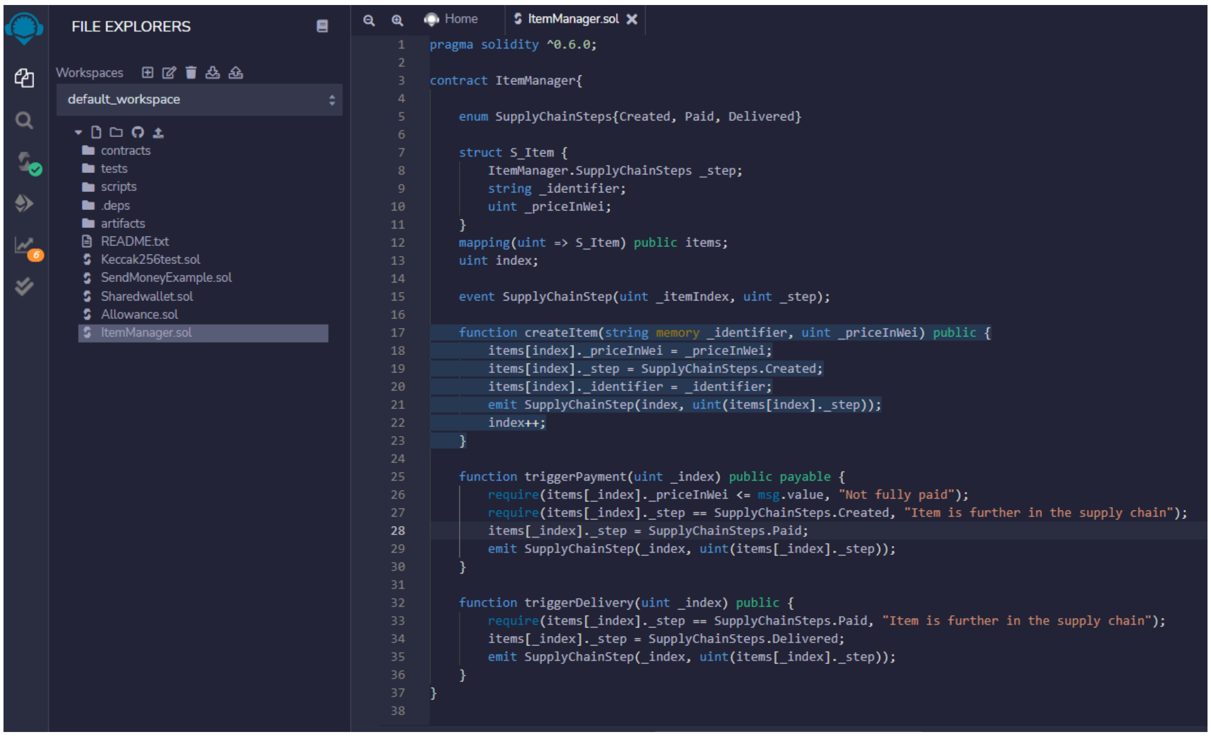
## 15.2 The ItemManager Smart Contract

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

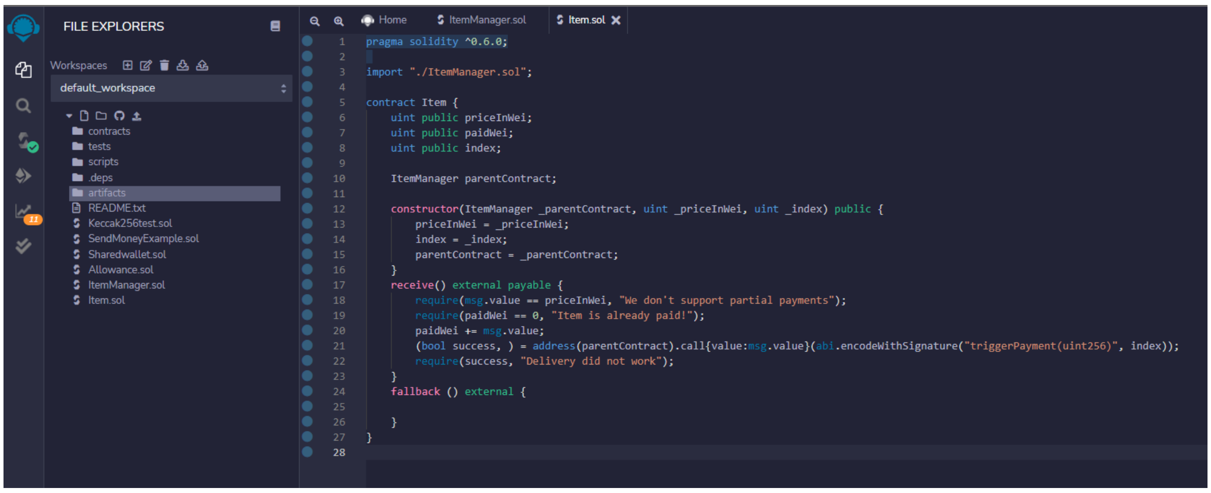


```
pragma solidity ^0.6.0;

contract ItemManager{

    enum SupplyChainSteps{Created, Paid, Delivered}

    struct S_Item {
        ItemManager.SupplyChainSteps _step;
        string _identifier;
        uint _priceInWei;
    }
    mapping(uint => S_Item) public items;
    uint index;

    event SupplyChainStep(uint _itemIndex, uint _step);

    function createItem(string memory _identifier, uint _priceInWei) public {
        items[index]._priceInWei = _priceInWei;
        items[index]._step = SupplyChainSteps.Created;
        items[index]._identifier = _identifier;
        emit SupplyChainStep(index, uint(items[index]._step));
        index++;
    }

    function triggerPayment(uint _index) public payable {
        require(items[_index]._priceInWei <= msg.value, "Not fully paid");
        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));
    }

    function triggerDelivery(uint _index) public {
        require(items[_index]._step == SupplyChainSteps.Paid, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Delivered;
        emit SupplyChainStep(_index, uint(items[_index]._step));
    }
}
```

With this it's possible to add items and pay them, move them forward in the supply chain and trigger a delivery. But that's something I don't like, because ideally I just want to give the user a simple address to send money to.

