# LAB : Deposit/Withdraw Ether

## Lab : Smart Contract Self Managing Funds

Di lab ini, Anda akan mempelajari cara Kontrak Cerdas mengelola dananya sendiri. Anda akan mengirim Eter ke Kontrak Cerdas Anda. Kemudian Kontrak Cerdas akan mengelola Eternya sendiri dan dapat menyampaikannya kepada orang lain. Ini seperti rekening bank dengan kode pemrograman yang melekat padanya.

Ini juga dapat digunakan untuk escrow Eter ke dalam Kontrak Cerdas. Pertama, kami akan melakukan contoh deposit/penarikan yang sangat sederhana, lalu Saya akan menunjukkan kepada Anda bagaimana Kontrak Cerdas dapat mengunci dana menggunakan fungsi penarikan yang diaktifkan waktu.

### What You Know At The End of The Lab

Memahami Alamat Kontrak dan objek pesan global

Bagaimana Kontrak Cerdas mengelola Dana

Cara Mengirim/Menarik Eter ke dan dari Kontrak Cerdas

## Smart Contract

Mari kita mulai dengan Kontrak Cerdas sederhana. Buat file baru di Remix dan rekatkan kode berikut:

Banyak hal baru di sini. Jangan khawatir, sebentar lagi semua hal itu akan sangat familiar bagi Anda. Saya terus menjelaskannya seiring berjalannya waktu, tetapi jika Anda tertarik segera, mari kita bahas satu per satu:

uint public balanceReceived : adalah variabel penyimpanan publik. Variabel publik akan membuat fungsi pengambil secara otomatis di Kepadatan. Jadi kami selalu dapat menanyakan konten variabel ini saat ini.

balanceReceived += msg.value : Objek msg adalah objek global yang selalu ada yang berisi beberapa informasi tentang transaksi yang sedang berlangsung. Dua properti yang paling penting adalah .value dan .sender. Mantan berisi jumlah Wei yang dulu dikirim ke kontrak pintar. Yang terakhir berisi alamat yang disebut Kontrak Cerdas. Kami akan menggunakan ini secara luas nanti, jadi, teruskan saja untuk saat ini.

function getBalance() public view returns(uint) : fungsi tampilan adalah fungsi yang tidak mengubah penyimpanan (hanya-baca) dan dapat mengembalikan informasi. Itu tidak perlu ditambang dan hampir gratis.

address(this).balance : Variabel bertipe address selalu memiliki properti bernama .balance yang memberi Anda jumlah eter disimpan di alamat itu. Itu tidak berarti Anda dapat mengaksesnya, itu hanya memberi tahu Anda berapa banyak yang disimpan di sana. Ingat, itu semua informasi publik. address(this) mengonversi instans Kontrak Cerdas menjadi sebuah alamat. Jadi, baris ini pada dasarnya mengembalikan jumlah Eter yang disimpan di Kontrak Cerdas itu sendiri.

Mari kita lihat apakah kita bisa melakukan sesuatu dengannya. Terapkan Kontrak Cerdas dan mainkan sedikit..

## Deploy and Use the Smart Contract

Hal pertama adalah menyebarkan Kontrak Cerdas. Kemudian kita akan melihat apakah kita dapat menyetor beberapa Eter dan mendapatkan saldo dari Smart Kontrak.

### Deploy the Smart Contract

Buka Plugin Deploy and Run Transactions dan terapkan Smart Contract ke dalam JavaScript VM:

Itu akan muncul di bagian bawah Plugin - Anda mungkin perlu memperluas instance kontrak:

### Send Ether To The Smart Contract

Sekarang saatnya untuk mengirim beberapa Eter ke Kontrak Cerdas!

Gulir ke atas ke bidang "nilai" dan masukkan "1" ke bidang input nilai dan pilih "eter" dari tarik-turun:

Kemudian gulir ke bawah ke Kontrak Cerdas dan tekan tombol merah "receiveMoney":

Amati juga terminal, lihat ada transaksi baru yang dikirim ke "jaringan" (walaupun hanya simulasi di

browser, tetapi itu akan sama dengan blockchain nyata).

