Nama : Putri Mellia Zahrani

NIM : 1103190143

**Lab 3 : Supply Chain**

1. **Real World Use Case for this Project**

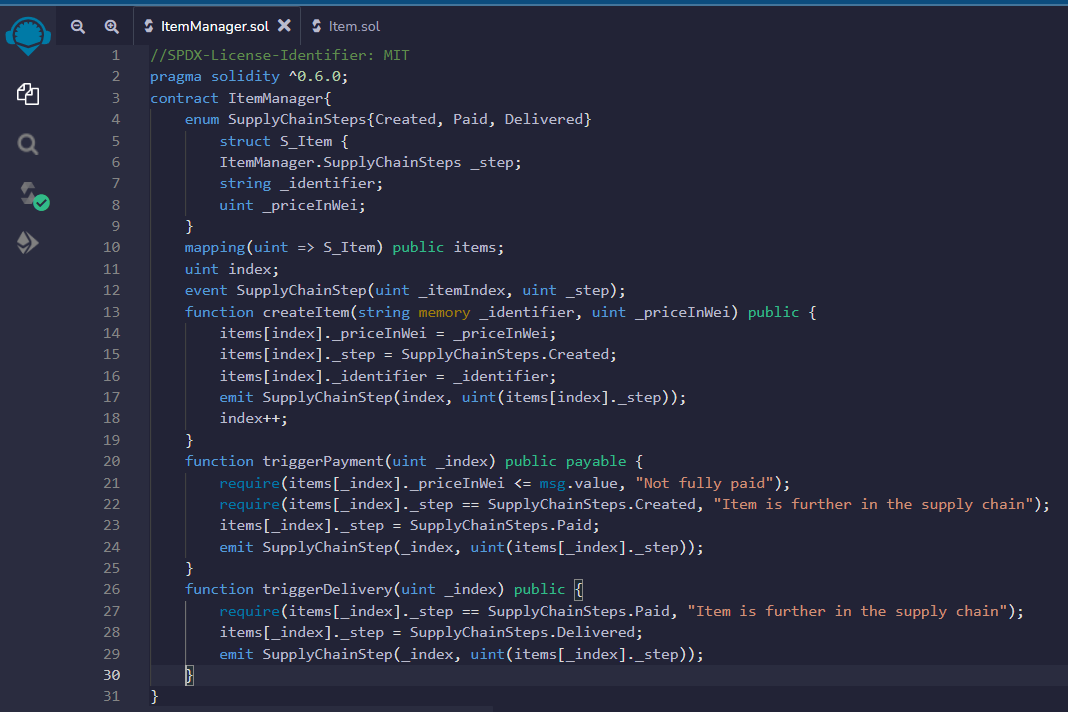
Penggunaan project secara nyata adalah :

1. Dapat menjadi bagian dari solusi rantai suplai
2. Pengiriman Otomatis setelah pembayaran
3. Penagihan pembayaran tanpa perantara
4. **Development Goal**

Tujuan Pengembangan :

1. Showcase Even-Triggers
2. Memahami fungsi tingkat rendah address.call.value()()
3. Pahami Alur Kerja dengan Truffle
4. Memahami Pengujian Unit dengan Truffle
5. Memahami Event dalam HTML
6. **The ItemManager Smart Contract**

Hal pertama yang kita butuhkan adalah “Management” Smart Contract, dimana kita dapat menambahkan item.