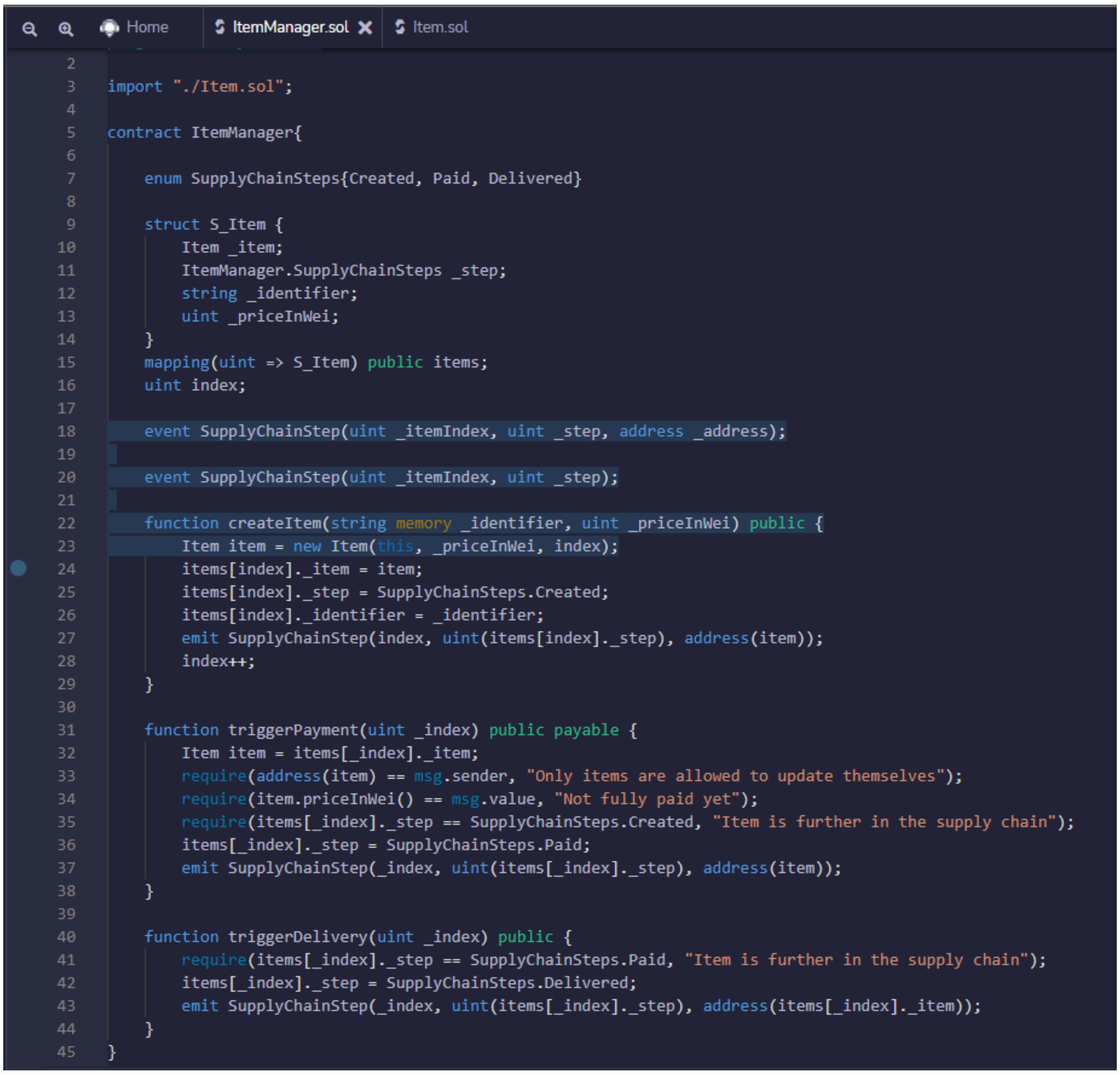
## 15.3 Item Smart Contract

Let's add another smart contract:



```
pragma solidity ^0.6.0;

import "./ItemManager.sol";

contract Item {
    uint public priceInWei;
    uint public paidWei;
    uint public index;

    ItemManager parentContract;

    constructor(ItemManager _parentContract, uint _priceInWei, uint _index) public {
        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 {

    }
}
```

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

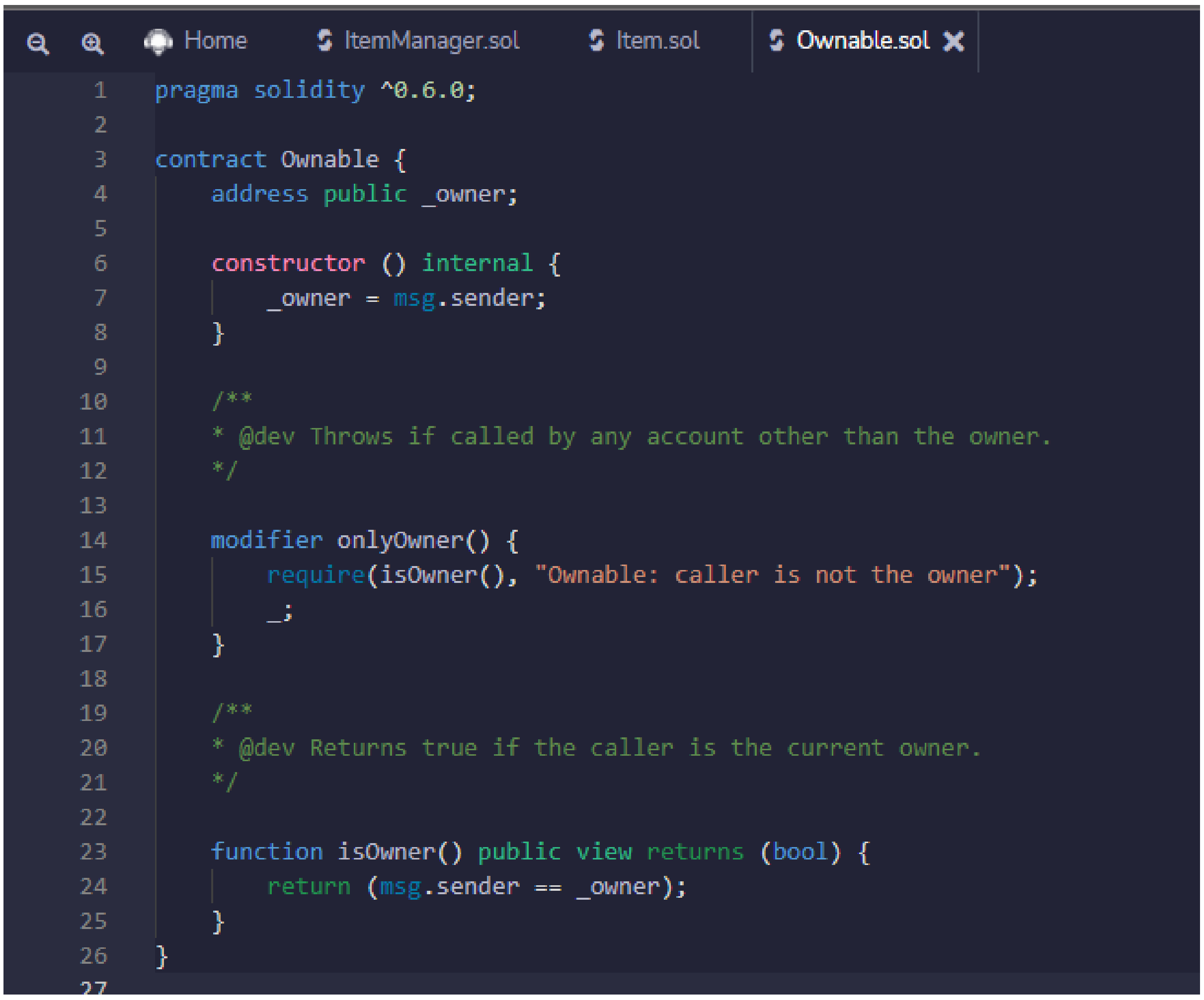


```

import "./Item.sol";

contract ItemManager{

    enum SupplyChainSteps{Created, Paid, Delivered}

    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
        uint _priceInWei;
    }
    mapping(uint => S_Item) public items;
    uint index;

    event SupplyChainStep(uint _itemIndex, uint _step, address _address);

    event SupplyChainStep(uint _itemIndex, uint _step);

    function createItem(string memory _identifier, uint _priceInWei) public {
        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));
    }

    function triggerDelivery(uint _index) public {
        require(items[_index]._step == SupplyChainSteps.Paid, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Delivered;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(items[_index]._item));
    }
}
```