### Check the Balance

Sekarang kami mengirim 1 Eter, atau 10^18 Wei, ke Kontrak Cerdas. Menurut kode kami, variabel balanceReceived dan fungsi getBalance() harus memiliki nilai yang sama.

Dan, memang, mereka melakukan:

Tapi bagaimana kita bisa mengeluarkan Eter lagi? Mari tambahkan metode Penarikan sederhana.

Coba sendiri dulu?

Anda ingin mencoba sendiri dulu? Berikut beberapa petunjuknya:

Kami menginginkan fungsi yang mengirimkan semua Eter yang disimpan dalam Kontrak Cerdas ke msg.sender (itulah alamat yang memanggil Kontrak Cerdas).

Karena Solidity 0.8 tidak dapat dibayar, jadi Anda perlu melakukan sesuatu seperti payable(msg.sender) , yang akan memberi Anda alamat yang mampu menerima Eter.

Jika Anda tidak tahu apa yang saya bicarakan, jangan khawatir - langsung saja ke halaman berikutnya

## Withdraw Ether From Smart Contract

Sejauh ini kami telah mengirim Eter ke Kontrak Cerdas kami. Tetapi saat ini tidak ada cara untuk mengeluarkan Ether kembali! Jadi apa selanjutnya?

Ya! Fungsi untuk menarik Ether akan bagus, ya?!

### Add a Withdraw Function

Mari tambahkan fungsi berikut ke Kontrak Cerdas:

Fungsi ini akan mengirimkan semua dana yang disimpan dalam Kontrak Cerdas ke orang yang memanggil fungsi "withdrawMoney()".

### Deploy The New Smart Contract

Mari kita coba ini:

1. Deploy versi baru dan kirim lagi 1 Ether ke Smart Contract.
2. Untuk menghindari kebingungan, saya sarankan Anda menutup Instance sebelumnya, kami tidak membutuhkannya lagi

Pada akhirnya, Anda akan mendapatkan satu Instance aktif dari Kontrak Cerdas Anda.

Prosedur yang sama seperti sebelumnya:

1. Masukkan "1 Eter" ke dalam kotak input nilai
2. tekan "receiveMoney" di Instans kontrak baru Anda

Saldo Anda harus 1 Eter lagi:

Bukan 1 Eter?

Jika saldo Anda 0, maka periksa kembali kolom nilai

Jika saldo Anda adalah 2 Eter, periksa kembali Instance kontrak yang berinteraksi dengan Anda!

### Withdraw Funds from the Smart Contract

Sekarang saatnya kita menggunakan fungsi baru kita! Tetapi untuk membuat segalanya lebih menarik, kami akan menarik ke Akun yang berbeda.

Pilih Akun kedua dari dropdown Akun:

Ini lebih dari 100 Eter sebelumnya! Kami mendapatkan 1 Eter kami melalui Kontrak Cerdas kami ke Akun lain! LUAR BIASA!

Mengapa tidak 101 Eter?

Apakah Anda bertanya-tanya mengapa Anda tidak memiliki 101 Eter di Akun Anda? Bagaimanapun, Anda memiliki 100 Eter sebelumnya, dan sekarang Anda menambahkan 1 Eter, jadi, mengapa bukan 101 Eter? Apakah Matematika yang Anda pelajari di sekolah tidak berguna?

Tidak, Matematika yang Anda pelajari di Sekolah sebenarnya berguna.

Apa yang dapat Anda amati di sini adalah konsep "Gas" di Ethereum Blockchain. Setiap transaksi di Ethereum sedikit mahal sedikit. Dan itu tidak berbeda di sini pada rantai simulasi. Prinsip yang sama berlaku. Berapa Gas yang Anda bayar, Anda bertanya-tanya?

Nah, detail transaksinya bisa kamu buka dan buktikan sendiri. Kami membahas ini - secara mendalam - nanti dalam kursus. saya juga membuat video dan posting blog khusus tentang ini jika Anda ingin menyelam lebih dalam sekarang.