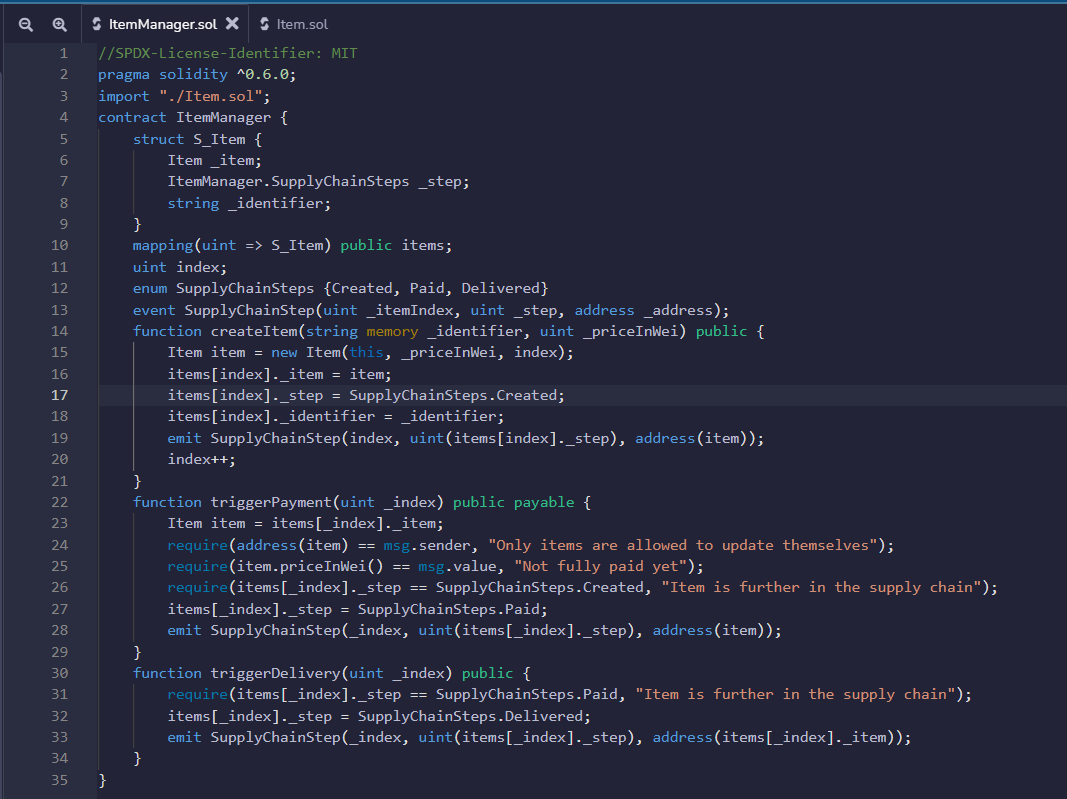
Dengan Smart Contract diatas kita sangat mungkin untuk menambahkan items dan membayarnya, memindahkannya ke depan dalam supply chain dan memicu pengiriman.

1. **Item Smart Contract**



Perhatikan bahwa call.value(msg.value)(abi.encodeWithSignature("triggerPayment(uint256)", index)) , karena perubahan pada Solidity versi 6.4 disarankan harus diubah menjadi call{value:msg.value}(abi.encodeWithSignature("triggerPayment(uint256)", indeks)).

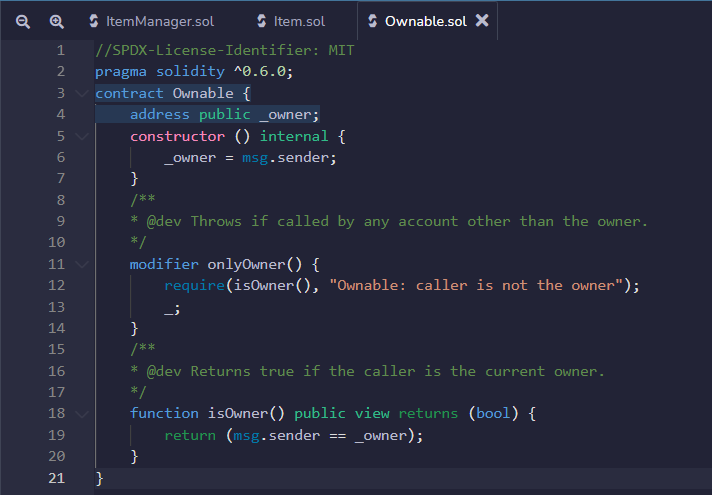
Ubah ItemManager Smart Contract untuk menggunakan Item Smart Contract tidak hanya Struct nya saja :



Sekarang dengan ini dapat memberi pelanggan alamat Smart Contract Item yang dibuat selama "createItem" dan dia akan dapat membayar langsung dengan mengirimkan X Wei ke Smart Contract. Tetapi Smart Contract belum terlalu aman. sehingga dibutuhkan semacam fungsi pemilik (owner).

1. **Ownable Functionality**

Biasanya kami akan menambahkan Smart Contract OpenZeppelin dengan fungsi Ownable. Tetapi pada saat menulis ini dokumen mereka belum diperbarui ke solidity 0.6. Jadi, alih-alih kami akan menambahkan fungsi Ownable kami sendiri sangat mirip dengan satu dari OpenZeppelin.



Kemudian ubah ItemManager sehingga semua fungsi, yang seharusnya dapat dieksekusi oleh "pemilik saja" memiliki modifier yang benar:



1. **Install Truffle**

Untuk menginstall truffle buka terminal (Mac/Linux) atau PowerShell (Windows10).

1. Tuliskan perintah berikut untuk melakukan instalasi truffle :

|  |
| --- |
| npm install -g truffle |
| Menggunakan output dimodul dikarenakan program yang saya jalankan error. |

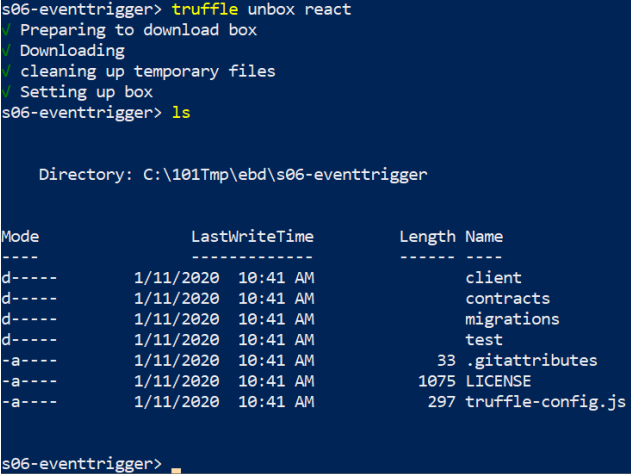
1. Kemudian, buat folder kosong dengan nama “s06-eventtrigger” dengan perintah dibawah :

|  |
| --- |
| mkdir s06-eventtrigger  cd s06-eventtrigger  ls |
|  |

1. Lalu unbox the react box

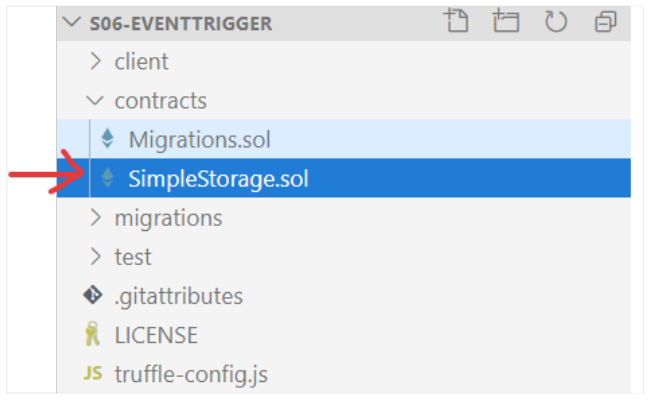
|  |
| --- |
| truffle unbox react |

Kemudian mengunduh repositori dan menginstal semua dependensi di folder saat ini:

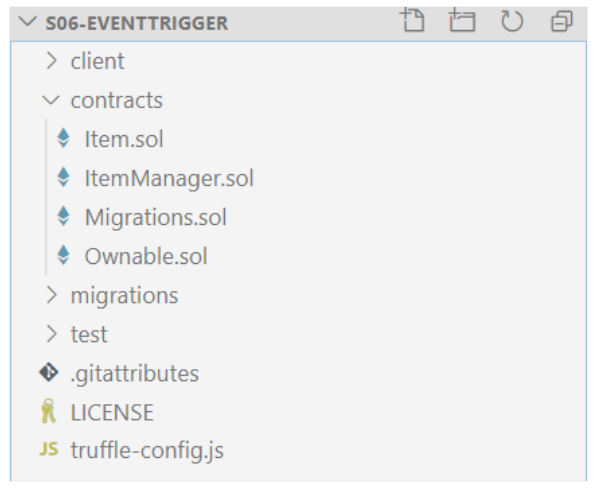


1. **Add Contract**

Hapus Smart Contract SimpleStorage yang ada tetapi biarkan file "Migrations.sol":



Tambahkan di file :



Kemudian ubah file "migrasi" di folder migrasi/:

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

module.exports = function(deployer) {

    deployer.deploy(ItemManager);

};

Ubah file truffle-config.js untuk mengunci versi kompiler tertentu:

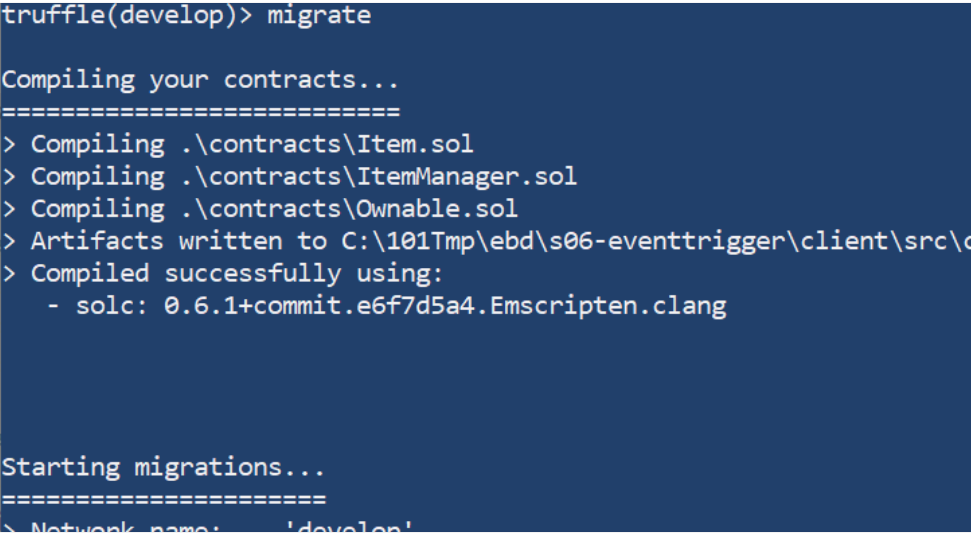
|  |
| --- |
| const path = require("path");    module.exports = {    // See <http://truffleframework.com/docs/advanced/configuration>    // to customize your Truffle configuration!    contracts\_build\_directory: path.join(\_\_dirname, "client/src/contracts"),    networks: {      develop: {        port: 8545      }    },    compilers: {      solc: {      version: "^0.6.0"      }    }  }; |

Run konsol pengembangan truffle untuk memeriksa apakah semuanya baik-baik saja dan dapat dimigrasikan. Di terminal/powershell jalankan

|  |
| --- |
| truffle develop |

Kemudian ketikkan

|  |
| --- |
| migrate |



1. **Modify HTML**

Sekarang saatnya memodifikasi HTML kita sehingga kita benar-benar dapat berinteraksi dengan Smart Contract dari Browser.

Buka "client/App.js" dan ubah beberapa hal di dalam file:

|  |
| --- |
| import React, { Component } from "react";  import ItemManager from "./contracts/ItemManager.json";  import Item from "./contracts/Item.json";  // import SimpleStorageContract from "./contracts/SimpleStorage.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,        );        // Set web3, accounts, and contract to the state, and then proceed with an        // example of interacting with the contract's methods.        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);      } |

Kemudian tambahkan form ke bagian HTML di ujung bawah file App.js, di fungsi "render".

|  |
| --- |
| 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>      );    } |

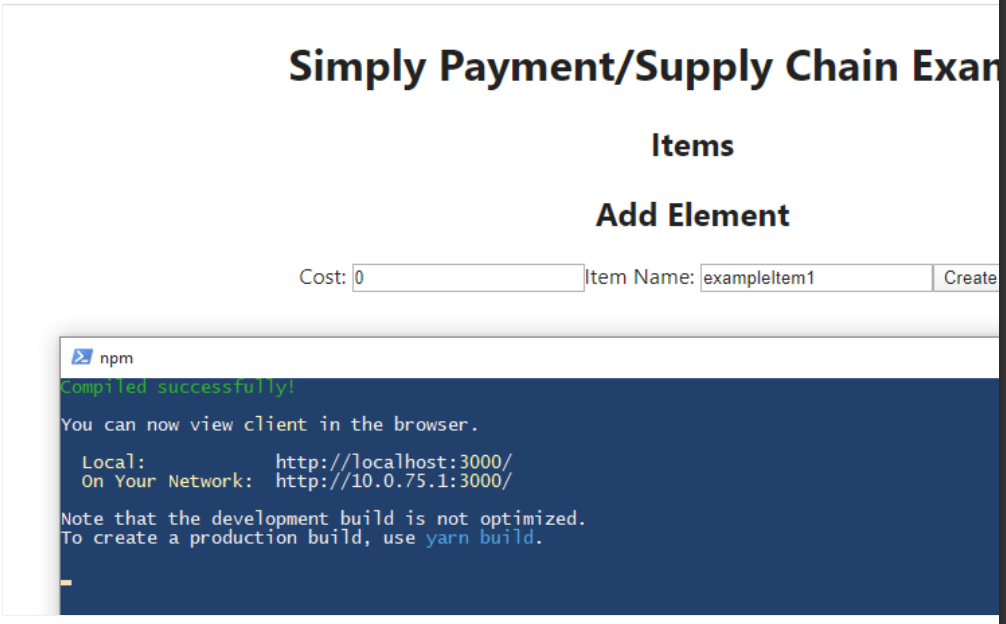
Tambahkan dua fungsi, satu untuk handleInputChange, sehingga semua variabel input diatur dengan benar. Dan satu untuk mengirim transaksi aktual ke jaringan:

|  |
| --- |
| 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 terminal atau poweshell lain (biarkan yang sudah Anda buka dengan truffle) dan pergi ke klien folder dan jalankan.

|  |
| --- |
| npm start |

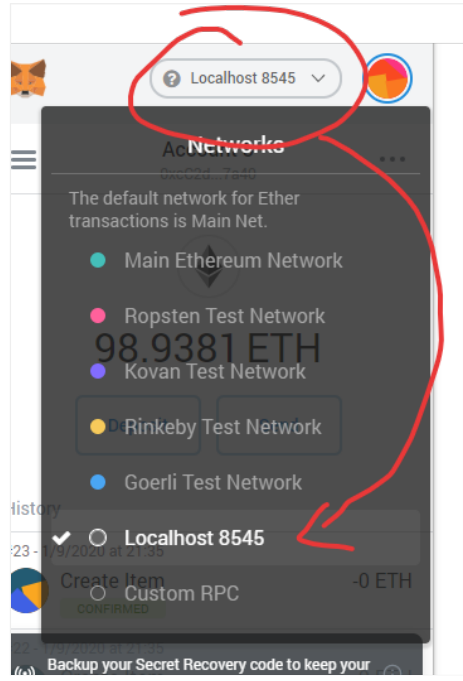
Dengan melakukan npm start maka akan memulai server pengembangan pada port 3000 dan akan membuka tab baru di browser Anda :



Jika Anda melihat pesan kesalahan bahwa jaringan tidak ditemukan atau kontrak tidak ditemukan di bawah alamat yang diberikan – jangan khawatir: Ikuti langkah selanjutnya di mana Anda mengubah jaringan di MetaMask!