Now with this we just have to give a customer the address of the Item Smart Contract created during "createItem" and he will be able to pay directly by sending X Wei to the Smart Contract. But the smart contract isn't very secure yet. We need some sort of owner functionality.

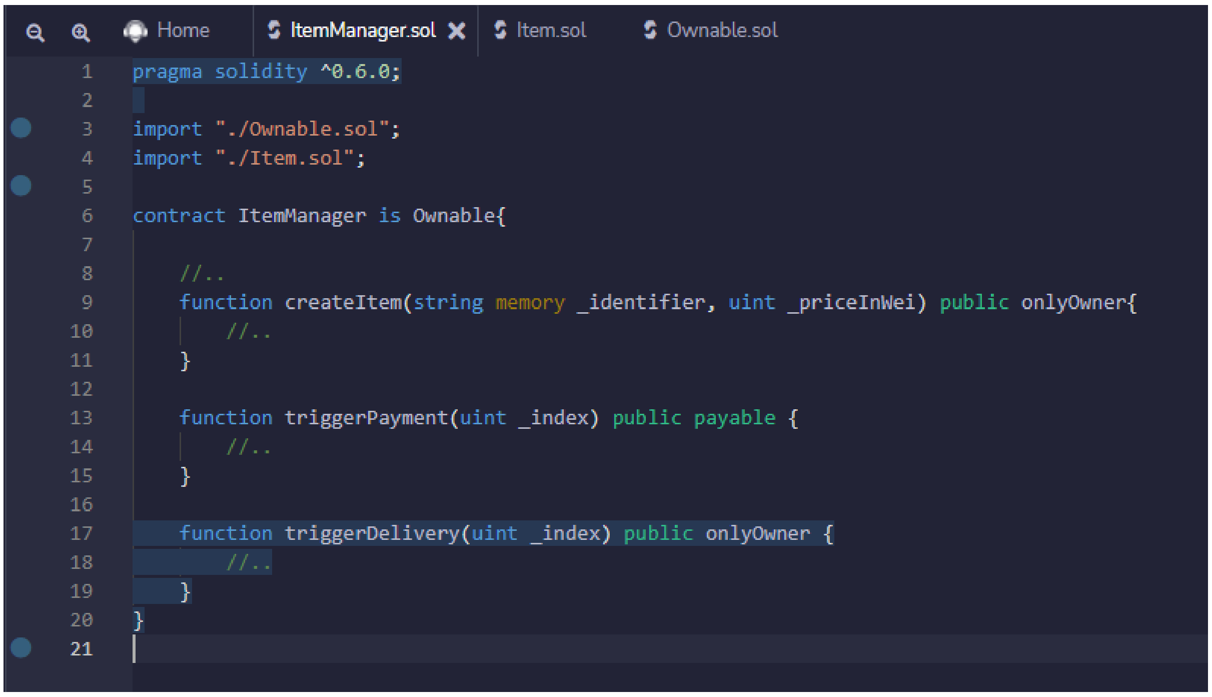
## 15.4 Ownable Functionality

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:



```
pragma solidity ^0.6.0;

contract Ownable {
    address public _owner;

    constructor () internal {
        _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);
    }
}
```

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

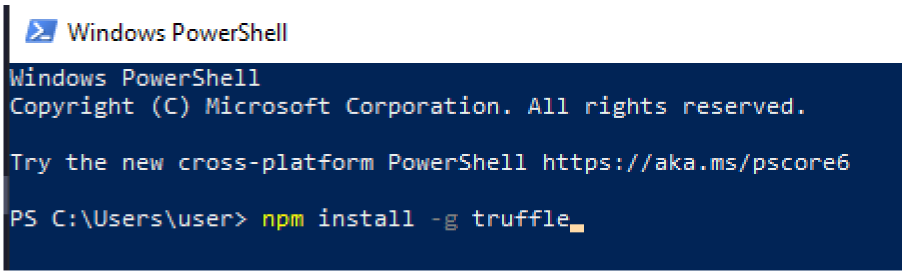


```
pragma solidity ^0.6.0;

import "./Ownable.sol";
import "./Item.sol";

contract ItemManager is Ownable{

    //..
    function createItem(string memory _identifier, uint _priceInWei) public onlyOwner{
        //..
    }

    function triggerPayment(uint _index) public payable {
        //..
    }

    function triggerDelivery(uint _index) public onlyOwner {
        //..
    }
}
```

## 15.5 Install Truffle

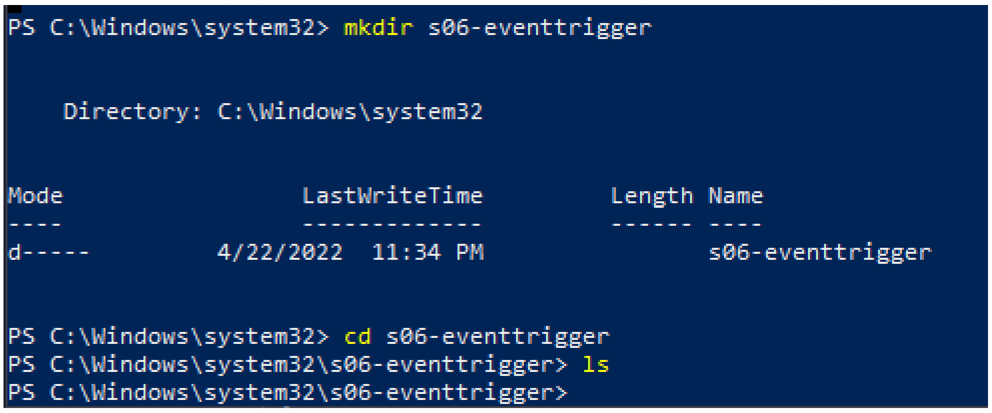
To install truffle open a terminal (Mac/Linux) or a PowerShell (Windows 10)

Type in: npm install -g truffle



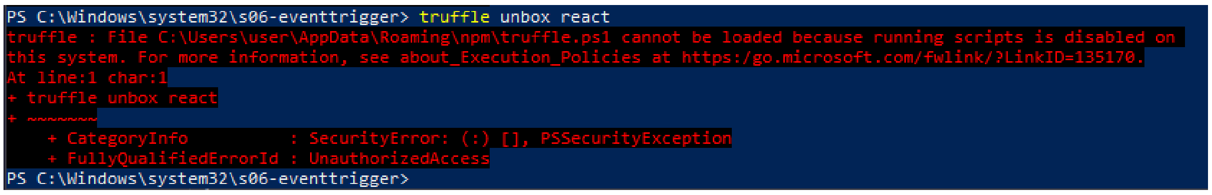
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

PS C:\Users\user> npm install -g truffle_
```

Then create an empty folder, in this case I am creating "s06-eventtrigger"



```
PS C:\Windows\system32> mkdir s06-eventtrigger


    Directory: C:\Windows\system32


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/22/2022  11:34 PM                s06-eventtrigger


PS C:\Windows\system32> cd s06-eventtrigger
PS C:\Windows\system32\s06-eventtrigger> ls
PS C:\Windows\system32\s06-eventtrigger>
```

And unbox the react box:

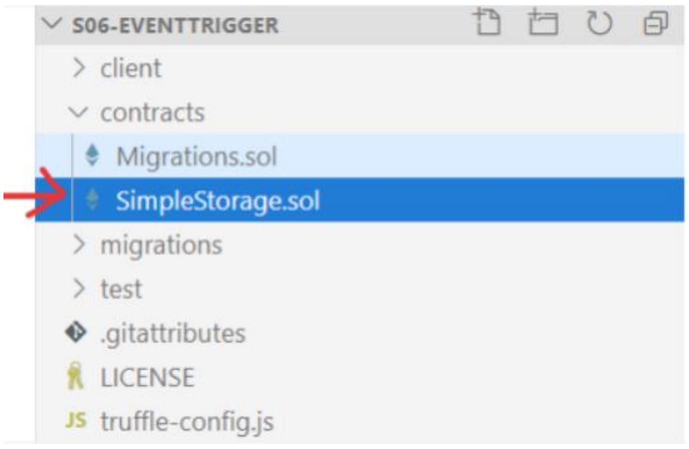


```
PS C:\Windows\system32\s06-eventtrigger> truffle unbox react
truffle : File C:\Users\user\AppData\Roaming\npm\truffle.ps1 cannot be loaded because running scripts is disabled on
this system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ truffle unbox react
+ ~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Windows\system32\s06-eventtrigger>
```

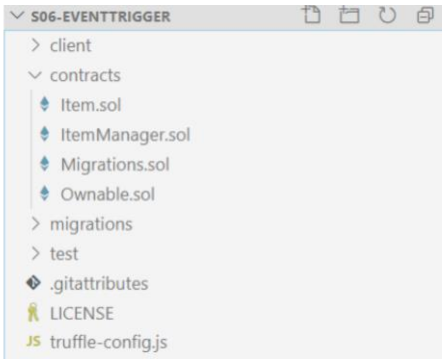
Failed to UNBOX the react box

## 15.6 Add Contracts

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

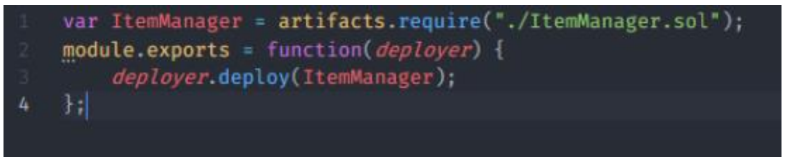


Add in our Files:



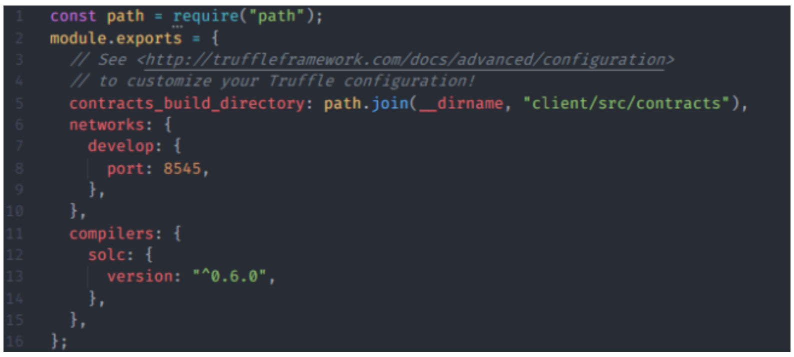
```
S06-EVENTTRIGGER
  > client
  v contracts
      Item.sol
      ItemManager.sol
      Migrations.sol
      Ownable.sol
  > migrations
  > test
  .gitattributes
  LICENSE
  truffle-config.js
```

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

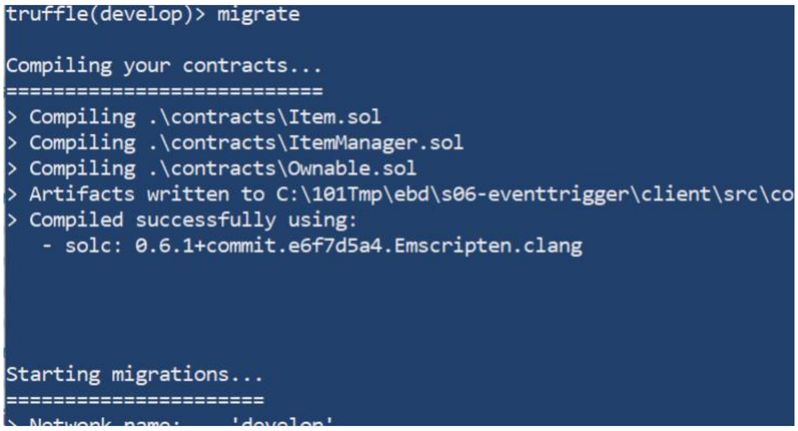