Sementara kita dapat menarik dana kita sekarang, seluruh fungsi itu sendiri sangat tidak berguna, bukan?! Siapapun dapat menarik dana ke nya Akun. Tidak ada pecahan dari Jumlah - semuanya, sangat tidak aman.

Saya masih berharap konsepnya sedikit lebih jelas sekarang!

Mari kita ke fungsi lain, yang memungkinkan kita mengirim jumlah penuh ke Alamat tertentu! Itu akan tetap tidak aman, tapi setidaknya mengajarkan konsep baru - satu per satu!

Coba sendiri dulu?

Jika Anda ingin mencoba sendiri terlebih dahulu, maka lakukan hal berikut:

1. Buat fungsi baru yang menggunakan satu alamat sebagai argumen
2. Jumlah penuh Eter yang disimpan di Kontrak Cerdas akan dikirim ke alamat ini

Baiklah, mari kita lakukan ini di halaman berikutnya!

## Withdraw To Specific Account

Sebelumnya kami memiliki Kontrak Cerdas kami hanya secara membabi buta mengirim Eter ke siapa pun yang menyebut Kontrak Cerdas "tarik Uang" fungsi. Mari kita perpanjang ini sedikit sehingga Dana dapat dikirim ke Rekening tertentu.

Itu masih belum aman, karena pada dasarnya siapa pun dapat berinteraksi dengan fungsi itu, tetapi ini selangkah lebih dekat!

Seperti yang Anda lihat, sekarang kami dapat menentukan Alamat tujuan pengiriman uang! Mari kita coba ini!

### Redeploy our Smart Contract

Tentu saja, kita perlu menerapkan kembali Kontrak Cerdas kita. Tidak ada pembaruan langsung (belum?!). Prosedur yang sama seperti sebelumnya:

1. Terapkan Kontrak Cerdas
2. Tutup Instance lama
3. Kirim 1 Eter ke Kontrak Cerdas (jangan lupa kolom input nilai!)
4. Pastikan Saldo muncul dengan benar.

### Test the “withdrawMoneyTo” function

Sekarang saatnya untuk menguji fungsi baru. Kami akan membuat segalanya sedikit lebih menarik : Kami akan menggunakan akun pertama kami untuk mengirim semua dana ke rekening ketiga. Mengapa? Karena kita bisa . Dan karena itu penting untuk memahami bagaimana biaya gas

bekerja - siapa yang membayar transaksi.

1. Pilih akun ketiga dari dropdown
2. Tekan ikon "salin" kecil:
3. Beralih kembali ke Akun pertama:
4. Tempel Akun yang Anda salin ke kolom input di sebelah "withdrawMoneyTo":

Tidak ada fungsi dengan nama itu?

Jika tidak ada fungsi dengan nama itu, kemungkinan besar Anda masih berinteraksi dengan Instance lama. Menyebarkan kembali atau bahkan memuat ulang seluruh halaman.

1. Tekan tombol "withdrawMoneyTo" dan lihat apa yang terjadi! Wow, tidak ada . Yah, hanya di permukaan!
2. Sekarang buka dropdown Akun. Lihat saldo Akun ketiga Anda? 101 Eter!!!

Mengapa ada 101 Eter dan bukan 100.999999999 beberapa? Karena kami mengirim transaksi dari Akun #1 ke Kontrak Cerdas,

menginstruksikan Kontrak Cerdas untuk mengirim semua dana yang tersimpan di Alamat Kontrak Cerdas ke Akun ketiga di . Anda

Daftar-Akun. Biaya gas dibayar oleh Akun #1. Akun #3 mendapat 1 Eter penuh!

### Time-Locked Withdrawals

Itu semua keren dan menyenangkan sejauh ini, tapi mari kita melangkah lebih jauh dan memperkenalkan block.timestamp . Objek gobal ini berisi stempel waktu ketika sebuah blok ditambang. Ini belum tentu stempel waktu saat ini ketika eksekusi terjadi. Mungkin beberapa detik off. Tapi itu masih cukup untuk melakukan penguncian.