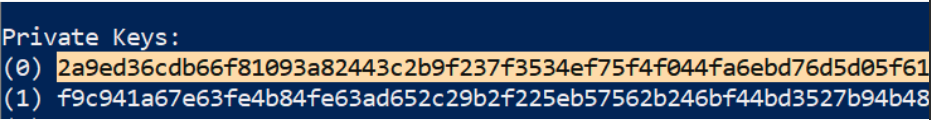
1. **Connect With MetaMask**

Pertama, hubungkan dengan MetaMask ke jaringan yang tepat :

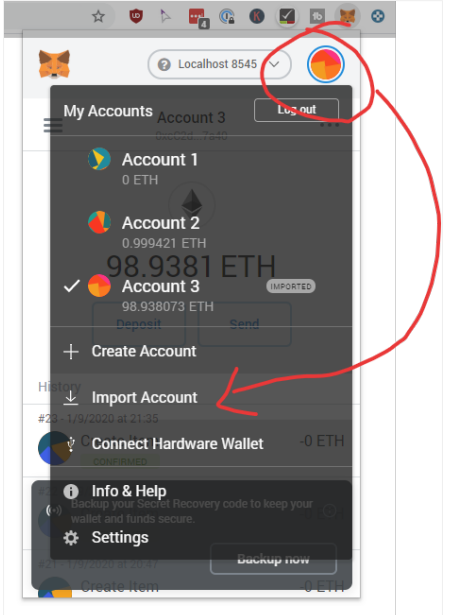


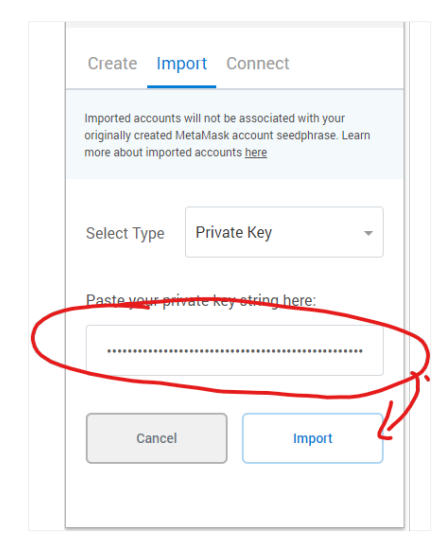
Saat kami memigrasikan kontrak pintar dengan konsol Pengembang Truffle, maka akun pertama di konsol pengembang truffle adalah “pemilik (owner)”. Jadi, perlu menonaktifkan MetaMask di Browser untuk berinteraksi dengan aplikasi atau perlu menambahkan kunci pribadi dari truffle developer console ke MetaMask.

Di Terminal/Powershell tempat Truffle Developer Console menjalankan scroll ke kunci pribadi di atas:



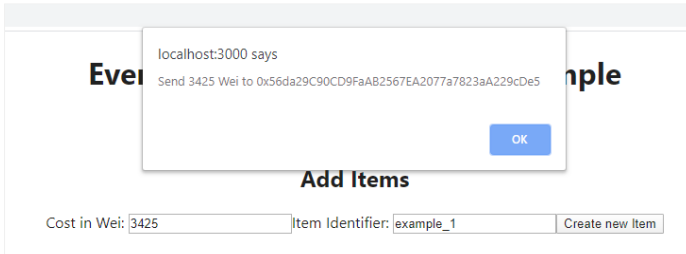
Copy private key dan masukkan ke MetaMask :





Kemudian Akun baru Anda akan muncul di sini dengan ~100 Eter di dalamnya.

Sekarang tambahkan Item baru ke Smart Contract. Anda harus disajikan dengan popup untuk mengirim pesan ke End-user.



1. **Listen to Payment**

Sekarang kita tahu berapa banyak yang harus dibayar ke alamat mana kita membutuhkan semacam umpan balik. Jelas kami tidak ingin menunggu sampai pelanggan memberi tahu kami bahwa dia membayar, kami ingin tahu langsung di tempat jika pembayaran terjadi.

Ada beberapa cara untuk memecahkan masalah khusus ini. Misalnya, Anda dapat melakukan polling pada kontrak pintar Item. Anda bisa menonton alamat di tingkat rendah untuk pembayaran masuk. Tapi bukan itu yang ingin kami lakukan.

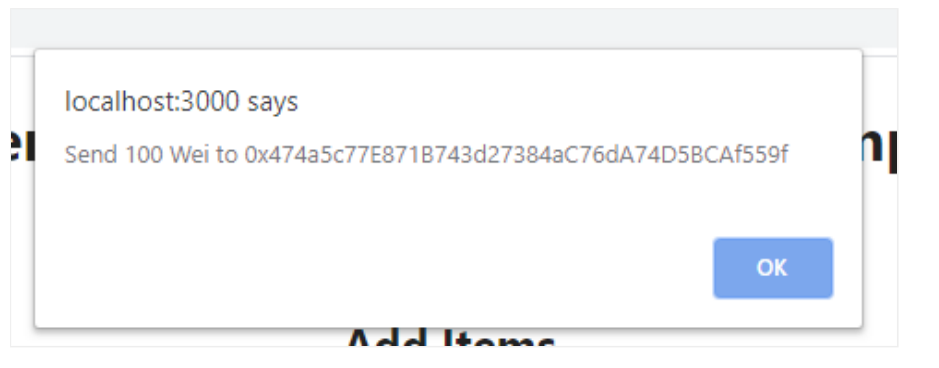
Yang kami inginkan adalah menunggu event "SupplyChainStep" dipicu with\_step == 1 (Berbayar). Mari tambahkan fungsi lain ke file App.js:

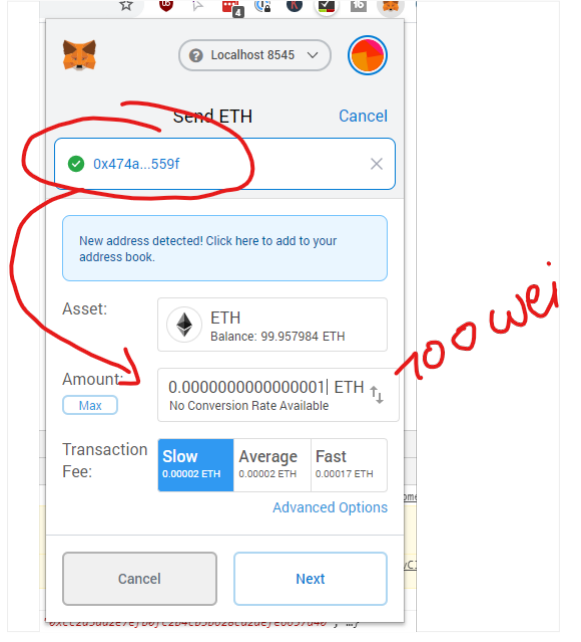
|  |
| --- |
| listenToPaymentEvent = () => {      let self = this;      this.itemManager.events.SupplyChainStep().on("data", async function(evt) {      if(evt.returnValues.\_step == 1) {      let item = await self.itemManager.methods.items(evt.returnValues.\_itemIndex).call();      console.log(item);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

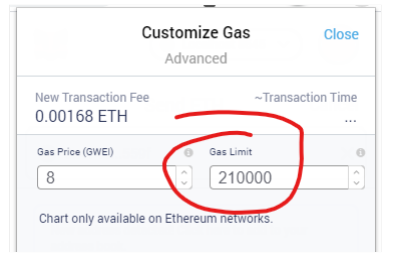
Dan panggil fungsi ketika kita menginisialisasikan aplikasi di “componentDidMount”:

|  |
| --- |
| 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);      } |

Setiap kali seseorang membayar barang tersebut, sebuah popup baru akan muncul memberitahu Anda untuk mengirimkannya. Anda juga dapat menambahkan ini ke halaman terpisah, tetapi untuk kesederhanaan, kami hanya menambahkannya sebagai pop up untuk menampilkan trigger-functionality :



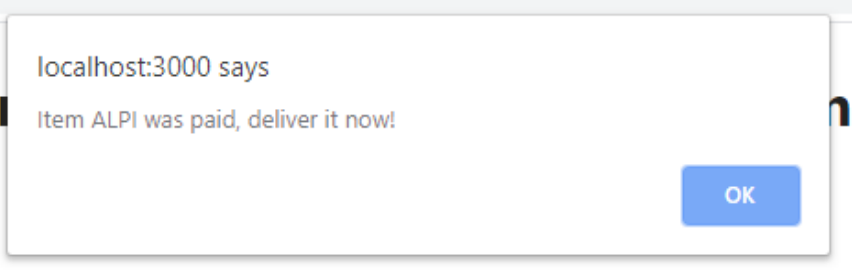




Ambil alamatnya, berikan kepada seseorang yang menyuruh mereka mengirim 100 wei (0,0000000000000001 Ether) dan sedikit lebih banyak gas ke alamat yang ditentukan. Anda dapat melakukannya melalui MetaMask atau melalui konsol truffle:

|  |
| --- |
| Web3.eth.sendTransaction({to: “ITEM\_ADDRESS”, value: 100, form: account[1], gas: 2000000}); |

Maka akan mucul popup pada website



1. **Unit Test**

Pertama-tama, hapus tes di folder "/test". Mereka adalah untuk smart contract penyimpanan paling sederhana yang tidak ada lagi.Kemudian tambahkan tes baru:

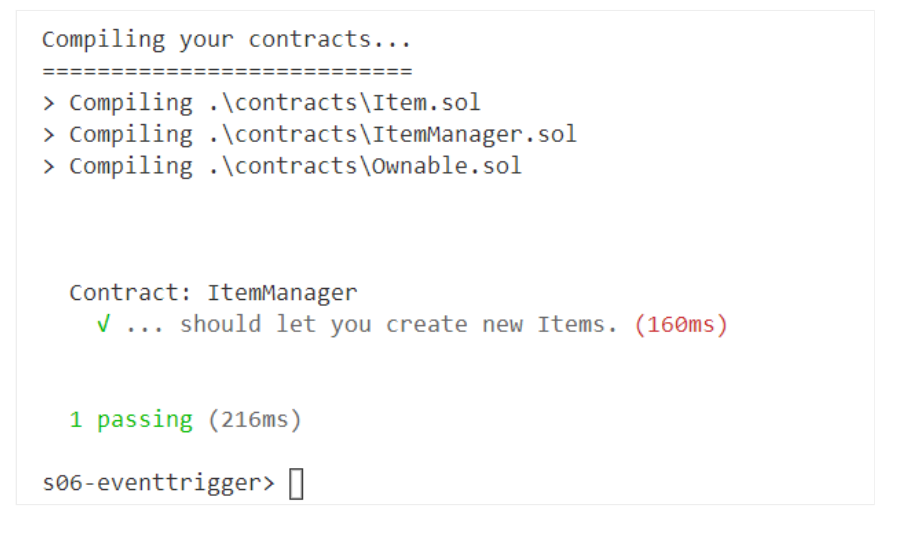
|  |
| --- |
| const ItemManager = artifacts.require("./ItemManager.sol");    contract("ItemManager", accounts => {  it("... should let you create new Items.", async () => {      const itemManagerInstance = await ItemManager.deployed();      const itemName = "test1";      const itemPrice = 500;     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

Perhatikan perbedaannya: Di web3js Anda bekerja dengan "instance.methods.createItem" sementara di truffle-contract Anda bekerja dengan "instance.createItem". Selain itu, acaranya juga berbeda. Di web3js Anda bekerja dengan result.events.returnValues dan di truffle- contract Anda bekerja dengan result.logs.args. Alasannya adalah bahwa truffle-contract sebagian besar mengambil API dari web3js 0.20 dan mereka melakukan refactor utama untuk web3js 1.0.0.

Biarkan konsol truffle development terbuka dan tulis di terminal yang baru :

|  |
| --- |
| truffle test |

Maka hasilnya akan seperti ini :



Begitulah cara kamu menambahkan unit tests ke Smart Contract.