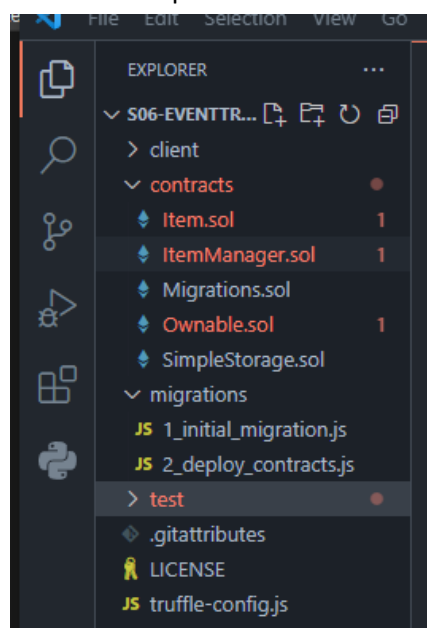
Mode                LastWriteTime         Length Name
----                -
d-----         21/04/2022    13:04             client
d-----         21/04/2022    13:03             contracts
d-----         21/04/2022    13:03             migrations
d-----         21/04/2022    13:03             test
-a-----         03/01/2022     09:13              33 .gitattributes
-a-----         03/01/2022    1075          LICENSE
-a-----         03/01/2022    297        truffle-config.js

PS H:\s06-eventtrigger>
```

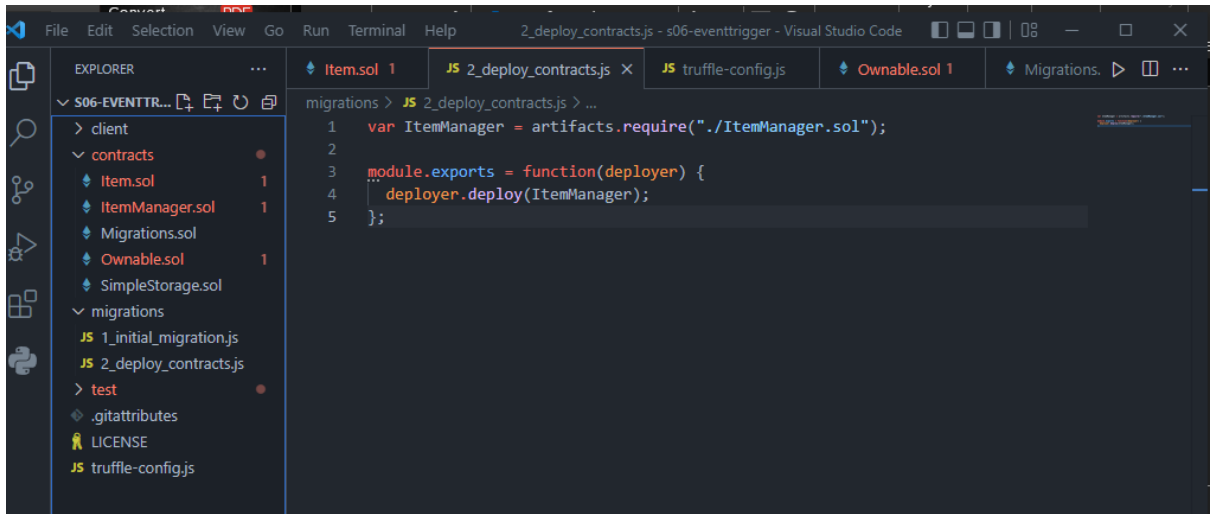
Langkah selanjutnya kita buka text editor kita, disini saya memakai visual studio code. Setelah di buka arahkan ke folder yang kita sudah buat sebelumnya lalu di dalam folder contract hapus file "SimpleStorage.sol"



Setelah menghapus file "SimpleStorage.sol" kita masukan kontrak kita yang sebelumnya sudah kita siapkan dari remix. Jika ada pesan error maka kita abaikan saja terlebih dahulu.

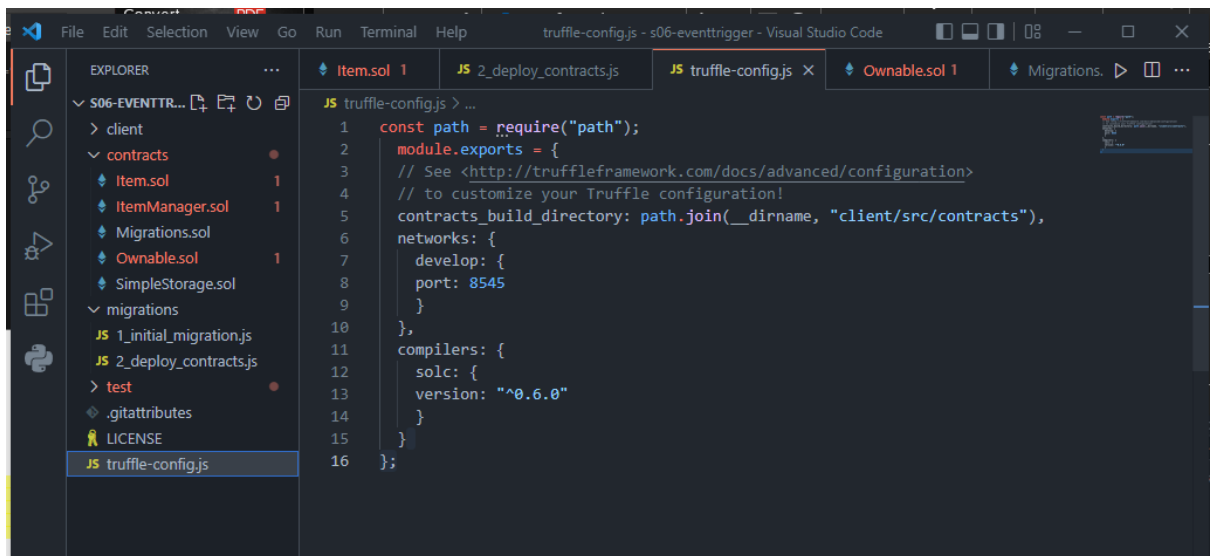


Setelah itu kita dapat merubah di environment kita agar dapat di compile



This screenshot shows the Visual Studio Code editor with the file explorer on the left and the editor window on the right. The file explorer shows a project structure with folders 'client', 'contracts', and 'migrations'. The 'contracts' folder contains 'Item.sol', 'ItemManager.sol', 'Ownable.sol', and 'SimpleStorage.sol'. The 'migrations' folder contains '1_initial_migration.js' and '2_deploy_contracts.js'. The editor window shows the content of '2_deploy_contracts.js'.

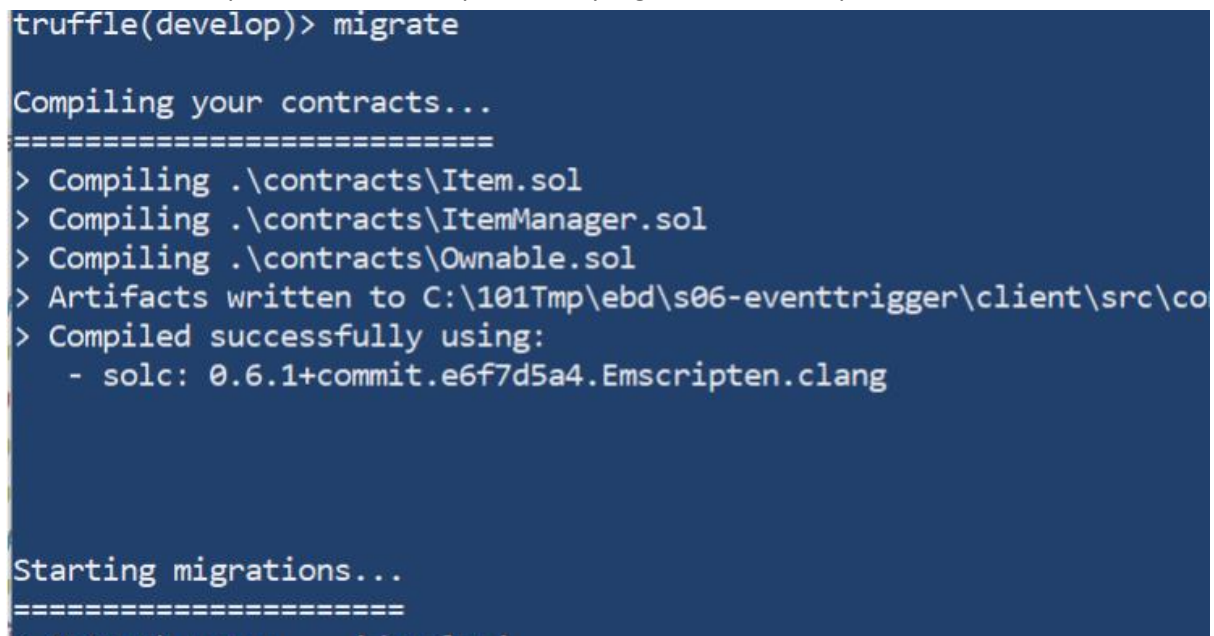
```
migrations > JS 2_deploy_contracts.js > ...
1  var ItemManager = artifacts.require("../ItemManager.sol");
2
3  module.exports = function(deployer) {
4    deployer.deploy(ItemManager);
5  };
```



This screenshot shows the Visual Studio Code editor with the file explorer on the left and the editor window on the right. The file explorer shows the same project structure as the previous screenshot. The editor window shows the content of 'truffle-config.js'.

```
JS truffle-config.js > ...
1  const path = require("path");
2  module.exports = {
3    // See <http://truffleframework.com/docs/advanced/configuration>
4    // to customize your Truffle configuration!
5    contracts_build_directory: path.join(__dirname, "client/src/contracts"),
6    networks: {
7      develop: {
8        port: 8545
9      }
10   },
11   compilers: {
12     solc: {
13       version: "^0.6.0"
14     }
15   }
16   };
```

Lalu dari sini kita dapat cek di console apakah ada yang masih belum tepat



This screenshot shows a terminal window with the output of the 'truffle(develop)> migrate' command. The output shows the compilation of the contracts and the successful migration.

```
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: 1: develop
```

3.6 Modifikasi HTML

Sekarang kita modifikasi HTML agar dapat berinteraksi dengan smartcontract kita di browser.

Buka di “client/app.js” dan modifikasi di bagian yang diberi warna kuning

```
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 ...
}
```

Lalu tambahkan render() seperti di bawah ini

```
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>

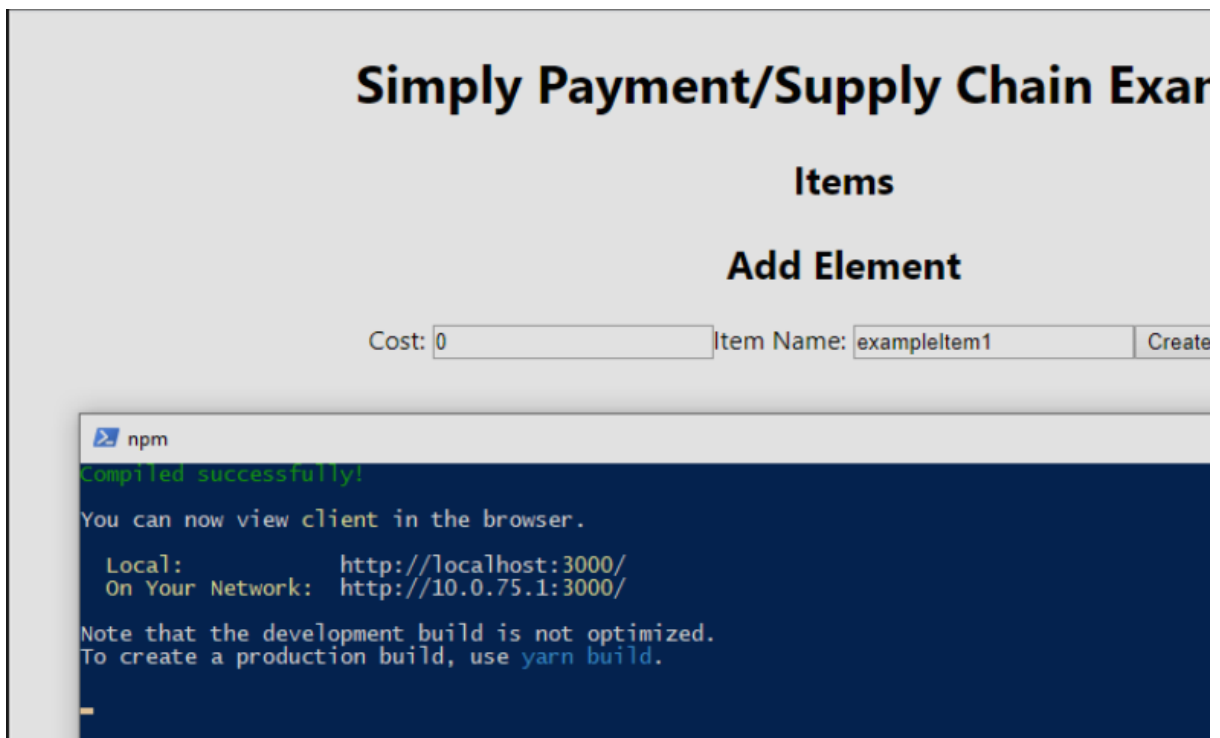
      <h2>Add Element</h2>
      Cost: <input type="text" name="cost" value={this.state.cost} onChange={this.handleInputChange} />
      Item Name: <input type="text" name="itemName" value={this.state.itemName} onChange={this.handleInputChange} />
      <button type="button" onClick={this.handleSubmit}>Create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

Tambahkan dua fungsi, satu untuk handleSubmit, sehingga all input variable dapat di set dengan benar

```
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
  });
};
```

Buka lagi terminal lalu jalankan npm dengan ini akan menjalankan server dengan port 300 di browser



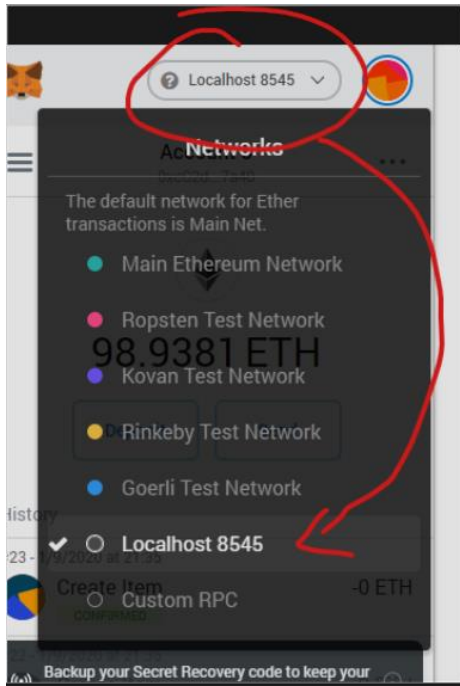
3.7 Koneksikan dengan MetaMask

3.7.1 Apa Yang Kita Kerjakan?

Pada bagian ini kita akan mengkoneksikan react app dengan metamask dengan keystore ke sign transaction

3.7.2 Langkah-langkah

Pertama-tama koneksikan dengan metamask dengan network yang benar

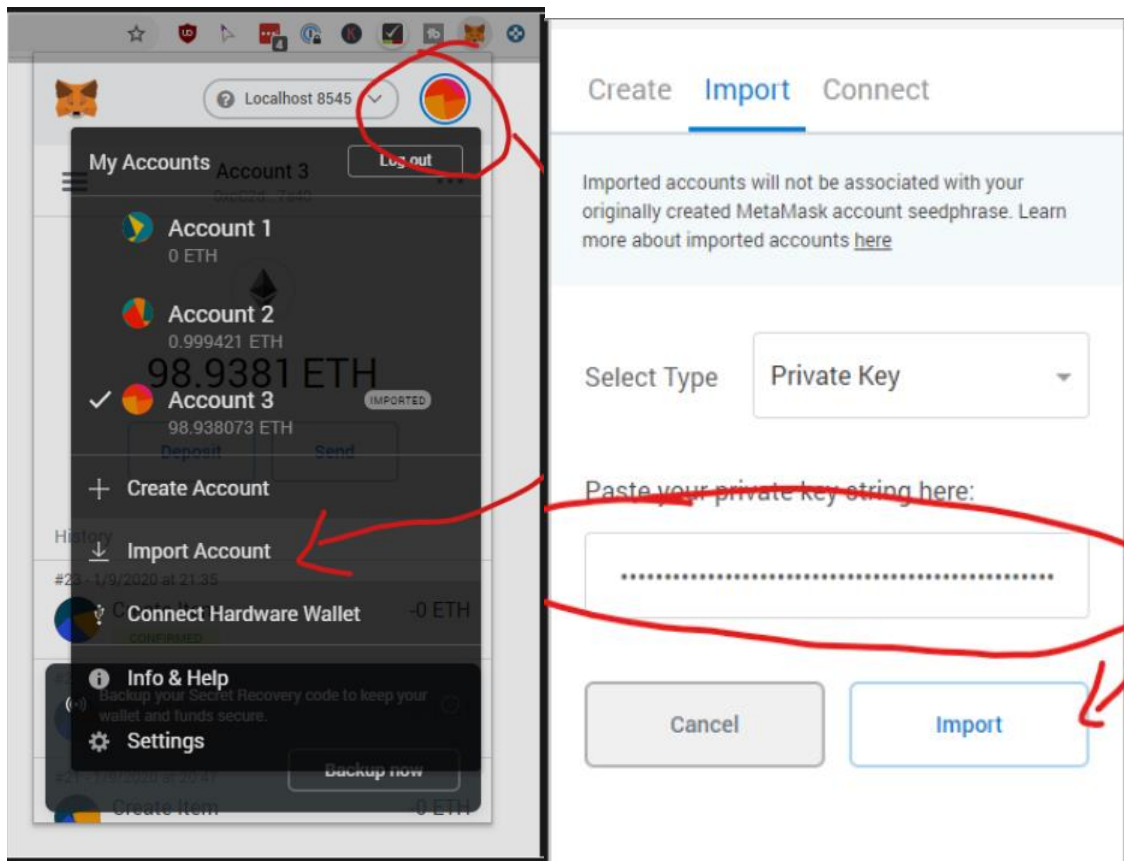


Saat kami memigrasikan kontrak pintar dengan konsol Pengembang Truffle, maka akun pertama di konsol pengembang truffle adalah pemilik". Jadi, kami menonaktifkan MetaMask di Browser untuk berinteraksi dengan aplikasi atau kami menambahkan kunci pribadi dari truffle konsol pengembang ke MetaMask. Di Terminal/Powershell tempat Truffle Developer Console menjalankan gulir ke kunci pribadi di atas:

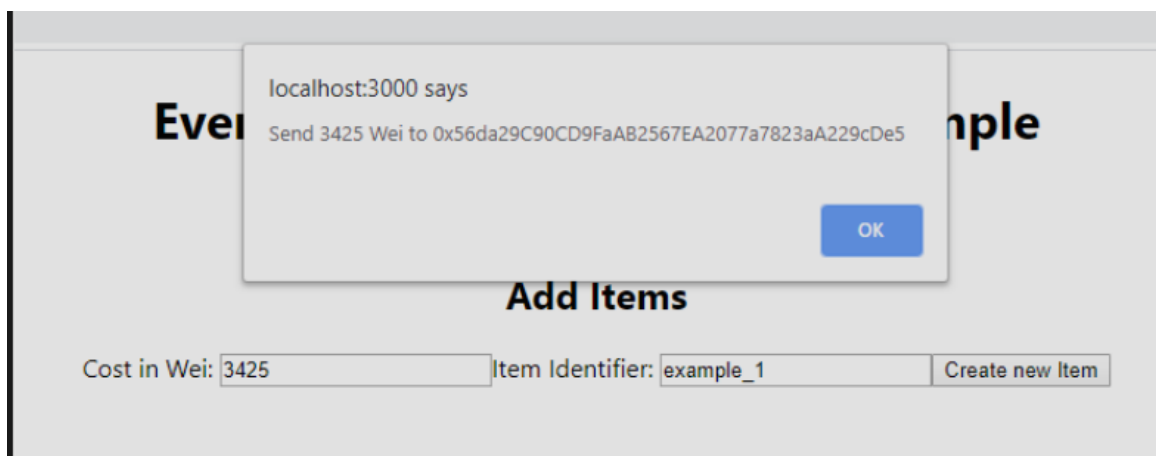
Private Keys:

```
(0) 2a9ed36cdb66f81093a82443c2b9f237f3534ef75f4f044fa6ebd76d5d05f61
(1) f9c941a67e63fe4b84fe63ad652c29b2f225eb57562b246bf44bd3527b94b48
```

Copy private key kita ke metamask



Dari sini seharusnya akun kita akan muncul dengan saldo 100 Ether didalamnya
Sekarang kita tambahkan new item ke smartcontract



3.8 Listen to Payment

Sekarang kita tahu berapa yang kita harus bayar untuk spesifik alamat
Tambahkan fungsi dibawah ini ke app.js

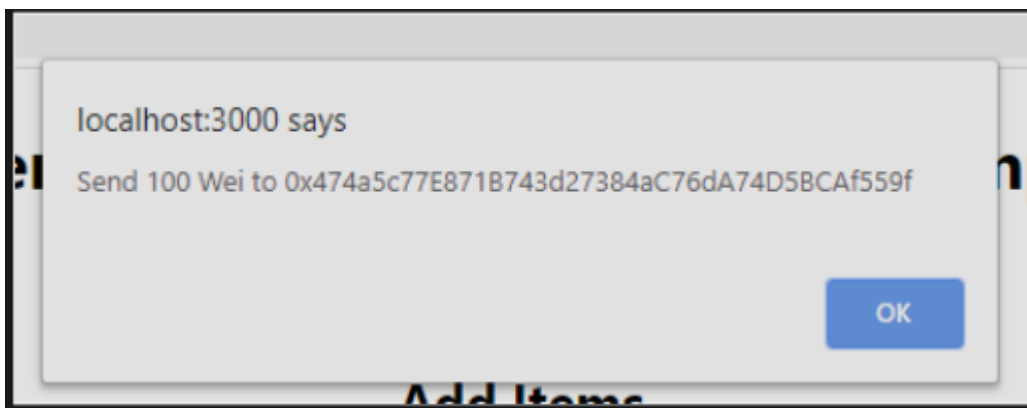
```
listenToPaymentEvent = () => {
  let self = this;
  this.itemManager.events_SUPPLY_CHAIN_STEP.on("data", async function(evt) {
    if(evt.returnValues._step == 1) {
      let item = await self.itemManager.methods.items(evt.returnValues._itemIndex).call();
      console.log(item);
      alert("Item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

Dan call fungsi ini Ketika kita inisialisasi aplikasi di “ComponenDidMount”

```
//-
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);
}
//...
```

Ketika ada seseorang membayart maka item akan akan pop up dan meberitahukan anda untuk mengirim



3.9 Unit Test

Pengujian unit itu penting, itu tidak mungkin. Tetapi bagaimana cara menulis tes unit? Ada sesuatu yang istimewa di Truffle tentang pengujian unit. Masalahnya adalah di suite pengujian Anda mendapatkan abstraksi kontrak menggunakan truffle-contract, sementara di aplikasi normal Anda bekerja dengan instance web3-contract. Mari kita terapkan unit test super sederhana dan lihat apakah kita bisa menguji item yang dibuat. Pertama-tama, hapus tes di folder "/test". Mereka adalah untuk kontrak pintar penyimpanan paling sederhana yang tidak ada lagi. Kemudian tambahkan tes baru:

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");
  });
});
```