Selanjutnya saya ingin menulis Kontrak Cerdas singkat yang hanya memungkinkan penarikan jika setoran terakhir lebih dari 1 Menit yang lalu.

Coba sendiri dulu?

Jika Anda ingin mencoba sendiri terlebih dahulu, maka kami memperpanjang Kontrak Cerdas dan menyimpan "block.timestamp" di suatu tempat. Penarikan hanya dapat terjadi jika "block.timestamp" saat penarikan lebih besar dari timestamp yang disimpan sebelumnya + 1 menit (itu adalah konstanta yang tersedia secara global di Solidity) Ini berpotensi tidak ada dalam video kursus. Latihan ini opsional.

## Withdraw Locking

Mari kita perpanjang Kontrak Cerdas kita untuk melakukan penguncian.

Anda akan melihat bahwa sangat mudah untuk membiarkan kode kami menangani beberapa logika khusus untuk mengizinkan/melarang tindakan tertentu

### Withdraw the Smart Contract

Yang kita butuhkan adalah menyimpan block.timestamp di suatu tempat. Ada beberapa metode untuk melakukan ini, saya lebih suka membiarkan pengguna tahu berapa lama itu terkunci. Jadi, alih-alih menyimpan stempel waktu setoran, saya akan menyimpan stempel waktu yang terkunci. Ayo lihat

Apa yang terjadi di sini:

### Deploy and Test the Smart Contract

Mari kita terapkan Kontrak Cerdas, prosedur yang sama seperti sebelumnya:

1. Terapkan versi Instans baru
2. Hapus Instance lama
3. Kirim 1 Ether ke Smart Contract (jangan lupa kolom value) dengan cara klik "receiveMoney"
4. Periksa Saldo!

Baiklah, sekarang apa? Nah, centang "lockedUntil" dengan mengklik tombol tersebut. Ini akan memberi Anda stempel waktu sampai tidak ada terjadi ketika Anda mengklik withdrawMoney atau withdrawMoneyTo .

1. Klik "withdrawMoney" - dan tidak ada yang terjadi. Saldo tetap sama sampai 1 Menit berlalu sejak Anda menekan "menerima uang".

Cobalah sendiri!

Tapi tidak melakukan apa-apa, tidak ada umpan balik, itu tidak benar-benar ramah pengguna. Anda akan belajar nanti tentang Pengecualian: Memerlukan, Menegaskan, Mengembalikan,

bagaimana kita bisa membuat ini lebih ramah pengguna.

Itu saja untuk saat ini, kerja bagus

# LAB : Shared Wallet

## Project Shared Wallet

Tunjangan Anak per hari/minggu/bulan untuk dapat membelanjakan dana dalam jumlah tertentu.

Majikan memberi karyawan tunjangan untuk biaya perjalanan mereka.

Bisnis memberi kontraktor tunjangan untuk menghabiskan anggaran tertentu.

### Development-Goal

Miliki kontrak pintar dompet on-chain.

Kontrak dompet ini dapat menyimpan dana dan memungkinkan pengguna menarik kembali.

Anda juga dapat memberikan "tunjangan" ke alamat pengguna tertentu lainnya.

Batasi fungsi untuk peran pengguna tertentu (pemilik, pengguna)

Gunakan kembali kontrak pintar yang ada yang sudah diaudit secara maksimal

## We Define the Basic Smart Contract

Ini adalah kontrak pintar yang sangat mendasar. Itu dapat menerima Eter dan dimungkinkan untuk menarik Eter, tetapi secara keseluruhan, tidak terlalu berguna cukup belum. Mari kita lihat apakah kita dapat meningkatkan ini sedikit di langkah berikutnya.

Sebelum Solidity 0.6 fungsi fallback hanya disebut "function() hutang eksternal" - Fungsi tanpa nama. Sejak Solidity 0.6 ada dua fungsi yang berbeda: satu disebut fallback dan yang lainnya disebut "menerima". Hanya "menerima" yang bisa menerima eter. Anda dapat membaca lebih lanjut tentang ini di panduan saya!