```javascript
var ItemManager = artifacts.require("./ItemManager.sol");
module.exports = function(deployer) {
    deployer.deploy(ItemManager);
};
```

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



```javascript
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",
    },
  },
};
```

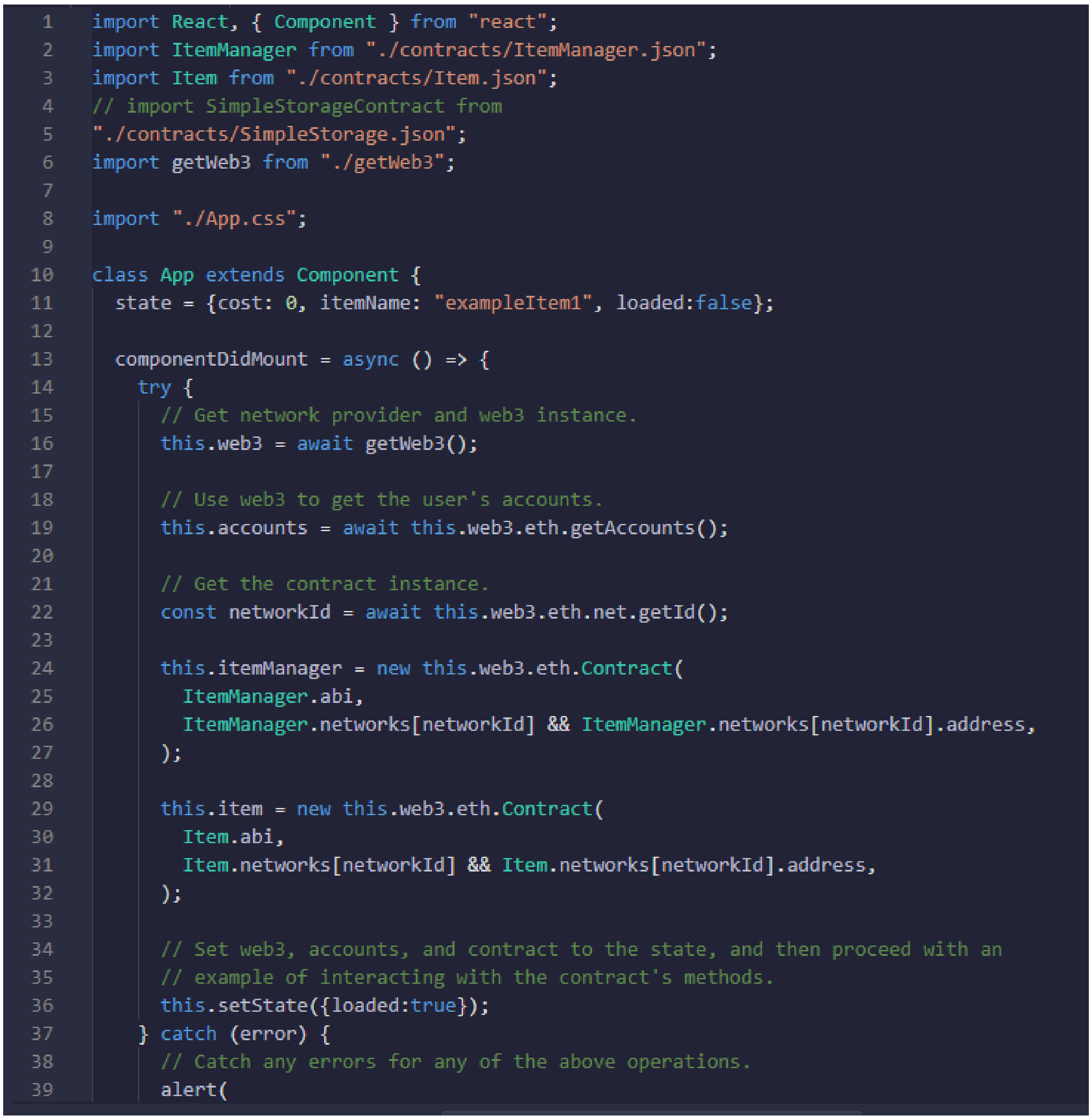


```
truffle(develop)> migrate

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'
```

## 15.7 Modify HTML

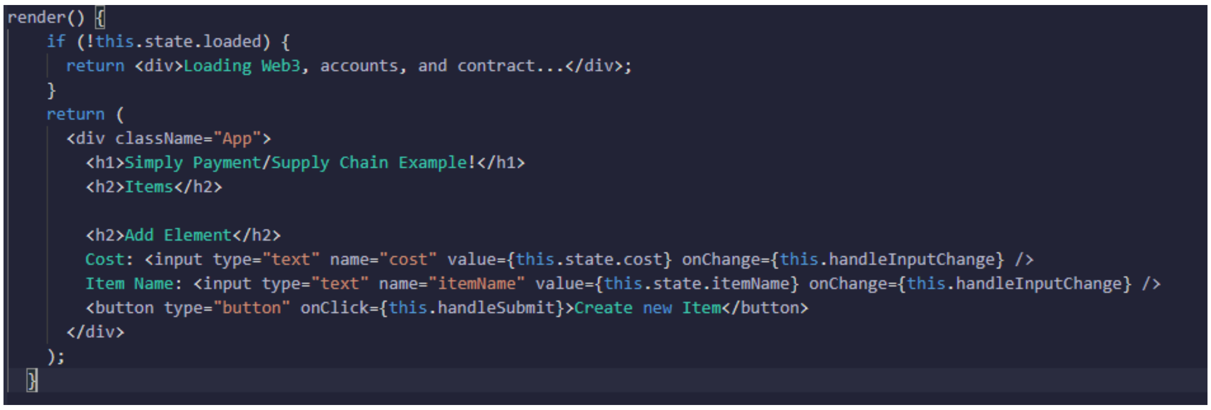
Now it's time that we modify our HTML so we can actually interact with the Smart Contract from the Browser

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



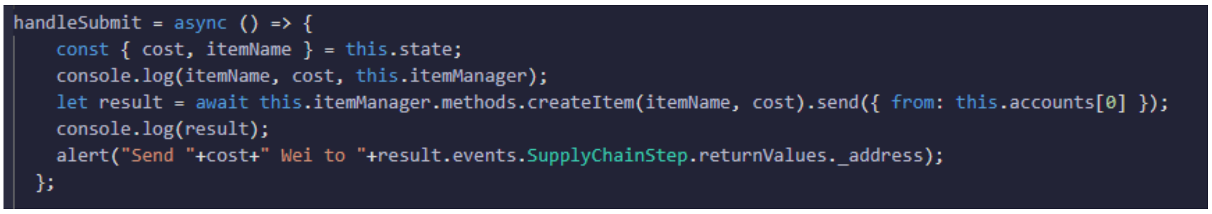
```
import React, { Component } from "react";
import ItemManager from "./contracts/ItemManager.json";
import Item from "./contracts/Item.json";
// import SimpleStorageContract from
"./contracts/SimpleStorage.json";
import getWeb3 from "./getWeb3";

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(
```

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

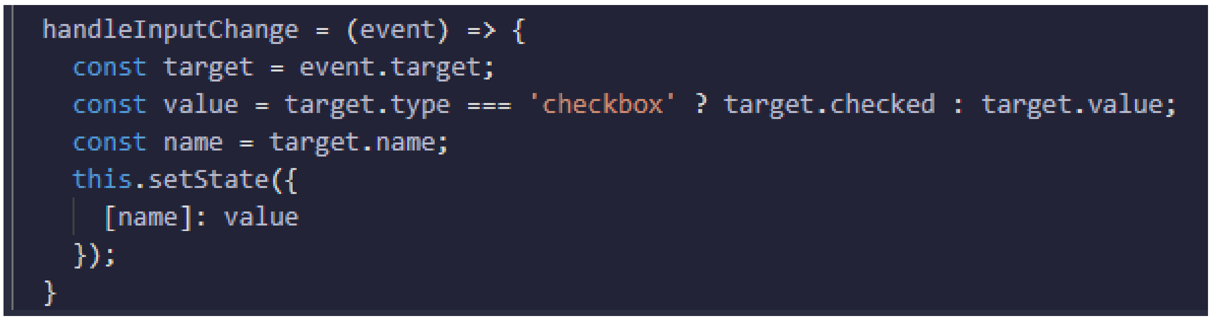


```
render() {
    if (!this.state.loaded) {
      return <div>Loading Web3, accounts, and contract...</div>;
    }
    return (
      <div className="App">
        <h1>Simply Payment/Supply Chain Example!</h1>
        <h2>Items</h2>

        <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>
    );
  }
```

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



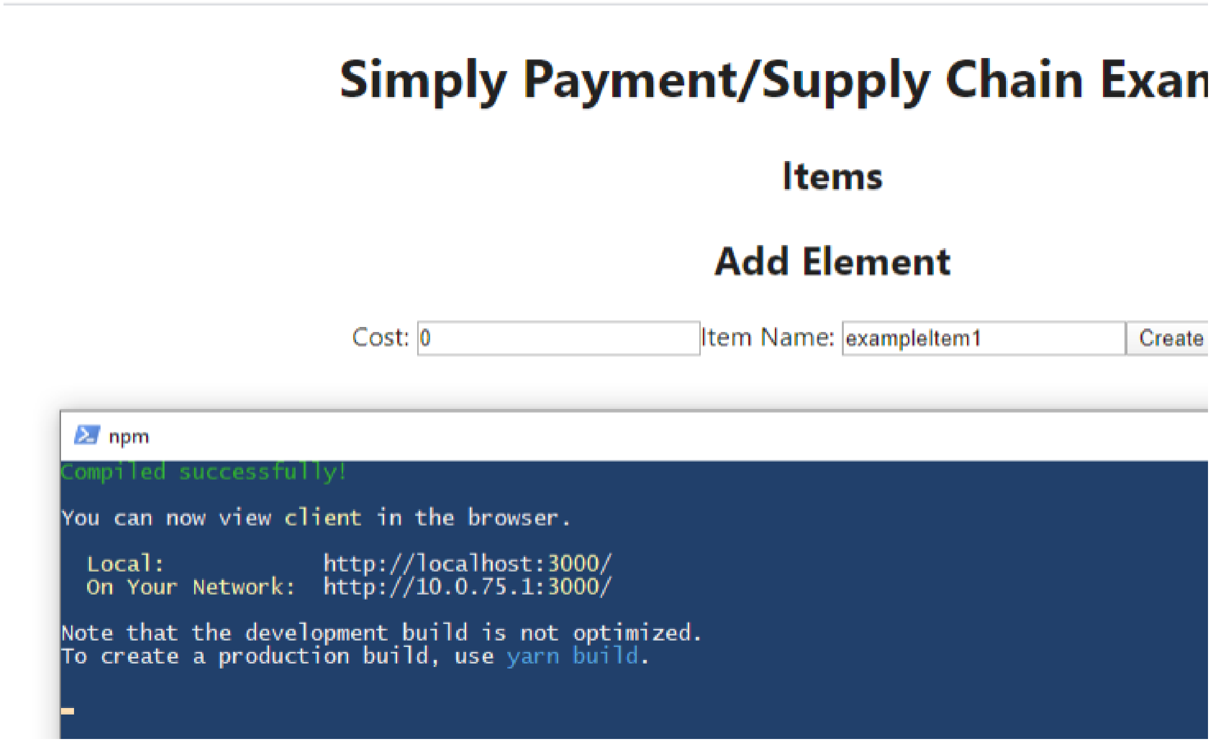
```
handleSubmit = async () => {
    const { cost, itemName } = this.state;
    console.log(itemName, cost, this.itemManager);
    let result = await this.itemManager.methods.createItem(itemName, cost).send({ from: this.accounts[0] });
    console.log(result);
    alert("Send "+cost+" Wei to "+result.events.SupplyChainStep.returnValues._address);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



```
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/powershell lain (biarkan yang sudah Anda buka dengan truffle) dan buka folder klien dan jalankan

Ini akan memulai server pengembangan pada port 3000 dan akan membuka tab baru di browser Anda:



Simply Payment/Supply Chain Exan

Items

Add Element

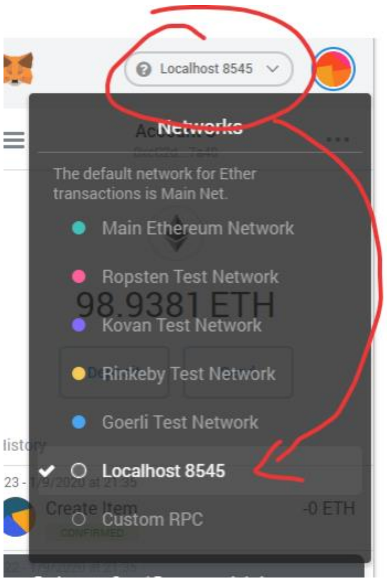
Cost: 0 Item Name: exampleItem1 Create

npm

```
Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.0.75.1:3000/

Note that the development build is not optimized.
To create a production build, use yarn build.
```

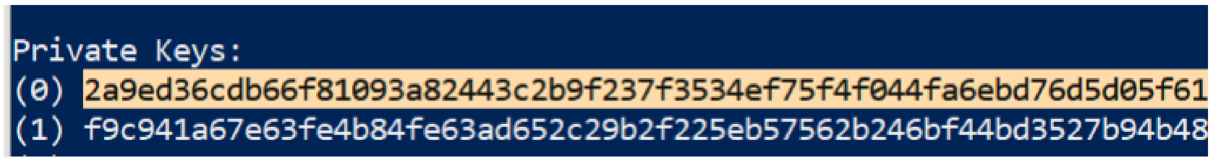
## 15.8 Connect with MetaMask

Pertama, hubungkan metamask dengan jaringan yang tepat :



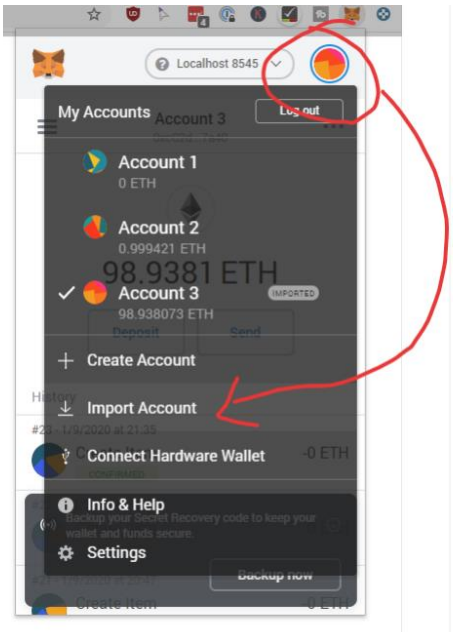
Saat kami memigrasikan kontrak pintar dengan konsol Pengembang Truffle, maka akun pertama di konsol pengembang truffle adalah “pemilik (owner)”. Jadi, perlu menonaktifkan MetaMask di Browser untuk berinteraksi dengan aplikasi atau perlu menambahkan kunci pribadi dari truffle developer console ke MetaMask.

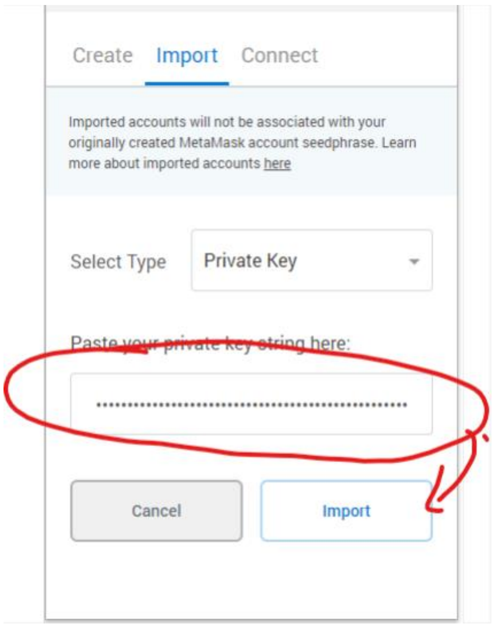
Di Terminal/Powershell tempat Truffle Developer Console menjalankan scroll ke kunci pribadi di atas:



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

Copy private key dan tambahkan ke metamask:





Imported accounts will not be associated with your originally created MetaMask account seedphrase. Learn more about imported accounts here

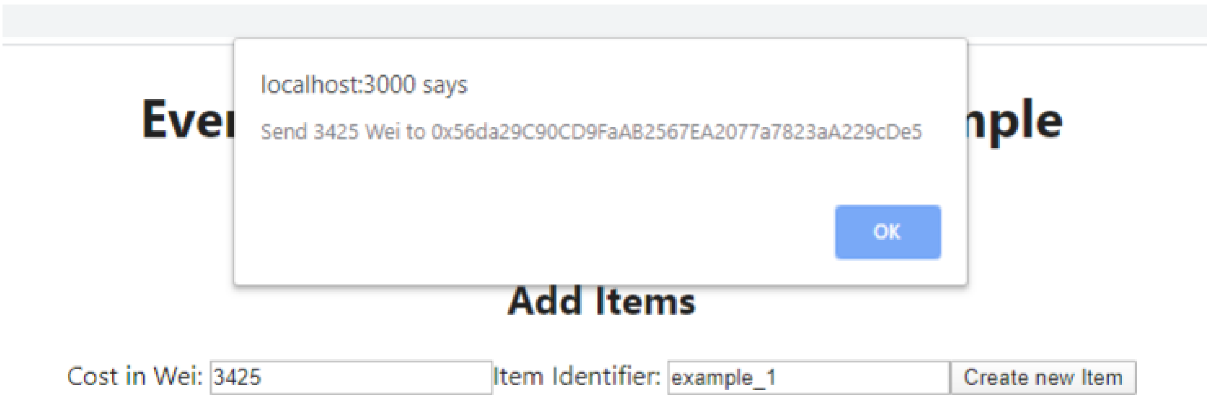
Select Type: Private Key

Paste your private key string here:

••••••••••••••••••••••••••••••••••••••••••••••••••••

Maka akun baru akan terisi dengan 100 ether didalamnya

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



localhost:3000 says

Send 3425 Wei to 0x56da29C90CD9FaAB2567EA2077a7823aA229cDe5

Cost in Wei: 3425 Item Identifier: example_1

15.9 Listen to Payments

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 jika pembayaran terjadi.

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

Yang kami inginkan adalah menunggu acara "SupplyChainStep" dipicu dengan _step == 1 (Berbayar).

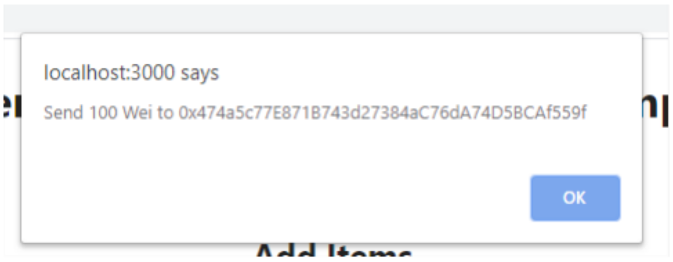
Mari tambahkan fungsi lain ke file App.js:

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

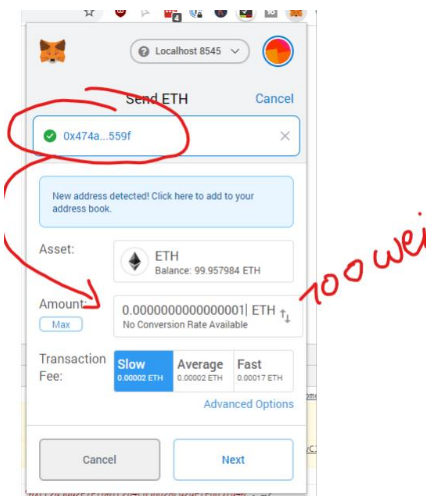
And call this function when we initialize the app in "componentDidMount":

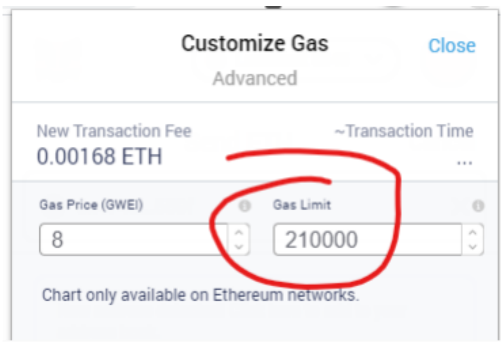
```
this.item = new this.web3.eth.Contract(
 ItemContract.abi,
 ItemContract.networks[this.networkId] && ItemContract.networks[this.networkId].address,
 );
 // Set web3, accounts, and contract to the state, and then proceed with an
 // example of interacting with the contract's methods.
 this.listenToPaymentEvent();
 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);
 }
```

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 sembulan peringatan untuk menampilkan fungsi pemicu:


localhost:3000 says

Send 100 Wei to 0x474a5c77E871B743d27384aC76dA74D5BCAf559f



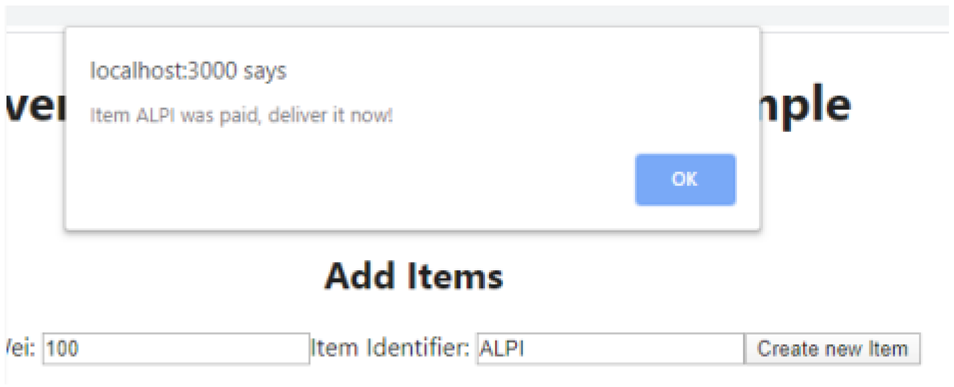


New Transaction Fee 0.00168 ETH
~Transaction Time ...
Gas Price (GWEI) 8
Gas Limit 210000
Chart only available on Ethereum networks.

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, from: accounts[1], gas: 2000000});
```

Maka akan muncul seperti ini :


localhost:3000 says

Item ALPI was paid, deliver it now!

Vei: 100 Item Identifier: ALPI

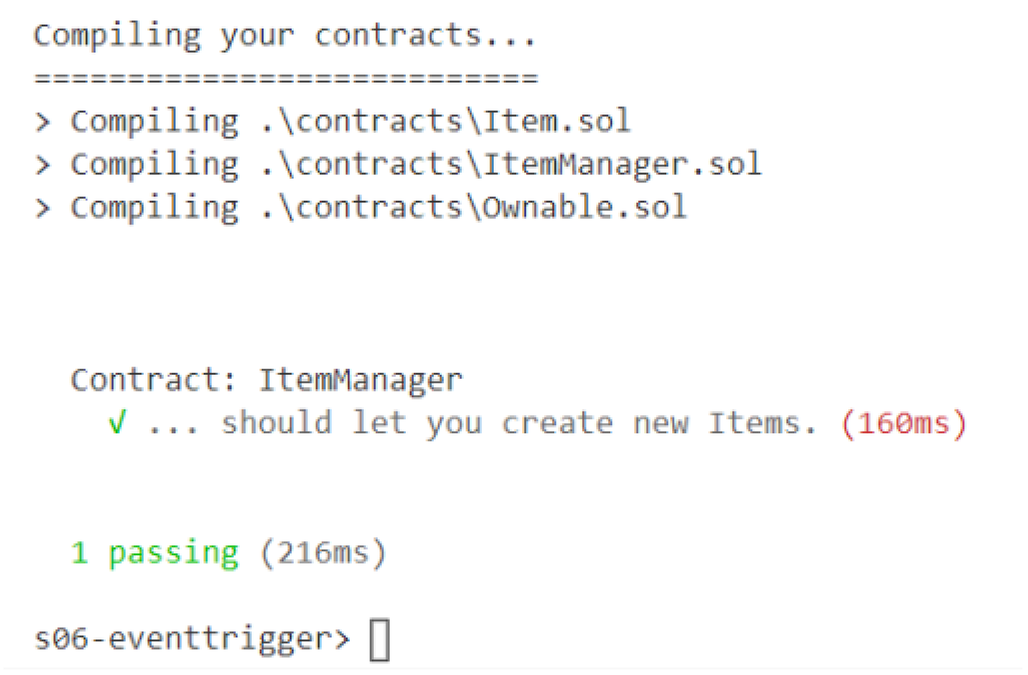
15.10 Unit Test

Pertama, 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.

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.