Juga, kode di lab ini telah di-porting ke Solidity 0.8.

Perubahan yang paling menonjol adalah penghapusan pustaka SafeMath, karena Solidity 0.8 tidak melakukan rollover integer otomatis lagi. Baca lebih lanjut tentang ini di topik tentang Overflow dan Underflow. Di lab ada catatan di mana Solidity 0.8 berubah masuk.

## Permissions : Allow only the Owner to Withdraw Ether

Pada langkah ini kami membatasi penarikan kepada pemilik dompet. Bagaimana kita bisa menentukan pemiliknya? Itu adalah pengguna yang menyebarkan kontrak pintar.

Ketahuilah bahwa Anda juga menambahkan pengubah "onlyOwner" ke fungsi withdrawMoney!

## Use Re-Usable Smart Contracts from OpenZeppelin

Memiliki logika pemilik secara langsung dalam satu kontrak pintar tidak mudah untuk diaudit. Mari kita uraikan menjadi bagian-bagian yang lebih kecil dan gunakan kembali kontrak pintar yang sudah diaudit dari OpenZeppelin untuk itu. Kontrak OpenZeppelin terbaru tidak memiliki

isOwner() fungsi lagi, jadi kita harus membuat sendiri. Perhatikan bahwa pemilik() adalah fungsi dari kontrak Ownable.sol.

## Permissions: Add Allowances for External Roles

Pada langkah ini kita menambahkan pemetaan sehingga kita dapat menyimpan alamat => jumlah uint. Ini akan seperti array yang menyimpan [0x123546...] alamat, ke nomor tertentu. Jadi, kita selalu tahu berapa banyak yang bisa ditarik seseorang. Kami juga menambahkan yang baru modifikator yang memeriksa: Apakah pemiliknya sendiri atau hanya seseorang dengan uang saku?

Apakah Anda menangkap bug?

Lihat fungsi withdrawMoney dan pikirkan baik-baik!

Dalam kuliah berikutnya kita akan sedikit meningkatkan kontrak pintar kita dan menghindari pengeluaran ganda.

## Improve/Fix Allowance to avoid Double-Spending

Tanpa mengurangi tunjangan penarikan, seseorang dapat terus menerus menarik jumlah yang sama berulang-ulang. Kami harus mengurangi uang saku untuk semua orang selain pemilik

## Improve Smart Contract Structure

Sekarang kami mengetahui fungsi dasar kami, kami dapat menyusun kontrak pintar secara berbeda. Untuk membuatnya lebih mudah dibaca, kita bisa istirahat fungsionalitas menjadi dua kontrak pintar yang berbeda.

Catatan

Perhatikan bahwa karena Allowance adalah Milik, dan SharedWallet adalah Allowance, oleh karena itu berdasarkan properti komutatif, SharedWallet juga Dapat Dimiliki.

Kedua kontrak masih dalam file yang sama, jadi kami tidak memiliki impor (belum). Itu sesuatu untuk kuliah lain nanti. Saat ini, bagian penting untuk dipahami adalah warisan.

## Add Events in the Allowances Smart Contract

Satu hal yang hilang adalah peristiwa.

## Add Events in the SharedWallet Smart Contract

Tentunya kami juga ingin mengadakan acara di dompet bersama kami, ketika seseorang menyetor atau menarik dana:

## Add the SafeMath Library safeguard Mathematical Operations

## Remove the Renounce Ownership functionality

Sekarang, mari kita hapus fungsi untuk menghapus pemilik. Kami hanya menghentikan ini dengan pengembalian. Tambahkan fungsi berikut ke SharedWallet

## Move the Smart Contracts into separate Files

Sebagai langkah terakhir, mari pindahkan kontrak pintar ke dalam file terpisah dan gunakan fungsi impor:

# LAB : Supply Chain Project

## Project Supply Chain

### Real-World Use-Case for this Project

Dapat menjadi bagian dari solusi rantai pasokan

Pengiriman Otomatis setelah pembayaran

Penagihan pembayaran tanpa perantara

## The ItemManager Smart Contract

Hal pertama yang kita butuhkan adalah Kontrak Cerdas "Manajemen", di mana kita dapat menambahkan item.

Dengan ini dimungkinkan untuk menambahkan item dan membayarnya, memindahkannya ke depan dalam rantai pasokan dan memicu pengiriman.

Tapi itu sesuatu yang saya tidak suka, karena idealnya saya hanya ingin memberikan alamat sederhana kepada pengguna untuk mengirim uang.

## Item Smart Contract

Mari tambahkan kontrak pintar lainnya:

Perubahan Soliditas

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)) .

Dan ubah Kontrak Cerdas ItemManager untuk menggunakan Kontrak Cerdas Item alih-alih hanya Struct:

Sekarang dengan ini kita hanya perlu memberi pelanggan alamat Kontrak Cerdas Item yang dibuat selama "createItem" dan dia akan dapat membayar langsung dengan mengirimkan X Wei ke Kontrak Cerdas. Tetapi kontrak pintar belum terlalu aman. Kita butuh semacam fungsi pemilik

## Ownable Functionality

Biasanya kami akan menambahkan Kontrak Cerdas OpenZeppelin dengan Fungsionalitas yang Dapat Dimiliki. 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 pengubah yang benar:

## Install Truffle

Untuk menginstal truffle buka terminal (Mac/Linux) atau PowerShell (Windows 10)

Petunjuk: Saya bekerja di sini dengan Truffle versi 5.1.8. Jika Anda ingin mengikuti versi yang sama persis, ketik npm install -g

truffle@5.1.8

Kemudian buat folder kosong, dalam hal ini saya membuat "s06-eventtrigger"

Dan buka kotak reaksi:

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

## Add Contracts

Hapus Kontrak Cerdas SimpleStorage yang ada tetapi biarkan file "Migrations.sol":

Tambahkan di File kami:

Kemudian ubah file "migrasi" di folder migrasi/:

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

Versi Kompilator

Anda dapat memilih versi untuk menjadi versi yang tepat seperti "0.6.4", atau Anda dapat menambahkan "^" untuk menentukan semua versi di atas yang lebih besar atau sama dengan 0.6.0, yang artinya akan mendownload versi 0.6.x terbaru.

Jalankan konsol pengembangan truffle untuk memeriksa apakah semuanya baik-baik saja dan dapat dimigrasikan. Di terminal/powershell jalankan dan kemudian cukup ketik

## Modify HTML

Sekarang saatnya kita memodifikasi HTML kita sehingga kita benar-benar dapat berinteraksi dengan Kontrak Cerdas dari Browser.

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

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

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

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:

Lihat Kesalahan?

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! Selama tidak ada kesalahan di terminal kami dan dikatakan "Berhasil dikompilasi" Anda siap melakukannya!

## Connect with MetaMask

### What We Do

Di bagian ini kami ingin menghubungkan Aplikasi React kami dengan MetaMask dan menggunakan MetaMask sebagai Keystore untuk menandatangani transaksi. Itu akan juga menjadi proxy ke blockchain yang benar

### Steps to follow

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". 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:

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

Sekarang mari tambahkan Item baru ke Kontrak Cerdas kita. Anda harus disajikan dengan popup untuk mengirim pesan ke pengguna akhir.

## 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 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 acara "SupplyChainStep" dipicu dengan \_step == 1 (Berbayar).

Mari tambahkan fungsi lain ke file App.js:

Dan panggil fungsi ini saat kita menginisialisasi aplikasi di "componentDidMount":

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:

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:

## 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:

Kontrak Truffle vs Web3js

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 dalam kontrak truffle Anda bekerja dengan result.logs.args. Alasannya adalah truffle-contract sebagian besar mengambil API dari web3js 0.20 dan mereka melakukannya

refactor utama untuk web3js 1.0.0.

Biarkan konsol pengembangan truffle terbuka dan ketik terminal/jendela powershell baru:

Seharusnya memunculkan tes seperti ini: