

Ethereum Blockchain

Developer Guide

**The Guide for Learning Ethereum Blockchain Development with Labs and
Explanations**

Thomas Wiesner

Copyright © 2016 - 2021 Thomas Wiesner

Table of contents

1. Practical Ethereum Development	7
1.1 Why This Guide Exists	7
1.2 Are You The Right Audience	8
1.3 What You Will Learn	8
1.4 I saw you also do Video-Courses?	8
1.5 Work in Progress	11
2. License and Re-Use	11
3. Your First Transaction	12
3.1 Lab: Download, Install and Use a Wallet to Create a Transaction	12
3.2 Installing MetaMask	14
3.3 Setup MetaMask	16
3.4 Get Test-Net Ether	21
3.5 Track Ether	29
3.6 Congratulations	33
4. Send Ether between Accounts	34
4.1 Lab: Send Ether between Accounts	34
4.2 Create a new MetaMask Account	35
4.3 Transfer Ether from Account 1 to Account 2	38
4.4 Track the Transaction using Etherscan	44
4.5 Congratulations	51
5. Remix	52
5.1 Lab: Write your first Smart Contract	52
5.2 Setup Remix	54
5.3 Create Your First Smart Contract	59
5.4 Deploy Smart Contract	62
5.5 Interact with the Smart Contract	68
5.6 Congratulations	72
6. Blockchain Networks	73
6.1 Lab: Use different Blockchain Networks	73
6.2 Smart Contract Example	75
6.3 Injected Web3	76
6.4 The JavaScript VM in Remix	82
6.5 Web3 Provider	85
6.6 Congratulations	93

7. Simple Variables	94
7.1 Lab: Solidity Basics	94
7.2 Understanding (Unsigned) Integers	95
7.3 Boolean Types in Solidity	106
7.4 Integer Overflow and Underflow	109
7.5 Address Types	116
7.6 String Types	121
7.7 Congratulations	124
8. LAB: Deposit/Withdraw Ether	125
8.1 Lab: Smart Contract Self Managing Funds	125
8.2 Smart Contract	126
8.3 Deploy and Use the Smart Contract	128
8.4 Withdraw Ether From Smart Contract	133
8.5 Withdraw To Specific Account	138
8.6 Withdrawal Locking	142
8.7 Congratulations	144
9. Smart Contract Life-Cycle	145
9.1 Lab: Starting, Pausing and Destroying/Stopping Smart Contracts	145
9.2 Sample Smart Contract	146
9.3 The Constructor	151
9.4 Pausing Smart Contract	157
9.5 Destroy Smart Contracts using <code>selfdestruct</code>	159
9.6 Congratulations	164
10. Simple Mappings Introduction	165
10.1 Lab: Mappings and Structs	165
10.2 Smart Contract	166
10.3 Access Mapping Variables	171
10.4 Address Keys in Mappings	173
11. Mappings and Structs	179
11.1 Mappings and Structs	179
11.2 Basic Smart Contract	181
11.3 Understand the Limitations	184
11.4 Add a Mapping	189
11.5 Add partial Withdrawals	201
11.6 Add Structs	206
12. Exception Handling	212
12.1 Exception Handling: Require, Assert and Revert in Solidity	212
12.2 The Smart Contract	213

12.3 Try the Smart Contract	216
12.4 Add a Require	218
12.5 Assert to check Invariants	220
12.6 Understanding Try/Catch	223
13. Fallback Function / Constructor / View and Pure	226
13.1 Fallback Function, View/Pure Functions, Constructor	226
13.2 Contract Example	227
13.3 The Receive Fallback Function	236
13.4 The Solidity Constructor	239
13.5 View and Pure Functions in Solidity	248
14. LAB: Shared Wallet	254
14.1 Project Shared Wallet	254
14.2 We Define the Basic Smart Contract	255
14.3 Permissions: Allow only the Owner to Withdraw Ether	256
14.4 Use Re-Usable Smart Contracts from OpenZeppelin	257
14.5 Permissions: Add Allowances for External Roles	258
14.6 Improve/Fix Allowance to avoid Double-Spending	259
14.7 Improve Smart Contract Structure	260
14.8 Add Events in the Allowances Smart Contract	261
14.9 Add Events in the SharedWallet Smart Contract	262
14.10 Add the SafeMath Library safeguard Mathematical Operations	263
14.11 Remove the Renounce Ownership functionality	264
14.12 Move the Smart Contracts into separate Files	265
14.13 Finishing Words	268
15. LAB: Supply Chain Project	269
15.1 Project Supply Chain	269
15.2 The ItemManager Smart Contract	271
15.3 Item Smart Contract	272
15.4 Ownable Functionality	274
15.5 Install Truffle	275
15.6 Add Contracts	278
15.7 Modify HTML	280
15.8 Connect with MetaMask	282
15.9 Listen to Payments	286
15.10 Unit Test	289
15.11 Congratulations	291
16. LAB: ERC20 Token Sale	292
16.1 Project Tokenization Overview	292

16.2 Truffle Initialization	294
16.3 ERC20 Smart Contract	296
16.4 Migration and Compilation	297
16.5 Unit Test ERC20	298
16.6 Add Crowdsale Contracts	303
16.7 Change the UnitTests To Support TokenSales	305
16.8 Crowdsale Unit-Test	307
16.9 Add in a Kyc Mockup	311
16.10 Frontend: Load Contracts to React	313
16.11 Update KYC	317
16.12 Deploy Smart Contracts With MetaMask	319
16.13 Buy Coffee Tokens	331
16.14 Deployment with Infura	342
17. LEARN: Proxies and Upgrades	348
17.1 Upgrade Smart Contract and Smart Contract Proxies	348
17.2 Introduction	349
17.3 The Problematic Smart Contract	350
17.4 Overview of Standards for Smart Contract Upgrades	353
17.5 Eternal Storage without Proxy	354
17.6 The First Proxy Contract	361
17.7 Understanding Storage and Storage Collisions	370
17.8 EIP-897: The first real Proxy	372
17.9 EIP-1822: Proxies without Storage Collision without common Storage Contracts	375
17.10 EIP-1967 Standard Proxy Storage Slots	380
17.11 EIP-1538: Transparent Contract Standard	381
17.12 EIP-2535: Diamond Standard	382
17.13 Metamorphosis Smart Contracts using CREATE2	388
17.14 Conclusion	401
18. LEARN: ERC721 Tokens with OpenZeppelin Presets	402
18.1 ERC721 with Physical Asset Shipping Part 1	402
18.2 The Project Idea	403
18.3 Truffle Setup	407
18.4 OpenZeppelin Preset Contracts	409
18.5 Token Time Based Locking	415
18.6 Unit Testing the ERC721 Token	421
18.7 Final Words and Next Part	424
19. LEARN: ERC721 Token Secondary Sales Royalties	425
19.1 ERC721 Part 2: Secondary Sales Royalties	425

19.2 The Token Improvement	426
19.3 Minimal ERC721 Token	428
19.4 Adding a Migration for the ERC721 Token	430
19.5 Secondary Royalties on OpenSea	434
19.6 Secondary Royalties on Rarible	438
19.7 Secondary Royalties on Mintable (ERC2981)	443
19.8 Token URI	445
19.9 Final Words	446

1. Practical Ethereum Development

Heyo! So, you thought "Blockchains" are a cool thing? You have no idea where to start? This whole thing is too hard to figure out with weird YouTube tutorials and outdated sites?

Guess what!?

You're at the right place! And with my help you're developing your own Smart Contracts in no time!



1.1 Why This Guide Exists

When I started with Blockchain development back in 2016, the landscape for tutorials was very very very scarce. There were many tools, almost all only half-working. And no real guides. There was no Remix, no MetaMask, no Infura, no Truffle, no Academies, no ConsenSys. No nothing back then. My start was AlethZero. It crashed every few minutes or so and some whacky guides on how to compile Smart Contracts.

And for the entirety of 2016 the price of Ether was between \$1 and \$7.

AlethZero in Action

AlethZero in Action from this [YouTube Video](#)

What I was looking for was a practical guide that takes me through typical steps as a Smart Contract and DApp developer. Something that takes me through the pitfalls. Something I can relate to as a developer.

I'm not trying to do something shady. I'm not trying to build another Silk Road. This guide is not about Libertarianism. I'm not a cryptography researcher.

I am a CTO with a strong development background trying to do practical stuff with that technology. I am not trying to make anyone geek out on how much it will f*ck up our traditional world of finance.

I didn't have any of those guides. And I set out to change that. Already in 2016. But my first attempts were not great. In fact, they were very bad. Now, 15 video courses later, hundreds of hours spent on creating tutorials and video materials (if not thousands of hours!), I believe it reached a point where I have a framework for learning this stuff. And showing it to others. And I want to keep going.

When you're a traditional (web-)dev, then it's quite a bit of new material to learn and dig through. The traditional trust-model changed: the underlaying flow of registration/authentication is almost reversed. Tools are different. Language is different. Boundaries of what's possible are narrower. The business goals may be the same, but the way to reach them is skewed, for the lack of a better word.

And this guide shall be your new best friend.

1.2 Are You The Right Audience

It's certainly not a complete reference or covers the Solidity documentation front2back. You won't get a PhD after working through my code examples. But it gives you a deep understanding of the technologies behind famous DeFi-projects like Uniswap, ERC20 and ERC721 Tokens, Blockchain Supply Chain Solutions, and many more things. Scalable things. Trustable things. Enterprise'ey workflows. Stuff that I would expect a developer would bring when he wants to be hired.

[Not me](#)

That's not me. That's a foto I blatantly copied from unsplash. If you made it this far, why not just go and do your first transaction in the next chapter?! Photo by [Campaign Creators](#) on [Unsplash](#)

How hard will it be to go through the guide? That depends on your prior knowledge about web development. If you're a total beginner: never written a single line of JavaScript, never heard of RESTful APIs? Then better look somewhere else. Blockchains are not the best way to get started with development, it's hard to access and many underlaying ideas require fundamental understanding of how the web works.

If you are a C, C#, C++, Java, etc programmer with 20 years on your shoulders, you'll probably have an easy time. If you come from PHP, some things will be new, some things might look easy.

One thing I can promise you: I'll try to show you "the right way"™ to do things in an ever changing and more-than-ever demanding environment and I hope it will be enough to spark your interest to learn more about selected topics.

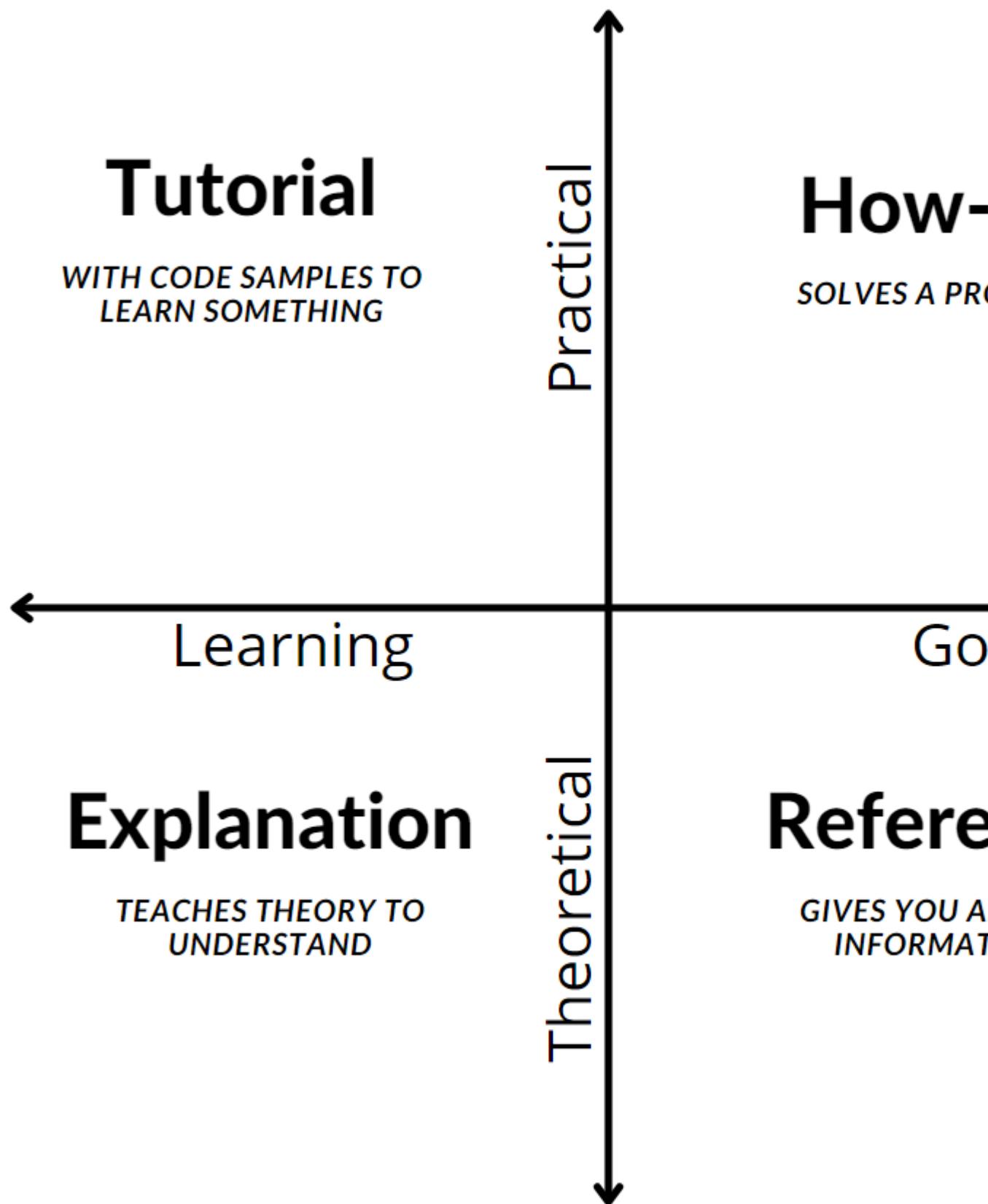
1.3 What You Will Learn

In these labs we're going through quite a bit and I'm not going to bullet-point every single tech and principle you're touching. Suffice to say, you'll have a workflow to develop your own projects. You'll have a fundamental understanding of Solidity. And you'll know the boundaries you're operating under (also called limitations).

At the end we'll run through a few full projects with Solidity on the Blockchain side and React on the Frontend side.

1.4 I saw you also do Video-Courses?

When you're looking to learn something, you can choose between with 4 different types of materials: Tutorials, How-To's, a full Reference and a theoretical explanation.



A school book about physics would be the classical "Theoretical Explanation". Yawn.

What's the difference between a Tutorial and a How-To?

A tutorial is learning oriented and a how-to is problem oriented. A tutorial is great for studying and a how-to is great to solve a specific problem when you're working. You wouldn't go to stackoverflow to learn a new skill, would you? And you would not take a 24h Udemy course to get that damn regex filter fixed, right?

The Solidity Documentation is a Reference, in my opinion. A pretty good one. Almost a Tutorial. What it lacks is teaching also about Blockchains and the tooling.

So, why an official (video) course on top of this guide?

This guide is a tutorial but it doesn't include a lot of theoretical knowledge. And it also doesn't include me directly showing, on video, how things are done.



This guide strips away most of the theoretical part and basically contains all the labs from of the video course [Ethereum Blockchain Development](#).

If videos are your thing, then check it out. I made it with my colleague and friend Ravinder Deol, who's just as much of a Blockchain enthusiast as I am.

Now a little self promo: We will take you from beginner to master. Here's why:

- This course is taught by the Co-Creator Of The Industry Standard Ethereum Certification.
- This course been updated to be 2021 Ready, so you'll be learning with all the latest tools.
- This course does not cut any corners, you will learn by building Real-World Projects in our labs.
- We've taught over 100,000 students in the cryptocurrency & blockchain ecosystem.
- Save Yourself Over \$10K, but still get access to the same materials as live bootcamps.
- This course is Constantly Updated with new content, projects and modules.

It's also a best-seller on Udemy and was picked up and transformed into corporate trainings and a book and instructor-led trainings, translated to chinese, probably pirated a few times, and more.

1.5 Work in Progress

Currently this written guide is a work in progress. I will update the following list as more and more chapters are being converted:

- ✓ Your First Transaction With MetaMask
- ✓ Your First Smart Contract with Remix
- ✓ Using different Blockchain Networks
- ✓ Solidity Basics: Integers, Boolean, Addresses, Strings
- ✓ LAB: Escrow - Deposit and Withdrawals
- ✓ Smart Contracts Life-Cycle: Starting, Pausing, Destroying Smart Contracts
- ✓ Understanding Mappings and Structs
- ✓ Exception Handling
- ✓ Constructor and Fallback Functions
- ✓ Solidity Advanced: Modifier, Inheritance, Constructors, Fallback
- ✓ Events and Return Variables
- ✓ LAB: Create a Shared Wallet
- ✓ LAB: Event Triggers / Supply Chain Example with Truffle 5 and Unit Tests
- ✓ Understand and Use Go-Ethereum Private Networks
- ✓ LAB: Asset Tokenization using OpenZeppelin and Truffle

Work in Progress

Please note, this site is a "work in progress" for the course "Ethereum Blockchain Developer Bootcamp With Solidity (2021)"

2. License and Re-Use

All my materials are original materials and it took several hundred hours to put them together. I didn't do it to earn money in the first place, but I am also not doing it solely for someone else to make money off my back.

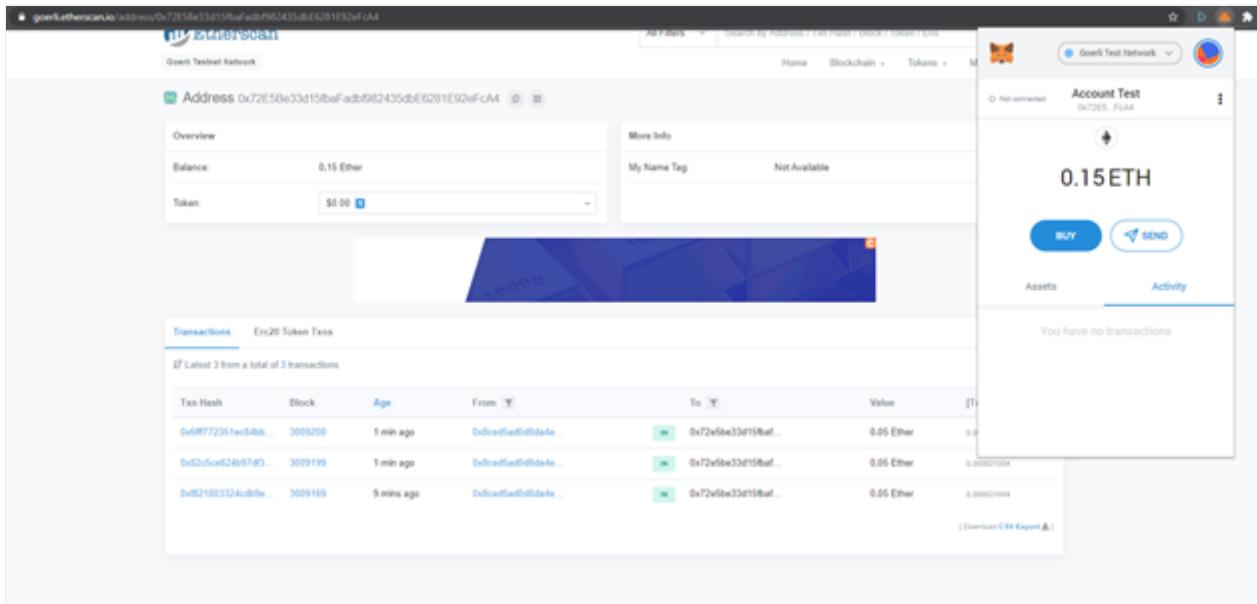
I offer sub-license agreements for commercial use and educational use. Reach out to me at thomas at vomtom dot at.

Last update: April 17, 2021

3. Your First Transaction

3.1 Lab: Download, Install and Use a Wallet to Create a Transaction

In this lab we are going to install MetaMask and create our first transaction.



The screenshot shows the MetaMask extension interface within a browser window. The top bar indicates "Connected" to the "Gnosis Test Network". The main area displays the account balance as "0.15 ETH". Below this, there are two buttons: "BUY" and "SEND". The "Activity" tab is selected, showing a list of recent transactions. The table lists three transactions from the last 9 minutes, each transferring 0.05 Ether to the address 0x72e50e33d15baFadbf9b2435dbf6281E92eFcA4. The "Transactions" tab is also visible.

3.1.1 What You Know At The End Of The Lab

How to Get Free Ether To Test Transactions

How To Securely Store Your Funds With Your Own Wallet

Interact With Different Blockchains

Industry Standard Way To Connect To Blockchains

Understand Public Information With Block Explorers

3.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.

2. An Internet connection

Brave Browser

If you are using a Brave Browser and you run into problems, then try to use Chrome instead.

3.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

3.1.4 Get Started

Let's get started by [installing MetaMask](#)

Last update: April 23, 2021

3.2 Installing MetaMask

Firstly, we are going to install MetaMask. That is a browser plugin which can securely store private keys and connect to different blockchains.

How this exactly works is something we discuss later. For now we just play around.

3.2.1 Download MetaMask

Open <https://metamask.io> and download the plugin for your browser

The image shows the official MetaMask website at <https://metamask.io>. The top navigation bar includes links for Team, Developers, Institutions, and FAQs. A large, stylized fox head logo is positioned next to the word "METAMASK". The main headline reads "A crypto wallet & gateway to blockchain apps". Below this, a subtext states "Start exploring blockchain applications in seconds. Trusted by over 1 million users worldwide." A prominent blue button labeled "Download now" is located at the bottom left. On the right side, a mobile device screen displays the MetaMask wallet interface. It shows "Account 1" with the address "0x5e...5924". The "TOTAL BALANCE" is listed as "\$228.77" with "1.29 ETH". There are "BUY" and "SEND" buttons. The "Assets" tab is selected, showing "1.2 ETH" worth "\$274.52 USD" and "0.20 DAI" worth "20 USD". A link "see your asset? Add asset" is visible at the bottom of the wallet interface.

Home > Extensions > MetaMask

MetaMask

Offered by: <https://metamask.io>

★★★★★ 1,677 | Productivity | 1,048,726 users

Overview Reviews Support Related

New Tab +

New Tab

marketing rome

MetaMask has been added

Use this extension by clicking the icon in the toolbar.

Manage shortcuts

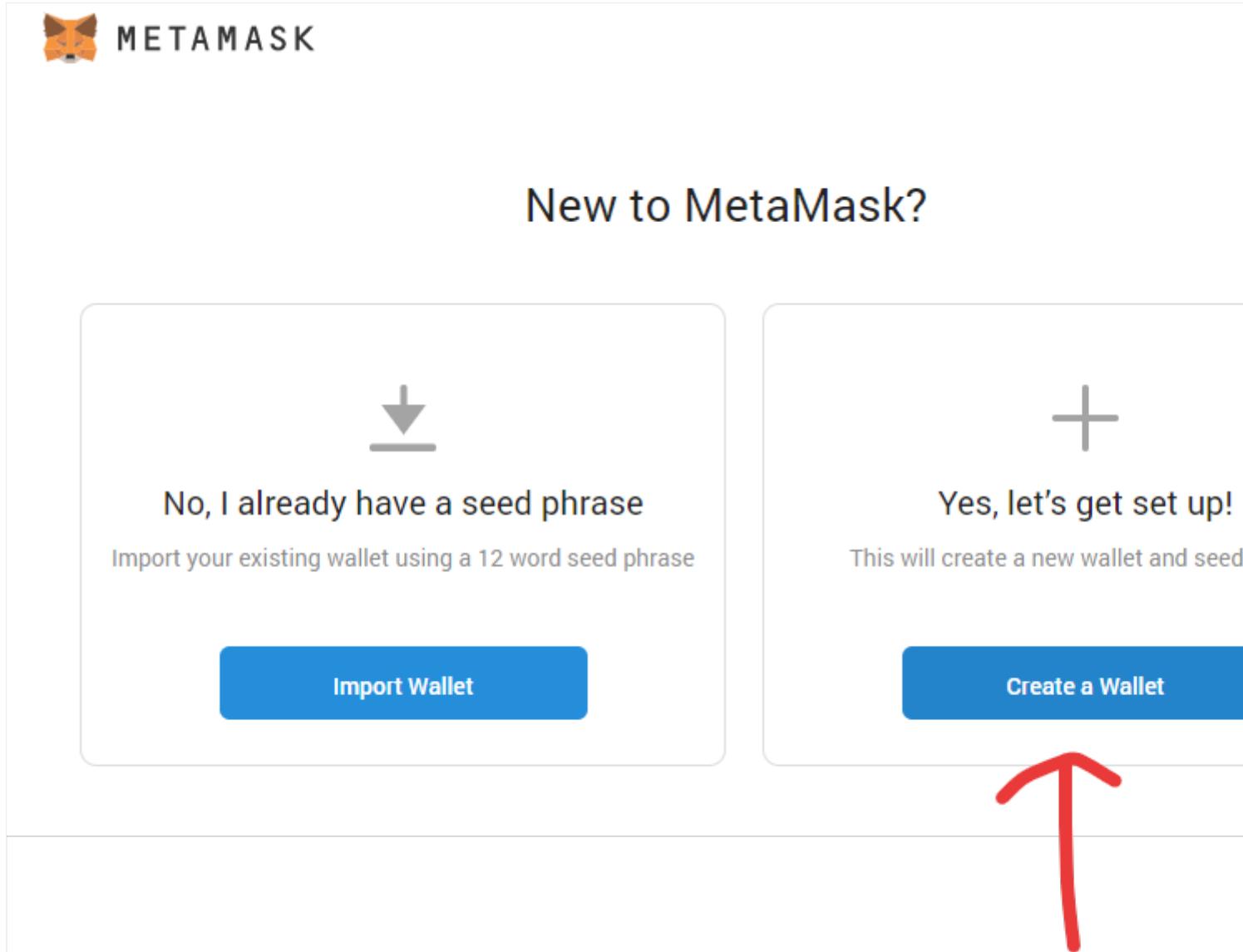
Perfect, that's it. Now, let's setup MetaMask and make it secure.

Last update: April 17, 2021

3.3 Setup MetaMask

When you install MetaMask, then it will automatically open up a "setup" page.

Hit "Begin" and walk through the setup-wizard. Let's create a new Wallet!



3.3.1 Statistical Information

If you want to send statistical information, is totally up to you, both is fine:



Help Us Improve MetaMask

MetaMask would like to gather usage data to better understand how our users interact with the extension. This data will be used to continually improve the usability and user experience of our product and the Ethereum ecosystem.

MetaMask will..

- ✓ Always allow you to opt-out via Settings
- ✓ Send anonymized click & pageview events
- ✓ Maintain a public aggregate dashboard to educate the community

- ✗ **Never** collect keys, addresses, transactions, balances, hashes, or any personal information
- ✗ **Never** collect your full IP address
- ✗ **Never** collect your location data from the Event

No Thanks

I agree

This data is aggregated and is therefore anonymous for the purposes of General Data Protection Regulation (EU) 2016/679. For more information in relation to our privacy practices, please see our [Privacy Policy here](#).

3.3.2 Set a Password

Create a new strong password. This password is used to encrypt your private keys. What private keys are exactly is discussed in a later section of the course, suffice to say though, they give access to all your Ether. So, better have a strong password here:



METAMASK

< Back

Create Password

New Password (min 8 chars)

Confirm Password

I have read and agree to the [Terms of Use](#)[Create](#)

3.3.3 Backup Phrase

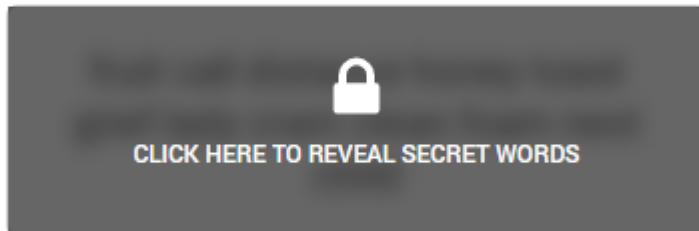
It would be better to safely store the secret phrase, but for sake of simplicity, let's just skip this for now:



Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.



Remind me later

Next

Seed Phrase

A seed phrase (or here: Backup Phrase) is usually a number of human-readable words (e.g. 12 words). This represents the "master key" to regain access to all your accounts. It is a simple algorithm to create a number of private keys based on your backup phrase. Don't worry if you don't know yet what this means - just remember: Never (like **never ever**) give out your seed phrase!

3.3.4 Final Screen

And you should be greeted with this screen:

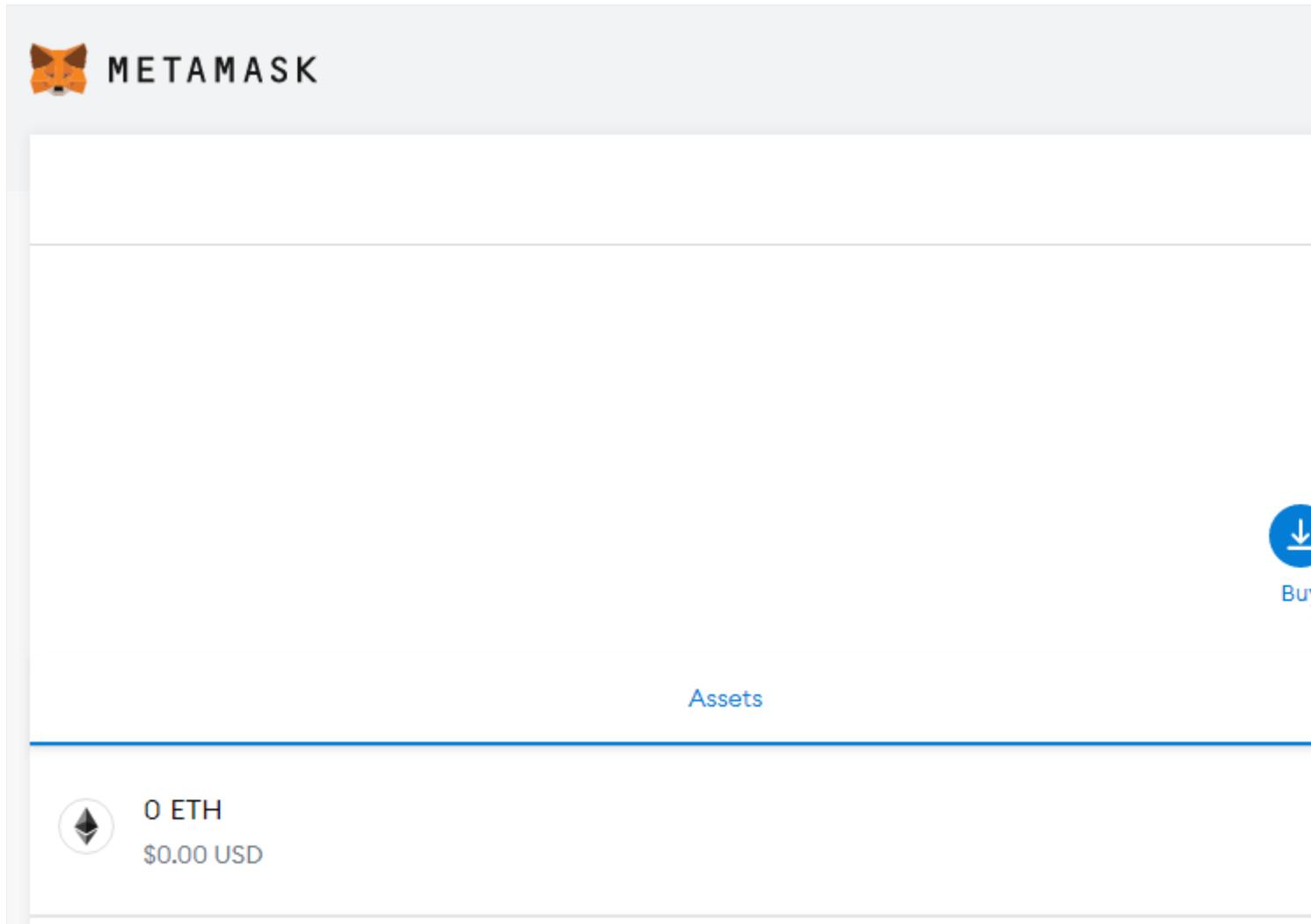
Tips:

Store this phrase in a password manager like 1Password.

Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down on multiple pieces of paper and store each in 2 - 3 different locations.

Memorize this phrase.

[Download this Secret Backup Phrase and keep it stored safely on an external encrypted hard drive or storage medium.](#)



Let's see now how we can use MetaMask to transfer Ether...

Last update: April 17, 2021

3.4 Get Test-Net Ether

If you have never worked with blockchains before, then the first confusing this you will encounter is: There is not one blockchain, but many different blockchains. I am talking about Ethereum Blockchains.

It's like having different databases. But only one is considered the "Main" Database, or "Mainnet".

There are also other blockchains, for testing different aspects. Each of those have usually a name and a specific network and chain id. There is no central list of them, because everyone can open their own blockchain, but here's a good [overview](#).

In this tutorial, we will use either Ropsten or Görli to get Test-Ether and start a transaction.

3.4.1 Get Görli Test-Ether

Switch the network to Goerli.

Network Selection

Attention here: some of the pictures have "Ropsten" selected, but the Ropsten test-network had a couple of hiccups, so I recommend Goerli instead!

List of ETH Testnet Faucets

Sometimes Faucets don't work as expected. Unfortunately there is nothing much that I can do about it. It is time intensive to run a faucet and usually it doesn't pay off economically. Here is a list of Faucets in case the one here doesn't work, you can probably switch to another one:

Ropsten: <https://faucet.metamask.io>

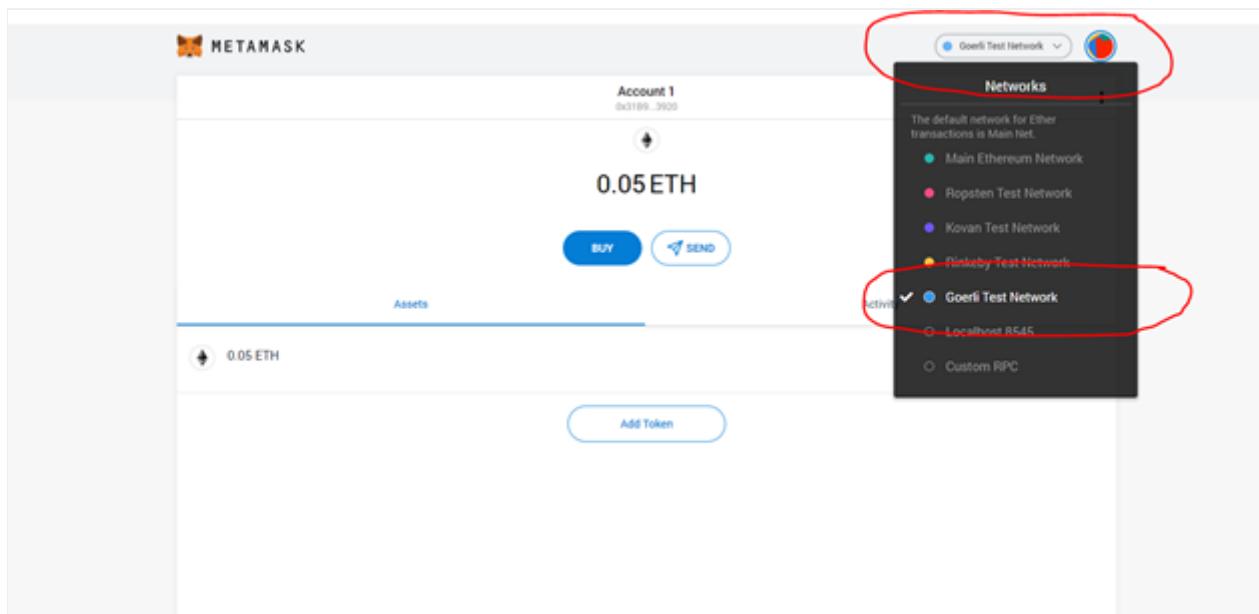
Rinkeby: <https://faucet.rinkeby.io> <https://www.rinkebyfaucet.com> <https://app.mycrypto.com/faucet> <https://faucets.chain.link/rinkeby>

Kovan: <https://gitter.im/kovan-testnet/faucet> basically post your eth address in the gitter chat

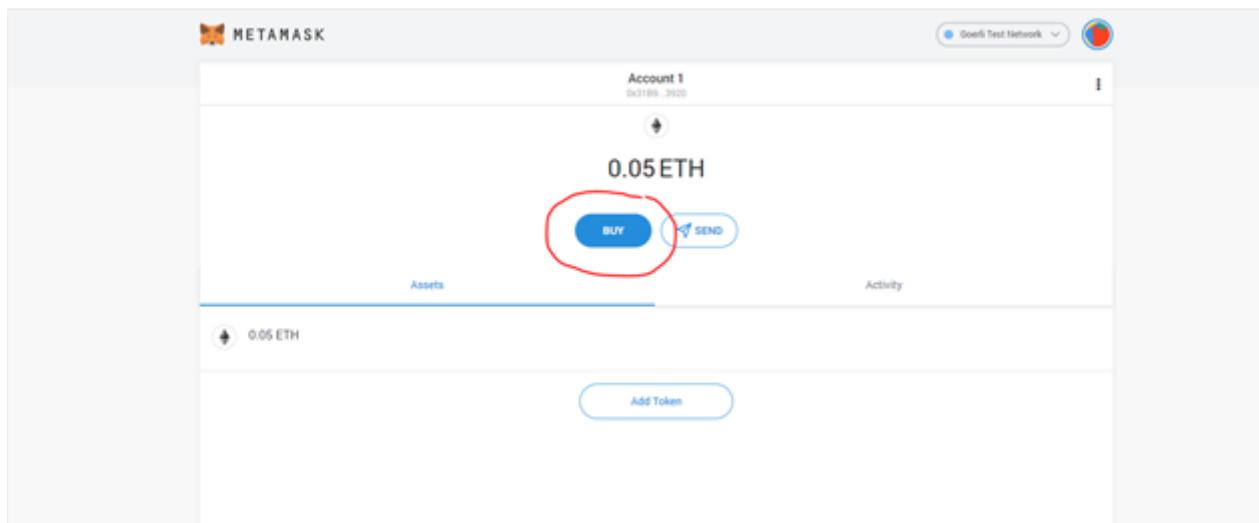
Görli: <https://goerli-faucet.slock.it/index.html> <https://faucet.goerli.mudit.blog>

Another "special edition" Faucet is maintained by Keir "Blockchain-Gandalf" Finlow-Bates, who also wrote a great book about Blockchains. He tries to maintain it as good as possible and it outputs Ropsten Ether: <https://moonborrow.com>

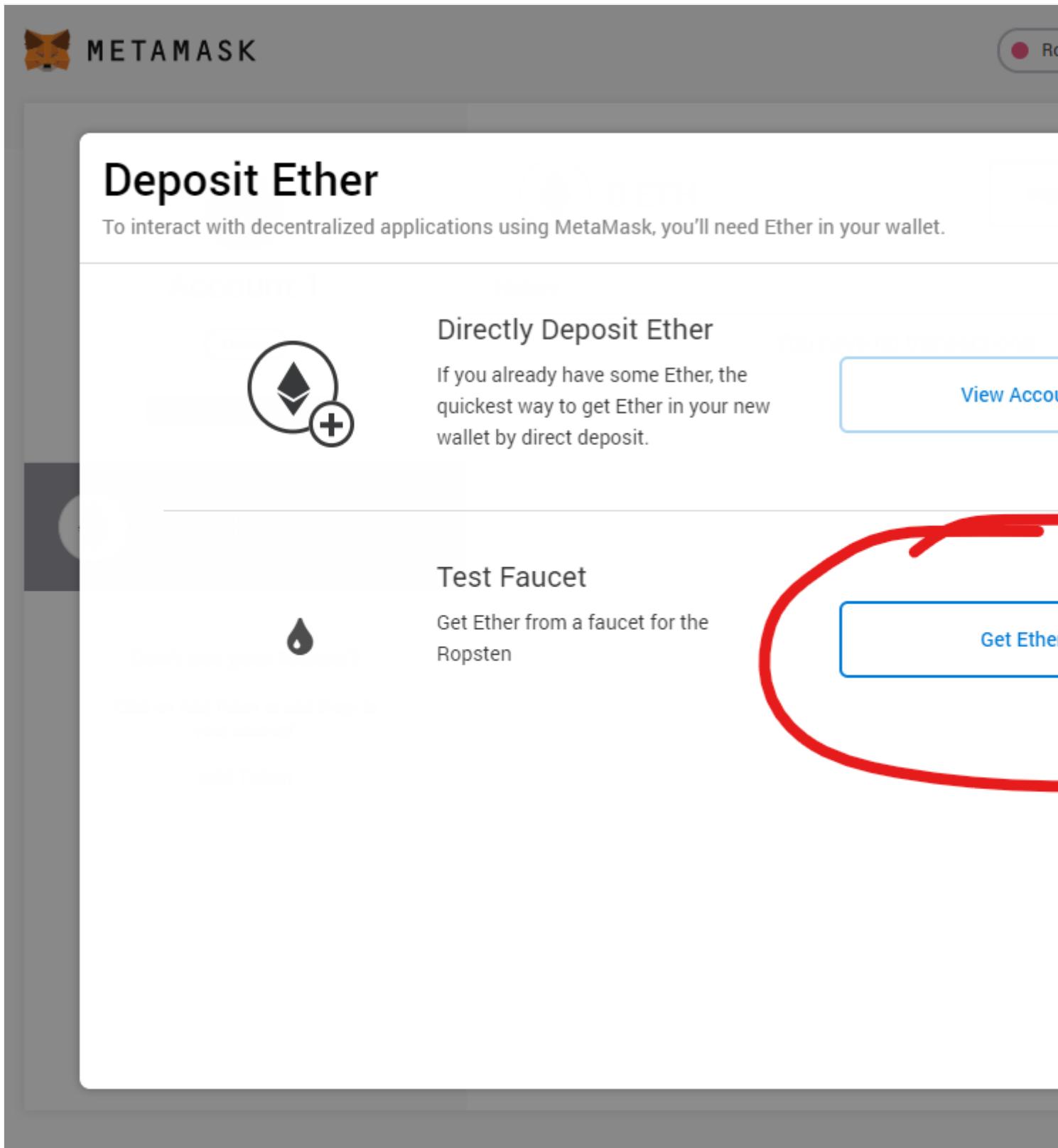
Kintsugi (Eth2.0): <https://kintsugi.themerge.dev>



Hit "BUY"



Click on "Get Ether"



The screenshot shows the MetaMask interface for depositing Ether. At the top left is the MetaMask logo (a fox head icon) and the word "METAMASK". On the top right is a red circular button with a white dot and the letters "Ro". Below the header, a large black header says "Deposit Ether". A sub-header below it reads "To interact with decentralized applications using MetaMask, you'll need Ether in your wallet." To the left of the "Directly Deposit Ether" section is an icon of a circle containing an Ethereum symbol with a plus sign next to it. To the right of the "Directly Deposit Ether" section is a blue button labeled "View Account". Below these sections is another section titled "Test Faucet" with an icon of a water droplet. To the right of the "Test Faucet" section is a blue button labeled "Get Ether". A large red oval has been drawn around the "Get Ether" button.

Deposit Ether

To interact with decentralized applications using MetaMask, you'll need Ether in your wallet.

Directly Deposit Ether

If you already have some Ether, the quickest way to get Ether in your new wallet by direct deposit.

Test Faucet

Get Ether from a faucet for the Ropsten

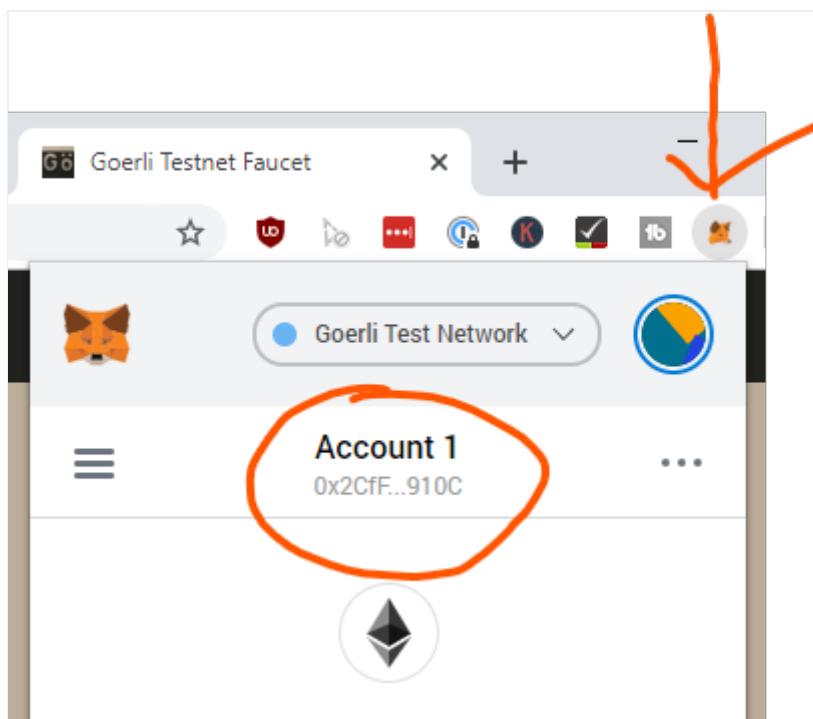
[View Account](#)

[Get Ether](#)

A new website should open up. That's the faucet to get Ether. A Faucet is like a "get free Ether" -- site. The Ethers are having no value, they are running under a "test" Blockchain, but they are great for getting your feet wet with transactions and how Wallets work.

The screenshot shows a web browser window with three tabs open, all titled "MetaMask". The active tab displays the "goerli-faucet.slock.it" website. The page has a dark background with a large, metallic-looking "Gö" logo. Below the logo, the text "GOERLI FAUCET" is displayed in a bold, white, sans-serif font. A rectangular input field with a light orange border contains the placeholder text "Enter address to receive funds here". To the right of this input field is a large, dark button with the text "REQUEST 0.05 GÖETH". At the bottom of the page, there is a reCAPTCHA interface consisting of a checkbox followed by the text "I'm not a robot". The footer of the page is black and contains the text "Powered by Slock.it - Send feedback to https://gith...".

Copy your Address from MetaMask by clicking directly on the address:



Paste it into the Goerli Faucet Value Field and hit "I'm not a robot" and "Request 0.05 GÖETH"

goerli-faucet.slock.it

Goerli Testnet Faucet



GOERLI FAUCET

0x2CfFF0C8C1aDbb9231c7F2b126066273960F910

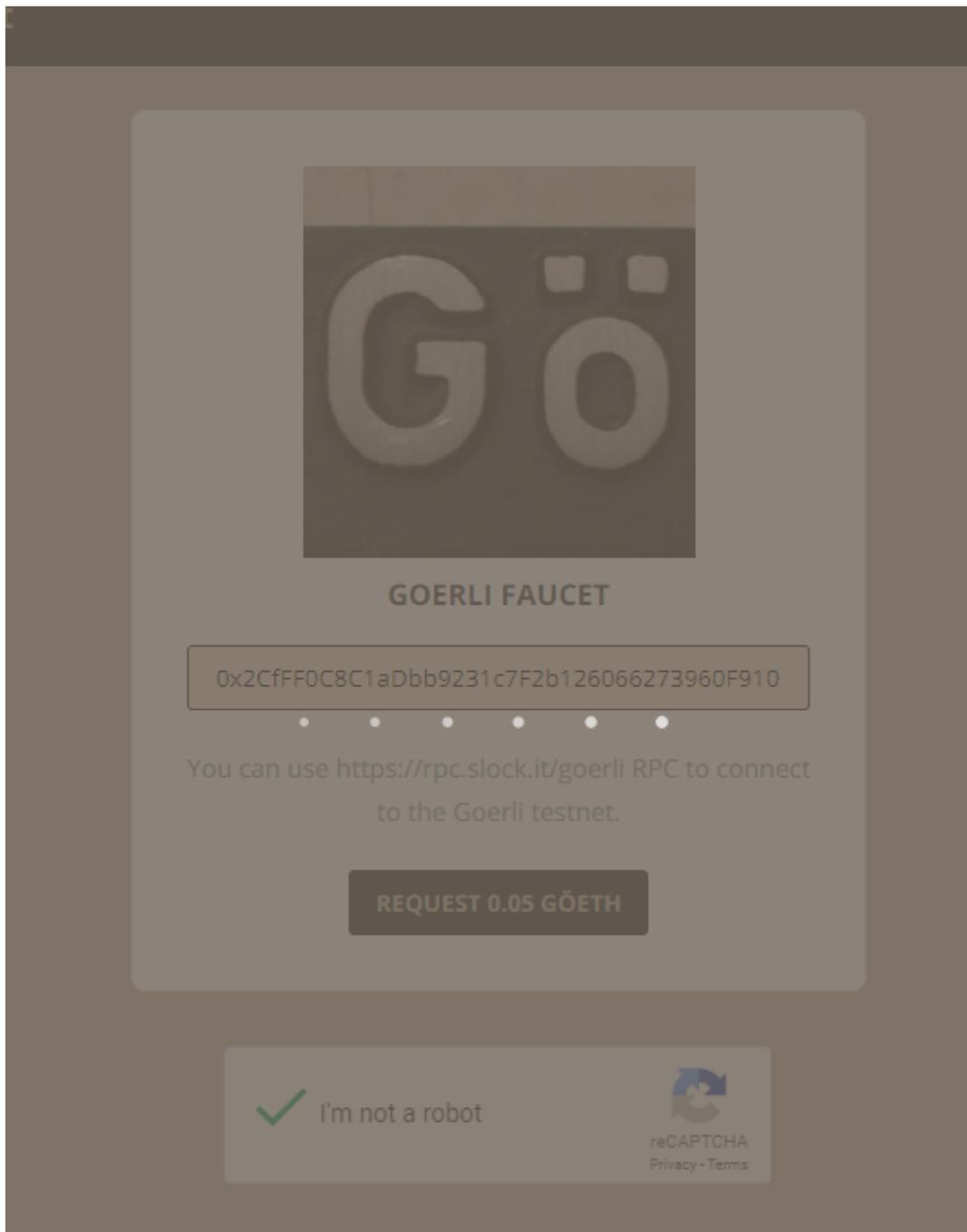
You can use <https://rpc.slock.it/goerli> RPC to connect
to the Goerli testnet.

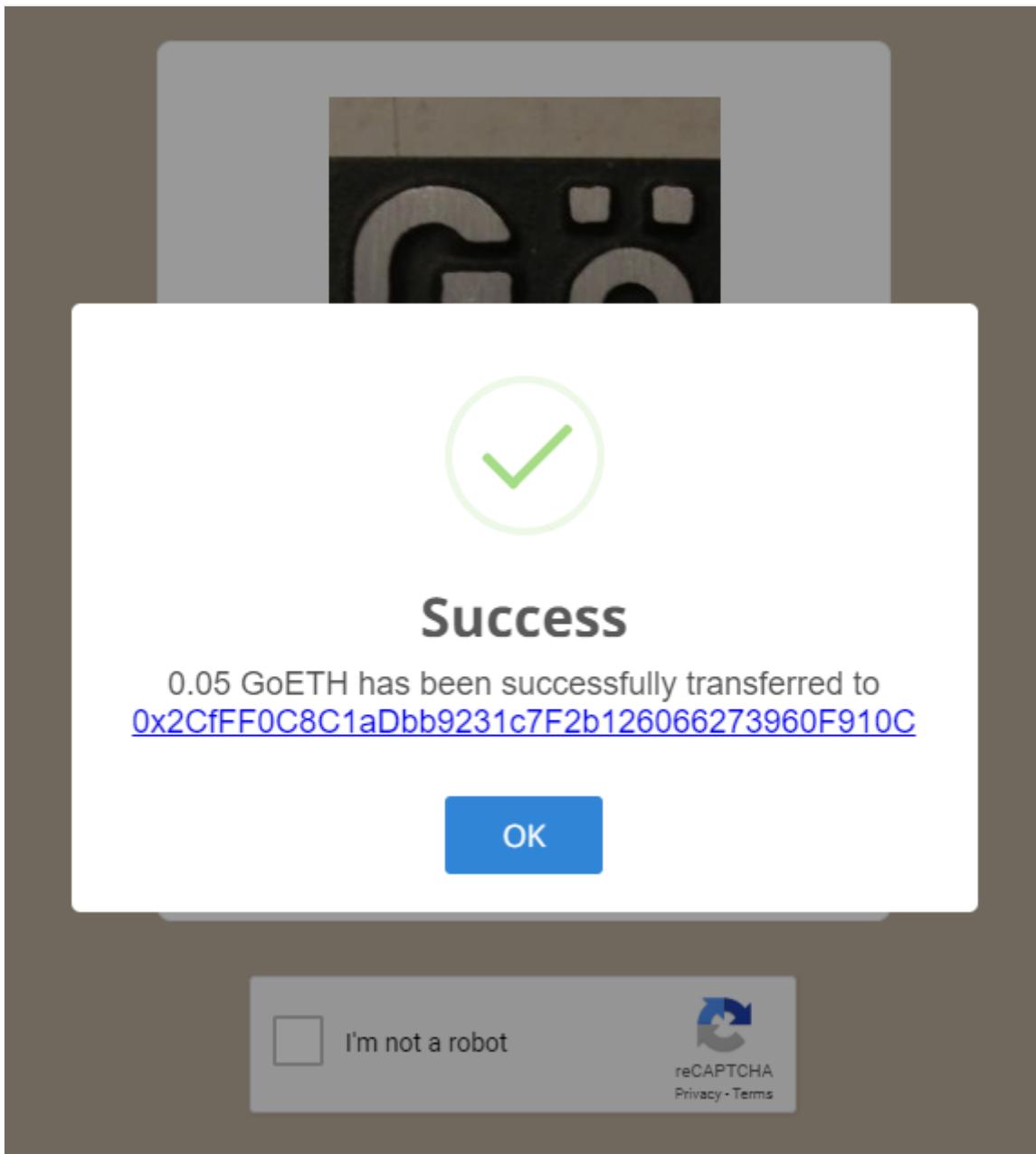
REQUEST 0.05 GÖETH

I'm not a robot



Wait until the popup appears...





Don't click the link of the transaction, most likely it will not really work anyways. Let's track our Incoming Transaction in the next step!

Last update: January 23, 2022

3.5 Track Ether

You might have heard it: all information on the Ethereum Blockchain is publicly visible information. So, if someone sends a transaction from A to B, then this is visible to all participants in the network.

There is specialized software to track those transactions, so called "Block explorers". One of them is [Etherscan](#).

Go to <https://etherscan.io/> and click the Ethereum logo at the top right and choose Goerli testnet.

The Ethereum Blockchain Explorer

All Filters ▾ Search by Address / Txn Hash / Block / Token / Ens

ETHER PRICE
\$1,227.37 @ 0.0341 BTC (-0.83%)

MARKET CAP
\$140,296,512,928

TRANSACTIONS
978.17 M (12.8 TPS)

DIFFICULTY
4,165.39 TH

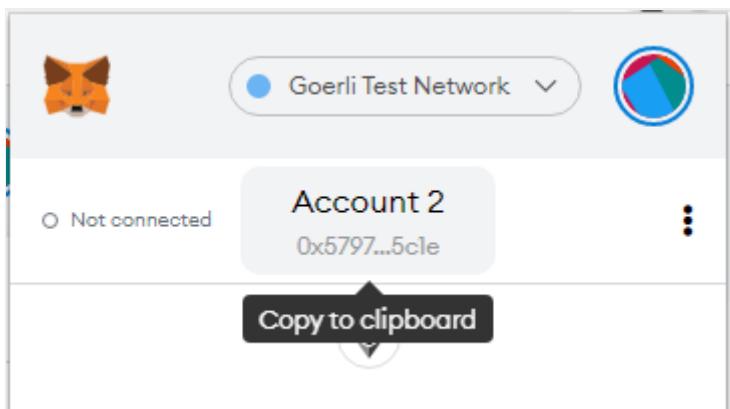
Latest Blocks

Bk 11681384 Miner Spark Pool 2.88572 Eth
40 secs ago 196 txns in 11 secs

Latest Transactions

Tx 0x28e...
40 secs

You should be at <https://goerli.etherscan.io/>. Copy and paste your address or copy the transaction hash from the previous step and paste it, either way, you should find a transaction that leads back to your wallet address:



The screenshot shows the Etherscan address page for the Goerli Testnet Network. The address is 0x5797B95C60Db61D3ed69b162733b2F008D695c1e. The page includes an "Overview" section with a balance of 0.05 Ether and a "Transactions" section showing one recent transaction.

Overview

Balance: 0.05 Ether

Transactions

1 Latest 1 from a total of 1 transactions

Txn Hash	Block	Age	From
0x39f5a4dff7f12d2c5af...	4128256	3 mins ago	0x8ced5ad0d8da4ec21...

 **Etherscan**

Goerli Testnet Network

Transaction Details

[Overview](#) [State](#)

[This is a Goerli Testnet transaction only]

⑦ Transaction Hash:	0x39f5a4dff7f12d2c5afed761934f059922e41963525b267a0e163de07
⑦ Status:	Success
⑦ Block:	4128256 <small>22 Block Confirmations</small>
⑦ Timestamp:	⑤ 5 mins ago (Jan-18-2021 08:35:10 PM +UTC)
⑦ From:	0x8ced5ad0d8da4ec211c17355ed3dbfec4cf0e5b9 
⑦ To:	0x5797b95c60db61d3ed69b162733b2f008d695c1e 
⑦ Value:	0.05 Ether (\$0.00)
⑦ Transaction Fee:	0.000021004 Ether (\$0.000000)
⑦ Gas Price:	0.0000000001 Ether (1 Gwei)

[Click to see More](#) ↓

You should see your transaction with the success message and all the details of the transaction.

Now open MetaMask from your browser and you should see some ETH in your wallet on a test-net.

[Video / Screenshots difference](#)

Note: I have 0.15ETH in my wallet, because I did this procedure 3 times for the screenshots.

The screenshot shows the Goerli Testnet Network interface on etherscan.io. The address is 0x72e5be33d15fbaf... and the balance is 0.15 Ether. There are three transactions listed, all from the same account to itself, each worth 0.05 Ether. The most recent transaction is 9 minutes ago.

Tx Hash	Block	Age	From	To	Value
0x0f772361ac84b...	3009200	1 min ago	0xdcaed4d0d0a...	0x72e5be33d15fbaf...	0.05 Ether
0x02c5ca124b91d0...	3009199	1 min ago	0xdcaed4d0d0a...	0x72e5be33d15fbaf...	0.05 Ether
0x0f21803324cd9e...	3009169	9 mins ago	0xdcaed4d0d0a...	0x72e5be33d15fbaf...	0.05 Ether

That's it. You have now installed a wallet and you have your first Ether ready. Let's carry on with the next steps!

Last update: April 17, 2021

3.6 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

4. Send Ether between Accounts

4.1 Lab: Send Ether between Accounts

In this lab you are going to send your first Ether between Accounts.

4.1.1 What You Know At The End Of The Lab

How to Create new Accounts in MetaMask

Send Money to different Accounts using MetaMask

Observe the Transaction using Etherscan.

4.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. Completed the Previous Lab with MetaMask and some Test-Ether

4.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

4.1.4 Get Started

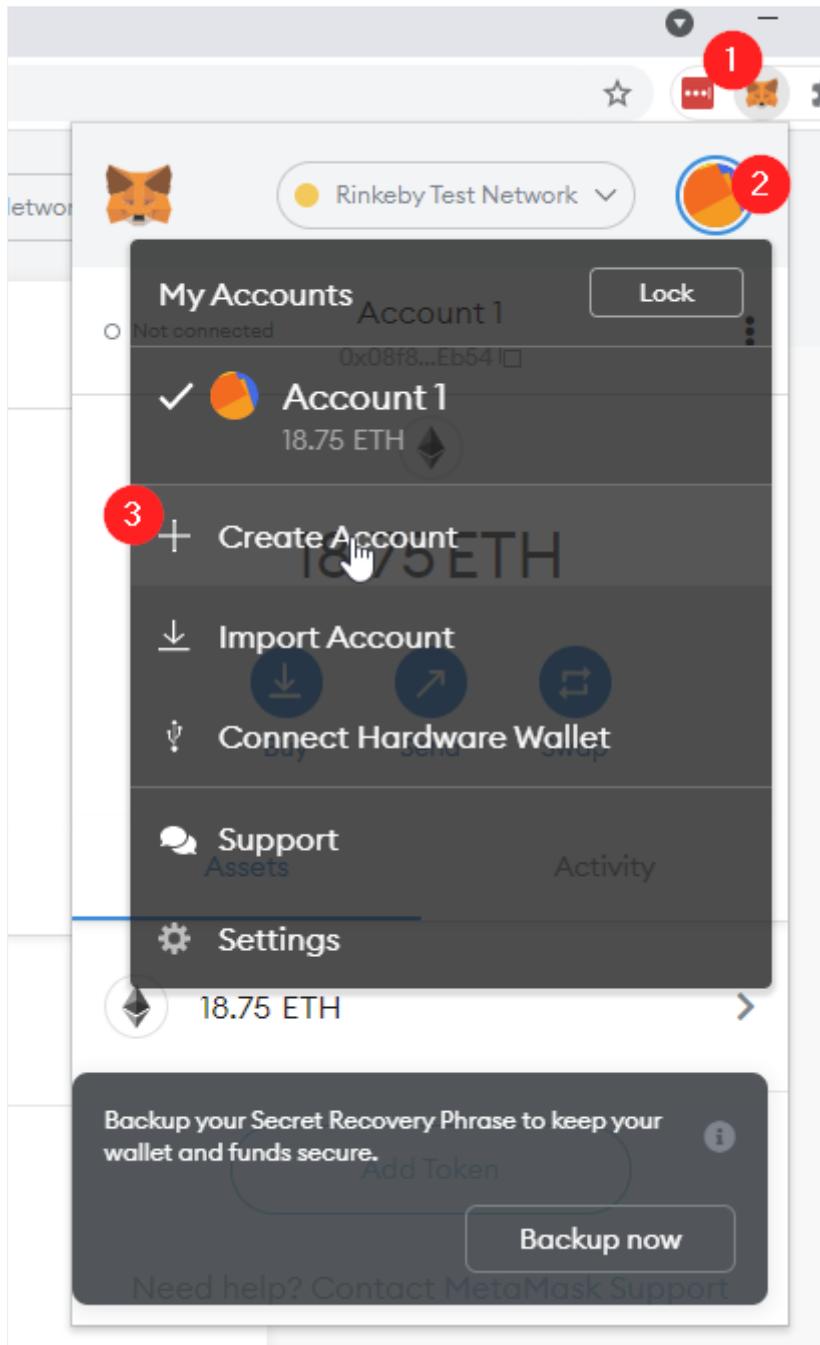
Let's get started [Creating a new Account](#)

Last update: January 3, 2022

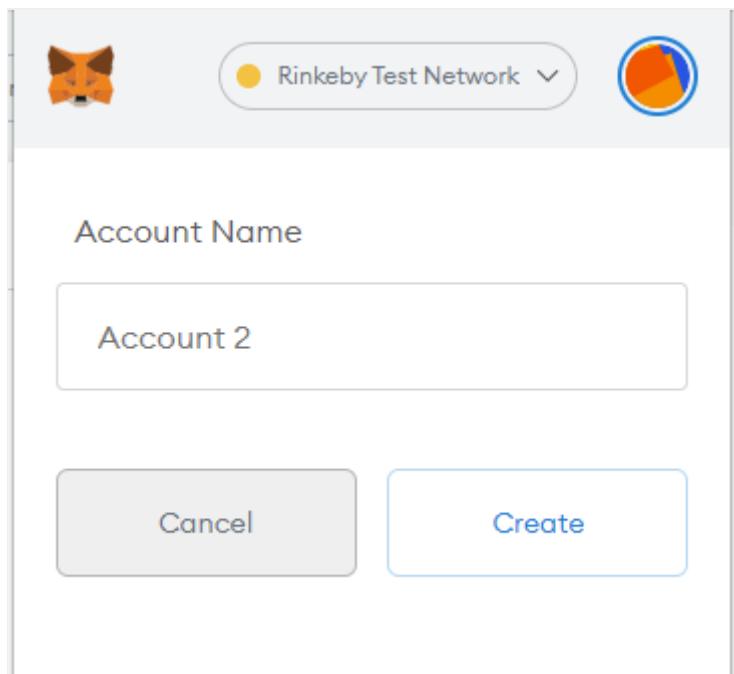
4.2 Create a new MetaMask Account

What we're trying to do in this lab is send some Ether from one account to another. But we don't want to waste the Ether. So, we are sending the Ether between two of our test accounts that we create ourselves.

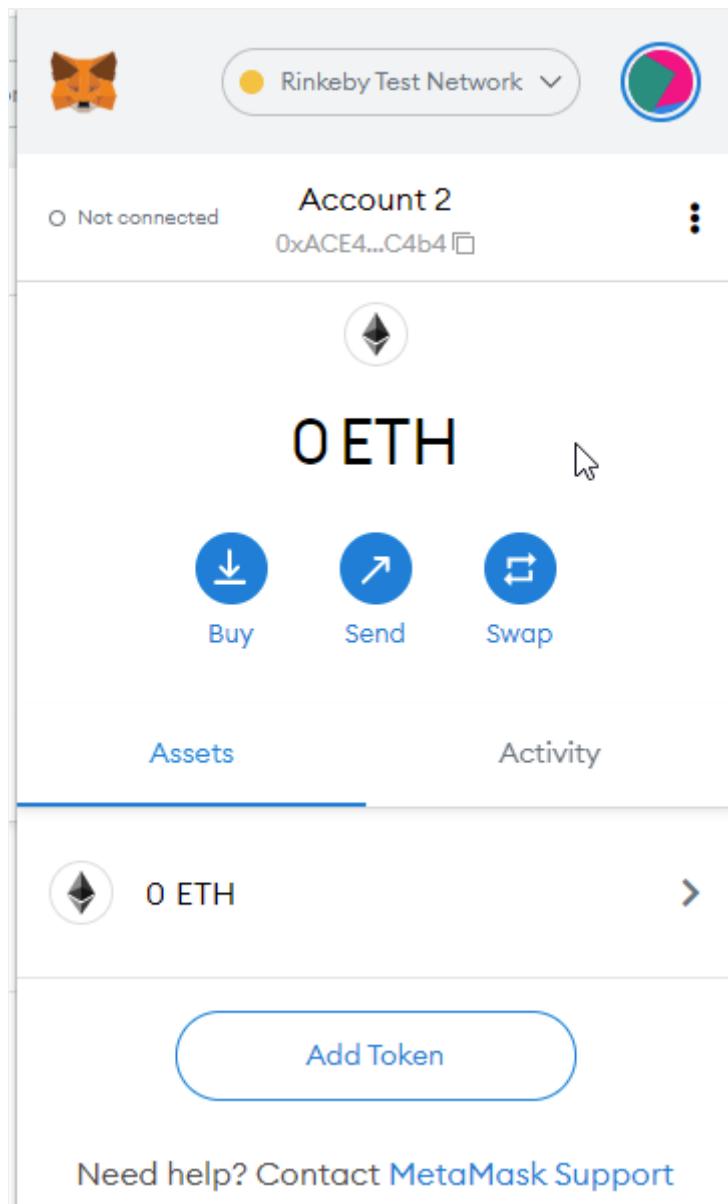
First, we need to create another account in MetaMask:



1. Open the MetaMask Plugin
2. Click on the little Account-Avatar on the top right corner
3. Hit "Create Account"



Hit "Create", you can leave the name to "Account 2" or whatever the default name is.



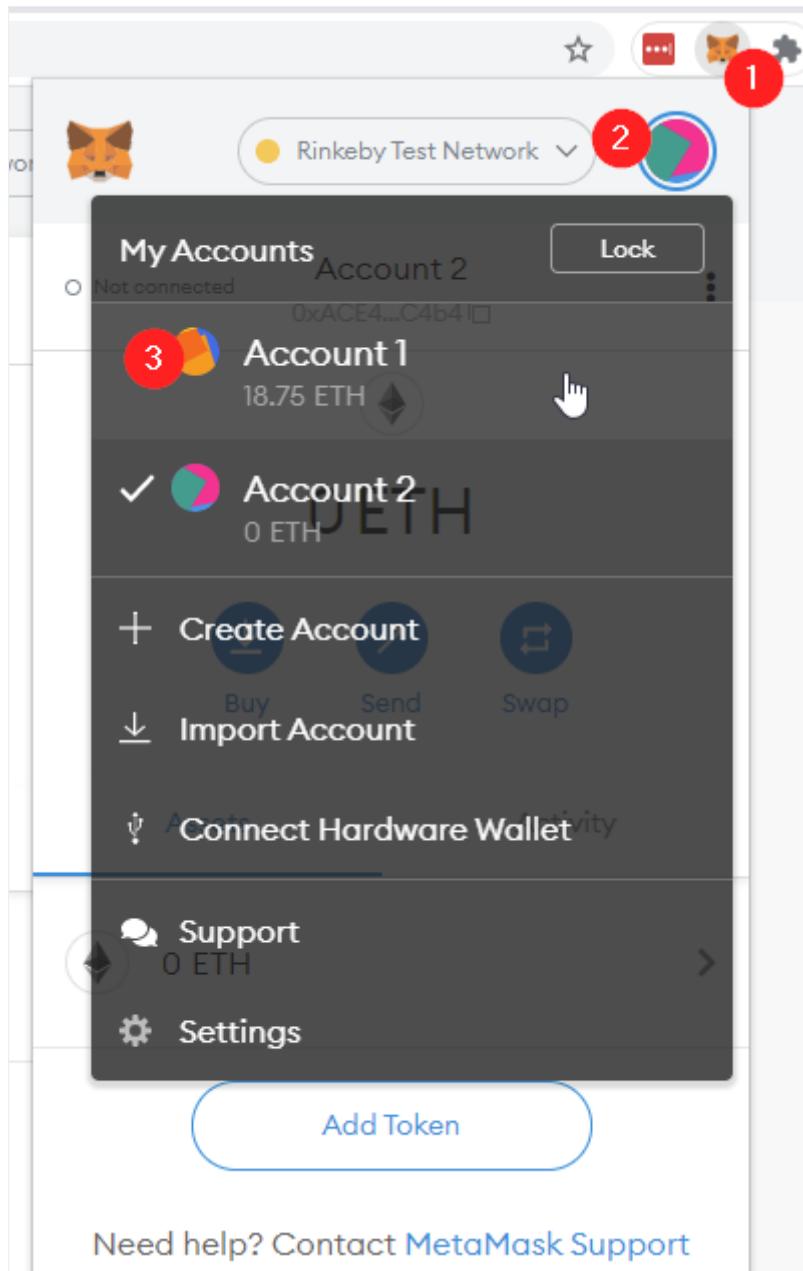
You now have two accounts. Now we can send Ether from one account to the other.

Last update: September 6, 2021

4.3 Transfer Ether from Account 1 to Account 2

You have two accounts now in MetaMask. You can switch between them. If you followed the previous tutorial, then you already have test-ether in your Account 1. Time to send some Ether from Account 1 to Account 2!

Switch back to Account 1:

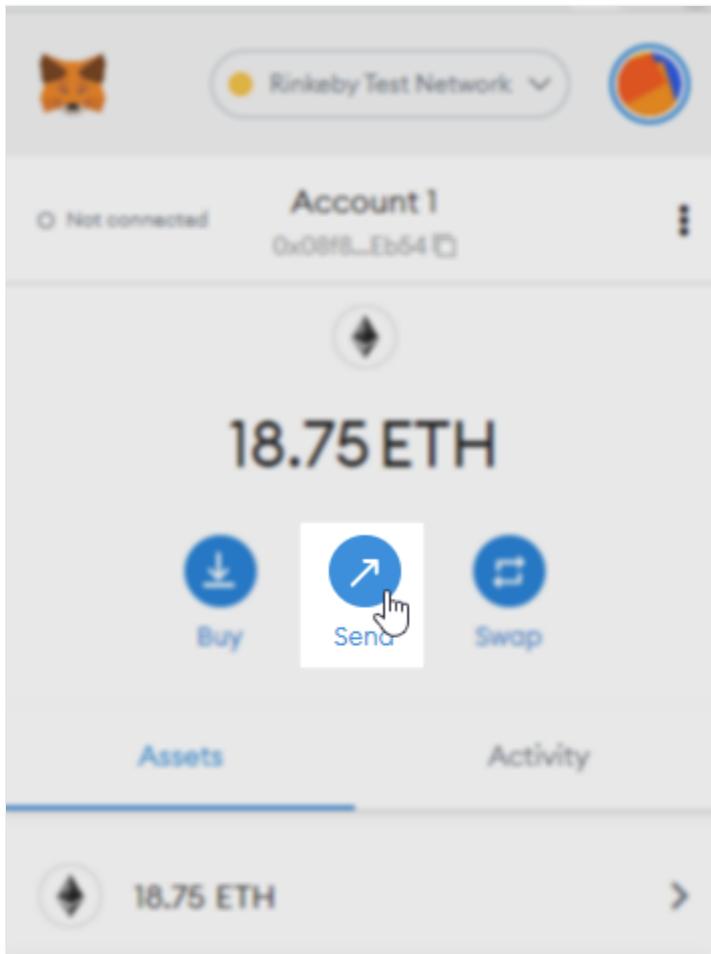


1. Open the MetaMask Extension
2. Click on the little Account avatar in the top right corner
3. Switch back to Account 1.

No Ether

If your Account 1 doesn't show any ether at all, run through the previous tutorial first to get some Ether on the Testnet. It doesn't have to be Ropsten, but on some network you need Ether, otherwise you can't complete this tutorial. [Previous Tutorial](#)

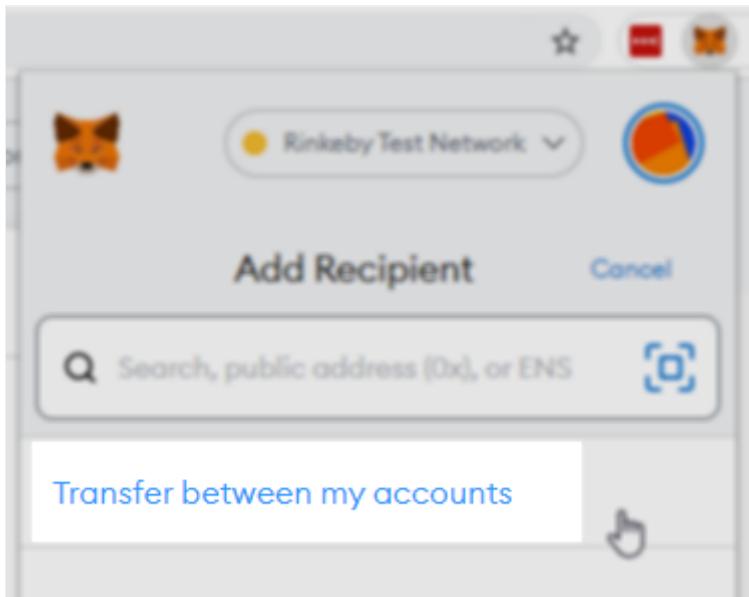
Click on the Send button:



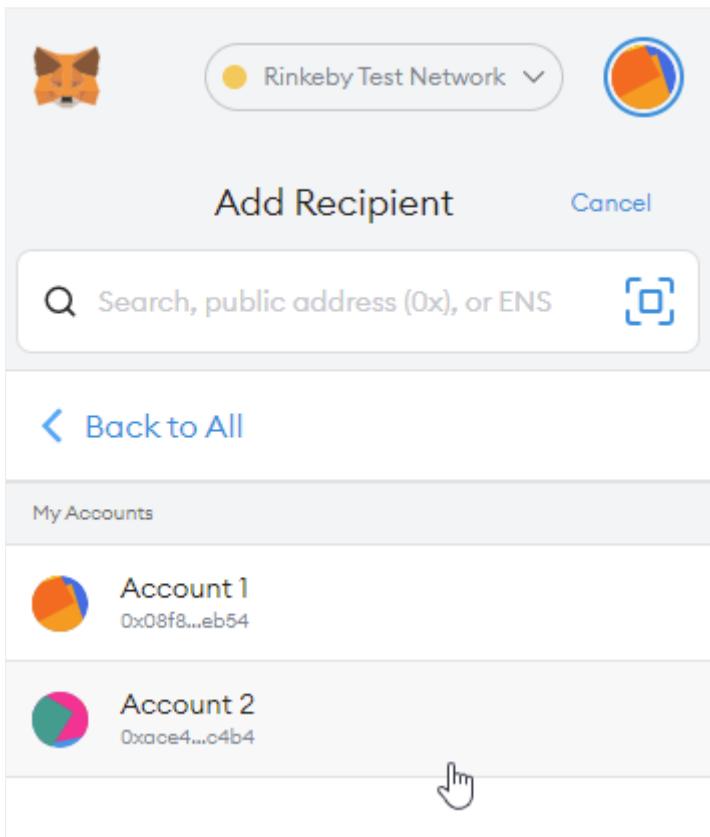
This will open up the Send-Page. Hit "Transfer Between my Accounts", because we want to transfer here between our Ethereum Accounts we have in MetaMask.

Transfer to other Account

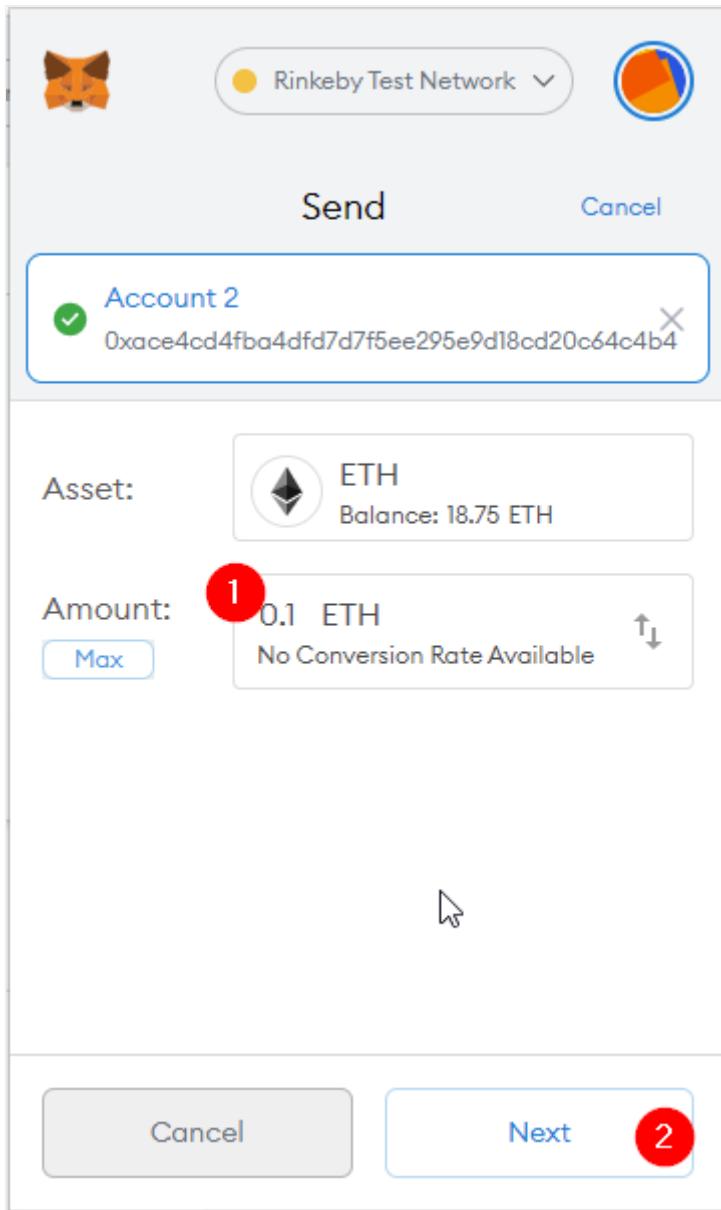
Of course, if you have to transfer Ether to different other Accounts, you can just paste the Ethereum Address into the input field, or even scan a QR code. We do this later on some time. Also feel free to just give it a try yourself. You can, for example, copy and paste your own Ethereum Address and skip the "Transfer between my Accounts" functionality.



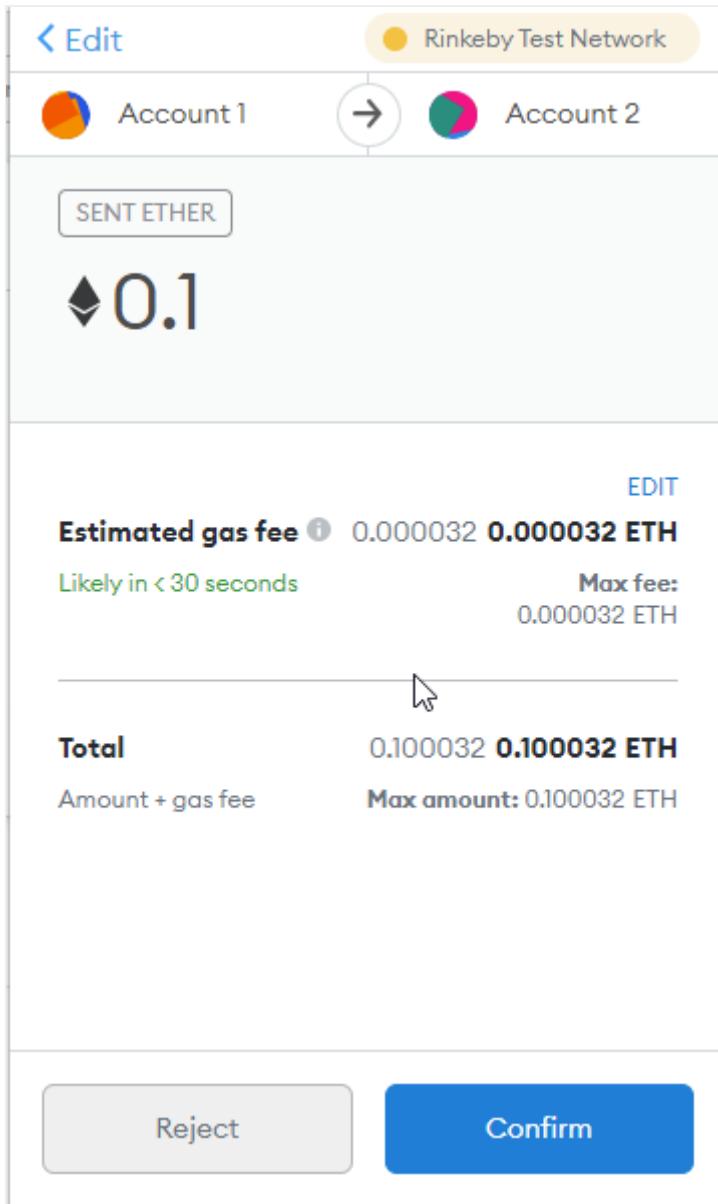
Select Account 2:



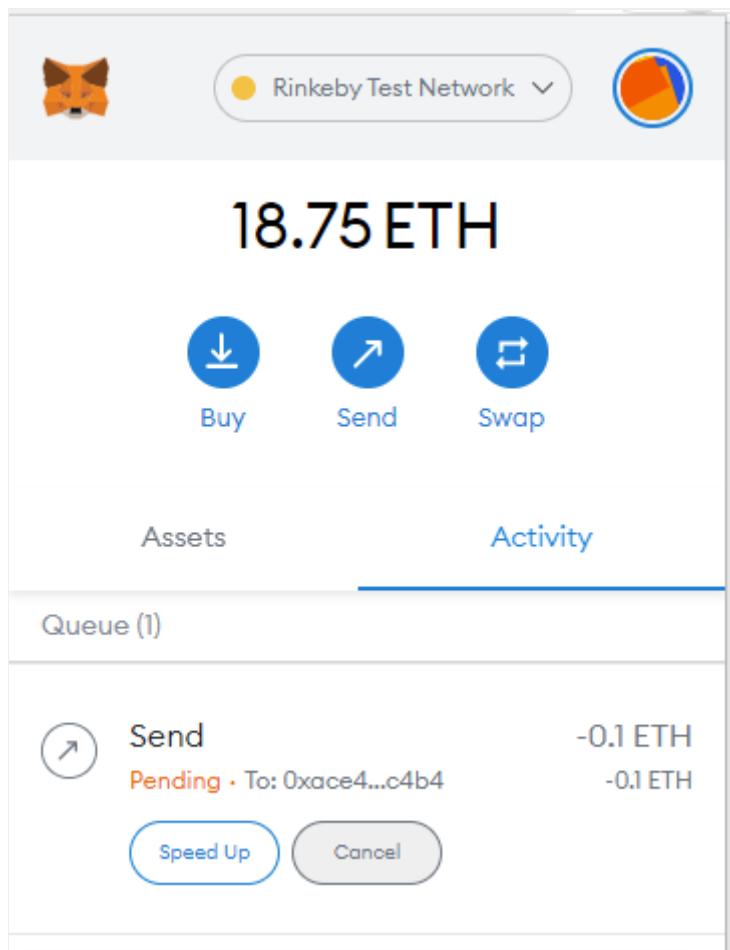
Ether 0.1 Eth and hit next:



Then have a look at the gas-cost overview. It might look a little bit different for you, but the essence is: You send 0.1 Eth, but it's not entirely for free. In this case the transfer costs (gas costs) amount to 0.000032 Eth:



Then simply hit "confirm". Now we can track the transaction using a Block Explorer. A Block Explorer is a public website that crawls the blockchain for new transactions and show them on a public website. Etherscan is one of many block explorers.

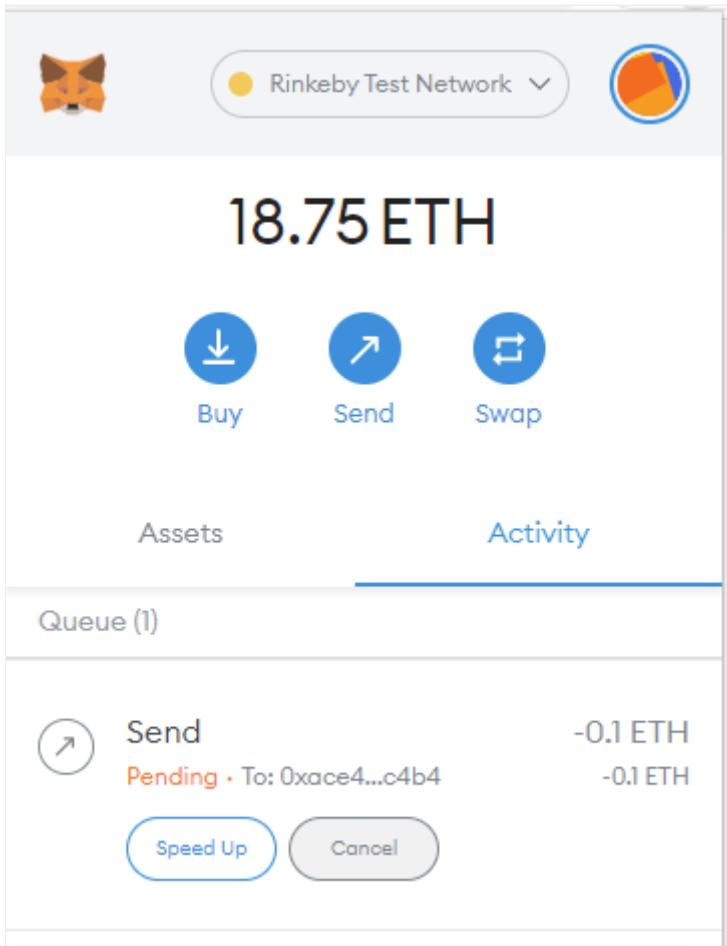


We do this next!

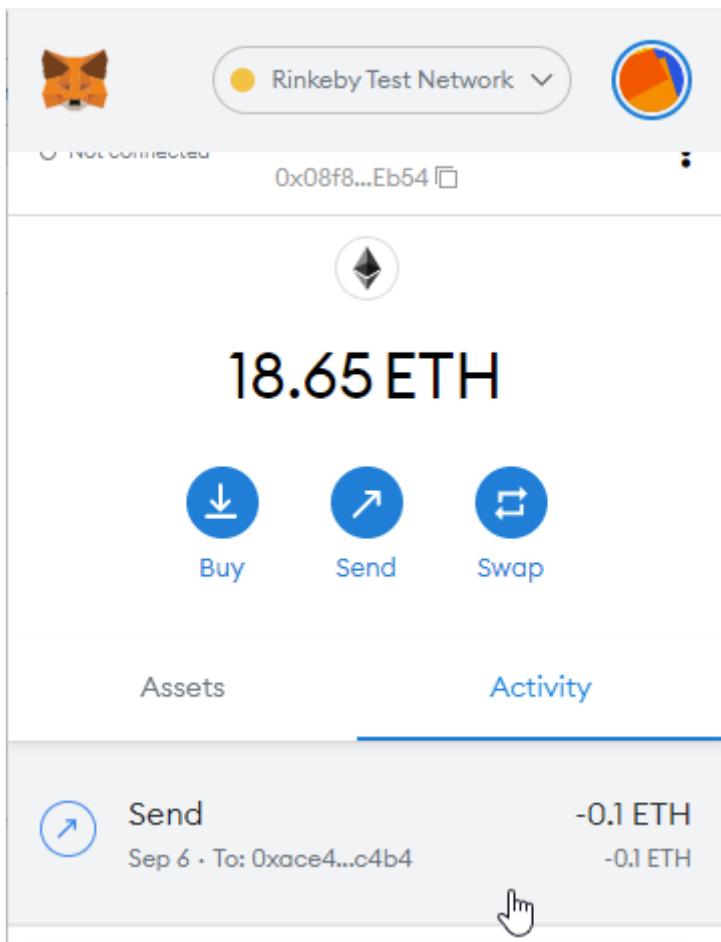
Last update: September 6, 2021

4.4 Track the Transaction using Etherscan

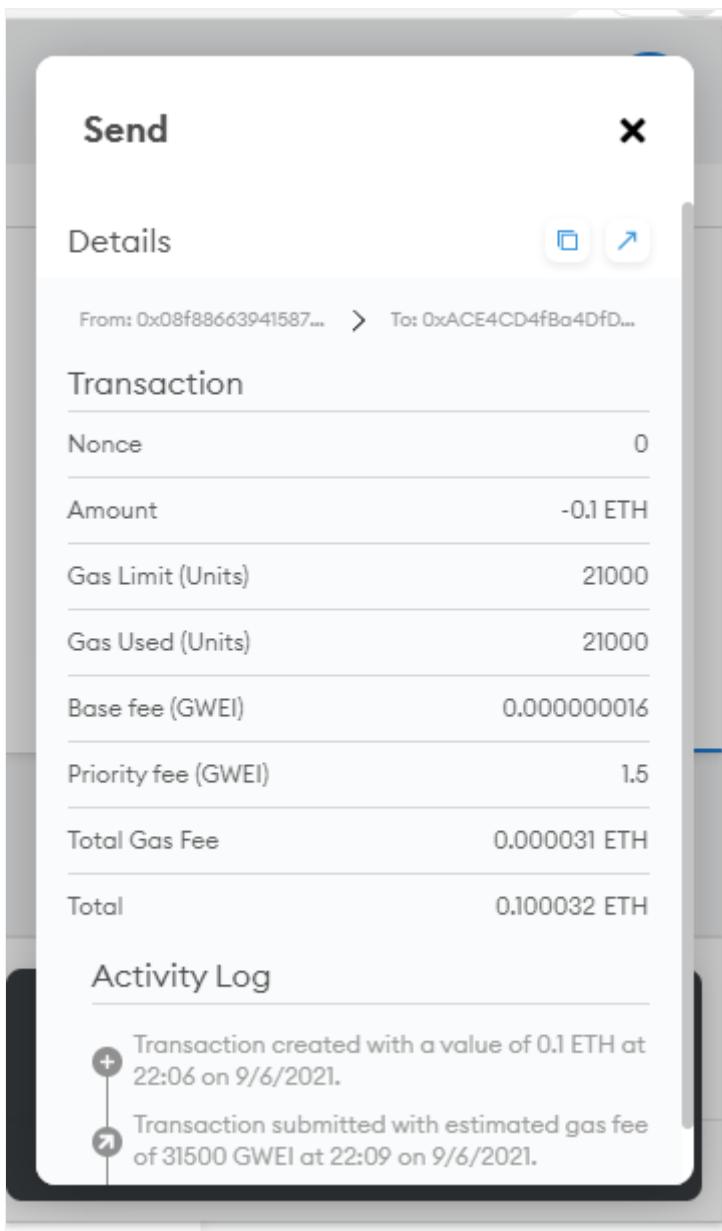
Now your MetaMask probably looks like this:



or, if the transaction already successfully mined, like this:



If you click on the entry (you might need to scroll down a bit), then you can open up a little modal window:



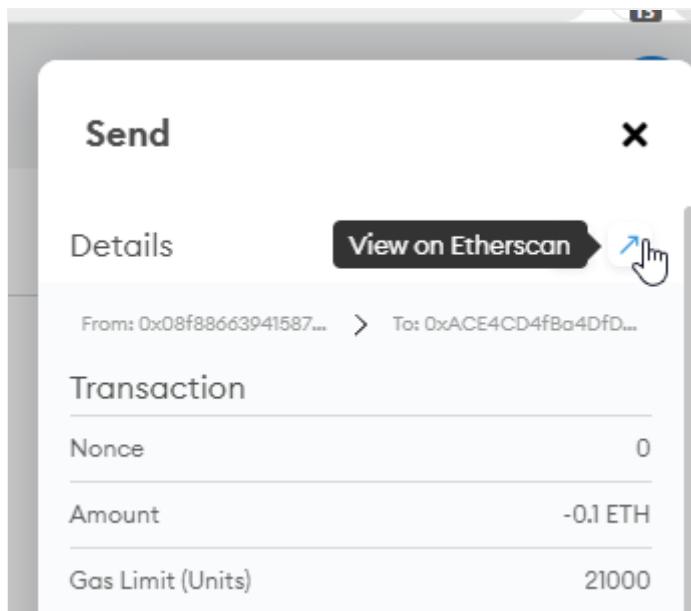
There are a few things you can see:

1. The Nonce. The Nonce is an ever increasing number on Ethereum that depicts how many transactions were sent from this account. Nonce 0 = First transaction .
2. Amount: We sent 0.1 Eth away
3. Gas: Simple transactions need 21000 gas (the gas stipend). More on that a little later, don't worry for now.
4. Base Fee and Priority Fee (Tip): We need to pay for gas with Eth. Its like using Water or Electricity and you need cash to pay for it. We're covering it later as well.
5. Total: The total amount we paid for it.

Basefee and Tip

Since [EIP-1559](#) we don't just do gas anymore, its broken down into a basefee that gets burned and a tip that goes to miners. It's still very similar to the previous gas auctions, just not everything goes to miners. More on that later as well.

But there is still some information missing. Like e.g. the block number or other transactions that were included in the block. That's something that block explorers can tell us! If you open up the link to Etherscan you can directly go to the transaction:



rinkeby.etherscan.io/tx/0xf5ab95d1cdcf15ae84caa852fffee502dcaae1c7dc86e4fafab790b5d5d37f5b



Etherscan

Rinkeby Testnet Network

Transaction Details

Overview	Access List	State
[This is a Rinkeby Testnet transaction only]		
② Transaction Hash:	0xf5ab95d1cdcf15ae84caa852fffee502dcaae1c7dc86e4fafab790b5d5d37f5b	
② Status:	Success	
② Block:	9247001	28 Block Confirmations
② Timestamp:	① 7 mins ago (Sep-06-2021 08:09:33 PM +UTC)	
② From:	0x08f88663941587cab55473086872b839bd09eb54	
② To:	0xace4cd4fba4dfd7d7f5ee295e9d18cd20c64c4b4	
② Value:	0.1 Ether (\$0.00)	
② Transaction Fee:	0.000031500000336 Ether (\$0.00)	
② Gas Price:	0.000000001500000016 Ether (1.500000016 Gwei)	
② Txn Type:	2 (EIP-1559)	
Click to see More ↓		

If you just want to see my transaction, you can click here: <https://rinkeby.etherscan.io/tx/0xf5ab95d1cdcf15ae84caa852fffee502dcaae1c7dc86e4fafab790b5d5d37f5b>

You can also click on the little "Click here to see more" link:

Overview	Access List	State
[This is a Rinkeby Testnet transaction only]		
② Transaction Hash:	0xf5ab95d1cdcf15ae84caa852fffee502dcaae1c7dc86e4fafab790b5d5	
② Status:	Success	
② Block:	9247001	28 Block Confirmations
② Timestamp:	7 mins ago (Sep-06-2021 08:09:33 PM +UTC)	
② From:	0x08f88663941587cab55473086872b839bd09eb54	📋
② To:	0xace4cd4fba4dfd7d7f5ee295e9d18cd20c64c4b4	📋
② Value:	0.1 Ether (\$0.00)	
② Transaction Fee:	0.000031500000336 Ether (\$0.00)	
② Gas Price:	0.000000001500000016 Ether (1.500000016 Gwei)	
② Txn Type:	2 (EIP-1559)	
② Gas Limit:	21,000	
② Gas Used by Transaction:	21,000 (100%)	
② Base Fee Per Gas:	16 wei (0.000000016 Gwei)	
② Max Fee Per Gas:	0.000000001500000023 Ether (1.500000023 Gwei)	
② Max Priority Fee Per Gas:	0.0000000015 Ether (1.5 Gwei)	
② Burnt Fees:	0.00000000000000336 Ether	
② Txn Savings:	0.00000000000000147 Ether	

You see all the information about the transaction, including in which block it was included and many more things we cover later in the course.

Last update: September 6, 2021

4.5 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: September 6, 2021

5. Remix

5.1 Lab: Write your first Smart Contract

In this lab you are going to write your very first smart contract. We are also going to deploy it to a blockchain.

The screenshot shows the Remix IDE interface. On the left is a sidebar with various icons: a blue headphones icon, a file icon, a green checkmark icon, a diamond icon, a chart icon with a '1' (highlighted in orange), a downward arrow icon, and a hand icon. The main area has tabs for 'Home' and 'MyContract.sol'. The code editor contains the following Solidity code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

The 'DEPLOY & RUN TRANSACTIONS' section includes fields for 'ENVIRONMENT' (set to 'Injected Web3' with 'Goerli (5) network'), 'ACCOUNT' (a dropdown menu), 'GAS LIMIT' (set to 3000000), 'VALUE' (0 wei), and a 'CONTRACT' dropdown set to 'MyContract - browser/MyContract.sol'. A large orange 'Deploy' button is prominent. Below it is a checkbox for 'Publish to IPFS'. The 'Transactions recorded' section shows 0 transactions. The 'Deployed Contracts' section is empty, displaying the message: 'Currently you have no contract instances to interact with.'

5.1.1 What You Know At The End Of The Lab

Understand Remix IDE and the Tooling

Get started with Smart Contract Development Fast and Easy

- ✿ Setup Remix the right way
- ⌚ No matter if you want to write in Solidity 0.5.x, 0.6.x, 0.7.x or 0.8.x!

Connect MetaMask and Remix to deploy to Görli

5.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. Completed the Previous Lab with MetaMask and some Test-Ether

5.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

5.1.4 Get Started

Let's get started by [Setting up Remix](#)

Last update: September 6, 2021

5.2 Setup Remix

First we need to setup Remix to have the correct plugins installed, activated and configured. Before we do that, some general information!

What's Remix anyways?

Remix, previously known as Browser-Solidity, is a browser based development environment for Smart Contracts. It comes with compilers for different solidity versions and a blockchain simulation. It also has plenty of other plugins. It's a great way to get started!

HTTP vs HTTPS

Be careful with the https vs http domain. Remix stores edited files in localstorage of the browser. If your smart contracts are suddenly gone, look at the protocol.

In this course we work with http, not https. This is especially important later when we do private blockchains which require CORS to be setup correctly.

5.2.1 Open Remix

Go to <http://remix.ethereum.org>. You should be greeted with the following page:

The screenshot shows the Remix Ethereum IDE interface. At the top, there's a browser-like header with the title "Remix - Ethereum IDE" and the URL "remix.ethereum.org/". Below this is the main workspace area.

FILE EXPLORERS

Workspaces + 🖍️ 🗑️

default_workspace

- contracts
- scripts
- tests
- README.txt

Help us to improve Remix IDE

An Opt-in version of [Matomo](#), an open source data used to improve Remix IDE.

We realize that our users have sensitive information privacy - your privacy - must be protected.

All data collected through Matomo is stored on our given to third parties. Our analytics reports are pu

We do not collect nor store any personally identifiable

For more info, see: [Matomo Analytics on Remix IDE](#)

You can change your choice in the [Settings panel](#)

Sure

IMPORT FROM:

- New File
- Open Files
- Connect to Localhost

Ciast GitHub Swaym ...

0 listen on network

- swarmigw
- remix (run remix.help() for more info)

If the popup shows up for you, then feel free to accept if you have no concerns over privacy violations. For our course we can do that.

Updates from Video

In the video we are using an older version of Remix. Currently, by default, Remix starts with the dark theme. In the videos you see the light theme. You can change this in the settings: Bottom left, scroll down, theme light.

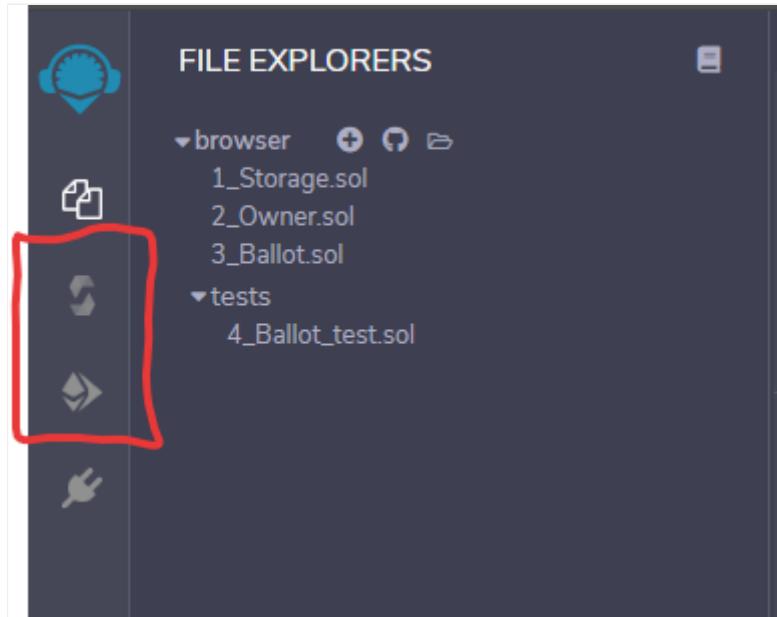
More importantly, in the videos we had to enable plugins. The most important plugins are now enabled by default. Below we're still making sure they are enabled, just in case.

5.2.2 Plugins

Remix is built with a pluggable architecture. All functions are done via plugins. The Compiler is a plugin, the blockchain connection is a plugin, the debugging functionality is a plugin and there are a lot of other plugins that might be useful.

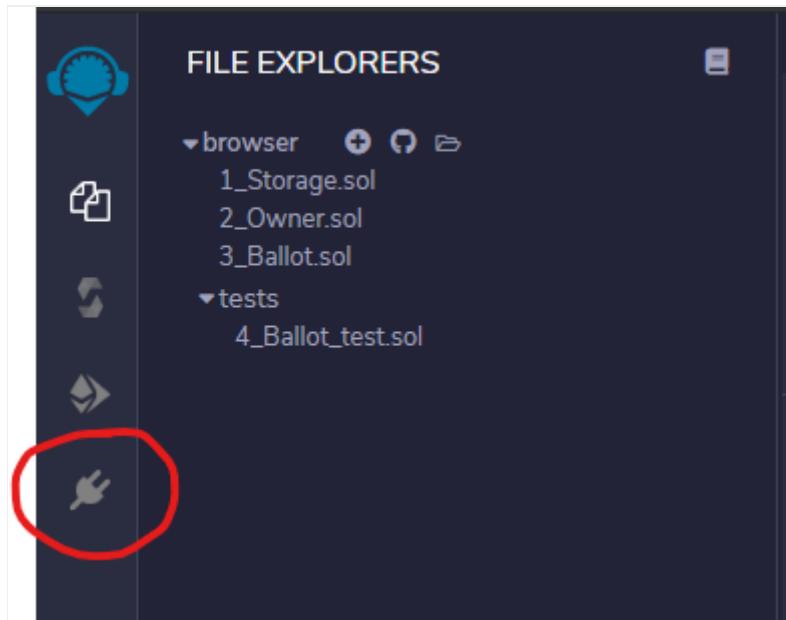
What you need in the next few chapters are

1. The Solidity compiler
2. The "Deploy & Run Transactions" Plugin

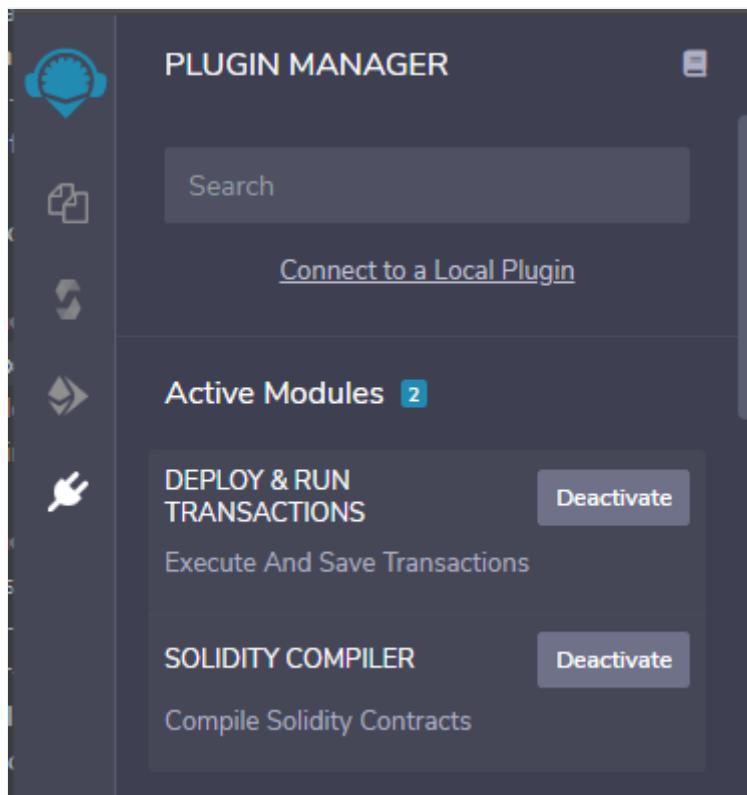


5.2.3 Enable Plugins

If the plugins are not showing up yet, then click on the plugin symbol and enable them:



And find the two plugins and activate them:



5.2.4 Configure the Compiler

In this chapter we are working with Solidity 0.8.1. The compiler will normally switch automatically based on the `pragma` line in your solidity files. But you can set a specific version, if necessary.

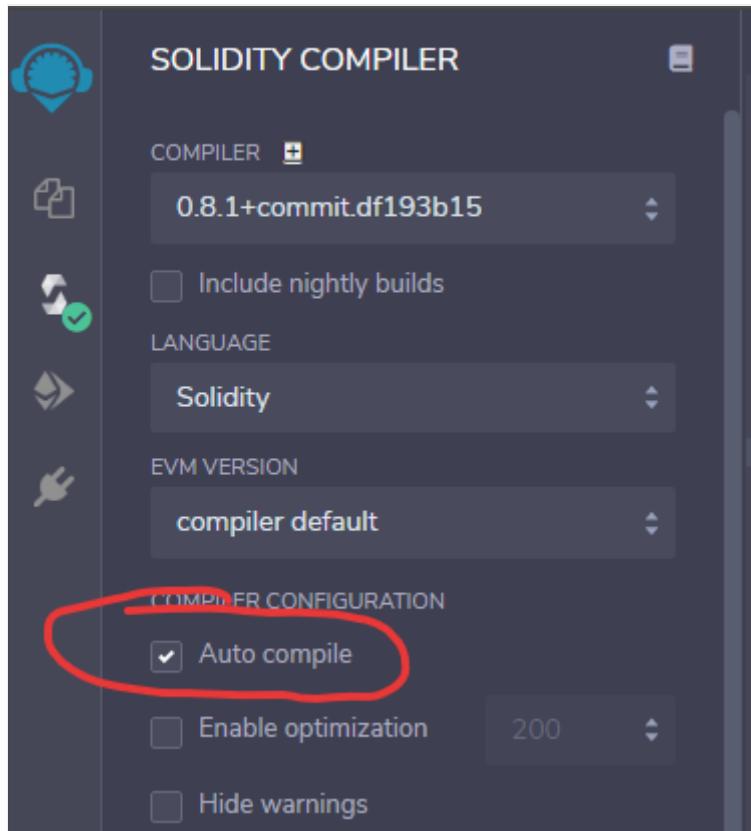
If you don't know what that is and don't want to wait several videos to understand what a pragma line is: In layman terms, it's here to configure your compiler. For example there's a version pragma, that tells the compiler "Hey, this source is made for compiler version XYZ". That's what we're going to use. Need more information? Either wait, or [read the official docs](#)

Switch Compiler Version

If it is necessary to switch compiler versions manually, you can always do this. You can either follow along in the videos, then use the compiler version the videos are using. Or you follow along this guide and use this solidity version.

New Compiler versions are published very frequently. It is very normal to find "outdated" solidity files around. Some very popular projects are using older solidity versions.

Make sure "auto-compile" is enabled:



Great! You're all set! Let's create your first file in the next section!

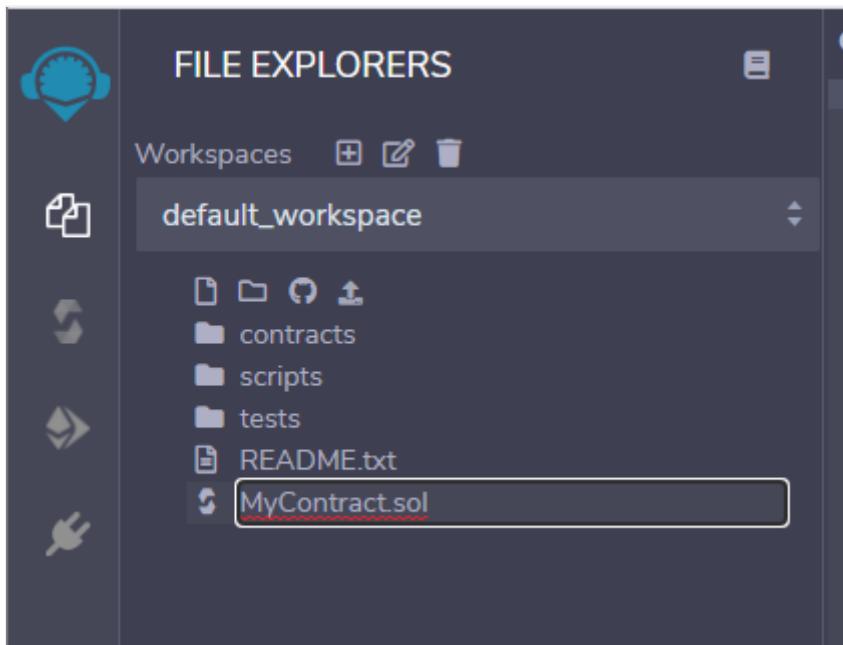
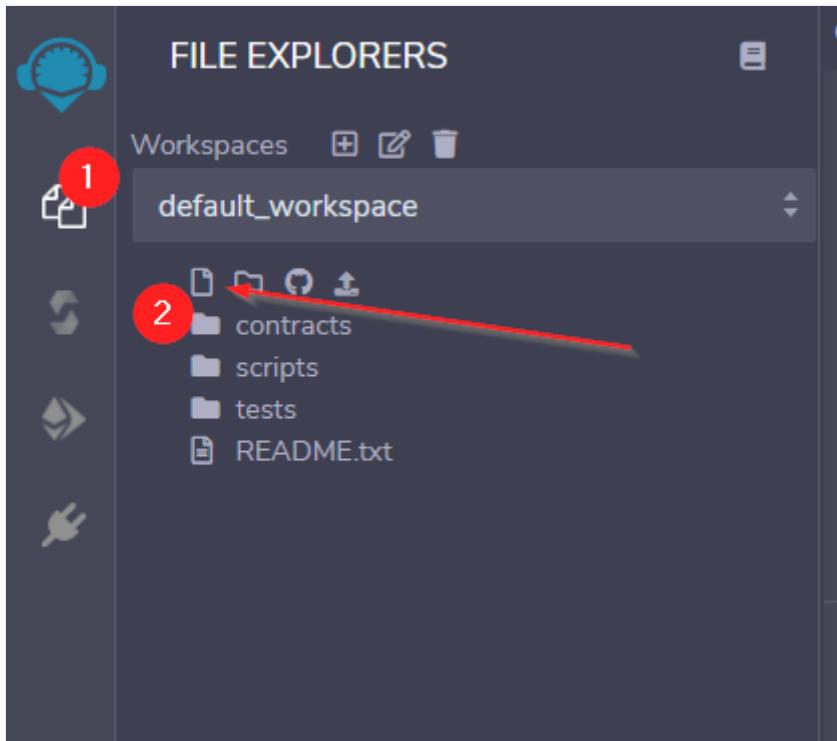
Last update: September 6, 2021

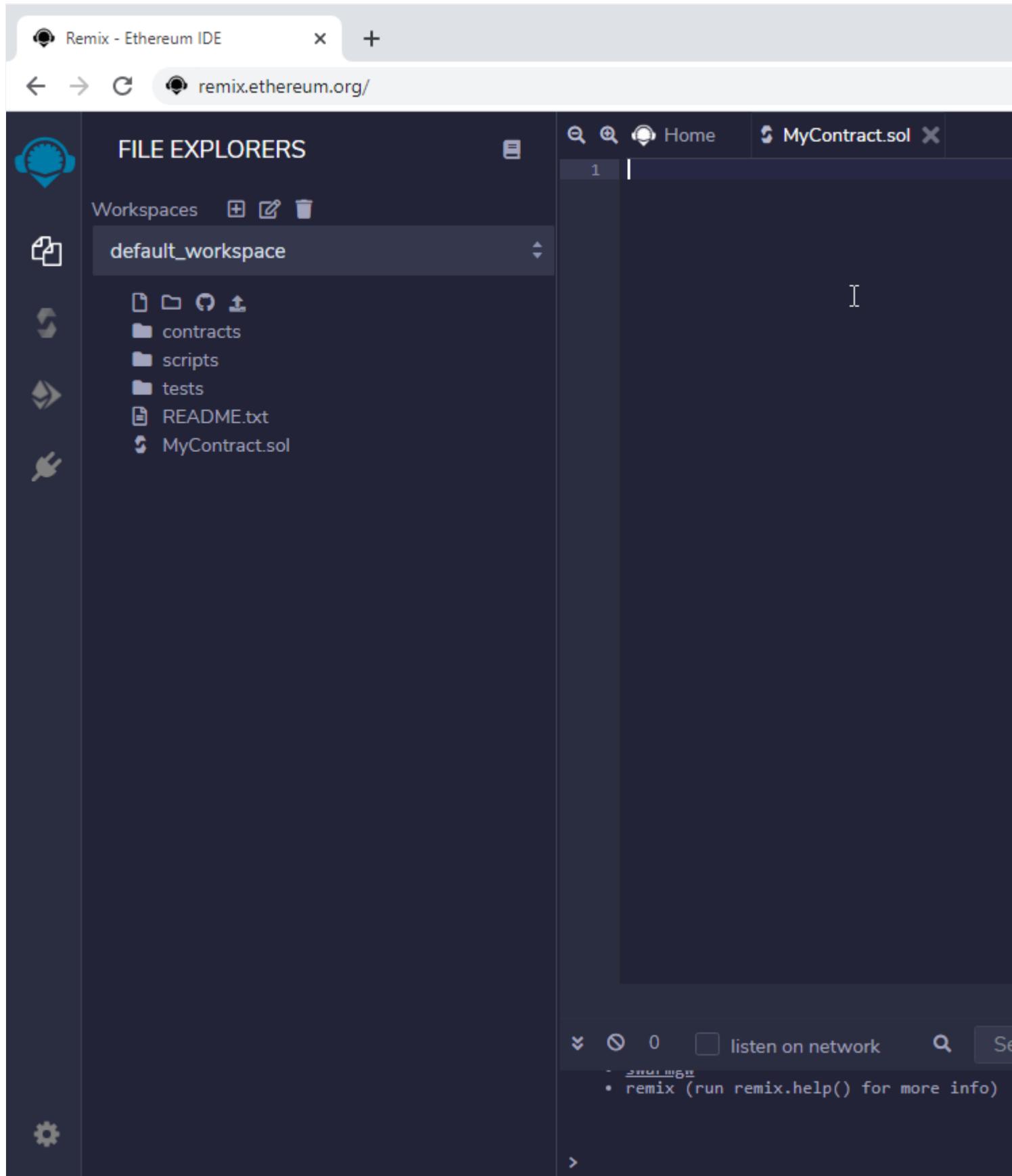
5.3 Create Your First Smart Contract

Now we are creating a new file and inserting some Solidity code. Don't worry if you don't fully understand everything - we have to start somewhere and we're here to play around. Just follow along, I promise everything will be clear later on!

5.3.1 Create A New File

Click on the plus icon in the code-editor and create a new file, name it "MyContract.sol". The sol-extension stands for Solidity.





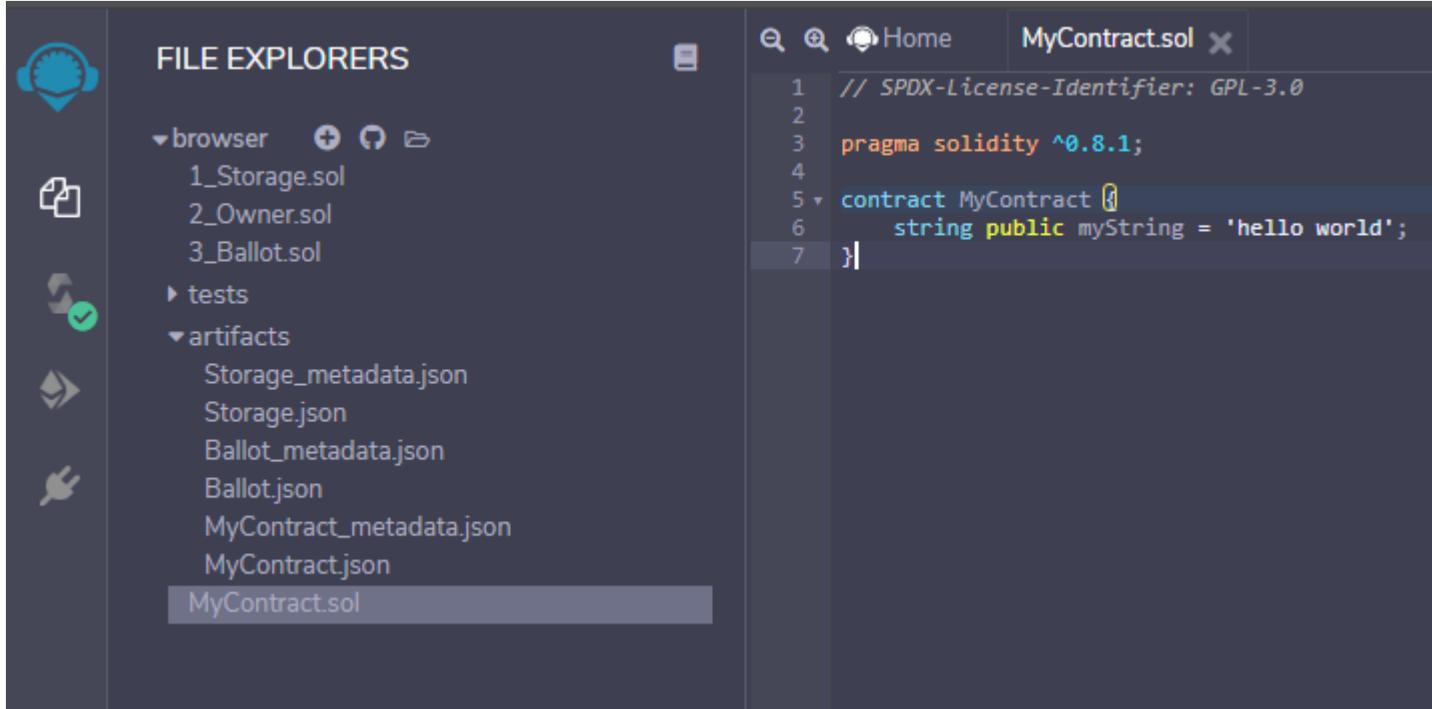
5.3.2 Add Solidity Hello World Code

Now add the following code to the file:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract MyContract {
    string public myString = 'hello world';
}
```

It should look like this and the "Compiler" Plugin should have a green checkmark badge. That's the icon in the left side panel:



Wondering what that all means?

This is a very basic version of a Smart Contract. Let's go through it line by line:

// SPDX-License-Identifier: GPL-3.0 : The [The Software Package Data Exchange® \(SPDX®\)](#) identifier is there to clearly communicate the license under which the Solidity file will be made available. Well, if you make it available. But you should. Smart Contracts transparency and trust greatly benefit from the source being published and sometimes it's not 100% clear under which license the source is out in the wild. The [SPDX identifier is optional](#), but recommended.

pragma solidity ^0.8.1 : The `pragma` keyword is for the compiler to enable certain features or check certain things. The version pragma is a safety measure, to let the compiler know for which compiler version the Solidity file was written for. It follows the [SemVer versioning standard](#). ^0.8.1 means >=0.8.1 and <0.9.0.

contract MyContract : That's the actual beginning of the Smart Contract. Like a Class in almost any other programming language.

string public myString = 'hello world' : That is a storage variable. It's public and Solidity will automatically generate a getter function for it - you'll see that in a minute!

Perfect! Let's proceed to deploy this Smart Contract to a Blockchain!

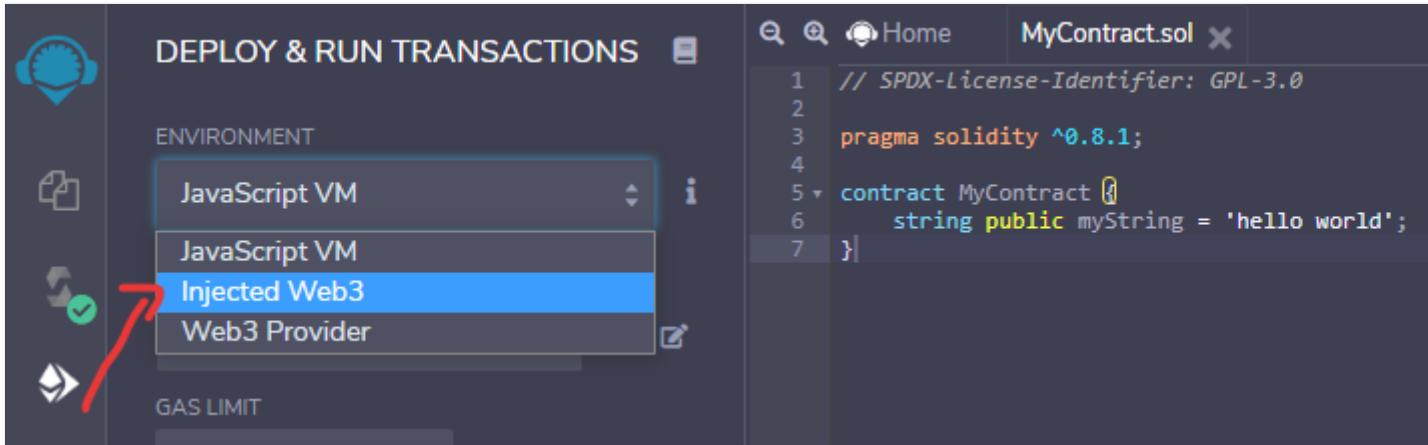
Last update: September 6, 2021

5.4 Deploy Smart Contract

Now it's time to deploy our Smart Contract. We will do this to a real blockchain. In the previous video we got our first Ether on a Test-Network. We will use them now to deploy the smart contract.

5.4.1 Connect MetaMask to Remix

Switch over to the "Deploy & Run Transactions" Plugin. We need to configure it, so it uses our MetaMask Wallet to access the Blockchain.



```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

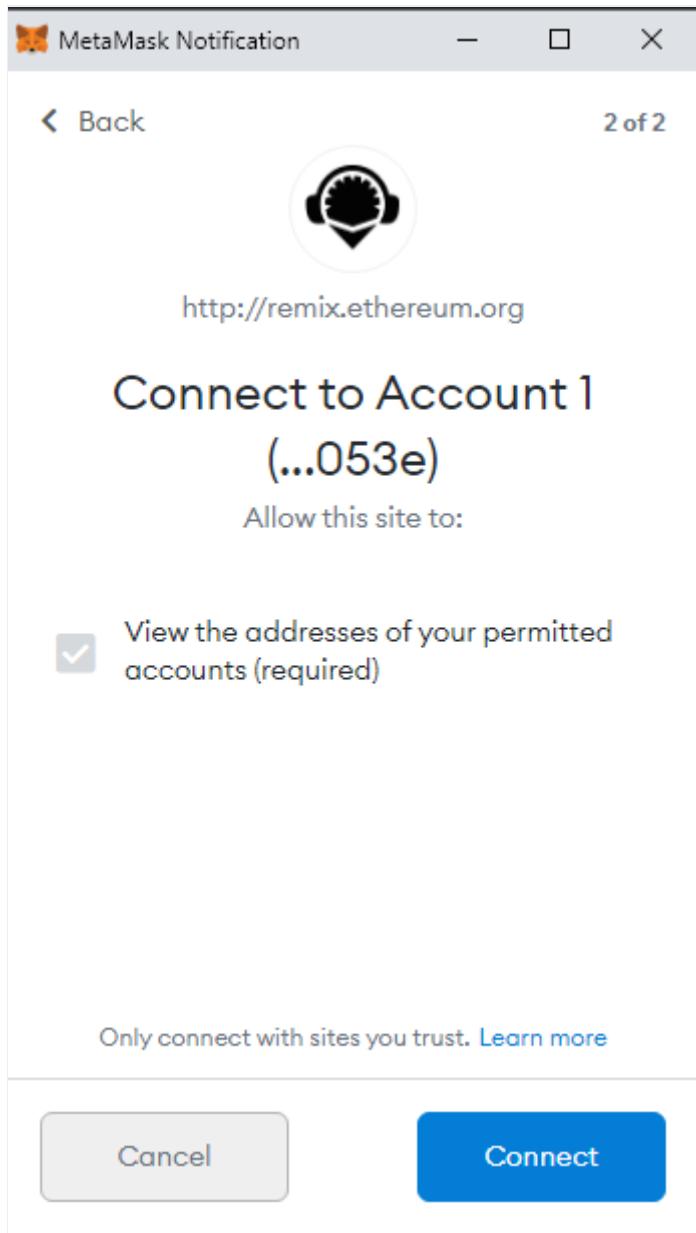
As soon as you do this, MetaMask should pop up and ask you to connect your account to Remix.

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" and "MyContract.sol". The "DEPLOY & RUN TRANSACTIONS" tab is active, showing fields for ENVIRONMENT (set to "Injected Web3" with "Goerli (5) network"), ACCOUNT (selected account), GAS LIMIT (3000000), VALUE (0 wei), CONTRACT (MyContract - browser/MyContracts.sol), and a Deploy button. Below this, there's a checkbox for "Publish to IPFS". The "OR" section allows selecting "At Address" or "Load contract from Address". Under "Transactions recorded", it says "0". The "Deployed Contracts" section shows a message: "Currently you have no contract instances to interact with." The "MyContract.sol" tab is open, displaying the following Solidity code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract MyContract {
    string public myString = 'hello world';
}
```

Click "Next" and the "Connect":



Now your account should pop-up in the dropdown under the Environment Selection:

The screenshot shows the Remix IDE interface. On the left, the "DEPLOY & RUN TRANSACTIONS" sidebar has an "ENVIRONMENT" section set to "Injected Web3" and a "Goerli (5) network" button. Below it is an "ACCOUNT" section with a dropdown menu showing "0x5AD...3053E (0.0271579...)". The main workspace on the right contains a Solidity code editor with the file "MyContract.sol" open. The code is as follows:

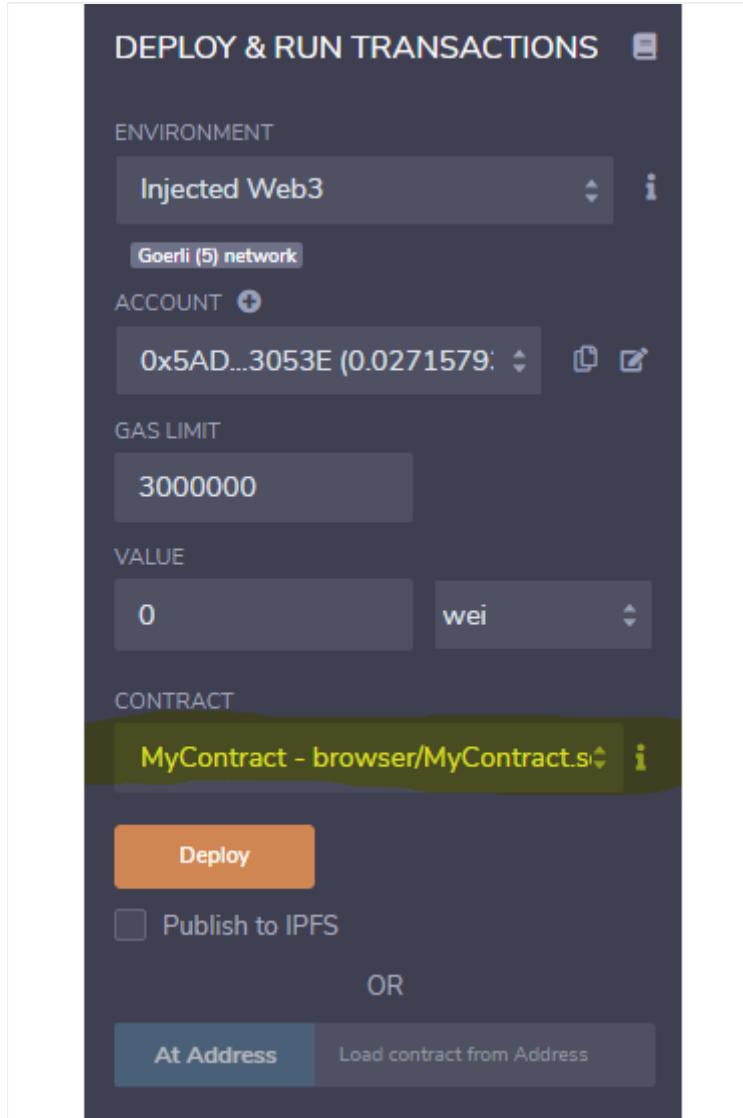
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

Account not showing up?

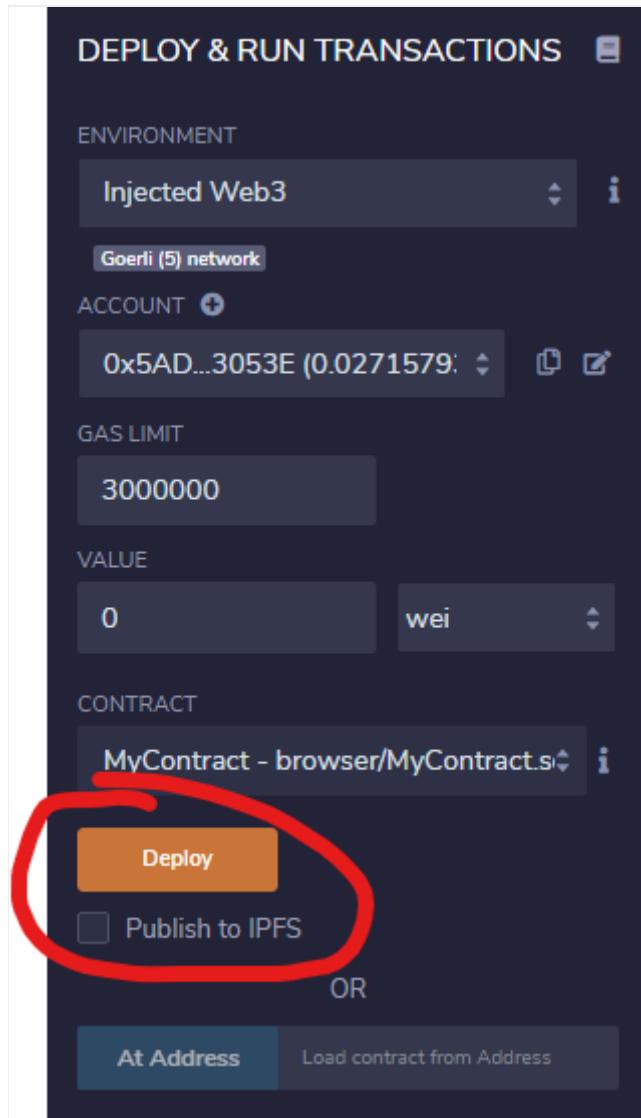
If your account doesn't show up, or MetaMask doesn't pop up, try to reload the page. There are sometimes caching issues.

5.4.2 Deploy the Smart Contract

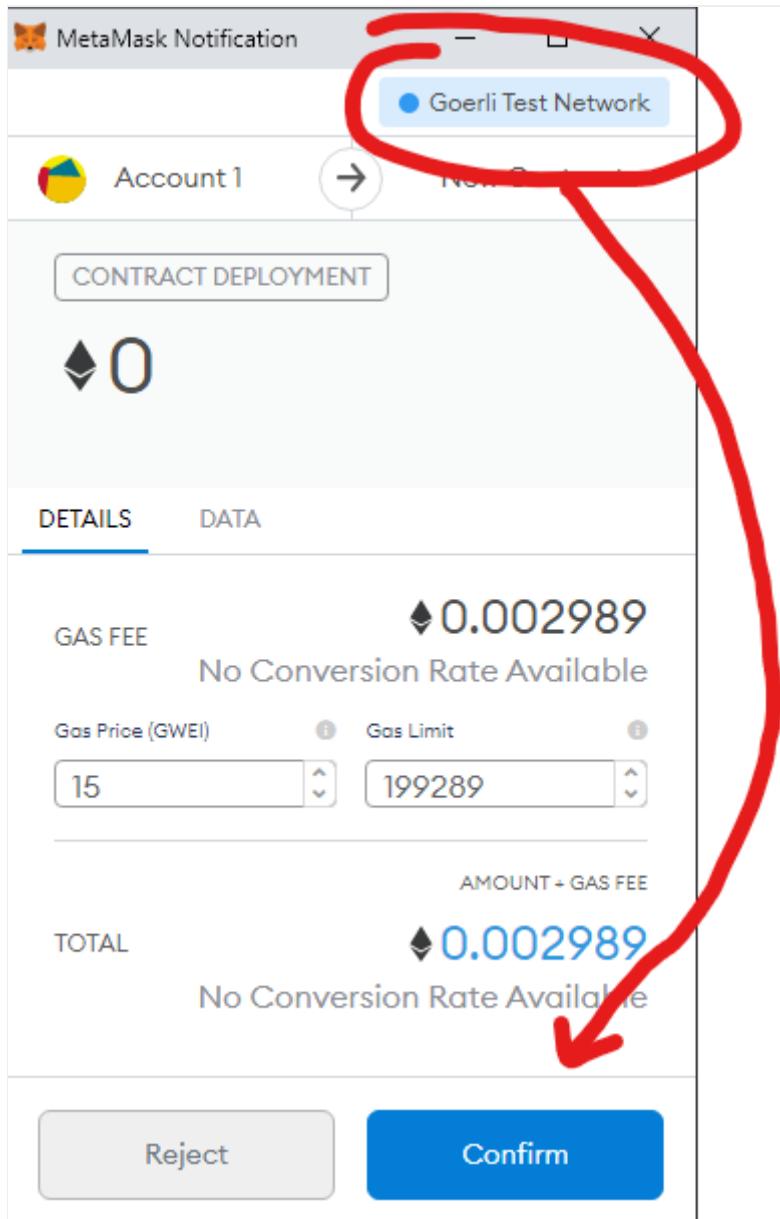
Let's deploy the Smart Contract now. First, make sure the correct Smart Contract is selected in the Dropdown:



Then simply hit "Deploy":



This should trigger MetaMask to ask you if you really want to send this transaction. Make sure the Görli Test-Network is selected and then simply hit "Confirm". If you selected the wrong network, then cancel the transaction, switch the network in MetaMask and hit "Deploy" again.



Perfect, now the transaction is on the way of getting mined. In the next section we will follow the transaction and interact with our smart contract!

Last update: September 6, 2021

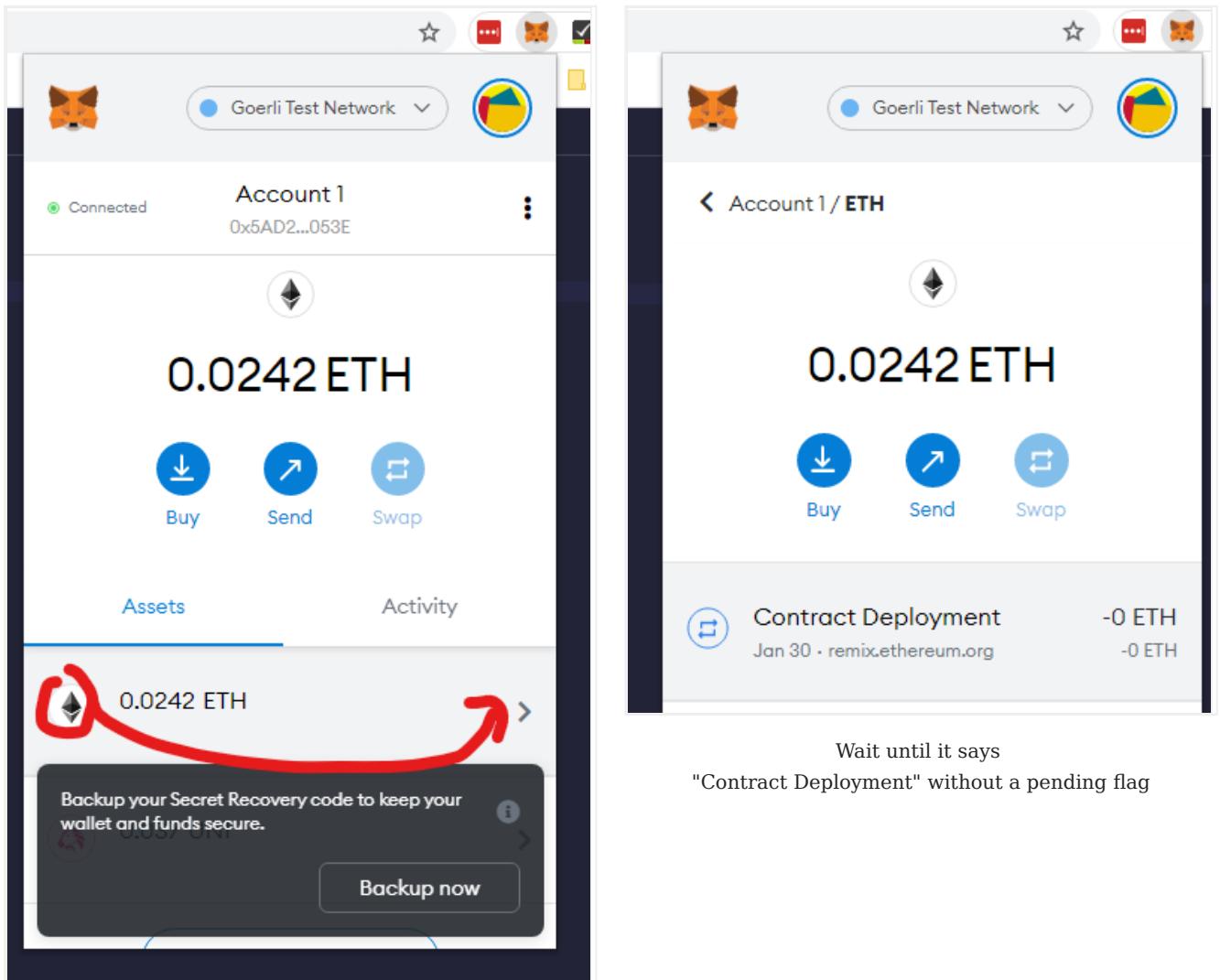
5.5 Interact with the Smart Contract

Now let's see if we can interact with the smart contract. Of course, at this point you have no idea what interaction actually means, so, let's just follow along.

5.5.1 Wait For Contract Readiness

First, we need to wait until the transaction is mined. We sent a transaction to the network, but before it's mined the contract won't be ready for any interaction. This can sometimes take a while, and sometimes it's really fast.

Wait until MetaMask says the Contract Deployment is complete. Open the MetaMask plugin in the top-right corner of Chrome, then check if the Smart Contract was already deployed:



Open MetaMask and go into the Ether Details

You will also see in Remix that the Contract is now ready:

ENVIRONMENT

Injected Web3
Goerli (5) network

ACCOUNT

0x5AD...3053E (0.0241686)

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

MyContract - browser/MyContract.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

MYCONTRACT AT 0X4BD...FA5BB (BLOCKCHAIN)

// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
 string public myString = 'hello world';
}

creation of MyContract pending...

<https://goerli.etherscan.io/tx/0xd7e479b675b966fa8e5>

[block:4197157 txIndex:0] from: 0x5AD...3053E

5.5.2 Interact With The Smart Contract

Now it's time to start our first interaction. In Remix we don't have to do any low-level things, it conveniently shows us buttons and input fields. You will later see how that works under the hood. We are covering it in the videos about the ABI Array.

Open the Dropdown on the left side:

The screenshot shows the Remix IDE interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and "MyContract.sol" on the right.

Left Panel (Deployment):

- ENVIRONMENT:** Set to "Injected Web3" (Goerli network).
- ACCOUNT:** Address 0x5AD...3053E with 0.0241686 ETH.
- GAS LIMIT:** Set to 3000000.
- VALUE:** Set to 0 wei.
- CONTRACT:** Selected "MyContract - browser/MyContracts.sol".
- Buttons:** "Deploy" (orange), "Publish to IPFS" (disabled).
- Transactions Recorded:** 1 transaction recorded.
- Deployed Contracts:** A list containing "MYC CONTRACT AT 0X4BD...FA5BB (BLOCKCHAIN)".

A red circle highlights the "Deployed Contracts" section.

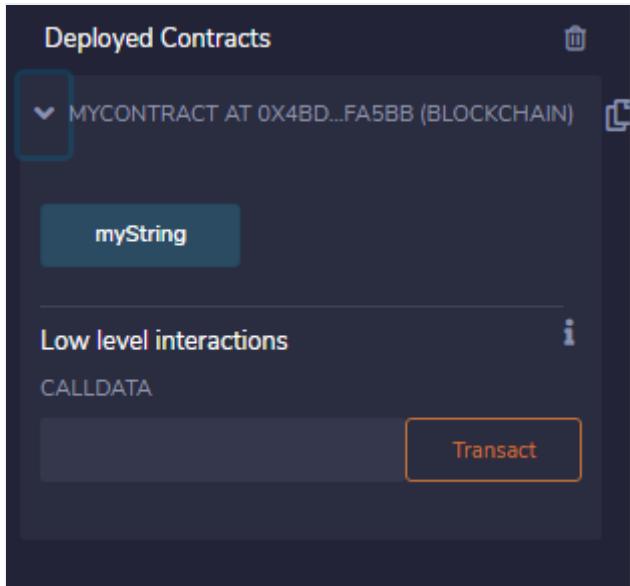
Right Panel (Code Editor):

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

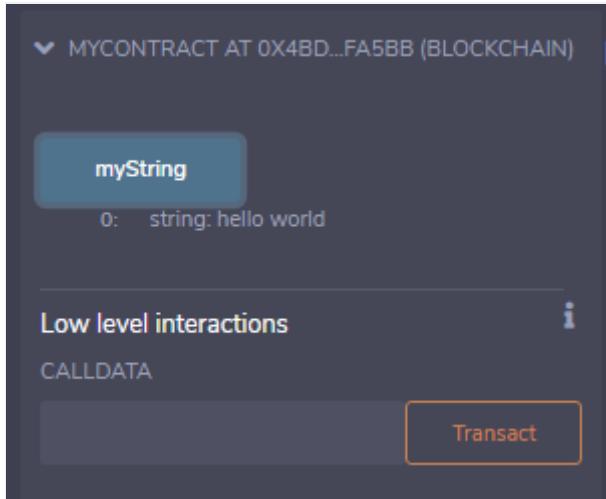
Bottom Panel (Logs):

- Logs: "creation of MyContract pending..." and a link to the transaction on Etherscan: <https://goerli.etherscan.io/tx/0xd7e479b675b966fa8e55>.
- Event log entry: "[block:4197157 txIndex:0] from: 0x5AD...3053E".

So that you can interact with the newly deployed Smart Contract:



Hit the "myString" Button and you will hopefully see that it returns "hello world" correctly.



This is it, your very first smart contract using Remix and the Görli Test-Network.

Last update: September 6, 2021

5.6 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

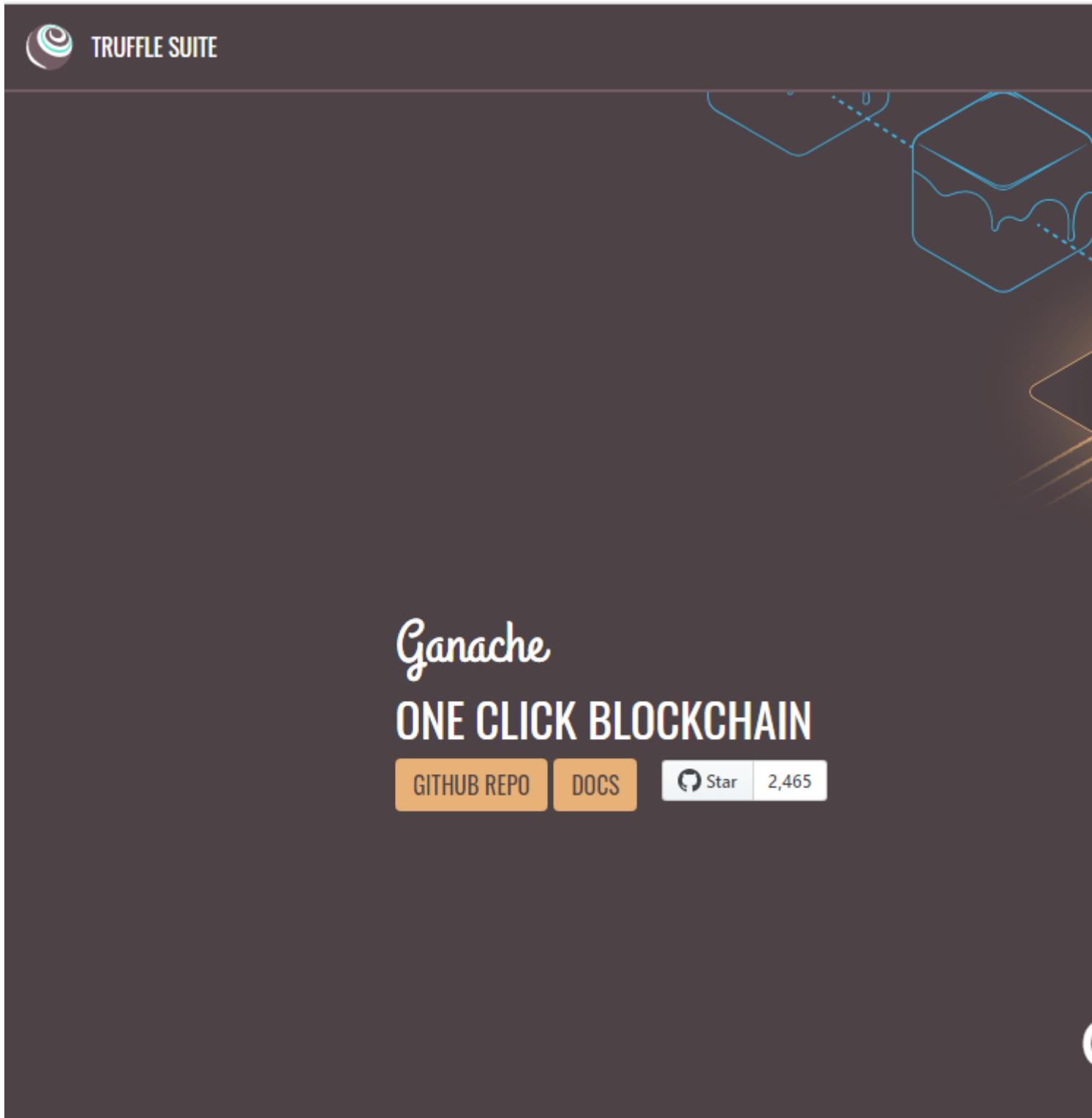
FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: September 6, 2021

6. Blockchain Networks

6.1 Lab: Use different Blockchain Networks

In this lab you are going to switch between different Blockchains. You are also setting up your own Development Blockchain.



It's a natural continuation from the previous Lab. We're going to deploy again to Görli, but we're also deploying to the JavaScript Virtual Machine. Then we're also directly connecting to a local Blockchain node, circumventing MetaMask and deploy our Smart Contract there in Ganache.

If all of that doesn't tell you anything - PERFECT! Follow along!

6.1.1 What You Know At The End Of The Lab

Understand Different Connection Methods

Download, Install and use Ganache

⚙️ Use JavaScript VM, Injected Web3 and Web3 Provider

Understand why Ganache is useful

6.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. About 15 Minutes of your precious time

6.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

6.1.4 Get Started

Let's get started [With A Smart Contract](#)

Last update: April 23, 2021

6.2 Smart Contract Example

We have to start with something, so we're going to use the exact same Smart Contract as we used in our previous Lab.

6.2.1 Simple Smart Contract

If you still have Remix open from the previous Lab, then you can keep re-using that smart contract, otherwise create a new file and paste the following content:

MyContract.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

Alright, let's deploy to Görli via MetaMask.

Try yourself!

Before you go to the next Lesson, try yourself to deploy to Görli via MetaMask.

Last update: April 17, 2021

6.3 Injected Web3

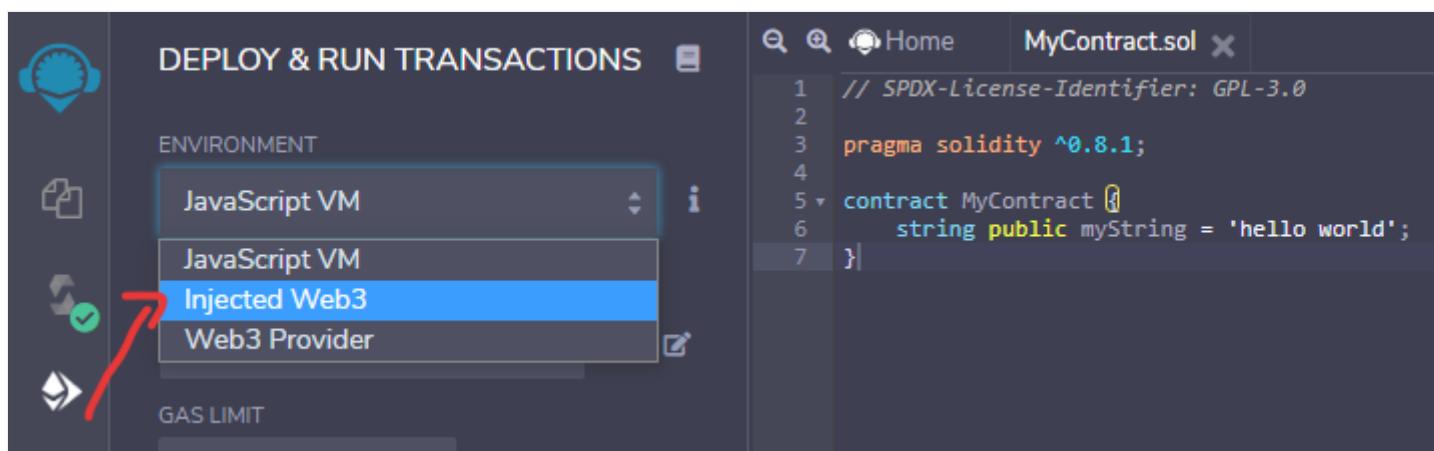
Tried yourself?

Did you try yourself before you opened this page? Did it work? Then directly try to see the difference when you deploy to the JavaScript VM and skip to the next page.

Alright, now we're going to deploy to Görli via MetaMask! This should look all too familiar from the previous Lab.

6.3.1 Connect MetaMask to Remix

Switch over to the "Deploy & Run Transactions" Plugin. We need to configure it, so it uses our MetaMask Wallet to access the Blockchain.



As soon as you do this, MetaMask should pop up and ask you to connect your account to Remix.

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area is titled "DEPLOY & RUN TRANSACTIONS". It includes sections for "ENVIRONMENT" (set to "Injected Web3" with "Goerli (5) network"), "ACCOUNT" (with a dropdown and copy/paste icons), "GAS LIMIT" (set to 3000000), "VALUE" (set to 0 wei), and "CONTRACT" (set to "MyContract - browser/MyContracts.sol"). A large orange "Deploy" button is prominent. Below it is a checkbox for "Publish to IPFS". The word "OR" is centered. Underneath, there are tabs for "At Address" (selected) and "Load contract from Address". A dropdown shows "Transactions recorded 0". A section for "Deployed Contracts" has a delete icon. At the bottom, a message box states: "Currently you have no contract instances to interact with."

ENVIRONMENT

Injected Web3

Goerli (5) network

ACCOUNT

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

MyContract - browser/MyContracts.sol

Deploy

Publish to IPFS

OR

At Address

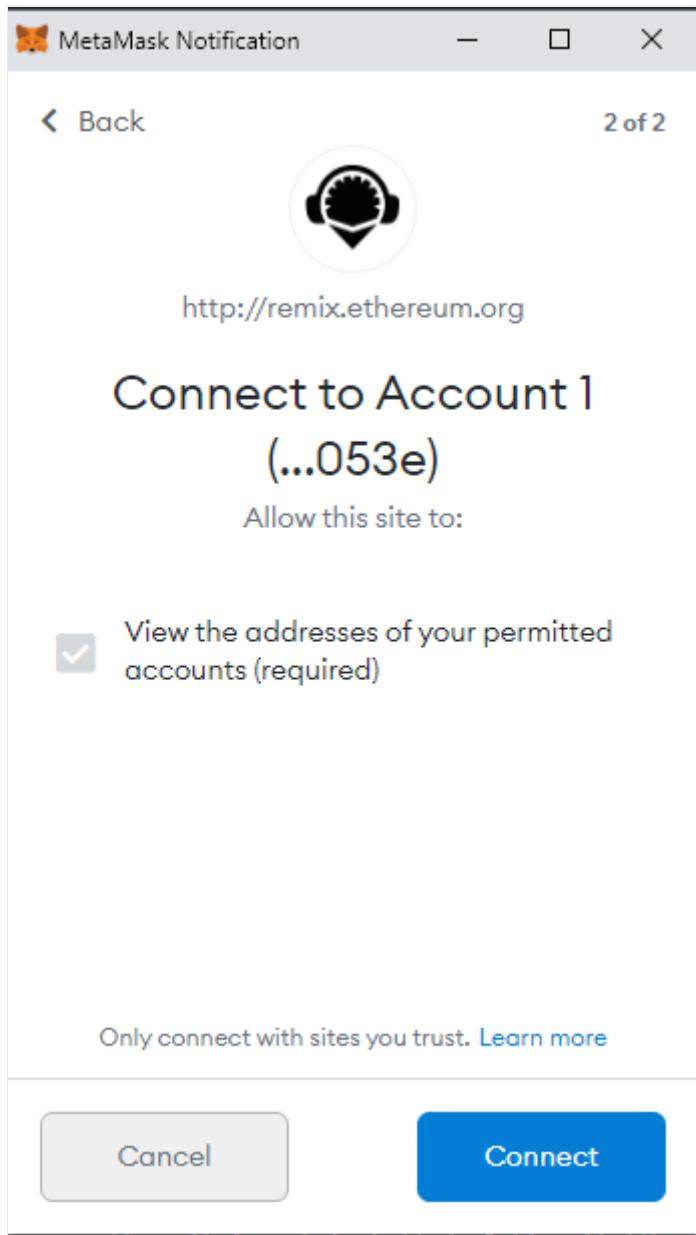
Load contract from Address

Transactions recorded 0

Deployed Contracts

Currently you have no contract instances to interact with.

Click "Next" and the "Connect":



Now your account should pop-up in the dropdown under the Environment Selection:

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with "DEPLOY & RUN TRANSACTIONS" and "ENVIRONMENT". The "ENVIRONMENT" dropdown is set to "Injected Web3" and shows "Goerli (5) network". Below it, the "ACCOUNT" dropdown shows "0x5AD...3053E (0.0271579...)". To the right, the code editor has a file named "MyContract.sol" with the following Solidity code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

Account not showing up?

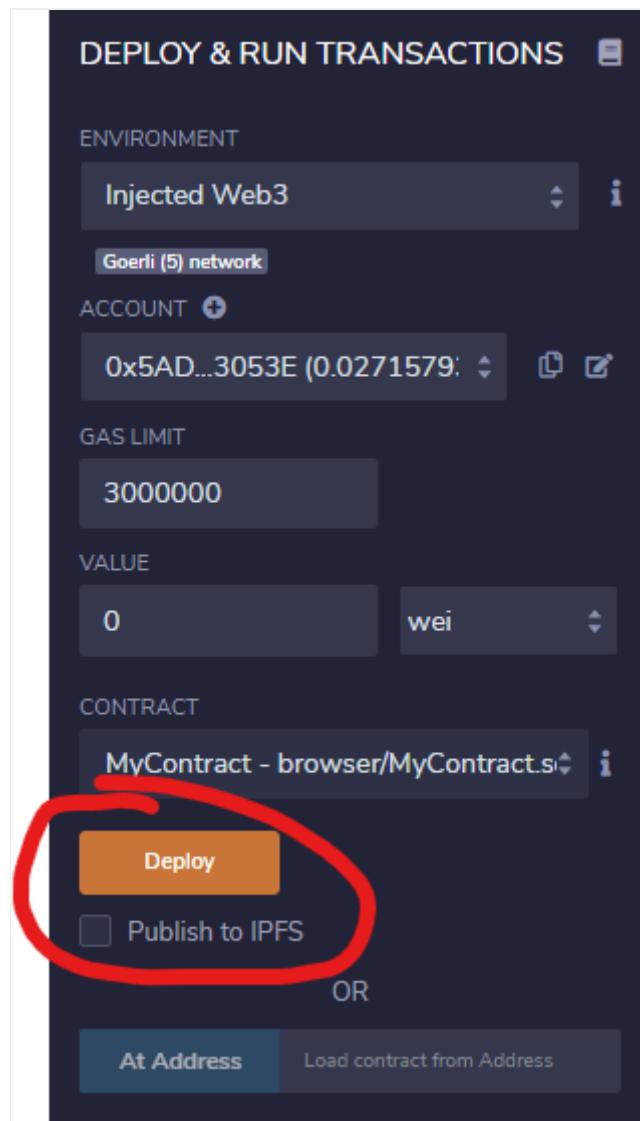
If your account doesn't show up, or MetaMask doesn't pop up, try to reload the page. There are sometimes caching issues.

6.3.2 Deploy the Smart Contract

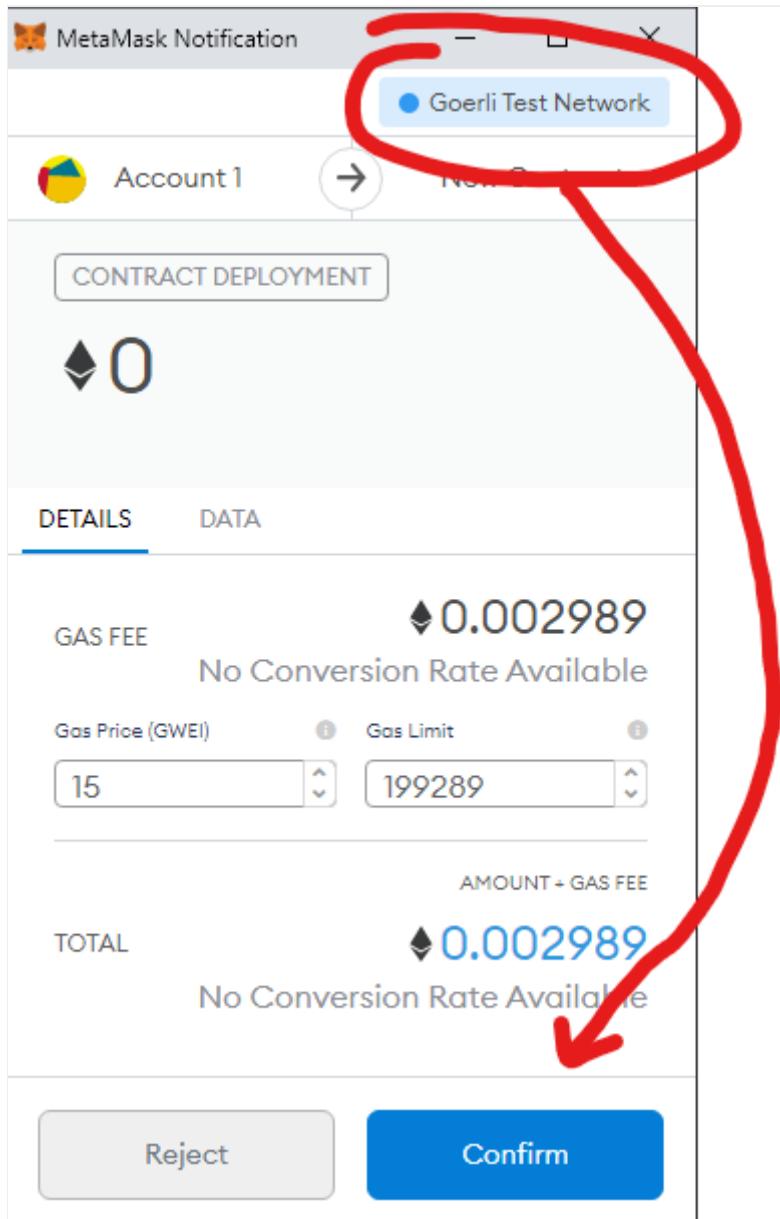
Let's deploy the Smart Contract now. First, make sure the correct Smart Contract is selected in the Dropdown:

The screenshot shows the "DEPLOY & RUN TRANSACTIONS" interface. The "ENVIRONMENT" section has "Injected Web3" selected in a dropdown, with "Goerli (5) network" shown below it. The "ACCOUNT" section shows an account address "0x5AD...3053E (0.0271579)". The "GAS LIMIT" is set to 3000000. The "VALUE" field is set to 0 wei. In the "CONTRACT" section, "MyContract - browser/MyContract.sol" is selected in a dropdown. Below it is an orange "Deploy" button. There is also a checkbox for "Publish to IPFS". At the bottom, there are two buttons: "At Address" and "Load contract from Address".

Then simply hit "Deploy":



This should trigger MetaMask to ask you if you really want to send this transaction. Make sure the Görli Test-Network is selected and then simply hit "Confirm". If you selected the wrong network, then cancel the transaction, switch the network in MetaMask and hit "Deploy" again.



Perfect, now the transaction is on the way of getting mined.

6.3.3 Time until Smart Contract is ready

See how long that takes? Mining on real Blockchains, even test-networks, can take a while. It's not very convenient for Development.

This is why there are alternatives out there, especially for Development!

Let's checkout the most basic one, the JavaScript VM next!

Last update: April 17, 2021

6.4 The JavaScript VM in Remix

The JavaScript VM is a simulated Blockchain Environment that only exists in your browser. It also only exists as long as you keep the browser-tab open. Close it or reload it, you start from scratch.

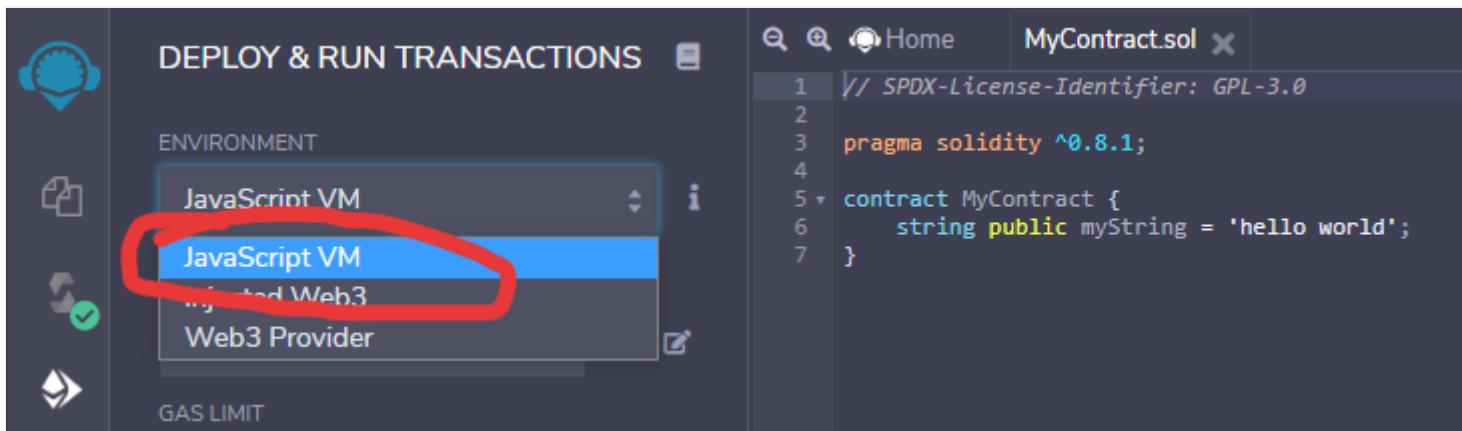
On the positive side: it's super fast! No waiting for Transactions to be mined. No complicated setup. It's just there and it works out of the box

On the negative side: There's only limited ways to connect to it. Once you reload everything is gone (non persistant). Sometimes things in the browser simulation work, which won't work on a real blockchain.

It's a great way to get started!

6.4.1 Deploy to JavaScript VM in Remix

Select the JavaScript VM from the Environment Dropdown in the "Deploy & Run Transactions" Plugin:



Then simply hit "deploy":

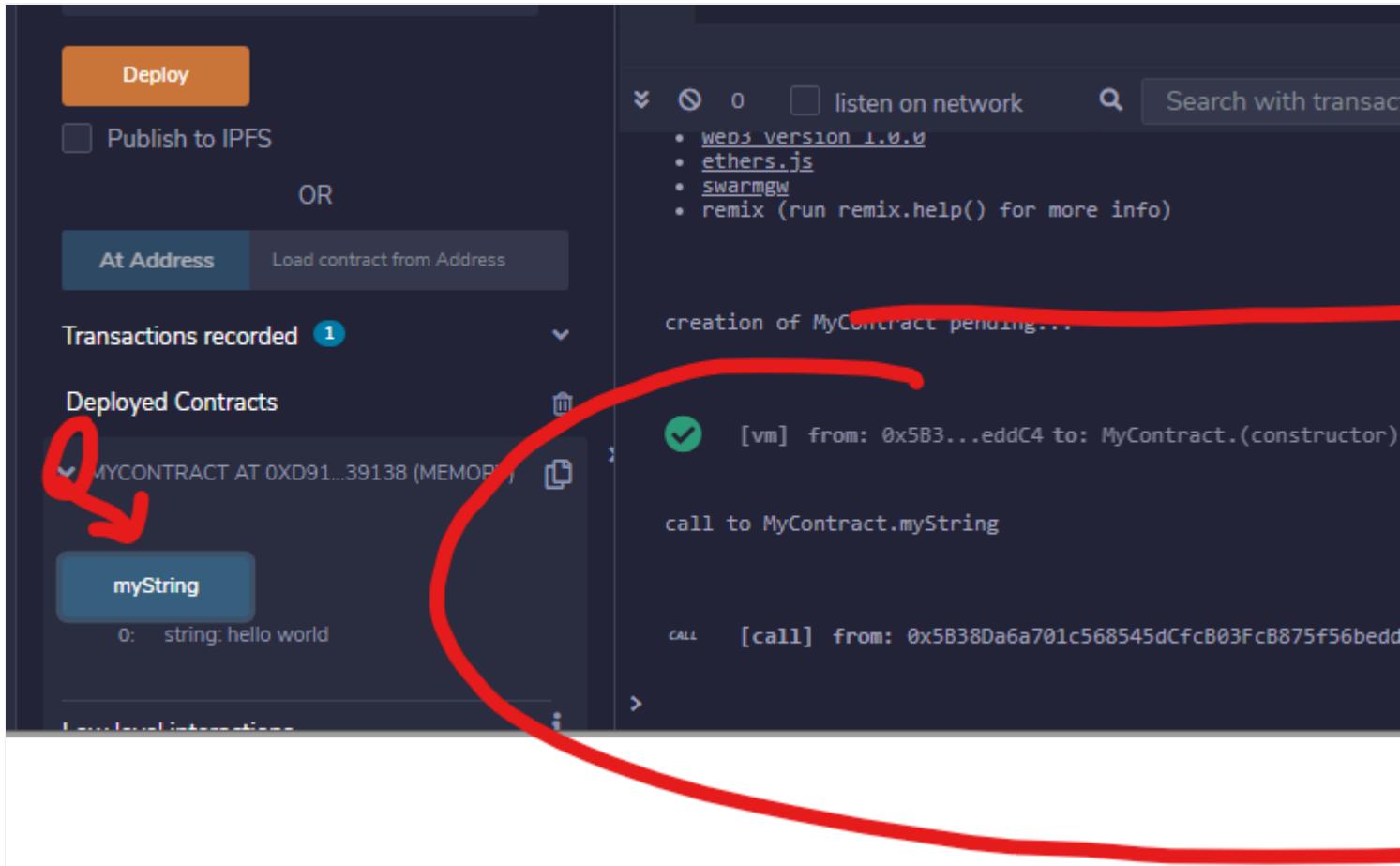
The screenshot shows the Ethereum Remix IDE interface. On the left, there's a sidebar with various icons for deploying, running, and interacting with contracts. The main area is titled "DEPLOY & RUN TRANSACTIONS". It includes sections for "ENVIRONMENT" (set to "JavaScript VM"), "ACCOUNT" (showing an account balance of "0x5B3...eddC4 (100 ether)"), "GAS LIMIT" (set to "3000000"), "VALUE" (set to "0 wei"), and "CONTRACT" (selected to "MyContract - browser/MyContract.sol"). A red oval highlights the "Deploy" button and the "Publish to IPFS" checkbox below it. To the right, the code editor shows the Solidity source code for "MyContract.sol":

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract MyContract {
    string public myString = 'hello world';
}
```

At the bottom, there are network status indicators and a search bar.

See how quickly that deployed? No MetaMask Popup. No wait time. It's just there. Bam!



But it's also not perfect, because there's no way to connect other tools to this blockchain. It's gone when you close the browser. All in all it's not perfect.

Let's checkout Ganache!

Last update: April 17, 2021

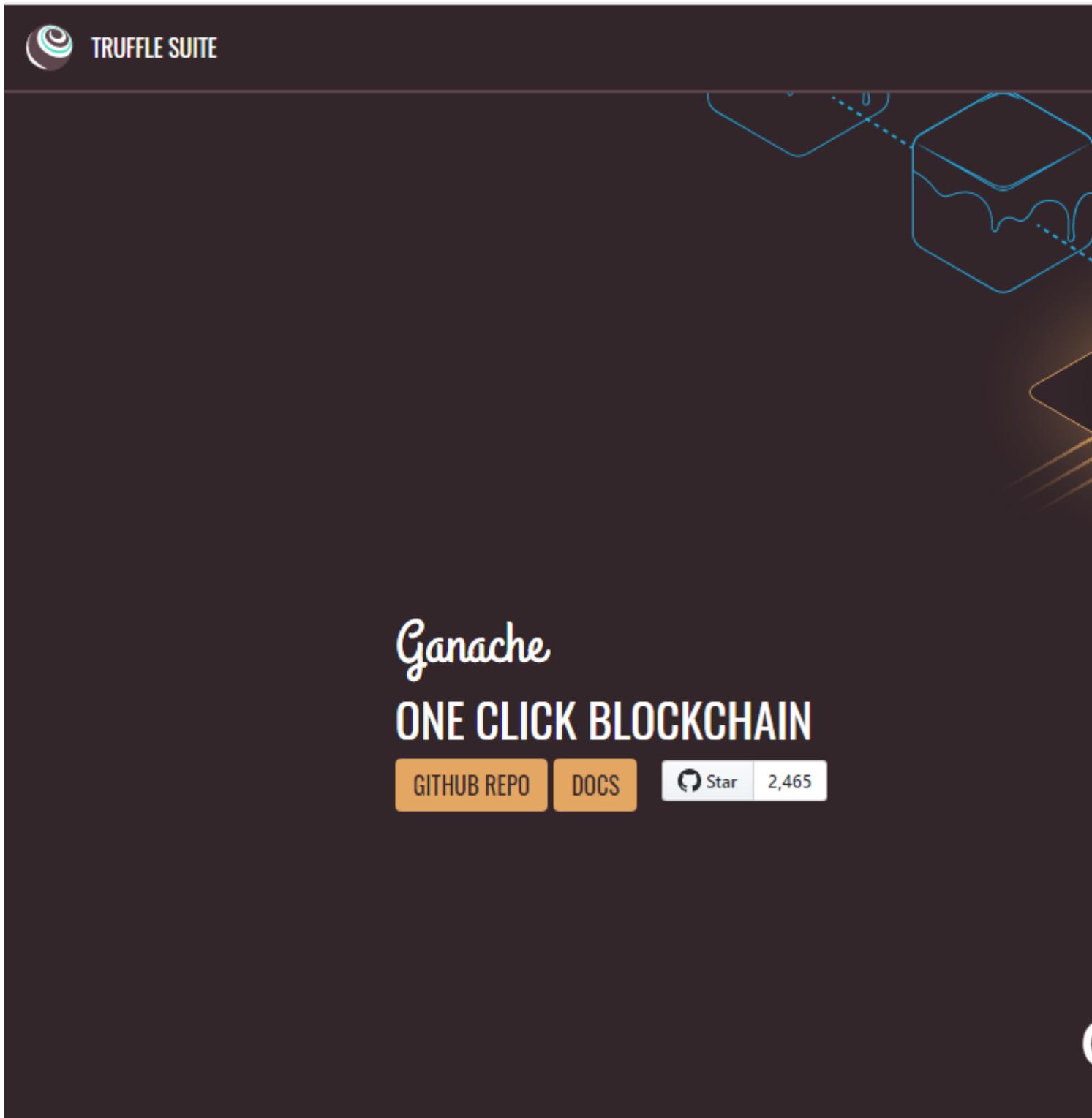
6.5 Web3 Provider

Connecting using a Web3 Provider actually establishes a connection to a software outside of the browser. Like you'd connect to a database. Or to any other API. It's either a RESTful interface or a WebSocket interface. And behind that interface sits a blockchain. This can be Go-Ethereum, Hyperledger Besu, Nethereum - or - Ganache for Development.

Let's start with Ganache!

6.5.1 Download and Install Ganache

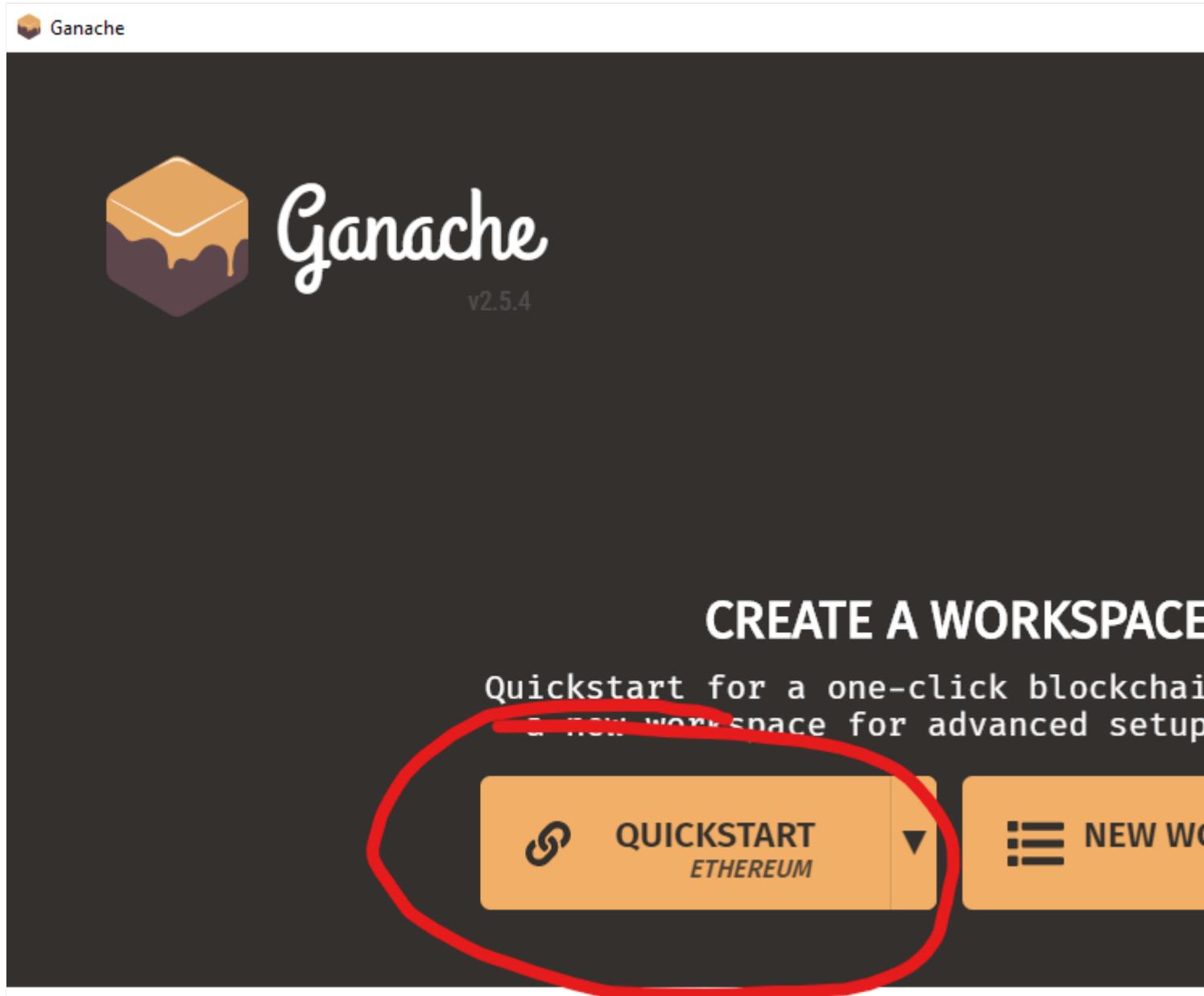
Go to <https://www.trufflesuite.com/ganache> and download Ganache for your Operating System. I am downloading it for Windows for this Lab. But there are also versions for MacOS and Linux. There's a UI Version and also a CLI (Command Line Interface) Version.



Run through the Setup Wizard and install it for your operating system.

6.5.2 Start Ganache

If you start ganache, it will ask you if you want to do a quickstart or actually start with a workspace. For now it's safe to say: We do a quickstart.

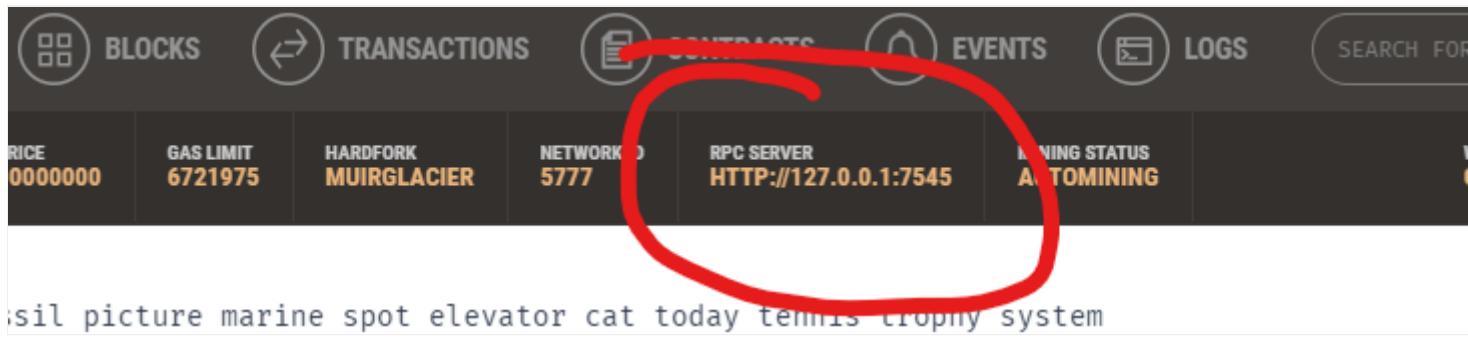


This will create 10 accounts and assign each 100 ether. The accounts are unlocked. All is ready!

Ganache						
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS	
CURRENT BLOCK 0	GAS PRICE 200000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING
MNEMONIC <small>?</small> offer swamp fossil picture marine spot elevator cat today tennis trophy system						
ADDRESS 0x7d0C4F167498781255A9b501802606a408ECC15				BALANCE 100.00 ETH		
ADDRESS 0xF3d4319e2866583bAb3eFc87B7882B72CF283B2c				BALANCE 100.00 ETH		
ADDRESS 0x12b00535C00F96A75bE99D61B5a4EB3BDc5d0219				BALANCE 100.00 ETH		
ADDRESS 0x4288d1c8fc2db1eBd9BEEA01cAEdA86913C9f858				BALANCE 100.00 ETH		
ADDRESS 0xC727AF1042DcEFd9411dE0FaF879fD551Ad41C33				BALANCE 100.00 ETH		
ADDRESS 0xd9Ab77AFe3854257690f288242f3c730258ac1a0				BALANCE 100.00 ETH		
ADDRESS 0xF259d60CB2192d86780c1392159f37709370A93F				BALANCE 100.00 ETH		

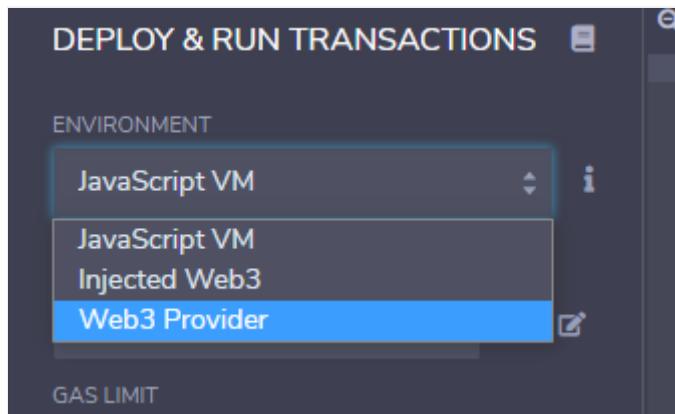
Ganache is now a Blockchain and a Wallet. Two in one. Anyone can connect to it using a Web3 Provider Method either via http RPC or WebSockets.

Pay attention to the RPC Endpoint:

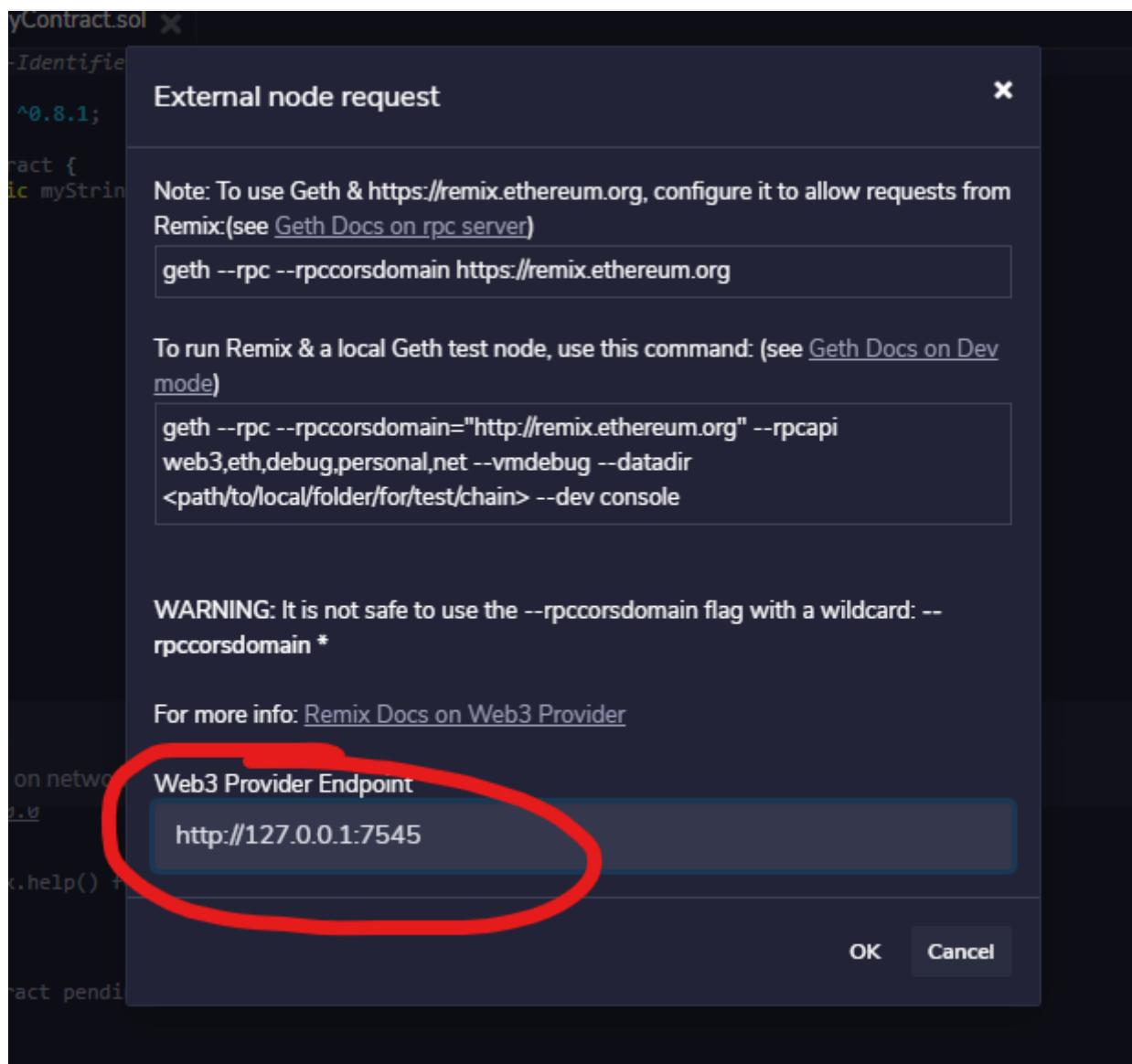


6.5.3 Connect Remix

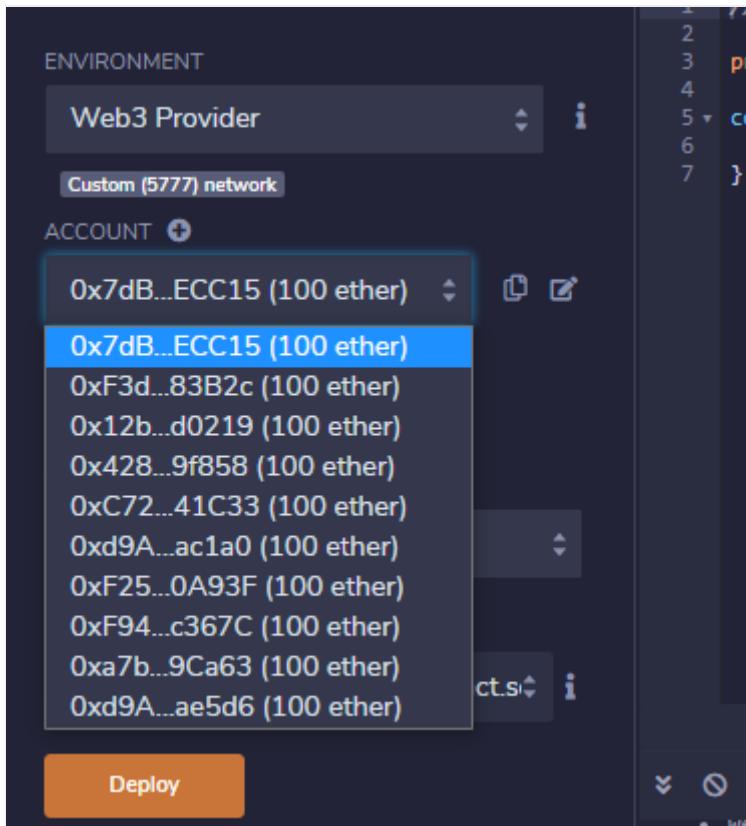
From the Environment Dropdown select now "Web3 Provider":



A new Popup will appear, and enter the RPC Server URL from Ganache. Pay attention to the Port number:



Your Accounts from Ganache should popup in the Accounts-Dropdown:



6.5.4 Deploy to Ganache

Now let's see what happens if we hit the Deploy Button:

CONTRACT
MyContract - browser/MyContract.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

MYCONTRACT AT 0X3B5...B2BAB (BLOCKCHAIN)

myString

Low level interactions

CALldata

Transact

0 listen on network

Search with transaction

[vm] from: 0x5B3...eddC4 to: MyContract.(constructor)
call to MyContract.myString

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56

creation of MyContract pending...

[block:1 txIndex:0] from: 0x7dB...ECC15 to: MyContract

Have you seen how fast that was?

Now you have the benefit of two things: An actual blockchain node, but fast like a development network.

Going forward it's probably best to use either the JavaScript VM or Web3 Provider with Ganache. The choice is yours, whatever you prefer. For ease of use, I'll use the JavaScript VM throughout the rest of the Solidity explanations.

Last update: April 17, 2021

6.6 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

7. Simple Variables

7.1 Lab: Solidity Basics

In this lab we're going to try different types of variables. You will get familiar with the very basic building blocks of a Smart Contract!

7.1.1 What You Know At The End Of The Lab

Familiarize yourself with Types of Variables

Get Insights into Solidity quirks and specials

Be able to bring your own ideas to life!

7.1.2 Prerequisites - You need:

1. Know how to use Remix

Solidity 0.8 Updates

The content has been updated for Solidity 0.8. If there are functional differences to the solidity files shown in the videos then they're highlighted.

In this guide a new smart contract is started every time we are doing a new type of variable. If you prefer to merge them together into one single file, feel free to do so.

7.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

7.1.4 Get Started

Let's get started

Last update: April 23, 2021

7.2 Understanding (Unsigned) Integers

In case you don't know what exactly integers are, then let's define first what we're talking about:

An integer is [...] a number that can be written without a fractional component. For example, 21, 4, 0, and -2048 are integers, while 9.75, ... is not. <https://en.wikipedia.org/wiki/Integer>

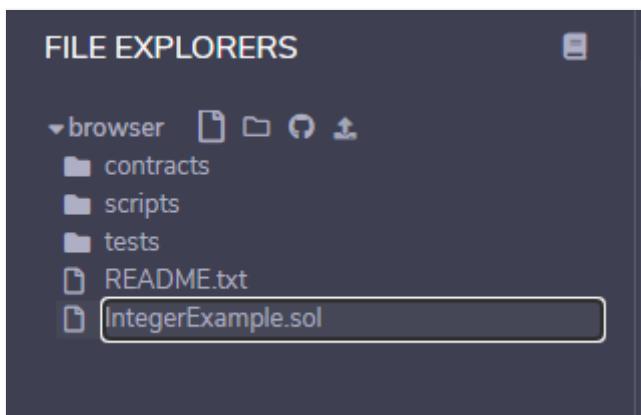
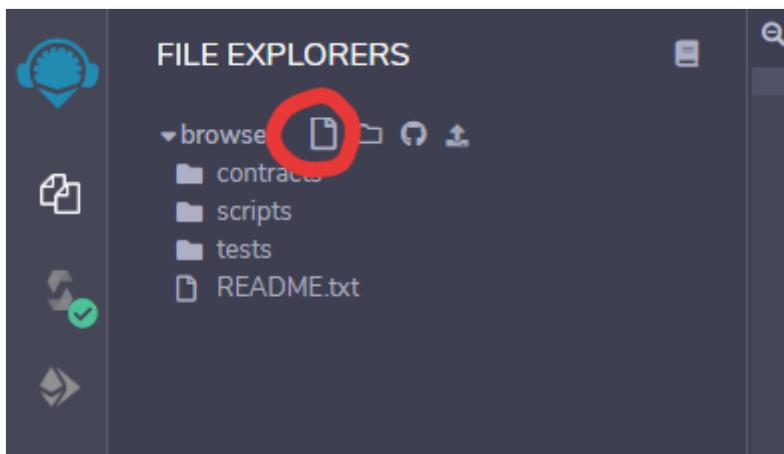
Integral types may be unsigned (capable of representing only non-negative integers) or signed (capable of representing negative integers as well) [https://en.wikipedia.org/wiki/Integer_\(computer_science\)](https://en.wikipedia.org/wiki/Integer_(computer_science))

In layman's terms: Integers are numbers without decimals. Unsigned integers cannot be negative.

Let's define a simple Smart Contract first so we can set an integer and then read the value again.

7.2.1 Smart Contract

Create a new file and name it IntegerExample.sol and fill in the following content:



IntegerExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract IntegerExample {
    uint public myUint;
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has two tabs: "FILE EXPLORERS" and "IntegerExample.sol". The "FILE EXPLORERS" tab shows a tree view of files under "browser": "contracts", "scripts", "tests", and "artifacts". Inside "artifacts", there are three JSON files: "IntegerExample_metadata.json", "IntegerExample.json", and "README.txt". The "IntegerExample.sol" file is also listed and is currently selected, highlighted with a blue bar at the bottom. The "IntegerExample.sol" tab on the right displays the Solidity code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract IntegerExample {
    uint public myUint;
}
```

There are two important things to see here:

1. The variable is not initialized, but as we will see in a moment, still has a default value
2. A public variable automatically creates a getter function in solidity.

7.2.2 Deploy the Smart Contract

Open the "Deploy & Run Transactions" Plugin. Make sure the correct Smart Contract is selected from the dropdown. Then simply hit "Deploy":

You will be able to observe a few things:

1. A new transaction will be sent and you can see that in the console of Remix (bottom right).
 2. Your Smart Contract is available in the "Deploy & Run Transactions" Plugin, at the bottom. You might need to uncollapse it.

The screenshot shows the Remix IDE interface. On the left, the "DEPLOY & RUN TRANSACTIONS" sidebar is open, displaying settings for the environment (JavaScript VM), account (0x5B3...eddC4), gas limit (3000000), value (0 wei), and the contract (IntegerExample). A large orange "Deploy" button is prominent. Below it, there's a checkbox for "Publish to IPFS". The "Deployed Contracts" section shows a single entry: "INTEGEREXAMPLE AT 0xD8B...33FA8 ()". A green button labeled "myUint" is visible next to this entry. A red circle with the number "2" is placed over this section. On the right, the code editor shows the Solidity source file "IntegerExample.sol" with the following code:

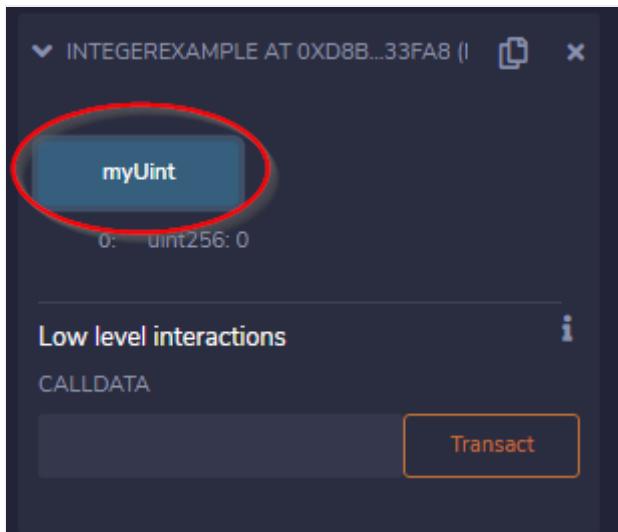
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract IntegerExample {
    uint public myUint;
}
```

The terminal window at the bottom shows the deployment process. It starts with "Contract deployed at address: 0xd9145CCE52D386f254917", followed by "creation of IntegerExample pending...", and then a transaction log: "[vm] from: 0x5B3...eddC4 to: IntegerExample.(0xf53...38e52)". A red circle with the number "1" is placed over the terminal window.

You can now interact with your smart contract.

7.2.3 Interact with the Smart Contract

To interact with your Smart Contract, Remix is automatically generating buttons for every function. If you click on "myUint", you call the auto-generated getter function from the public variable called "myUint".



Standard Workflow

This is a standard workflow, change the smart contract, redeploy. Currently, there are no in-place updates. For every change we do, we have to deploy a new version of the Smart Contract.

7.2.4 Write a Setter Function

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract IntegerExample {
    uint public myUint;
    function setMyUint(uint _myUint) public {
        myUint = _myUint;
    }
}
```

Deploy a new version of the Smart Contract - simply hit "Deploy" again. You will see that you have two instances of your Smart Contract on the bottom of the "Deploy & Run Transactions" Plugin. You can safely close the old version - the new version is on the bottom.

DEPLOY & RUN TRANSACTIONS

VALUE
0 wei

CONTRACT
IntegerExample - browser/IntegerExam

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 2

Deployed Contracts

▼ INTEGEREXAMPLE AT 0XD8B...33FA8 (I)

myUint

0: uint256: 0

Low level interactions

CALLDATA

Tran

► INTEGEREXAMPLE AT 0XF8E...9FBE8 (M)

[vm] from: 0x5B3...eddC4 to: IntegerExample.(c hash: 0xa1d...06959)

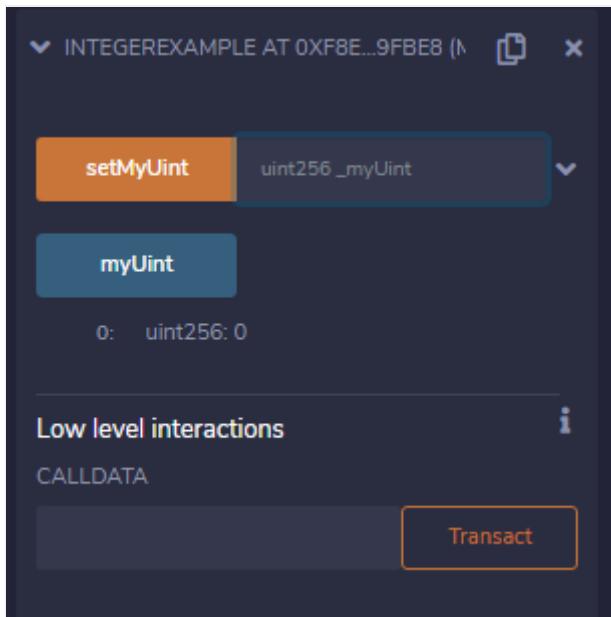
Old Instance

New Instance

The screenshot shows the Truffle UI interface with two open contracts:

- INTEGEREXAMPLE AT 0xD8B...33FA8 ()**:
 - myUint**: uint256: 0
 - Low level interactions**
 - CALldata**
 - Transact** button
- setMyUint**:
 - uint256 _myUint**
 - myUint**: uint256: 0
 - Low level interactions**
 - CALldata**
 - Transact** button

If you click on "myUint" it will be initially 0 again.



Let's update it to 5. You can enter the number "5" into the input field next to "setMyUint", then click on "setMyUint":



If you do so, you can again observe the console of Remix on the bottom right that it sent a transaction.

DEPLOY & RUN TRANSACTIONS

gas limit: 3000000

value: 0 wei

contract: IntegerExample - browser/IntegerExam...

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded (3)

Deployed Contracts

▼ INTEGEREXAMPLE AT 0XF8E...9FBE8 (M)

- setMyUint 5
- myUint

0: uint256: 0

Low level interactions

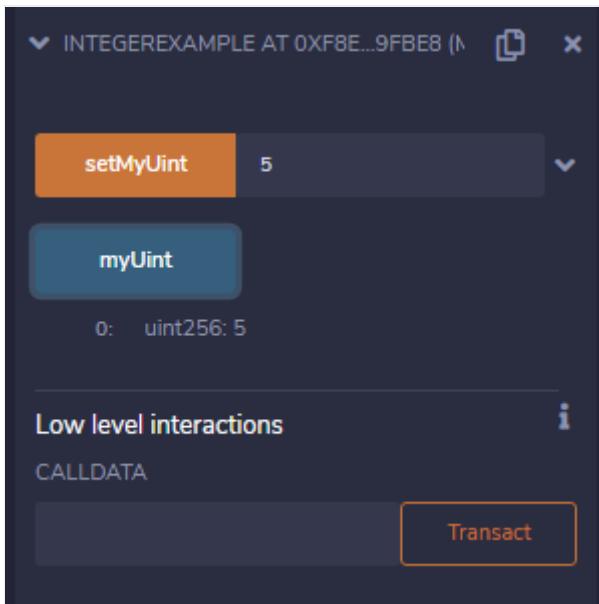
CALldata

Transact

I am a transaction

[vm] from: 0x5B3...eddC4 to: IntegerExample.set data: 0xe49...00005 logs: 0 hash: 0xc03...933de

When you click on "myUint" now, it should show you "5" instead of "0".



When you click on a function that only reads a value, then no transaction is sent to the network, but a call. You can see this in the console on the right side again.

```

setMyUint      5
myUint
0: uint256: 5

Low level interactions
CALldata
Transact

[tx] 0x49...00005 logs: 0 has
call to IntegerExample.myUint
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
data: 0x065...40f7e
  
```

I'm a "call"

Transaction vs Call

A transaction is necessary, if a value in a Smart Contract needs to be updated (writing to state). A call is done, if a value is read. Transactions cost Ether (gas), need to be mined and therefore take a while until the value is reflected, which you will see later. Calls are virtually free and instant.

Great! Now you know the basic workflow, how to deploy and how to update your code during development. But working only with Integers is a bit boring. Let's level up a bit and learn some more types before doing our first project.

Please note: In the next sections I will silently assume you are deploying a version of a Smart Contract and delete the old instance, whenever we do some changes. I recommend repeating this a few times so you don't forget it.

Last update: April 17, 2021

7.3 Boolean Types in Solidity

Boolean Types can be true or false. Let's define a simple Smart Contract to play around:

7.3.1 Boolean Smart Contract Example

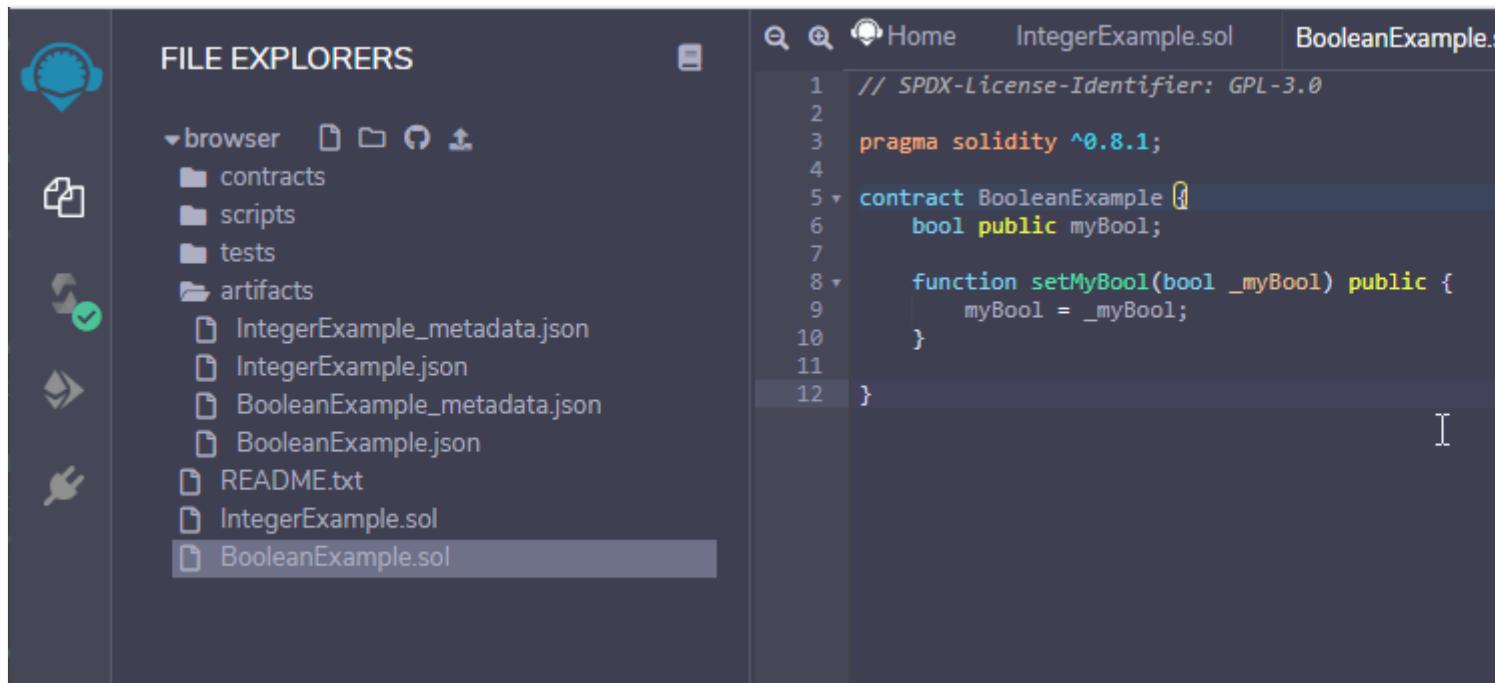
Create a new Smart Contract in Remix, name it BooleanExample.sol and add the following content:

BooleanExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract BooleanExample {
    bool public myBool;

    function setMyBool(bool _myBool) public {
        myBool = _myBool;
    }
}
```



7.3.2 Deploy the Smart Contract

Head over to "Run & Deploy Transactions" Plugin and deploy this Smart Contract.

When you click on "myBool" you will see that the variable is initialized with its default value, which is "false".

The screenshot shows the Truffle UI interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and a code editor on the right.

Left Section (Deployment):

- Gas:** 5000000
- Value:** 0 wei
- Contract:** BooleanExample - browser/BooleanExample.sol
- Buttons:** Deploy, Publish to IPFS
- OR**
- At Address:** Selected
- Transactions recorded:** 4
- Deployed Contracts:** BOOLEANEXAMPLE AT 0X7EF...8CB47
 - setMyBool:** bool _myBool
 - myBool:** 1 (highlighted with a red circle)
 - 0: bool: false
- Low level interactions:** CALLDATA, Transact

Right Section (Code Editor):

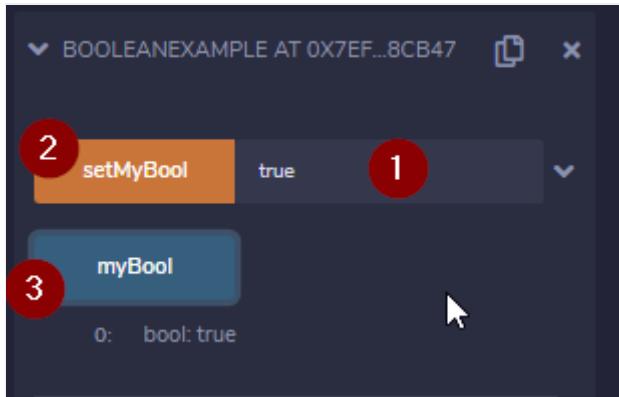
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract BooleanExample {
    bool public myBool;
    function setMyBool(bool _myBool) public {
        myBool = _myBool;
    }
}
```

Bottom Panel:

- 0: listen on network
- hash: 0xed2...9aace
- call to BooleanExample.myBool
- CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875...
- data: 0x6de...ecdbe

Now let's update the variable to "true":

1. enter "true" into the input field
2. hit the "setMyBool" button
3. see if the variable was updated by clicking on "myBool" again



Boolean Operators

With boolean types you have the standard operators at your disposal. Operators: `!` (logical negation), `&&` (logical conjunction, "and"), `||` (logical disjunction, "or"), `==` (equality), `!=` (inequality).

We will use this in the upcoming projects.

Now you know two very basic types, and we could do already a lot with it, but before we head into our first project, let's see some more peculiarities which you only have in Solidity.

Last update: April 17, 2021

7.4 Integer Overflow and Underflow

7.4.1 What are Overflows or Underflows?

In previous versions of Solidity (prior Solidity 0.8.x) an integer would automatically roll-over to a lower or higher number. If you would decrement 0 by 1 (0-1) on an unsigned integer, the result would not be -1, or an error, the result would simple be: MAX(uint).

For this example I want to use uint8. We haven't used different types of uint yet. In our previous example worked with uint256, but bear with me for a moment. Uint8 is going from 0 to $2^8 - 1$. In other words: uint8 ranges from 0 to 255. If you increment 255 it will automatically be 0, if you decrement 0, it will become 255. No warnings, no errors. For example, this can become problematic, if you store a token-balance in a variable and decrement it without checking.

Let's do an actual example!

7.4.2 Sample Smart Contract

Create a new Smart Contract with the name "RolloverExample.sol". We're going to use Solidity 0.7 for this example, as in Solidity 0.8 this behavior is different.

RolloverExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.7.0;

contract RolloverExample {
    uint8 public myUint8;

    function decrement() public {
        myUint8--;
    }

    function increment() public {
        myUint8++;
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has two tabs: "FILE EXPLORERS" and "CONTRACTS". Under "FILE EXPLORERS", there's a tree view of files and folders. Under "contracts", there are sub-folders for "scripts", "tests", and "artifacts". The "artifacts" folder contains several JSON files: IntegerExample_metadata.json, IntegerExample.json, BooleanExample_metadata.json, BooleanExample.json, RolloverExample_metadata.json, and RolloverExample.json. Below these are README.txt, IntegerExample.sol, BooleanExample.sol, and RolloverExample.sol. The RolloverExample.sol file is currently selected, highlighted with a blue bar at the bottom.

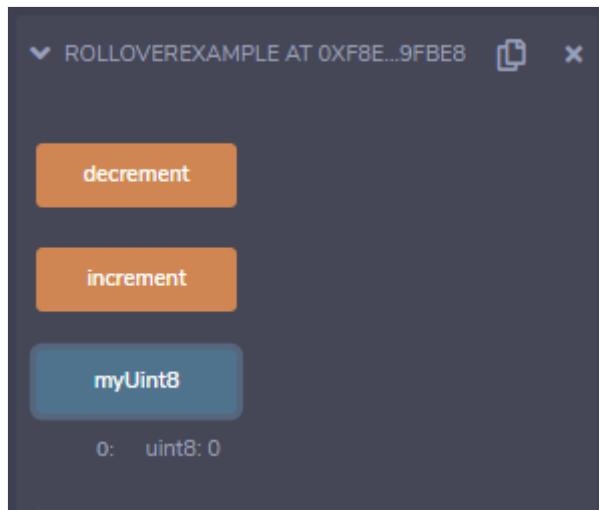
```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity 0.7.0;
3
4 contract RolloverExample {
5     uint8 public myUint8;
6
7     function decrement() public {
8         myUint8--;
9     }
10
11    function increment() public {
12        myUint8++;
13    }
14 }

```

Let's deploy the Smart Contract and see what happens if we call decrement.

Initially, myUint8 is 0:



If you press the "decrement" button, then the myUint8 is decremented by one. Let's see what happens, and also observe the Remix console:

The screenshot shows the VomTom IDE interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and a code editor on the right.

Left Panel (Deploy & Run Transactions):

- Top Bar:** Includes icons for headphones, a checkmark, and a gear, followed by "DEPLOY & RUN TRANSACTIONS" and a refresh button.
- Buttons:** "Publish to IPFS" and "OR".
- Address Selection:** "At Address" (selected) and "Load contract from Address".
- Transactions Recorded:** Shows "Transactions recorded 4".
- Deployed Contracts:** Shows "ROLLOVEREXAMPLE AT 0XF8E...9FBE8" with a copy icon and delete icon.
- Contract Interaction Buttons:** "decrement" (orange), "increment" (orange), and "myUint8" (blue).
- Low Level Interactions:** Shows "CALLDATA" and a "Transact" button.
- Bottom Left Icon:** A gear icon.

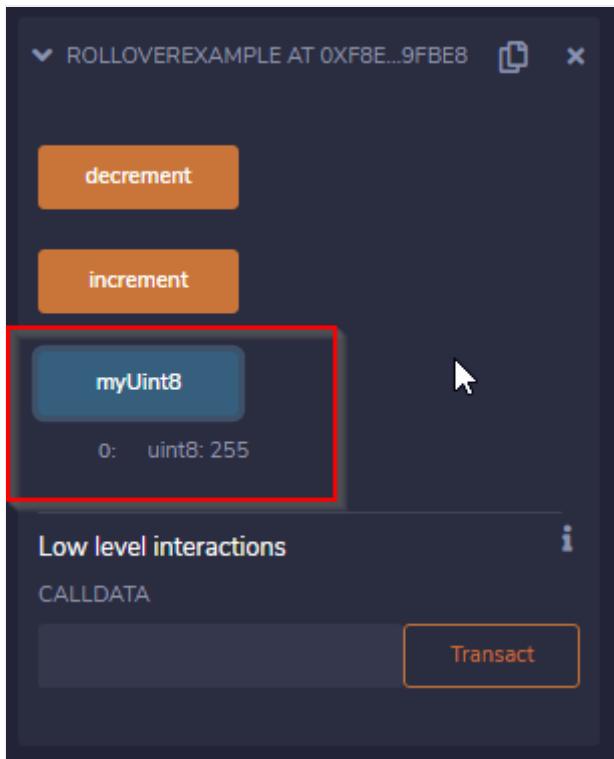
Right Panel (Code Editor):

- Header:** Home, MorpherOracle.sol, RolloverExample.sol
- Code:**

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity 0.7.0;
3
4 contract RolloverExample {
5     uint8 public myUint8;
6
7     function decrement() public {
8         myUint8--;
9     }
10
11    function increment() public {
12        myUint8++;
13    }
14 }
```
- Logs:**
 - CALL: data: 0xe2a...4b2af
 - transact to RolloverExample.decrement pending ...
 - [vm] from: 0x5B3...eddC4 to: RolloverExample.decrement pending ...
 - data: 0x2ba...eceb7 logs: 0 hash: 0x4ed...c2e8f
- Bottom Right:** Search bar with placeholder "Search with..."

The transaction works perfectly. No errors occur. If you retrieve the value for myUint8 then you see it's 255:



Increment Example

Try yourself what happens when you increment again. Does it roll over again without a warning?

Now you see one of the quirks with Solidity. It's not completely unique to Solidity, but definitely something to be aware of. This is where Libraries like [SafeMath](#) were invented, which you will see later.

But what happened in Solidity 0.8?

7.4.3 Solidity 0.8 Difference

In Solidity 0.8, the compiler will automatically take care of checking for overflows and underflows. Let's run the same example with Solidity 0.8. Create a new file and fill in the following Smart Contract:

RolloverExampleSol08.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.0;

contract RolloverExample2 {
    uint8 public myUint8;

    function decrement() public {
        myUint8--;
    }

    function increment() public {
        myUint8++;
    }
}
```

Deploy the new version and try to hit "decrement". Now you will get an error in the Remix console:

The screenshot shows the VomTom IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Publish. The main panel is titled "DEPLOY & RUN TRANSACTIONS". It has a "Deploy" button, a checkbox for "Publish to IPFS", and options to "At Address" or "Load contract from Address". Below this, it shows "Transactions recorded" (6) and "Deployed Contracts" for "ROLLOVEREXAMPLE AT 0X7EF...8CB47". Under the deployed contract, there are buttons for "decrement" (with a red circle containing '1'), "increment", and "myUint8". A "Transact" button is also present. The right side of the interface is a code editor showing the Solidity code for the RolloverExample contract:

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity 0.8.0;
3
4 contract RolloverExample {
5     uint8 public myUint8;
6
7     function decrement() public {
8         myUint8--;
9     }
10
11    function increment() public {
12        myUint8++;
13    }
14 }

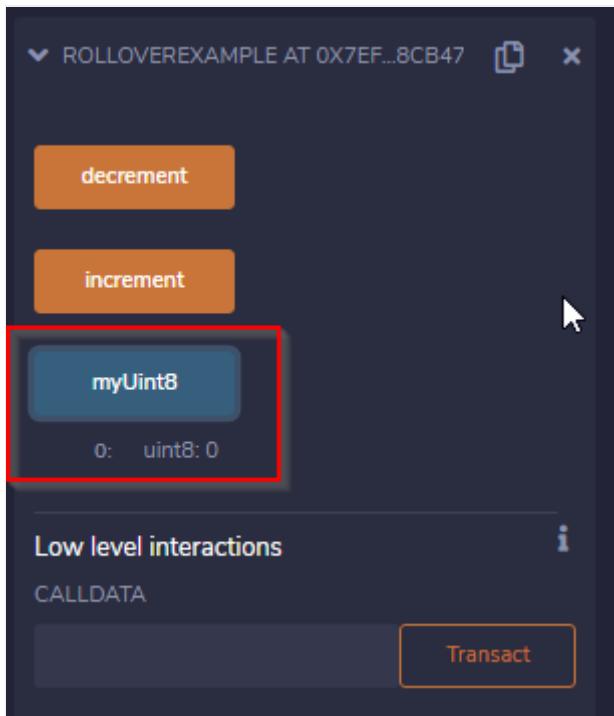
```

Below the code editor, there are transaction details: "0" transactions, a checkbox for "listen on network", and a search bar. A red circle with '2' is placed over the transaction log output, which shows a failed transaction due to a VM error.

Transaction Log:

[vm] from: 0x5B3...eddC4 to: RolloverExample.d...
 data: 0x2ba...eceb7 logs: 0 hash: 0x466...d0eeef
 transact to RolloverExample.decrement errored: VM error state. Note: The called function should be payable if current balance. Debug the transaction to get more info

Your variable "myUint8" will remain 0, because it cannot roll over anymore:



But what if you actually want to roll over? Then there is a new "unchecked" block you can wrap your variables around:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.0;

contract RolloverExample2 {
    uint8 public myUint8;

    function decrement() public {
        unchecked {
            myUint8--;
        }
    }

    function increment() public {
        unchecked {
            myUint8++;
        }
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, publishing to IPFS, and managing contracts. The main area is titled "DEPLOY & RUN TRANSACTIONS". It has a "Deploy" button, a checkbox for "Publish to IPFS", and a section for "Transactions recorded" and "Deployed Contracts". A dropdown menu shows "ROLLOVEREXAMPLE AT 0X358...D5EE3". Below this, there are two buttons: "decrement" and "increment". Underneath them is a field labeled "myUint8" with the value "0: uint8: 255". There's also a section for "Low level interactions" with a "Transact" button.

MorpherOracle.sol

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity 0.8.0;
3
4 contract RolloverExample {
5     uint8 public myUint8;
6
7     function decrement() public {
8         unchecked {
9             myUint8--;
10        }
11    }
12
13    function increment() public {
14        unchecked {
15            myUint8++;
16        }
17    }
18 }

```

RolloverExample.sol

```

data: 0x2ba...eceb7 logs: 0 hash: 0x1e6...2a373
call to RolloverExample.myUint8
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875
data: 0xe2a...4b2af

```

Then everything works again as you know it from previous Solidity versions. This is such an important quirks of Solidity that I wanted to bring it up early. Now, let's go back to some lighter topics.

Last update: January 3, 2022

7.5 Address Types

One type, which is very specific to Ethereum and Solidity, is the type "address".

Ethereum supports transfer of Ether and communication between Smart Contracts. Those reside on an address. Addresses can be stored in Smart Contracts and can be used to transfer Ether from the Smart Contract to to an address stored in a variable.

That's where variables of the type address come in.

In general, a variable of the type address holds 20 bytes. That's all that happens internally. Let's see what we can do with Solidity and addresses.

7.5.1 Smart Contract Example

Let's create a new Smart Contract to have an example.

AddressExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.1;

contract AddressExample {
    address public myAddress;

    function setAddress(address _address) public {
        myAddress = _address;
    }

    function getBalanceOfAccount() public view returns(uint) {
        return myAddress.balance;
    }
}
```

Deploy the Smart Contract with Remix to the JavaScript VM:

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has a title "DEPLOY & RUN TRANSACTIONS". It includes a "Deploy" button, a checkbox for "Publish to IPFS", and a dropdown for "At Address" or "Load contract from Address". Below that, it says "Transactions recorded 9" and "Deployed Contracts" with a list item "ADDRESSEXAMPLE AT 0xD2A...FD005". Under "Low level interactions", there are buttons for "setAddress", "getBalanceOf...", and "myAddress", each with a dropdown menu. A "Transact" button is at the bottom. On the right, there's a code editor with Solidity code for "AddressExample", a transaction history section with a pending creation transaction and a successful deployment transaction, and a search bar.

Important Concepts

As you continue, please pay special attention to the following few concepts here which are really important and different than in any other programming language:

1. The Smart Contract is stored under its own address
2. The Smart Contract can store an address in the variable "myAddress", which can be its own address, but can be any other address as well.
3. All information on the blockchain is public, so we can retrieve the balance of the address stored in the variable "myAddress"
4. The Smart Contract can transfer funds from his own address to another address. But it cannot transfer the funds from another address.
5. Transferring Ether is fundamentally different than transferring ERC20 Tokens, as you will see later.

Before you continue, read the statements above and keep them in mind. These are the most mind-blowing facts for Ethereum newcomers.

Let's run the Smart Contract and get the balance of addresses programatically.

7.5.2 Run the Smart Contract

What we're going to do is to access the address in the accounts-list programatically from within the Smart Contract. We will:

1. Copy the Address from the Accounts-List
2. Update the "myAddress" variable in the Smart Contract
3. Get the Balance of the address stored

Let's copy your first address from the Accounts-List. Click the little "copy" icon next to your Account:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.1;

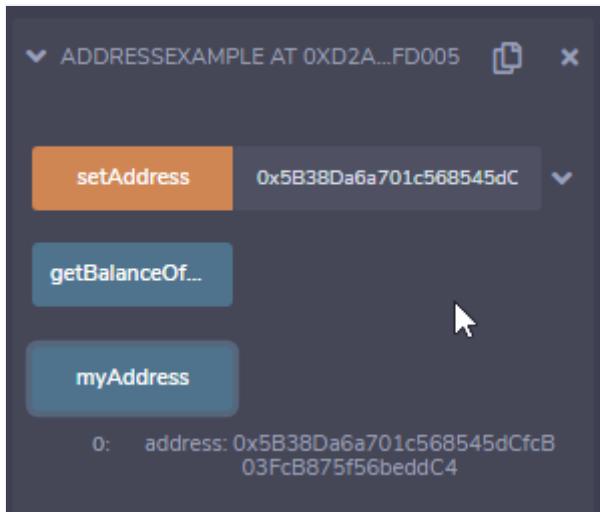
contract AddressExample {
    address public myAddress;

    function setAddress(address _address) public {
        myAddress = _address;
    }

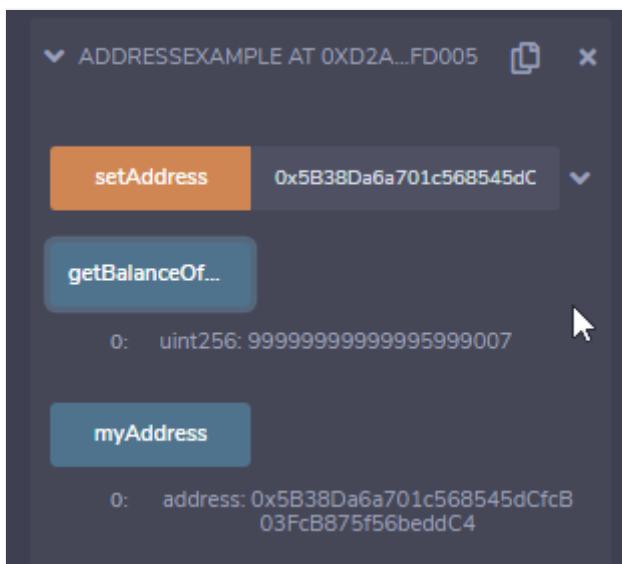
    function getBalanceOfAccount() public view returns(uint) {
        return myAddress.balance;
    }
}
```

Then paste it into the input field next to "setAddress" and then click the "setAddress" button:

When you hit the "myAddress" button, it should show you your address:



Now, let's check the balance, click on "getBalanceOfAccount" button and it should show you the balance in Wei:



Ethereum Denominations

A short reminder on [Ethereum Denominations](#). Wei is the smallest, Ether = 10^{18} Wei.

Unit	Wei Exp	Wei
wei	1	1
Kwei	10^3	1,000
Mwei	10^6	1,000,000
Gwei	10^9	1,000,000,000
Ether	10^{18}	1,000,000,000,000,000,000

Your balance will be very similar to the one in the picture above, probably around 99.999999-some Ether. Why not 100 Ether, or where do the Ether come from? The JavaScript VM is a simulated environment that will "give" you 100 Ether to play. Every transaction costs a little bit of Ether in Gas-Costs, which we will cover later.

Later on you will see how a Smart Contract can manage Ether which are sent to the address of the Smart Contract. Let's discuss an example using Strings next!

Last update: April 17, 2021

7.6 String Types

Strings are actually Arrays, very similar to a bytes-array. If that sounds too complicated, let me break down some quirks for you that are somewhat unique to Solidity:

1. Natively, there are no String manipulation functions.
2. No even string comparison is natively possible
3. There are libraries to work with Strings
4. Strings are expensive to store and work with in Solidity (Gas costs, we talk about them later)
5. As a rule of thumb: try to avoid storing Strings, use Events instead (more on that later as well!)

If you still want to use Strings, then let's do an example!

7.6.1 Example Smart Contract

Let's create the following example Smart Contract, store a String and retrieve it again:

StringExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.1;

contract StringExample {
    string public myString = 'hello world!';

    function setMyString(string memory _myString) public {
        myString = _myString;
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons: a blue headphones icon at the top, followed by a file icon, a circular progress bar with a checkmark, a diamond icon, a plug icon, a value input field (3000000), a gas limit input field (0 wei), and a dropdown for the contract name (StringExample - browser/StringsExam). Below these are buttons for "Deploy" and "Publish to IPFS". The word "OR" is centered between two options: "At Address" and "Load contract from Address". At the bottom, there are sections for "Transactions recorded" (1) and "Deployed Contracts".

On the right, the main area displays a Solidity code editor with the following content:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.1;

contract StringExample {
    string public myString = 'hello world!';

    function setMyString(string memory _myString)
        myString = _myString;
}
```

Below the code editor, there are search and filter controls, and a message indicating the creation of the contract is pending. A transaction log entry is shown at the bottom:

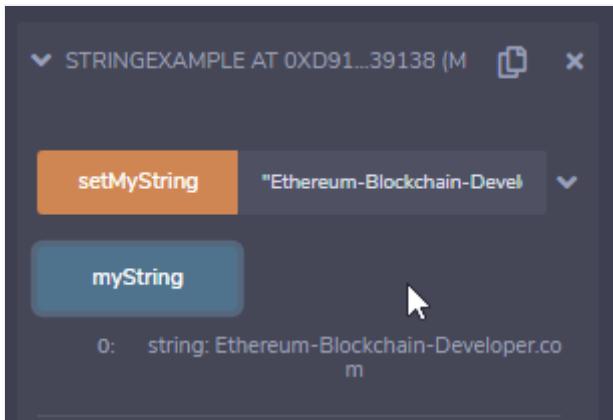
[vm] from: 0x5B3...eddC4 to: StringExample.(constructor) hash: 0xa7a...0b44a

Deploy the Smart Contract into the JavaScript VM and retrieve the String. It should output "hello world!", since it was initialized with this value:

```
setMyString string „myString“  
myString  
0: string: hello world!
```

Now we can overwrite the String with another String:

Write any String you want into the input box next to "setMyString" - for example "Ethereum-Blockchain-Developer.com". Then hit the "myString" button again:



Later, in the Gas-Cost section, you will see how inefficient it is to use Strings. I will also introduce some better ways storing Strings in a trustable way with Events on the Blockchain.

Last update: April 17, 2021

7.7 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

8. LAB: Deposit/Withdraw Ether

8.1 Lab: Smart Contract Self Managing Funds

In this lab you are going learn how a Smart Contract manages its own funds. You will send Ether to your Smart Contract. Then the Smart Contract will manage its own Ether and will be able to relay it to anyone else. It's like a bank account with programming code attached to it.

This can also be used to escrow Ether into a Smart Contract. First we'll do a very simple deposit/withdrawal example, then I'll show you how a Smart Contract can lock funds using a time-activated withdrawal functionality.

8.1.1 What You Know At The End Of The Lab

Understand Contract Addresses and the global msg-object

How Smart Contracts manage Funds

How to Send/Withdraw Ether to and from Smart Contracts

8.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. About 30 Minutes of your precious time

8.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD9E>

8.1.4 Get Started

Let's get started [With A Smart Contract](#)

Last update: April 23, 2021

8.2 Smart Contract

Let's start with a simple Smart Contract. Create a new file in Remix and paste the following code:

SendMoneyExample.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract SendMoneyExample {
    uint public balanceReceived;
    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }
    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
}
```

The screenshot shows the Remix IDE interface. On the left, there is a sidebar with various icons: a blue headphones icon, a white document icon, a green checkmark icon, a diamond icon, and a wrench icon. Below these are the labels "FILE EXPLORERS" and "ARTIFACTS". The "ARTIFACTS" section is expanded, showing a list of files: README.txt, IntegerExample.sol, BooleanExample.sol, RolloverExample.sol, AddressExample.sol, StringsExample.sol, and SendMoneyExample.sol. The "SendMoneyExample.sol" file is highlighted with a gray background. To the right of the sidebar is the main workspace. At the top of the workspace is a tab bar with "SendMoneyExample.sol" and a close button. Below the tab bar is a code editor with numbered lines. The code is as follows:

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.1;
4
5 contract SendMoneyExample {
6     uint public balanceReceived;
7
8     function receiveMoney() public payable {
9         balanceReceived += msg.value;
10    }
11
12    function getBalance() public view returns(uint) {
13        return address(this).balance;
14    }
15 }
16

```

At the bottom of the workspace, there are several status indicators: a dropdown arrow, a network icon, a "0" count, a "listen on network" checkbox, a search icon, and a "Search with" dropdown. A note at the bottom states: "The following libraries are accessible: • web3 version 1.0.0 • ethers.js • swarmgw • remix (run remix.help() for more info)".

A lot of new stuff in here. Don't worry, in a bit all of those things will be very familiar to you. I keep explaining them as we go along, but if you're interested right away then let's go through this one line at a time:

```
uint public balanceReceived : is a public storage variable. A public variable will create a getter function automatically in Solidity. So we can always query the current content of this variable.
```

`balanceReceived += msg.value`: The msg-object is a global always-existing object containing a few informations about the ongoing transaction. The two most important properties are .value and .sender. Former contains the amount of Wei that was sent to the smart contract. Latter contains the address that called the Smart Contract. We will use this extensively later on, so, just keep going for now.

```
function getBalance() public view returns(uint) : a view function is a function that doesn't alter the storage (read-only) and can return information. It doesn't need to be mined and it is virtually free of charge.
```

`address(this).balance`: A variable of the type address always has a property called .balance which gives you the amount of ether stored on that address. It doesn't mean you can access them, it just tells you how much is stored there. Remember, it's all public information. `address(this)` converts the Smart Contract instance to an address. So, this line essentially returns the amount of Ether that are stored on the Smart Contract itself.

Let's see if we can do something with it. Deploy the Smart Contract and play around a bit...

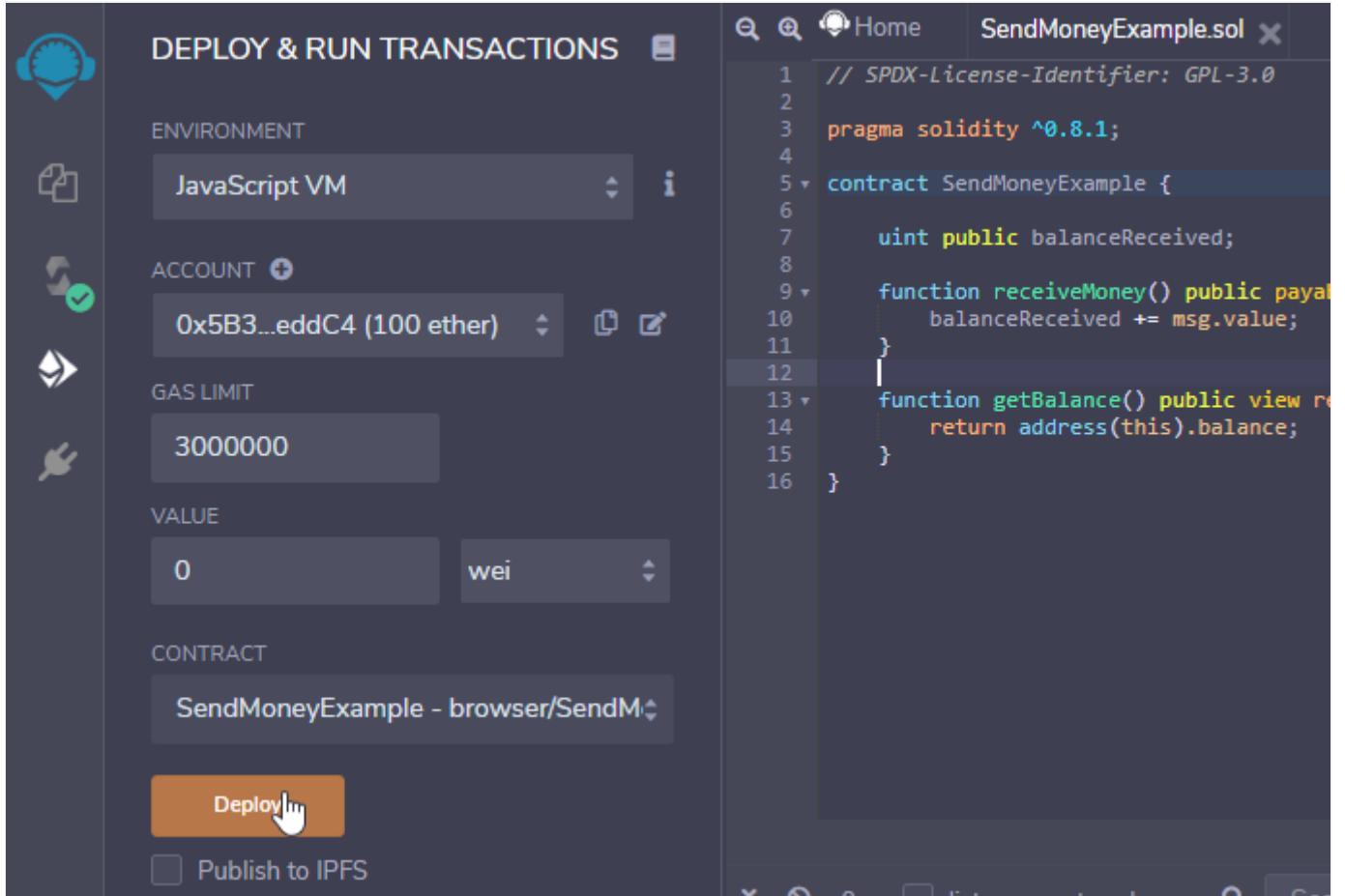
Last update: April 17, 2021

8.3 Deploy and Use the Smart Contract

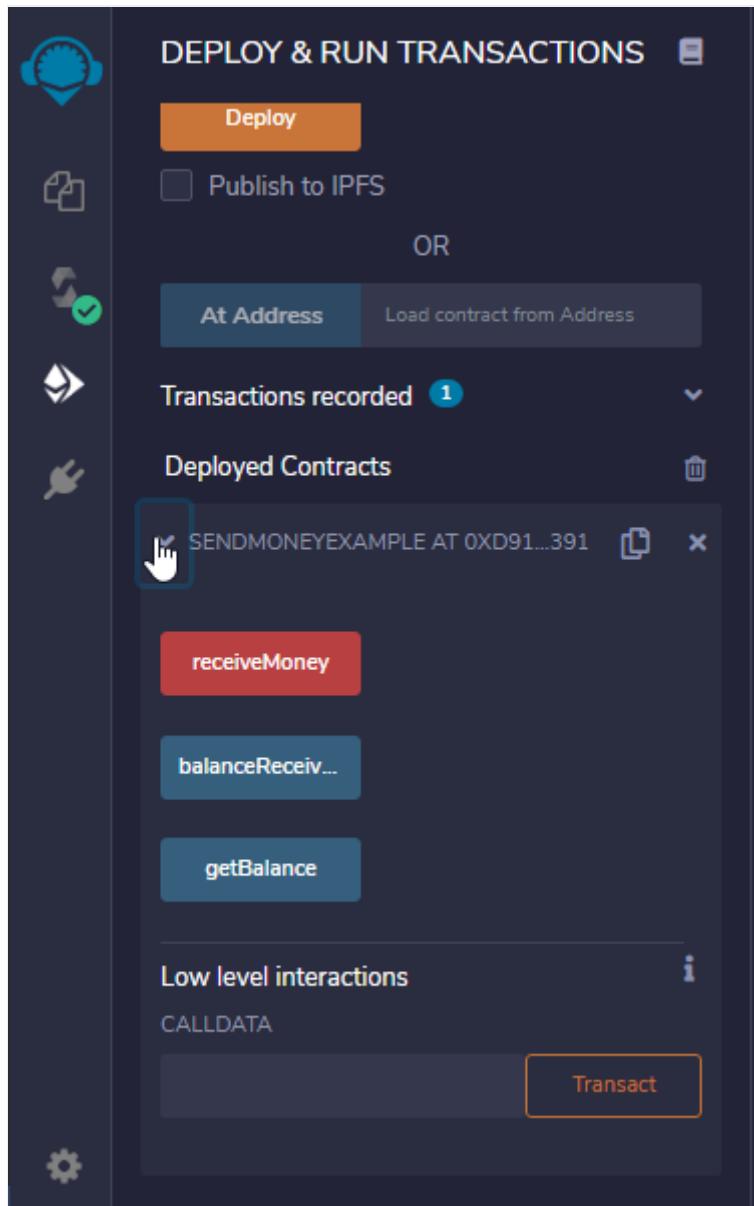
First thing is to deploy the Smart Contract. Then we'll see if we can deposit some Ether and get the balance from the Smart Contract.

8.3.1 Deploy the Smart Contract

Head over to the Deploy and Run Transactions Plugin and deploy the Smart Contract into the JavaScript VM:



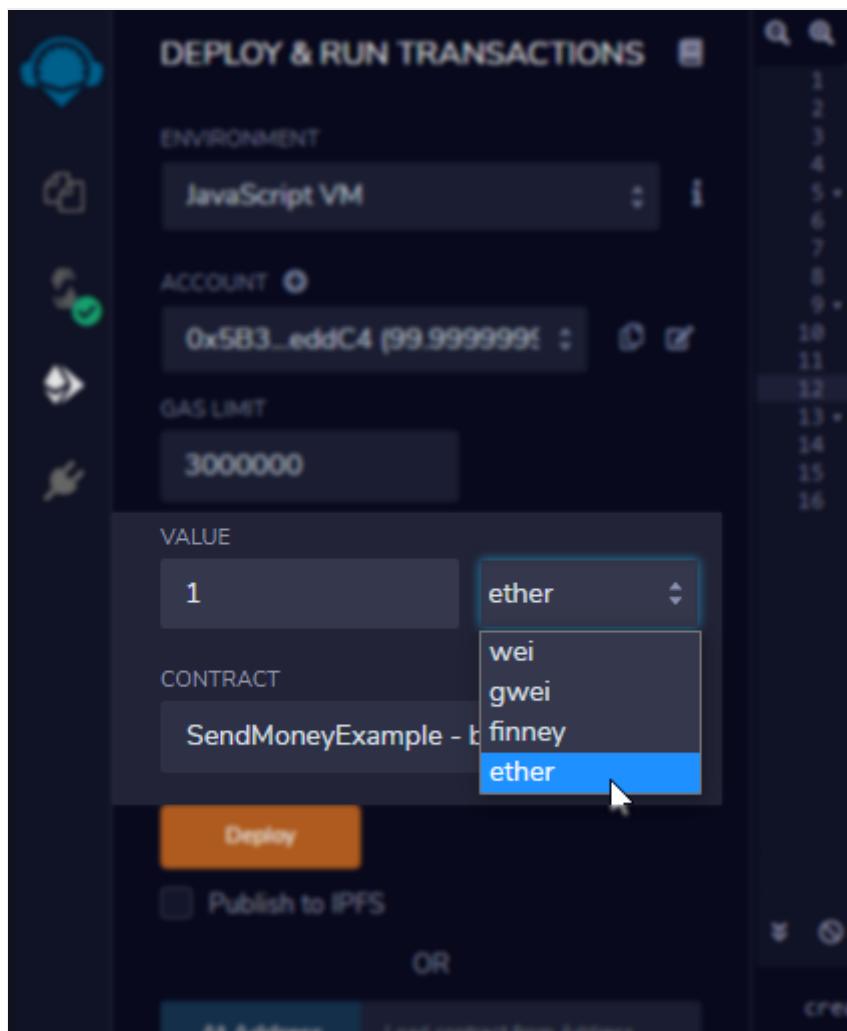
It should appear at the bottom of the Plugin - you probably need to expand the contract instance:



8.3.2 Send Ether To The Smart Contract

Now it is time to send some Ether to the Smart Contract!

Scroll up to the "value" field and put "1" into the value input field and select "ether" from the dropdown:



Then scroll down to the Smart Contract and hit the red "receiveMoney" button:

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying contracts, publishing to IPFS, and managing deployed contracts. A red arrow points from the sidebar to the 'receiveMoney' button in the main content area. The main content area has tabs for 'Deploy & Run Transactions' and 'SendMoneyExample.sol'. The 'Deploy & Run Transactions' tab is active, showing a 'Deploy' button, a checkbox for 'Publish to IPFS', and sections for 'Transactions recorded' and 'Deployed Contracts'. Under 'Deployed Contracts', a dropdown menu is open for 'SENDMONEYEXAMPLE AT 0xD91...391', showing three buttons: 'receiveMoney' (highlighted with a red arrow), 'balanceReceiv...', and 'getBalance'. Below these are sections for 'Low level interactions' and 'CALLDATA', each with a 'Transact' button. The right side of the screen shows the Solidity code for the contract:

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.1;
4
5 contract SendMoneyExample {
6     uint public balanceReceived;
7
8     function receiveMoney() public payable {
9         balanceReceived += msg.value;
10    }
11
12    function getBalance() public view returns(uint) {
13        return address(this).balance;
14    }
15}
16

```

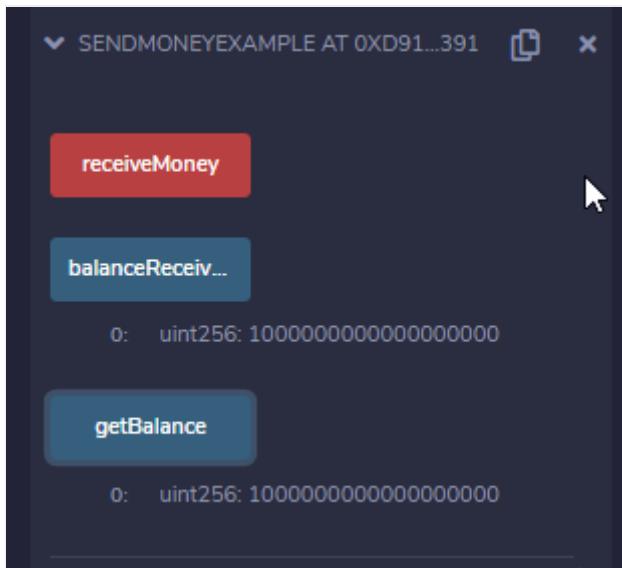
Below the code, there's a status bar with a green checkmark, a pending transaction count (0), a 'listen on network' checkbox, and a search bar.

Also observe the terminal , see that there was a new transaction sent to "the network" (although just a simulation in the browser, but it would be the same with a real blockchain).

8.3.3 Check the Balance

Now we sent 1 Ether, or 10^{18} Wei, to the Smart Contract. According to our code the variable `balanceReceived` and the function `getBalance()` should have the same value.

And, indeed, they do:



But how can we get the Ether out again? Let's add a simple Withdrawal method.

Try it yourself first?

You want to try yourself first? Here are some hints:

We want a function that sends all Ether stored in the Smart Contract to the `msg.sender` (that's the address that calls the Smart Contract).

Since Solidity 0.8 that is non-payable, so you'd need to do something like `payable(msg.sender)`, which would give you an address that is capable of receiving Ether.

If you have no clue what the heck I'm talking about, don't worry - just head over to the next page.

Last update: April 17, 2021

8.4 Withdraw Ether From Smart Contract

So far we have sent Ether to our Smart Contract. But there is currently no way to get Ether back out again! So, what's next? Yes! A function to withdraw Ether would be good, ey?!

8.4.1 Add a Withdraw Function

Let's add the following function to the Smart Contract:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract SendMoneyExample {
    uint public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function withdrawMoney() public {
        address payable to = payable(msg.sender);
        to.transfer(getBalance());
    }
}
```

This function will send all funds stored in the Smart Contract to the person who calls the "withdrawMoney()" function.

8.4.2 Deploy the new Smart Contract

Let's try this:

1. Deploy the new version and send again 1 Ether to the Smart Contract.
2. To avoid confusion I recommend you close the previous Instance, we won't need it anymore

The screenshot shows the Remix IDE interface with the following details:

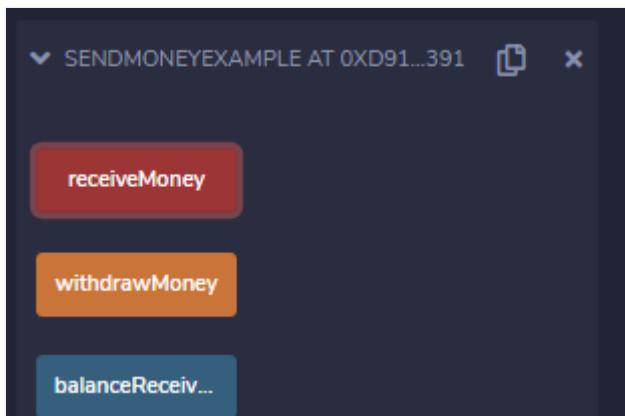
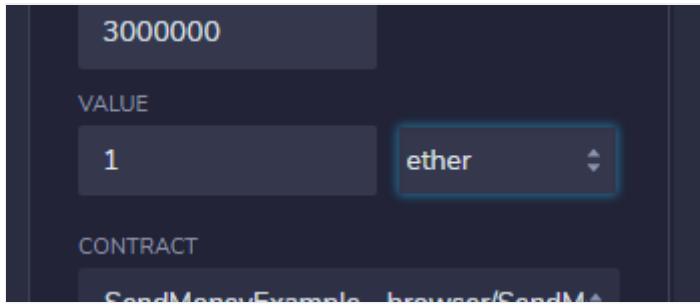
- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.99999999999999)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** SendMoneyExample - browser/SendM...
- Deploy Button:** An orange button labeled "Deploy" is highlighted with a red circle and the number 1.
- At Address:** The selected option for deployment.
- Transactions recorded:** 1
- Deployed Contracts:** A list containing "SENDMONEYEXAMPLE AT 0xD91...391".
- receiveMoney:** A red button associated with the deployed contract.
- Solidity Code:**

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract SendMoneyExample {
    uint public balanceReceived;
    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }
    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
    function withdrawMoney() public {
        address payable to = payable(msg.sender);
        to.transfer(getBalance());
    }
}
```
- Logs:**
 - creation of SendMoneyExample pending...
 - [vm] from: 0x5B3...eddC4 to: SendMoneyExample...

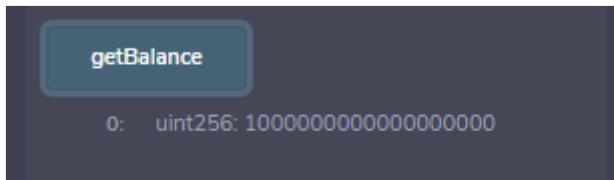
At the end you should end up with one active Instance of your Smart Contract.

The same procedure as before:

1. Put in "1 Ether" into the value input box
2. hit "receiveMoney" in your new contract Instance



Your balance should be 1 Ether again:



Not 1 Ether?

If your balance is 0, then double check the value field

If your balance is 2 Ether, then double check the contract Instance you are interacting with!

8.4.3 Withdraw Funds from the Smart Contract

Now it's time we use our new function! But to make things more exciting, we're going to withdraw to a different Account.

Select the second Account from the Accounts dropdown:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract SendMoneyExample {
    uint public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }

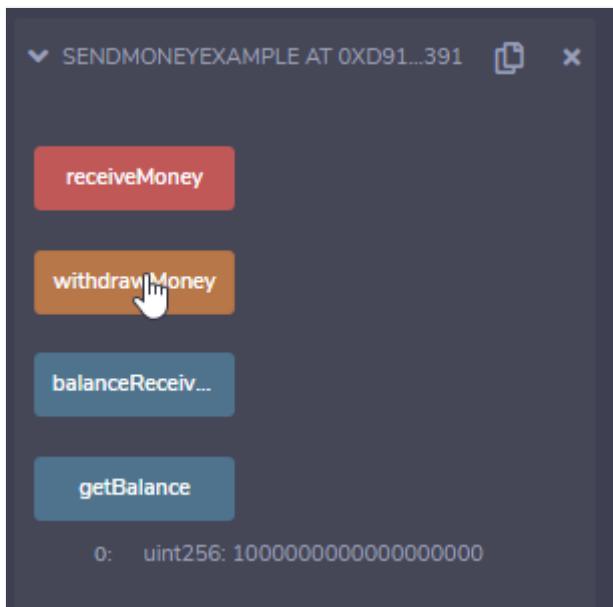
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }

    function withdrawMoney() public {
        address payable to = payable(msg.sender);
        to.transfer(getBalance());
    }
}
```

Transactions recorded 2

[vm] from: 0x5B3...eddC4 to: SendMoneyExample at 0xD91...391 logs: 0 hash: 0xb0...227ae

Then hit the "withdrawMoney" button:



Observe the amount of Ether you have now in your Account:

The screenshot shows the Truffle UI's 'Deploy & Run Transactions' interface. On the left, there are several icons: a stopwatch, a file, a gear, a checkmark, a diamond, and a hand. The main area has tabs for 'ENVIRONMENT' (set to 'JavaScript VM') and 'ACCOUNT' (which shows a list of accounts). One account is selected: '0xAb8...35cb2 (100.999999999999759095 ether)'. Below this, a list of other accounts is visible, including '0x5B3...eddC4 (98.999999999999759095 ether)' and '0xAb8...35cb2 (100.99999999999970987 ether)' (which is also highlighted with a blue selection bar).

It's more than the previous 100 Ether! We got our 1 Ether through our Smart Contract into another Account! AWESOME!

Why not 101 Ether?

Are you wondering why you don't have 101 Ether in your Account? After all, you had 100 Ether before, and now you added 1 Ether, so, why is it not 101 Ether? Is the Math you learned in school worthless?

No, the Math you learned in School comes in handy actually.

What you can observe here is the concept of "Gas" on the Ethereum Blockchain. Every transaction on Ethereum costs a little bit. And it's not different here on a simulated chain. Same principles apply. How much is the Gas you paid, you're wondering? Well, you can open the transaction details and see for yourself. We're covering this - in depth - later on in the course. I also made a dedicated [video and blog post](#) about this if you want to deep dive right now.

While we can withdraw our funds now, the whole function itself is pretty useless, isn't it?! Anyone can withdraw funds to his Account. There are no fractions of the Amount - all in all, pretty insecure.

I still hope the concept is a bit clearer now!

Let's to another function, which allows us the send the full amount to a specific Address! It will still be insecure, but at least teaches a new concept - one at a time!

Try yourself first?

If you want to try yourself first, then do the following:

1. Create a new function that takes one address as argument
2. The full amount of Ether stored on the Smart Contract will be sent to this address

Alright, let's do this on the next page!

8.5 Withdraw To Specific Account

Previously we had our Smart Contract just blindly send the Ether to whoever called the Smart Contracts "withdrawMoney" function. Let's extend this a bit so that the Funds can be send to a specific Account.

It's still not secure, as basically anyone could interact with that function, but it's one step closer!

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.1;

contract SendMoneyExample {

    uint public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function withdrawMoney() public {
        address payable to = payable(msg.sender);
        to.transfer(getBalance());
    }

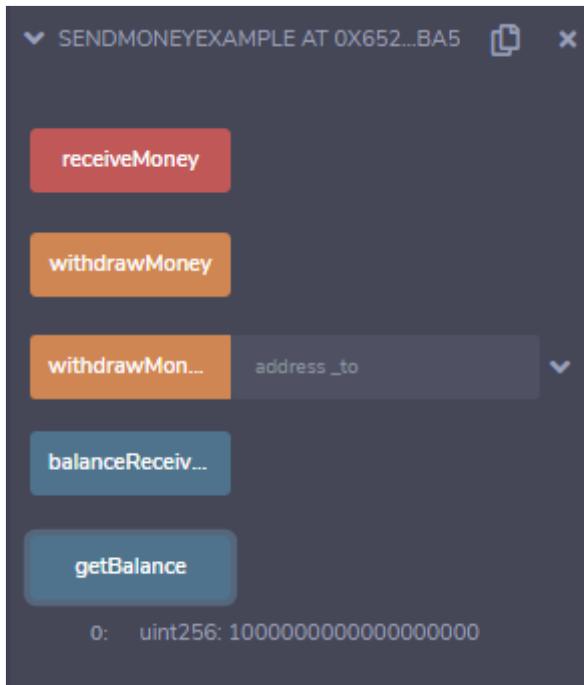
    function withdrawMoneyTo(address payable _to) public {
        _to.transfer(getBalance());
    }
}
```

As you can see, we can now specify an Address the money will be transferred to! Let's give this a try!

8.5.1 Redeploy our Smart Contract

Of course, we need to re-deploy our Smart Contract. There are no live-updates (yet?!). Same procedure as before:

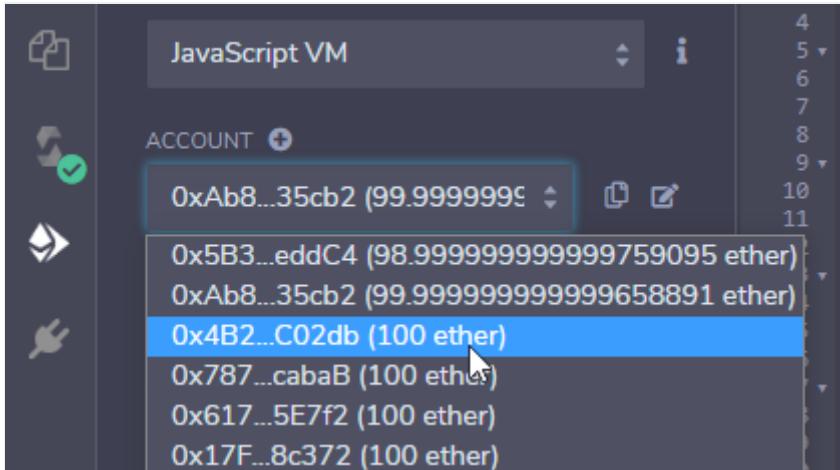
1. Deploy the Smart Contract
 2. Close the old Instance
 3. Send 1 Ether to the Smart Contract (don't forget the value input field!)
 4. Make sure the Balance shows up correctly.



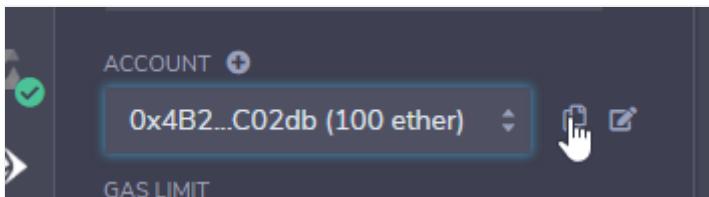
8.5.2 Test the "withdrawMoneyTo" function

Now it's time to test the new function. We're going to make things a bit more exciting : We're going to use of first account to send all funds to the third account. Why? Because we can . And because it's important to understand how gas fees are working - who is paying for the transaction.

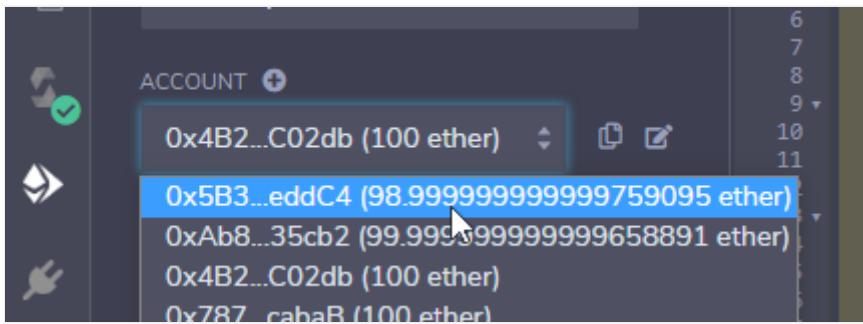
1. Select the third account from the dropdown



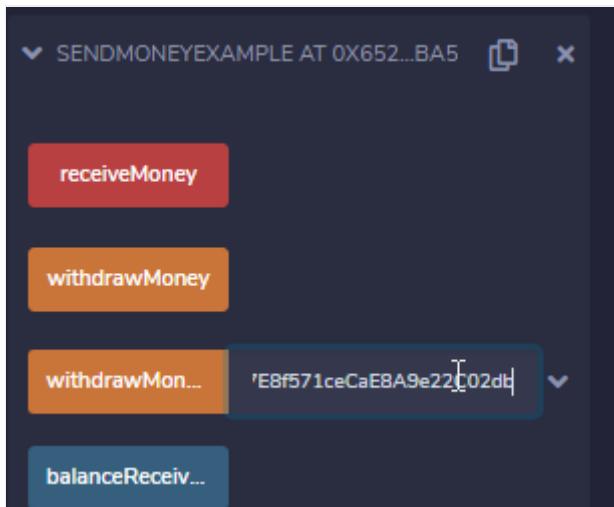
1. Hit the little "copy" icon:



1. Switch back to the first Account:



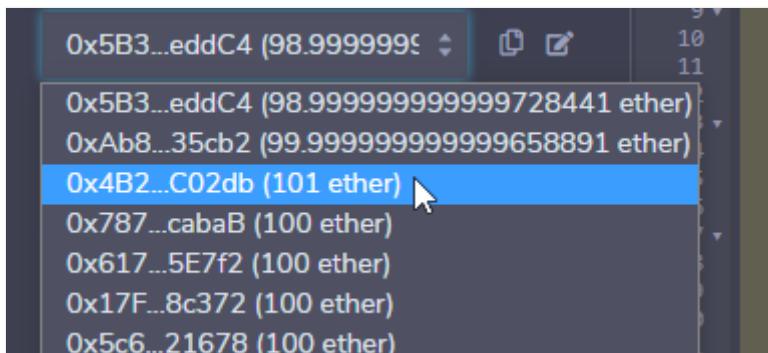
1. Paste the Account you copied into the input field next to "withdrawMoneyTo":



No function with that name?

If there's no function with that name, then you are most likely still interacting with the old Instance. Re-Deploy or even reload the whole page.

1. Hit the "withdrawMoneyTo" button and see what happens! Wow, nothing . Well, only on the surface!
2. Now open the Accounts dropdown. See the balance of your third Account? 101 Ether!!!



Why is there 101 Ether and not 100.999999999some? Because we sent a transaction from Account #1 to the Smart Contract, instructing the Smart Contract to send all funds stored on the Address of the Smart Contract to the third Account in your Account-List. Gas fees were paid by Account #1. Account #3 got 1 full Ether!

8.5.3 Time-Locked Withdrawals

That's all cool and fun so far, but let's go one step further and introduce `block.timestamp`. This global object contains the timestamp when a block was mined. It's not necessarily the current timestamp when the execution happens. It might be a few seconds off. But it's still enough to do some locking.

Next up I want to write a short Smart Contract that only allows withdrawal if the last deposit was more than 1 Minute ago.

Try yourself first?

If you want to try yourself first, then we extend the Smart Contract and store the "block.timestamp" somewhere. Withdrawals can only happen if the "block.timestamp" during withdrawal is greater than the previously stored timestamp + 1 minutes (that is a globally available [constant in Solidity](#))

This is potentially not in the course videos. This exercise is optional.

Last update: April 17, 2021

8.6 Withdrawal Locking

Let's extend our Smart Contract to do some locking.

You will see that it is very easy to let our code take care of some specific logic to allow/disallow certain actions.

8.6.1 Extend the Smart Contract

What we need is to store the `block.timestamp` somewhere. There are several methods to go about this, I prefer to let the user know how long is it locked. So, instead of storing the deposit-timestamp, I will store the `lockedUntil` timestamp. Let's see what happens here:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.1;

contract SendMoneyExample {

    uint public balanceReceived;
    uint public lockedUntil;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
        lockedUntil = block.timestamp + 1 minutes;
    }

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function withdrawMoney() public {
        if(lockedUntil < block.timestamp) {
            address payable to = payable(msg.sender);
            to.transfer(getBalance());
        }
    }

    function withdrawMoneyTo(address payable _to) public {
        if(lockedUntil < block.timestamp) {
            _to.transfer(getBalance());
        }
    }
}
```

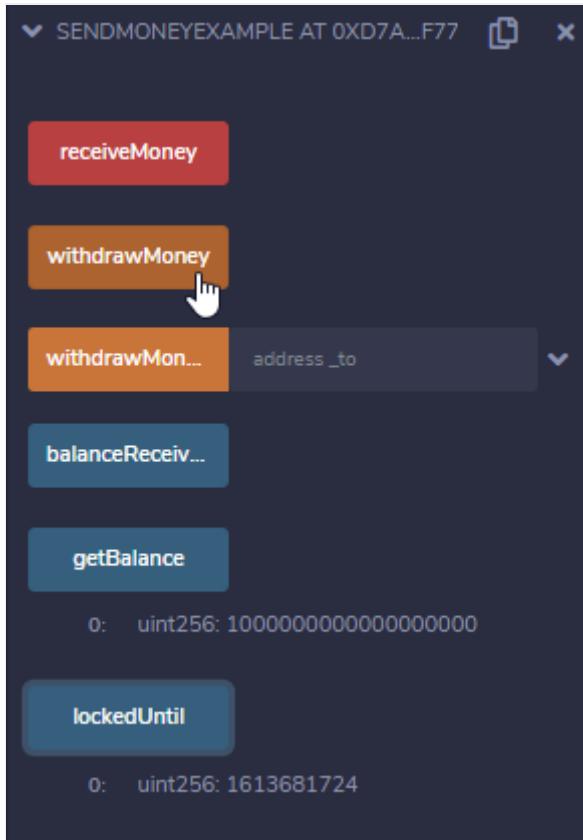
8.6.2 Deploy and Test the Smart Contract

Let's deploy the Smart Contract, same procedure as before:

1. Deploy a new Instance version
2. Remove the old Instance
3. Send 1 Ether to the Smart Contract (don't forget the value field) by clicking on "receiveMoney"
4. Check the Balance!

Alright, now what? Well, check the "lockedUntil" by clicking on the button. It will give you the timestamp until nothing happens when you click `withdrawMoney` or `withdrawMoneyTo`.

1. Click "withdrawMoney" - and nothing happens. The Balance stays the same until 1 Minute passed since you hit "receiveMoney".



Try it yourself!

But doing nothing, no feedback, that's not really user-friendly. You will learn later about [Exceptions: Require, Assert, Revert](#), how we can make this more user-friendly.

That's it for now, good job!

Last update: April 17, 2021

8.7 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

9. Smart Contract Life-Cycle

9.1 Lab: Starting, Pausing and Destroying/Stopping Smart Contracts

One things that is repeatedly asked: How can we stop Smart Contracts? What happens if we stop using Smart Contracts? How to destroy Smart Contracts? And many more questions are answered in these few exercises!

9.1.1 What You Know At The End Of The Lab

How to destroy a Smart Contract for good using `selfdestruct`

❖ The Smart Contract life-cycle (can we actually update?)

When we can interact with Smart Contracts and when not

☺ The Solidity Constructor

9.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. Know how to use Remix a bit

9.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

9.1.4 Get Started

Let's get started!!! I know you can do it!

Last update: April 23, 2021

9.2 Sample Smart Contract

Before we begin, let's setup a sample Smart Contract. Just enough to get something working for us to have a simplistic example.

Create a new file in Remix. I've named mine "StartingStopping.sol", but obviously, you can name your file any way you'd like.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract StartStopUpdateExample {
    function sendMoney() public payable {

    }

    function withdrawAllMoney(address payable _to) public {
        _to.transfer(address(this).balance);
    }
}
```

9.2.1 What does it do?

We have two functions:

`sendMoney()` : This function can receive Ether, it's a payable function.

`withdrawAllMoney(...)` : Very similar to our previous example, this function will automatically withdraw all available funds stored at the address of the Smart Contract to the variable in the as function argument given address. What a sentence! In other words: It sends all Ether to the "`_to`" address.

9.2.2 Deploy the Smart Contract

Let's deploy the Smart Contract to the "JavaScript VM" in Remix. Head over to the "Deploy & Run Transactions" Plugin and hit Deploy:

Perfect! Let's try to send some Funds around!

9.2.3 Use the Smart Contract

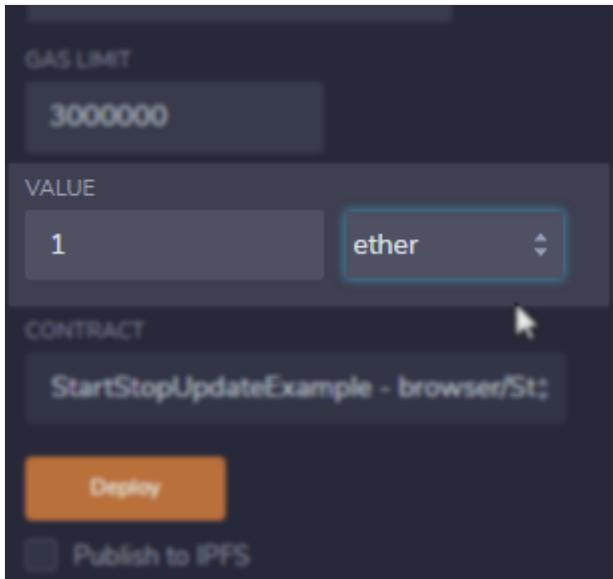
As before, we're going to:

1. Send Money to the Smart Contract using Account #1
 2. Withdraw the Money using any other Account

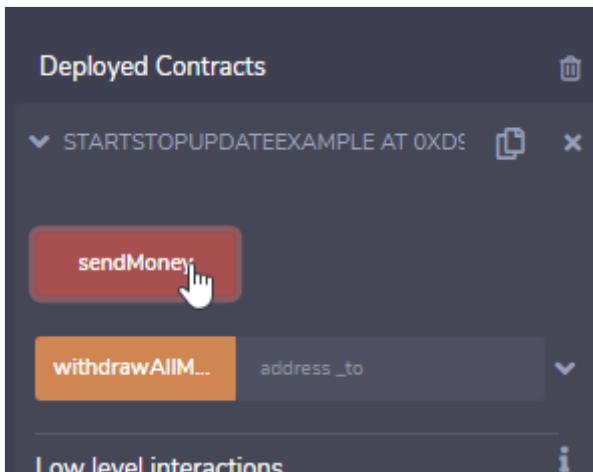
Already sounds scarily unsecure. If you come from traditional backend development, you should shiver now. But worry not, we'll get to safe heavens soon!

Alright, so, start by sending some funds to the Smart Contract...

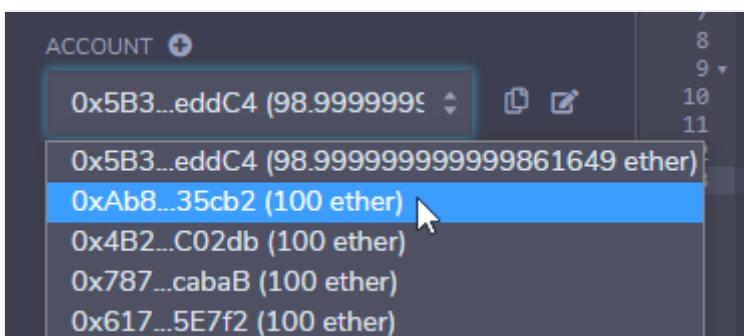
1. Enter 1 Ether into the value field:



1. send it to the "sendMoney" function of your Smart Contract:



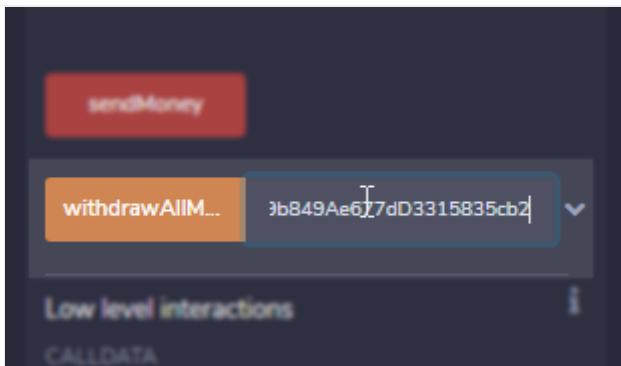
1. Select the second Account in the Accounts-Dropdown:



1. Copy the Address:



1. Paste the Address into the "withdrawAllMoney" input field:



1. Click the "withdrawAllMoney" button.
2. Have a look if you have >100 eth in your Account #2 of the Accounts-Dropdown:



So, just to summarize how **insecure** that is:

1. Someone funded the Smart Contract
2. But everyone can withdraw to any address of their choice?

That sounds pretty bad, right?

We can do better than that!!! In the next exercise we're going to restrict withdrawals to the person who owns the Smart Contract.

Try yourself first?

Want to give it a try yourself? Great idea!

So, here's what you need:

1. A variable that stores the address of the person who deploys the Smart Contract.
2. a **constructor**. This get's called when the Smart Contract is deployed. It's named `constructor() {...}`. Inside you set the address to the `msg.sender`.
3. a require in the `withdrawAllMoney` function. We're talking about Exceptions later in the course extensively, so don't worry too much about the internal workings. Make sure that the Address that calls the `withdrawAllMoney` function is the same as stored in the variable that is set by the constructor.

Alright, off to the solution: next page!

Last update: April 17, 2021

9.3 The Constructor

One thing we haven't really talked about yet is the constructor.

It's something you need to understand before we proceed!

The constructor is a special function. It is automatically called during Smart Contract deployment. And it can never be called again after that.

It also has a special name! It's simply called `constructor() { ... }`.

Let's see how that works to our advantage. Let's extend the Smart Contract we wrote before to make it a bit more secure.

9.3.1 Securing our Smart Contract using a simple Ownership-Model

We are going to set a storage variable to the address that deployed the Smart Contract. Then we will `require()` that the person interacting with `withdrawAllMoney` is the same as the one who deployed the Smart Contract.

Extend our Smart Contract:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract StartStopUpdateExample {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function sendMoney() public payable {

    }

    function withdrawAllMoney(address payable _to) public {
        require(owner == msg.sender, "You cannot withdraw.");
        _to.transfer(address(this).balance);
    }
}
```

So much new stuff in there! Let's dig through it line by line:

`constructor()` : is a special function that is called only once during contract deployment. It still has the same global objects available as in any other transaction. So in `msg.sender` is the address of the person who deployed the Smart Contract

`require(owner == msg.sender, "You cannot withdraw.")` : That might be a bit early, but this is how you trigger Errors (or throw Exceptions) in Solidity. If the require evaluates to false it will stop the transaction, roll-back any changes made so far and emit the error message as String.

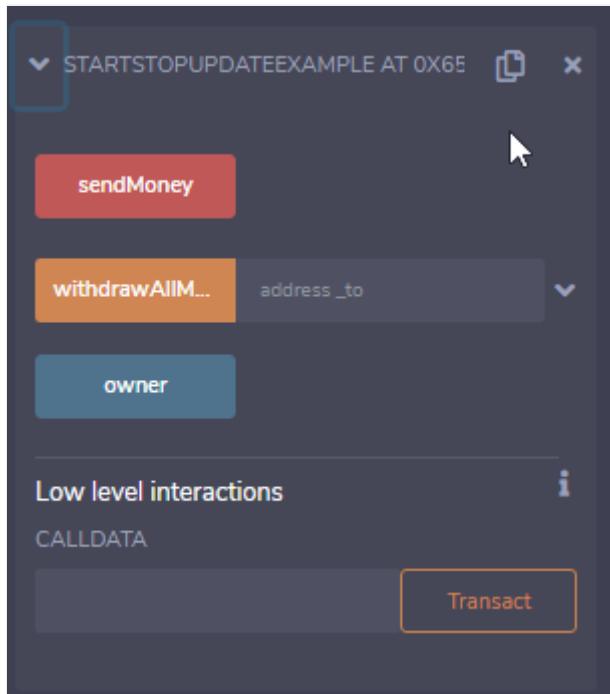
Everyone can send Ether to our Smart Contract. But only the person who deployed the Smart Contract can withdraw. Secure and Smart - Let's try this!

9.3.2 Testing our new Smart Contract

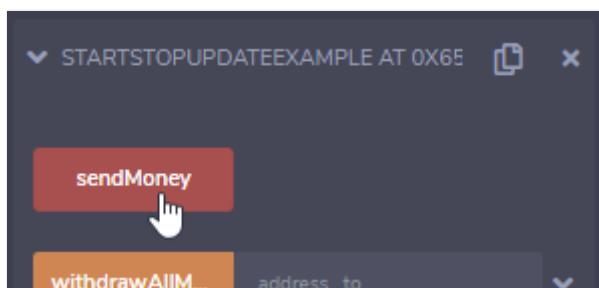
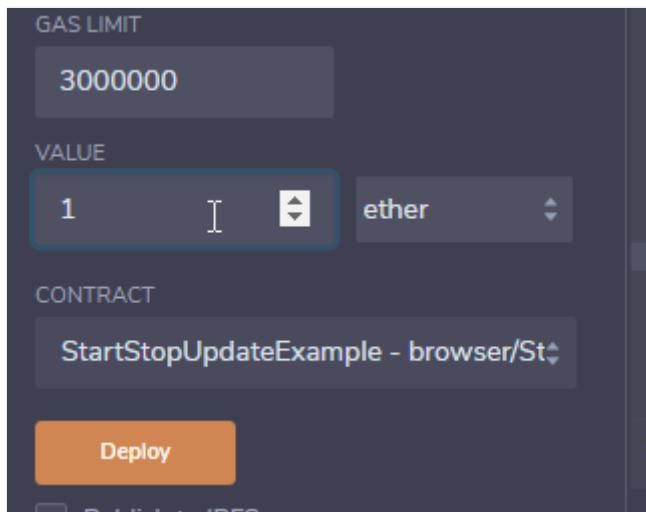
We're going to do the following:

Deploy the new Smart Contract

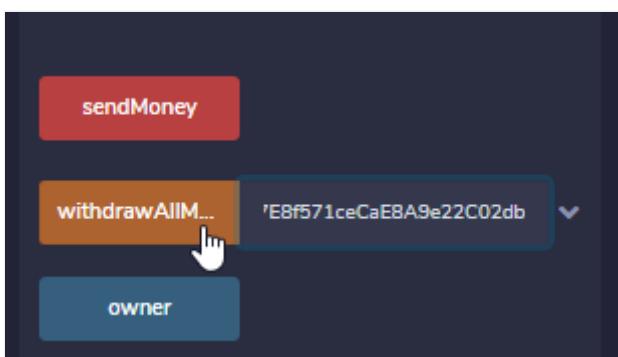
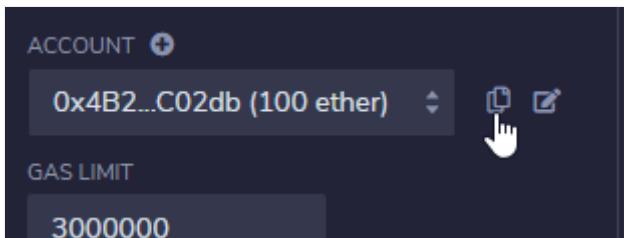
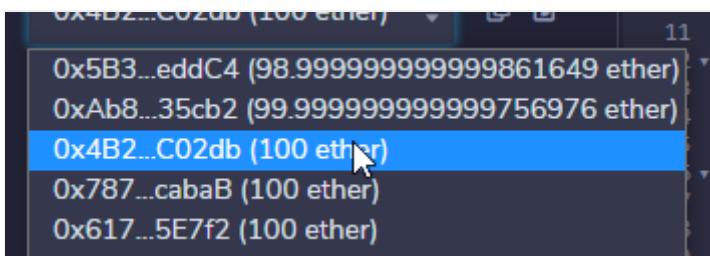
1.



2. Send Ether to the Smart Contract



3. Try to use another Account to withdraw



Note: Don't switch back, use the other Account to call withdrawAllMoney

4. Observe the Error Message

The screenshot shows the Truffle UI for the `STARTSTOPUPDATEEXAMPLE` contract. On the left, there are three buttons: `sendMoney` (red), `withdrawAllMoney` (orange), and `owner` (blue). The `withdrawAllMoney` button has a cursor hovering over it. On the right, the transaction details are shown:

- From:** `0xAb8...35cb2`
- To:** `StartStopUpdateExample`
- Value:** `1000000000000000000 wei`
- Data:** `0x0ad...c02db`
- Logs:** `0`
- Hash:** `0xf2...`

The status of the transaction is `pending`. Below the transaction details, an error message is displayed:

transact to StartStopUpdateExample.withdrawAllMoney errored
reverted to the initial state. Reason provided by the contract:
 get more information.

Switch back to the original Account

5.

The screenshot shows a list of Ethereum accounts in a dropdown menu. The account '0xAb8...35cb2 (99.99999999999999756976 ether)' is highlighted with a blue selection bar, indicating it is the active account.

Note: I used the second account in my account-dropdown to deploy the Smart Contract, so I am switching back to this one

6. Withdraw and see it works

The screenshot shows the Remix IDE interface. On the left, the Solidity code for 'StartStopUpdateExample' is displayed. A red circle with the number 1 highlights the 'withdrawAllMoney' button. On the right, the transaction history shows two entries:

- [vm] from: 0x4B2...C02db to: StartStopUpdateExample value: 0 wei data: 0x0ad...c02db logs: 0 hash: 0...
- [vm] from: 0xAb8...35cb2 to: StartStopUpdateExample value: 0 wei data: 0x0ad...c02db logs: 0 hash: 0...

Amazing, right! A very simplistic access rule that denies access to anyone except the one who deployed the Smart Contract.

Of course, you can spin that further. You can create a whole Access Model around this. [OpenZeppelin did this in their Ownable Models](#)

So, what's next? Let's see if we can pause our Smart Contract!

Can you think of a solution?

The Ethereum Virtual Machine has no functionality to pause Smart Contracts on a protocol-level. So, we need to think of a solution "in code".

Can you think of something? Maybe a boolean that pauses any deposits and withdrawals when it's true?

Give it a try!

Alright, off to Pausing Smart Contracts...

Last update: April 17, 2021

9.4 Pausing Smart Contract

A Smart Contract cannot be "paused" on a protocol-level on the Ethereum Blockchain. But we can add some logic "in code" to pause it.

Let's try this!

9.4.1 Smart Contract Pausing Functionality

Let us update our Smart Contract and add a simple Boolean variable to see if the functionality is paused or not.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract StartStopUpdateExample {

    address public owner;
    bool public paused;

    constructor() {
        owner = msg.sender;
    }

    function sendMoney() public payable {

    }

    function setPaused(bool _paused) public {
        require(msg.sender == owner, "You are not the owner");
        paused = _paused;
    }

    function withdrawAllMoney(address payable _to) public {
        require(owner == msg.sender, "You cannot withdraw.");
        require(paused == false, "Contract Paused");
        _to.transfer(address(this).balance);
    }
}
```

Content wise, there isn't much new. We have a function that can update the `paused` variable. That function can only be called by the owner. And `withdrawAllMoney` can only be called if the contract is not paused.

Of course, this doesn't make much sense, other than being some sort of academic example. BUT! It makes sense if you think of more complex functionality, like a Token Sale that can be paused. If you have `require(paused == false)` in all your customer facing functions, then you can easily pause a Smart Contract.

9.4.2 Try the Smart Contract

Of course we're also trying if our functionality works!

1. Deploy a new Instance
2. Update `paused` to true
3. Try to `withdrawAllMoney` with any of your accounts, it won't work!

The screenshot shows the Truffle UI interface for a deployed smart contract named 'STARTSTOPUPDATEEXAMPLE'. On the left, there's a sidebar titled 'Deployed Contracts' with a dropdown menu. Below it are four buttons: 'sendMoney' (red), 'setPaused' (orange, currently set to 'true'), 'withdrawAll...' (orange, address: 0xA6B483F64d9C6d1EcF9b), and 'owner' (blue). At the bottom of this sidebar is a 'Low level interactions' section with 'CALLDATA' and a 'TRANSACTION' button.

The main right-hand area displays the Solidity code for the contract:

```

14
15
16
17
18
19
20
21
22
23
24
25
}
function setPaused(bool _paused) public {
    require(msg.sender == owner, "You are not the owner");
    paused = _paused;
}

function withdrawAllMoney(address payable _to) public {
    require(owner == msg.sender, "You cannot withdraw");
    require(paused == false, "Contract Paused");
    // Transfer funds from this contract to the owner
}

```

Below the code, there are several transaction logs:

- A green log: 'transact to StartStopUpdateExample.withdrawAllMoney errored: EIP-160 (argument="address", value="", code=INVALID_ARGUMENT, version=0x1, error=0x0, code=INVALID_ARGUMENT, version=abi/5.0.7)'.
- A grey log: 'transact to StartStopUpdateExample.withdrawAllMoney pending ...'.
- A red error log: '[vm] from: 0xAB8...35cb2 to: StartStopUpdateExample.withdrawAllMoney value: 0 wei data: 0x0ad...35cb2 logs: 0 hash: 0xd57...b3'.
- A grey log: 'transact to StartStopUpdateExample.withdrawAllMoney errored: VM error: revert reverted to the initial state. Reason provided by the contract more information.'

In the next and last exercise for this Lab I want to destroy our Smart Contract.

Last update: April 17, 2021

9.5 Destroy Smart Contracts using `selfdestruct`

We can not erase information from the Blockchain, but we can update the current state so that you can't interact with an address anymore going forward. Everyone can always go back in time and check what was the value on day X, but, once `selfdestruct` is called, you can't interact with a Smart Contract anymore.

Let's try this!

Potential Removal of SELFDESTRUCT

There might be an [Ethereum Protocol update](#) coming ahead which removes the SELFDESTRUCT functionality all-together. As writing this, it's not out there (yet), but might be soon, so take the following lab with this in mind.

9.5.1 Update our Smart Contract

Let's update our Smart Contract and add a `selfdestruct` function. This function takes one argument, an address. When `selfdestruct` is called, all remaining funds on the address of the Smart Contract are transferred to that address.

```
contract StartStopUpdateExample {
    address public owner;
    bool public paused;

    constructor() {
        owner = msg.sender;
    }

    function sendMoney() public payable {

    }

    function setPaused(bool _paused) public {
        require(msg.sender == owner, "You are not the owner");
        paused = _paused;
    }

    function withdrawAllMoney(address payable _to) public {
        require(owner == msg.sender, "You cannot withdraw.");
        require(paused == false, "Contract Paused");
        _to.transfer(address(this).balance);
    }

    function destroySmartContract(address payable _to) public {
        require(msg.sender == owner, "You are not the owner");
        selfdestruct(_to);
    }
}
```

On the surface it looks very similar to our `withdrawAllMoney` function, with one major difference: Once you call `destroySmartContract`, the address of the Smart Contract will contain no more code. You can still send transactions to the address and transfer Ether there, but there won't be any code that could send you the Ether back.

Let's try this!

9.5.2 Try our new Smart Contract

1. Deploy a new Instance
2. Send some Ether there by interacting with `sendMoney`
3. Try to call `destroySmartContract` and provide your own Account, so Ether are sent back.

A Transaction should go through smoothly!

The screenshot shows the Truffle UI interface for a deployed contract named "STARTSTOPUPDATEEXAMPLE AT 0x84". The left sidebar lists several functions: "destroySmartContract", "sendMoney", "setPaused", "withdrawAllMoney", "owner", "paused", and "Low level interactions". The "sendMoney" button is highlighted with a red background. The "Paused" status is shown as a teal box. The "Low level interactions" section includes a "CALLDATA" button and a "Transact" button.

The right side displays the Solidity code for the contract:

```
18 require(msg.sender == owner, "You are not the owner");
19 paused = _paused;
20 }
21
22 function withdrawAllMoney(address payable _to) public {
23     require(owner == msg.sender, "You cannot withdraw");
24     require(paused == false, "Contract Paused");
25     _to.transfer(address(this).balance);
26 }
27
28 function destroySmartContract(address payable _to) public {
29     require(msg.sender == owner, "You are not the owner");
30     selfdestruct(_to);
31 }
32 }
33 }
```

Below the code, there are two transaction logs:

- Log 1: A pending transaction to "sendMoney" with a value of 1000000000000000000 wei, from address 0xAb8...35cb2 to the contract at 0x84. The status is marked with a green checkmark.
- Log 2: A pending transaction to "destroySmartContract" with a value of 0 wei, from address 0xAb8...35cb2 to the contract at 0x84. The status is marked with a green checkmark.

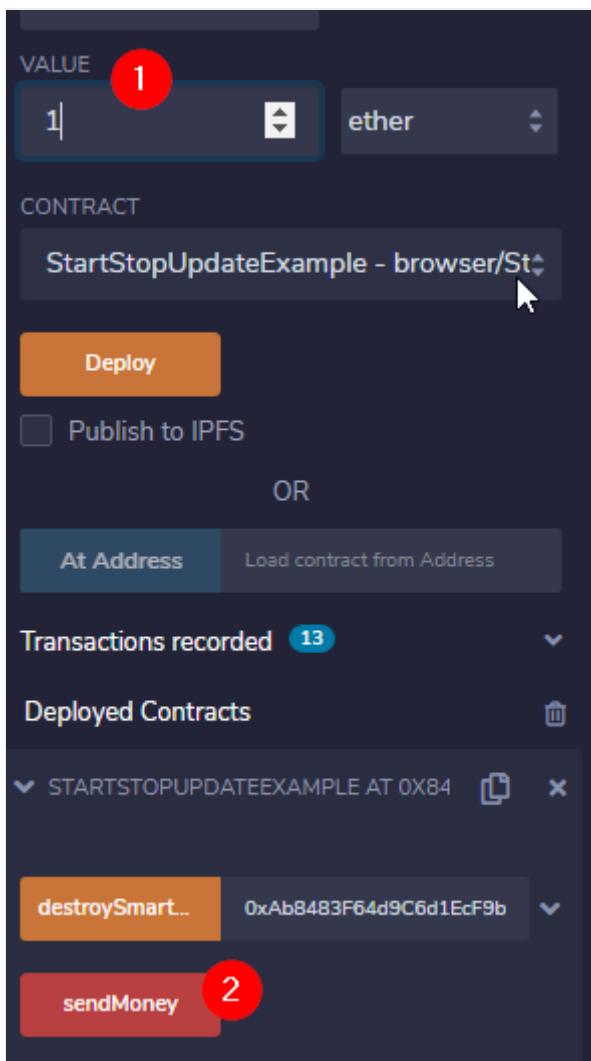
9.5.3 Interacting with Destroyed Smart Contracts

Now comes the part that is puzzling a lot of newcomers to Ethereum.

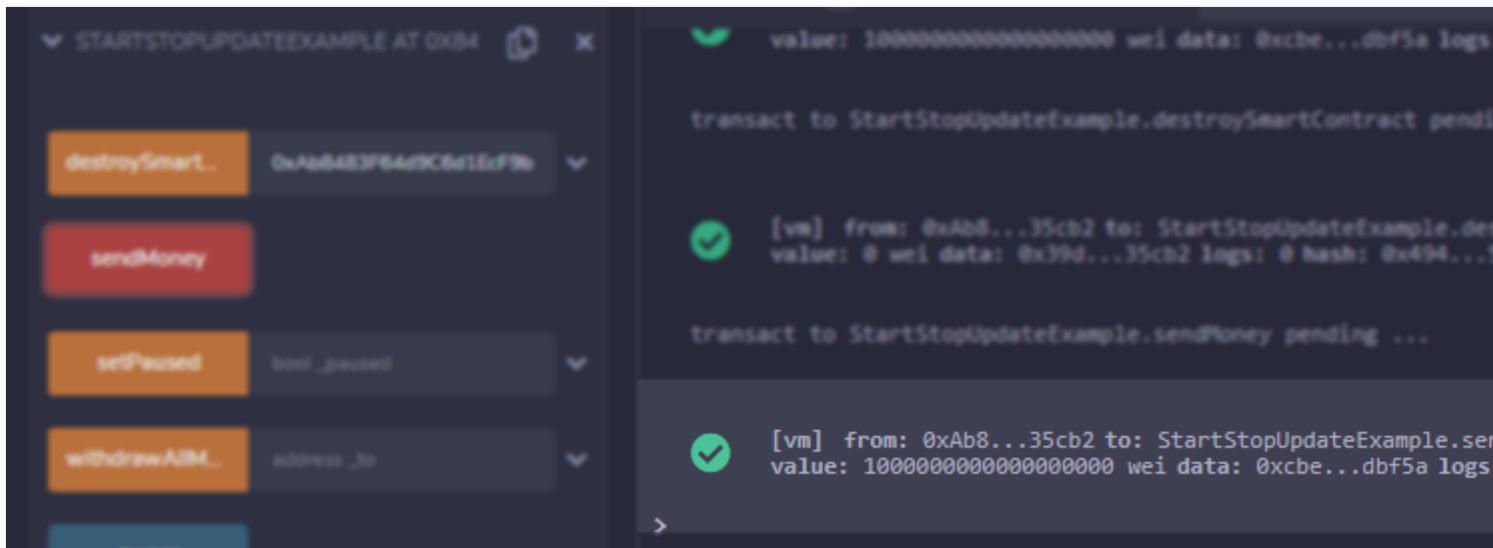
What happens if you try to send Ether again using the "sendMoney" function? Will there be an error or not?

There won't be an error! Internally you are sending Ether to an address. Nothing more.

Try it! Use the same contract instance, don't redeploy. Just enter 1 Ether and hit the `sendMoney` button.



and see the log console:



Works perfectly fine!

But try to get your Ether out again!

Also this transaction goes through smoothly! But wait!

Check your Balance!

It isn't updated...

0xAb8...35cb2 (98.999999999999981649 ether)			18	require paused
0xAb8...35cb2 (98.99999999999998998795 ether)			19	
0xAb8...35cb2 (98.99999999999998998795 ether)			20	
0x4B2...C02db (100.9999999999999976101 ether)				function wi
0x787...cabab (100 ether)				require
0x617...5E7f2 (100 ether)				require _to tra

You locked one Ether at the Smart Contract address for good. There's no more code running on that Address that could send you the Funds back.

So, be careful with the Contract life-cycle!

9.5.4 Re-Deployment of Smart Contracts to the Same Address

Once scenario, which is not in the course videos, is in-place upgrades. Since the [CREATE2](#) Op-Code was introduced, you can pre-compute a contract address.

Without CREATE2, a contract gets deployed to an address that is computed based on your address + your nonce. That way it was guaranteed that a Smart Contract cannot be re-deployed to the same address.

With the CREATE2 op-code you can instruct the EVM to place your Smart Contract on a specific address. Then you could call selfdestruct(), thus remove the source code. Then re-deploy a different Smart Contract to the same address.

This comes with several implications: when you see that a Smart Contract includes a `selfdestruct()` then simply be careful. Those implications will become more and more apparent as you progress through the course, especially when we talk about the ERC20 Token allowance. At this stage it is too early to discuss them all, but if you want to read on about it, checkout [this article](#).

Alright, that's enough of a detour into advanced topics.

Last update: April 17, 2021

9.6 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

10. Simple Mappings Introduction

10.1 Lab: Mappings and Structs

In this lab you are going learn how mappings work.

Mappings are a little bit like arrays or hash maps. You can store key/value pairs.

10.1.1 What You Know At The End Of The Lab

You'll know how mappings work.

You'll learn a new way to create account-white lists.

10.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. About 30 Minutes of your precious time

10.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

10.1.4 Get Started

Let's get started!

Last update: April 23, 2021

10.2 Smart Contract

We have to start with something, so we're going to start with the simplest of all examples possible. A simple mapping.

Create a new file in Remix and put in the following content:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract SimpleMappingExample {
    mapping(uint => bool) public myMapping;
    mapping(address => bool) public myAddressMapping;

    function setValue(uint _index) public {
        myMapping[_index] = true;
    }

    function setMyAddressToTrue() public {
        myAddressMapping[msg.sender] = true;
    }
}
```

The screenshot shows the Remix IDE interface. On the left, the "SOLIDITY COMPILER" panel is open, displaying compiler settings: Compiler version 0.8.3+commit.8d00100c, Language Solidity, EVM Version compiler default, Auto compile checked, Enable optimization unchecked, and Hide warnings unchecked. A large blue button at the bottom of this panel says "Compile mapping.sol". Below the compiler panel, the "CONTRACT" section shows "SimpleMappingExample (mapping.sol)". Underneath are three buttons: "Publish on Swarm" (with a swarm icon), "Publish on Ipfs" (with an IPFS icon), and "Compilation Details". At the bottom of the panel are links for ABI and Bytecode. On the right side of the interface, the code editor displays the Solidity contract "mapping.sol". The code defines a contract named SimpleMappingExample with two mappings: myMapping (uint to bool) and myAddressMapping (address to bool). It includes functions setValue and setMyAddressToTrue. The Remix terminal at the bottom shows a welcome message and information about accessible libraries like web3, ethers.js, and swarmgw.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
contract SimpleMappingExample {
    mapping(uint => bool) public myMapping;
    mapping(address => bool) public myAddressMapping;
    function setValue(uint _index) public {
        myMapping[_index] = true;
    }
    function setMyAddressToTrue() public {
        myAddressMapping[msg.sender] = true;
    }
}
```

- Welcome to Remix 0.11.4 -

You can use this terminal to:

- Check transactions details and start debugging.
- Execute JavaScript scripts:
 - Input a script directly in the command line interface
 - Select a Javascript file in the file explorer in the command line interface
 - Right click on a JavaScript file in the file explorer

The following libraries are accessible:

- [web3 version 1.0.0](#)
- [ethers.js](#)
- [swarmgw](#)
- [remix \(run remix.help\(\) for more info\)](#)

10.2.1 Run the Smart Contract

Mappings are an interesting datatype in Solidity. They are accessed like arrays, but they have one major advantage: All key/value pairs are initialized with their default value.

If you have a look at the example Smart Contract, you'll see that we have two mappings.

One, that maps uint256 to booleans, that's called `myMapping`. Another one that maps addresses to booleans, that we called `myAddressMapping`.

We can access a mapping with the brackets []. If we want to access the key "123" in our `myMapping`, then we'd simply write `myMapping[123]`.

Our mappings here are public, so Solidity will automatically generate a getter-function for us. That means, if we deploy the Smart Contract, we will automatically have a function that looks technically like this:

```
function myMapping(uint index) returns (bool) {  
    return myMapping[index];  
}
```

We don't need to explicitly write the function. Also not for `myAddressMapping`. Since both are public variables, Solidity will add these auto_magic_ally.

Let's run the Smart Contract and see what happens!

Head over to the "Deploy and Run Transactions" Plugin and run the Smart Contract:

Remix - Ethereum IDE

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0x5B3...eddC4 (99.9999999)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
SimpleMappingExample - mapping.sol

Deploy 2

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

SIMPLEMAPPINGEXAMPLE AT 0xD91...

setMyAddress...

setValue uint256 _index

myAddressMa... address

myMapping uint256

Low level interactions

CALL DATA

mapping.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract SimpleMappingExample {
    mapping(uint => bool) public myMap;
    mapping(address => bool) public myAddressMapping;

    function setValue(uint _index) public {
        myMap[_index] = true;
    }

    function setMyAddressToTrue() public {
        myAddressMapping[msg.sender] = true;
    }
}
```

0 listen on network

Select a Javascript file in the file explorer interface
Right click on a JavaScript file in the file explorer interface

The following libraries are accessible:

- [web3 version 1.0.0](#)
- [ethers.js](#)
- [swarmgw](#)
- [remix \(run remix.help\(\) for more info\)](#)

creation of SimpleMappingExample pending...

[vm] from: 0x5B3...eddC4 to: SimpleMappingExample at 0xD91...a1982
hash: 0xd80...a1982

Copyright © 2016 - 2021 Thomas Wiesner

Perfect, let's read and write to the uint => bool mapping `myMapping` first.

Last update: April 23, 2021

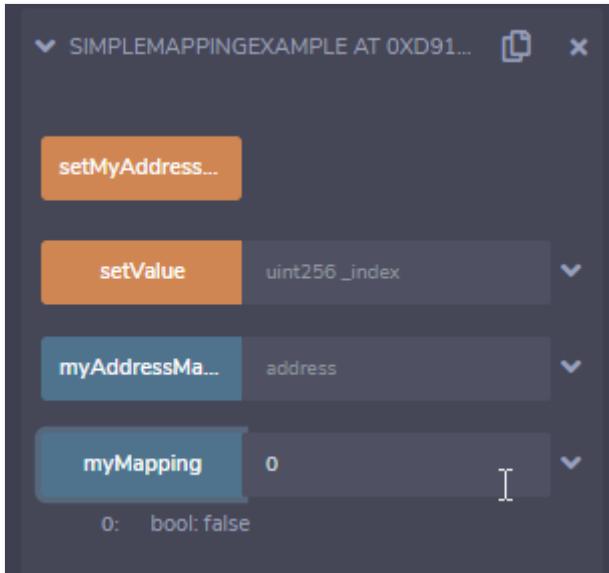
10.3 Access Mapping Variables

Mappings are accessed like arrays, but there are no index-out-of-bounds-exceptions. If you don't know what that means, don't worry, it was a joke for Java Developers.

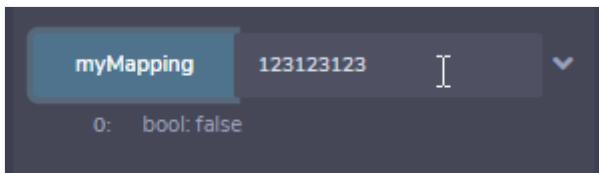
All possible key/value pairs are already initialized. You can simply access any key and get back "false", since that's the default boolean value.

Give it a try?

Enter "0" next to "myMapping" and hit the button. It will return false.



The same happens with "1", "2", or "123123123123123". Any index will return false, because we didn't write a value there yet.

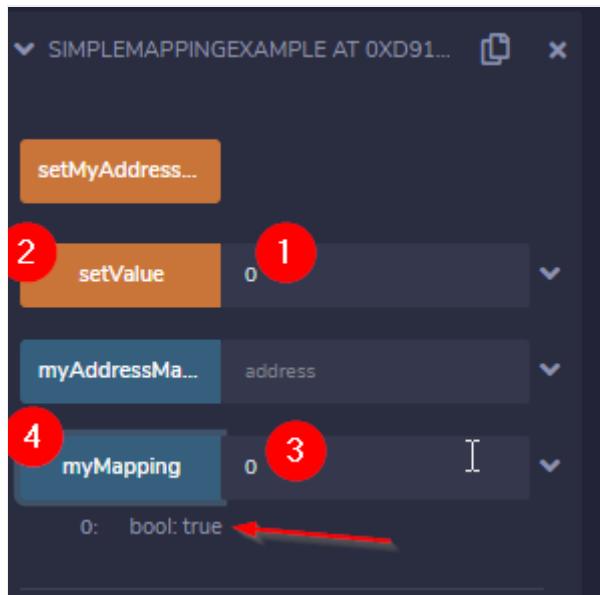


10.3.1 Write to a Mapping

If you have a look at our simple function `setValue(uint _index)`, then you will see you will write "true" to an `_index`. So, if the `_index = 0`, then `myMapping[0]` will be set to true. Again, no need to initialize anything here, it will just assign the value.

Give it a try!

1. Set the value of `myMapping[0]` to true, by entering "0" next to the "setValue" button
2. Hit the `setValue` button
3. retrieve the `myMapping[0]` value again, by entering "0" next to the "myMapping" button
4. It should return true



Last update: April 17, 2021

10.4 Address Keys in Mappings

If you come from the traditional Development world, then integer keys are nothing particularly interesting probably. It's very much like using an array or a hash map or something similar.

Internal Storage of Mappings

Here's a little advanced detour to how mappings and arrays are stored internally in the EVM.

Array data is located starting at keccak256(p) and it is laid out in the same way as statically-sized array data would: One element after the other, potentially sharing storage slots if the elements are not longer than 16 bytes. Dynamic arrays of dynamic arrays apply this rule recursively.

The value corresponding to a mapping key k is located at keccak256(h(k) . p) where . is concatenation and h is a function that is applied to the key depending on its type:

1. for value types, h pads the value to 32 bytes in the same way as when storing the value in memory.
2. for strings and byte arrays, h computes the keccak256 hash of the unpadded data.

Find more information here on the Solidity page: https://docs.soliditylang.org/en/v0.8.3/internals/layout_in_storage.html?highlight=storage#mappings-and-dynamic-arrays

Addresses are a cool thing in Solidity. They are like a bank account number, an IBAN if you wish. You know who transacts with your Smart Contract and the Smart Contract knows who you are.

The cool thing is, addresses can be keys for arrays and mappings. And in our example we map addresses to boolean values. We could use this for white-listing for example. So, if an address is allowed to do a certain action in our Smart Contract then we can white-list it.

Let's see how that behaves. First, let's check the value for your own address:

1. Copy the Address from the Dropdown - there is a little copy icon next to the dropdown, press it
2. Paste the Address into the input field next to "myAddressMapping"
3. Press the Button
4. The value should be "false"

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM

ACCOUNT

0x5B3...eddC4 (99.9999999)

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

SimpleMappingExample - mapping.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 2

Deployed Contracts

SIMPLEMAPPINGEXAMPLE AT 0xD91...

setMyAddress...

setValue uint256 _index

myAddressMa... 0x5B38Da6a701c568545dC

0: bool: false

myMapping uint256

0: bool: true

1

2

3

4

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract SimpleMappingExample {

mapping(uint => bool) public myMap;

mapping(address => bool) public myAddressMapping;

function setValue(uint _index) public {

myMapping[_index] = true;

}

function setMyAddressToTrue() public {

myAddressMapping[msg.sender] = true;

}

}

0

listen on network

[vm] from: 0x5B3...eddC4 to: SimpleMappingExample at 0xD91... logs: 0 hash: 0xd15...98b6f

call to SimpleMappingExample.myMapping

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875 data: 0x7c6...00000

call to SimpleMappingExample.myAddressMapping

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875 data: 0xb92...eddC4

10.4.1 Set Address Keys in a Mapping

Let's have a closer look at the function `setMyAddressToTrue`:

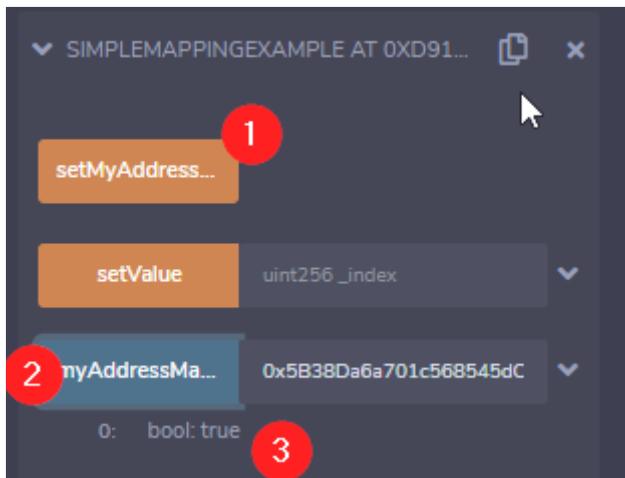
```
function setMyAddressToTrue() public {
    myAddressMapping[msg.sender] = true;
}
```

This function does several things:

1. it accesses the global `msg`-object and gets the senders address. So, if you are interacting with a specific address, then for the Smart Contract that address will be in `msg.sender`
2. It accesses the `myAddressMapping` mapping and sets the value "true" for the current address.

Let's give it a try!

1. simply click the "setMyAddressToTrue" button. There is no input field, because the address will automatically be the one you use to interact with the Smart Contract
2. Retrieve the value again
3. It should be true



10.4.2 Mappings of mappings

Because of the way mappings are stored within the EVM, it's possible to create mappings of mappings as well. For example:

```
mapping (uint => mapping(uint => bool)) uintUintBoolMapping;
```

Let's try a little coding exercise to get a feeling for mappings yourself!

Code Exercise

Try to use the mapping above, which is not public, and create both a getter and a setter function for it.

In case you get stuck, you can see the solution below.

Solution Code Exercise

```
mapping(uint => mapping(uint => bool)) uintUintBoolMapping;

function setUintUintBoolMapping(uint _index1, uint _index2, bool _value) public {
    uintUintBoolMapping[_index1][_index2] = _value;
}

function getUintUintBoolMapping(uint _index1, uint _index2) public view returns (bool) {
    return uintUintBoolMapping[_index1][_index2];
}
```

You might have seen, you can't name the getter function with the same name as the variable. But if you are making the variable public, then Solidity automatically creates a getter function with the same name as the variable. Peculiar detail to keep in mind here.

Alright, that's it for this lab. Congratulations, you created your first mapping and know how to access mappings.

DEPLOY & RUN TRANSACTIONS

VALUE
0 wei

CONTRACT
SimpleMappingExample - mapping.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 5

Deployed Contracts

SIMPLEMAPPINGEXAMPLE AT 0xD7A...  

- setMyAddress...**
- setUintUIntBo...** 0,0,true
- setValue** uint256 _index
- getUintUIntBo...** 0,0
 - 0: bool: true
- myAddressMa...** address
- myMapping** uint256

Low level interactions 

CALldata 

Home mapping.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract SimpleMappingExample {
    mapping(uint => bool) public myMapping;
    mapping(address => bool) public myAddressMapping;
    mapping(uint => mapping(uint => bool)) public uintUintBoolMapping;

    function setUintUIntBoolMapping(uint _index1, uint _index2, bool _value) public {
        uintUintBoolMapping[_index1][_index2] = _value;
    }

    function getUintUIntBoolMapping(uint _index1, uint _index2) public view returns(bool) {
        return uintUintBoolMapping[_index1][_index2];
    }

    function setValue(uint _index) public {
        myMapping[_index] = true;
    }

    function setMyAddressToTrue() public {
        myAddressMapping[msg.sender] = true;
    }
}
```

0 listen on network  Search with

[vm] from: 0x5B3...eddC4 to: SimpleMappingExample
hash: 0x190...2d4ce

transact to SimpleMappingExample.setUintUIntBoolMapping

[vm] from: 0x5B3...eddC4 to: SimpleMappingExample
value: 0 wei data: 0xbc6...00001 logs: 0 hash: 0x

call to SimpleMappingExample.getUintUIntBoolMapping

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875 to: SimpleMappingExample.getUintUIntBoolMapping

Last update: April 17, 2021

11. Mappings and Structs

11.1 Mappings and Structs

So far we have only used mappings and simple data types. So, we can now map addresses to booleans, or uint to addresses or any combination of that.

But what if you want to define something a bit more complex?

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.4;
4
5 contract MappingsStructExample {
6
7     struct Payment {
8         uint amount;
9         uint timestamp;
10    }
11
12     struct Balance {
13         uint totalBalance;
14         uint numPayments;
15         mapping(uint => Payment) payments;
16    }
17
18     mapping(address => Balance) public balanceReceived;
19
20
21     function getBalance() public view returns(uint) {
22         return address(this).balance;
23     }
24
25     function sendMoney() public payable {
26         balanceReceived[msg.sender].totalBalance += msg.value;
27
28         Payment memory payment = Payment(msg.value, block.timestamp);

```

Let's say you work on the latest DeFi Project and your project does deposits and withdrawals. You not only want to save the amount that was deposited, but also by whom and possibly some more information.

You could define several mappings for this using simple data types. But there is also another way: Structs.

With structs you can define your own Datatype.

11.1.1 What You Know At The End Of The Lab

How you can define your own datatypes using structs

How to combine structs and mappings to get the most out of your Smart Contract functionality

-
- 11.1.2 Prerequisites - You need:
1. Chrome or Firefox browser.
 2. An Internet connection
 3. About 60 Minutes of your precious time

11.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

11.1.4 Get Started

Let's get started!

Updated to Solidity 0.8

The following Smart Contracts have been updated to Solidity 0.8 and might deviate slightly from the Videos. As always, all and everything will be explained so you don't need to worry to miss anything.

Last update: April 23, 2021

11.2 Basic Smart Contract

Let's create a Smart Contract we start with again. It's slightly different than in the previous Lab, because I actually want to do something with it. By the end of the lab we have some sort of Payment Smart Contract can keep track of deposits and withdrawals in an easy way using mappings and structs.

Create the following file in Remix:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract MappingsStructExample {
    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
    function sendMoney() public payable {
    }
    function withdrawAllMoney(address payable _to) public {
        _to.transfer(address(this).balance);
    }
}
```

The Smart Contract is very simplistic. It allows everyone to send Ether to the Smart Contract. But it also allows everyone to transfer all funds from the Smart Contract to any address. Not very secure - yet.

SOLIDITY COMPILER

COMPILER 

0.8.4+commit.c7e474f2 

Include nightly builds

LANGUAGE 

Solidity 

EVM VERSION 

compiler default 

COMPILER CONFIGURATION

Auto compile

Enable optimization 200 

Hide warnings

Compile mappingstructs.sol 

Home  mapping.sol  mappingstructs.sol 

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.3;
4
5 contract MappingsStructExample {
6
7     function getBalance() public view
8         return address(this).balance;
9     }
10
11
12     function sendMoney() public payable
13         }
14
15     function withdrawAllMoney(address _to) {
16         _to.transfer(address(this).balance);
17     }
18 }
```

CONTRACT 

MappingsStructExample (mappingstructs.sol) 

Publish on Swarm 

Publish on Ipfs 

Compilation Details

 ABI  Bytecode



- Welcome to Remix 0.11.4 -

You can use this terminal to:

- Check transactions details and start debugging.
- Execute JavaScript scripts:
 - Input a script directly in the command line
 - Select a Javascript file in the file explorer
 - Right click on a JavaScript file in the file explorer

The following libraries are accessible:

- [web3 version 1.0.0](#)
- [ethers.js](#)
- [swarmgw](#)
- [remix](#) (run remix.help() for more info)

Let's play around with this Smart Contract and see how it works with depositing and withdrawing money, and how we can make it more secure.

Last update: April 23, 2021

11.3 Understand the Limitations

In this step we basically execute the Smart Contract and see how it behaves, before we improve the Smart Contract with mapping and structs.

11.3.1 Deploy the Smart Contract

Head over to the "Deploy & Run Transactions" Plugin and Deploy the Smart Contract

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.3;
4
5 contract MappingsStructExample {
6
7     function getBalance() public view
8         return address(this).balance;
9     }
10
11     function sendMoney() public payable
12     {
13     }
14
15     function withdrawAllMoney(address _to) public
16         _to.transfer(address(this).balance);
17     }
18 }
```

0 listen on network Search with

- Input a script directly in the command line
- Select a Javascript file in the file explorer
- Right click on a JavaScript file in the file explorer

The following libraries are accessible:

- [web3 version 1.0.0](#)
- [ethers.js](#)
- [swarmgw](#)
- [remix \(run remix.help\(\) for more info\)](#)

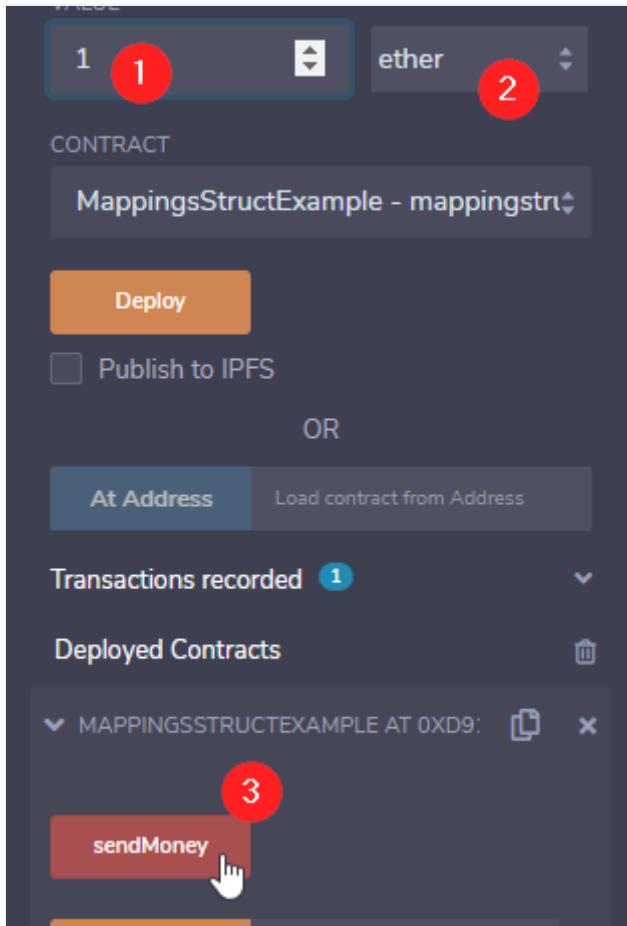
creation of MappingsStructExample pending...

Now you should have your Smart Contract deployed and an instance under "Deployed Contracts"

11.3.2 Deposit some Funds

Let's send some Ether to the Smart Contract. The Contract will manage its own funds.

1. Enter "1" in the value field
2. Select "Ether" from the Dropdown
3. Hit the "sendMoney" Button



11.3.3 Withdraw Money

Our Smart Contract is extremely simple at this stage. It allows everyone to send money to the Smart Contract. It also allows everyone to withdraw everything. That's not very good obviously, unless you operate some sort of charity. Let's give it a try. Let's withdraw all money to our second account using our second account:

1. Select Account#2 from the Accounts Dropdown
2. Copy the Address
3. Paste the Address next to "withdrawAllMoney"
4. hit the "withdrawAllMoney" Button
5. Observe the increased Ether Amount for that Account

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (98.999999999999798581 ether) (1)

- 0x5B3...eddC4 (98.999999999999798581 ether)
- 0xAB8...35cb2 (100 ether) (highlighted)
- 0x4B2...C02db (100 ether)
- 0x787...cabaB (100 ether)
- 0x617...5E7f2 (100 ether)
- 0x17F...8c372 (100 ether)
- 0x5c6...21678 (100 ether)
- 0x03C...D1Ff7 (100 ether)
- 0x1aE...E454C (100 ether)
- 0x0A0...C70DC (100 ether)
- 0xCA3...a733c (100 ether)
- 0x147...C160C (100 ether)
- 0x4B0...4D2dB (100 ether)
- 0x583...40225 (100 ether)
- 0xdD8...92148 (100 ether)

At Address Load contract from Address

DEPLOY & RUN TRANSACTIONS

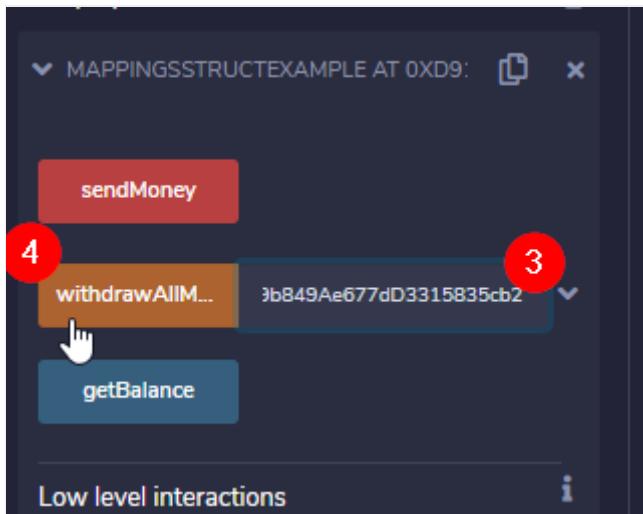
ENVIRONMENT: JavaScript VM

ACCOUNT: 0xAB8...35cb2 (100 ether) (2)

GAS LIMIT: 3000000

VALUE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract MappingExample {
    mapping(uint256 => string) public myMapping;
    function setMapping(uint256 key, string memory value) public {
        myMapping[key] = value;
    }
    function getMapping(uint256 key) public view returns (string memory) {
        return myMapping[key];
    }
}
```



ACCOUNT	BALANCE
0xAb8...35cb2 (100.999999999999781 ether)	100.999999999999781 ether
0x5B3...eddC4 (98.9999999999999798581 ether)	98.9999999999999798581 ether
0xAb8...35cb2 (100.99999999999996981 ether)	100.99999999999996981 ether
0x4B2...C02db (100 ether)	100 ether
0x787...cabaB (100 ether)	100 ether
0x617...5E7f2 (100 ether)	100 ether
0x17F...8c372 (100 ether)	100 ether
0x5c6...21678 (100 ether)	100 ether

So, everyone can do everything with the Smart Contract. Let's make it a bit more secure!

Try yourself first

Want to give it a try yourself first before you proceed?

Extend the Smart Contract and use a Mapping to track who send how much to the Smart Contract.

Then only allow withdrawals for the amount deposited.

So, if address 0x123... deposits 1 Ether, then address 0x123... can withdraw 1 Ether again. Nobody else. And not more.

Use a mapping(address => uint) for this and track the funds with msg.value.

Last update: April 23, 2021

11.4 Add a Mapping

Let's make our Smart Contract inherently secure. We will allow only withdrawals for previously done deposits. It's the first step towards understanding Token Contracts, which you will see later!

Add the following things to the Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract MappingsStructExample {
    mapping(address => uint) public balanceReceived;

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawAllMoney(address payable _to) public {
        uint balanceToSend = balanceReceived[msg.sender];
        balanceReceived[msg.sender] = 0;
        _to.transfer(balanceToSend);
    }
}
```

To understand what's going on here:

When someone sends money using the "sendMoney" function, we track the msg.value (amount in Wei) with the balanceReceived mapping for the person who interacted with the Smart Contract.

If that same person tries to withdraw money again using "withdrawAllMoney", we look in that mapping how much he sent there previously, then reset the mapping and send the amount.

Re-Entrancy and Checks-Effects-Interaction Pattern

You are eventually wondering why we don't do the following:

```
function withdrawAllMoney(address payable _to) public {
    _to.transfer(balanceReceived[msg.sender]);
    balanceReceived[msg.sender] = 0;
}
```

This follows the so-called Checks-Effects-Interaction pattern. As a rule of thumb: You interact with outside addresses last, no matter what. Unless you have a trusted source. So, first set your Variables to the state you want, as if someone could call back to the Smart Contract before you can execute the next line after .transfer(...). Read more about this here: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

11.4.1 Deploy the new Smart Contract

Head over to the "Deploy & Run Transactions" Plugin and deploy a new Instance.

DEPLOY & RUN TRANSACTIONS

0xAb8...35cb2 (100.99999€)

GAS LIMIT
3000000

VALUE
0 ether

CONTRACT
MappingsStructExample - mappingstructsexample.sol

OR

Load contract from Address

Transactions recorded 4

Deployed Contracts

- ▶ MAPPINGSSTRUCTEXAMPLE AT 0XD91
- ▼ MAPPINGSSTRUCTEXAMPLE AT 0X652

address _to

address

Low level interactions

CALldata

Home mapping.sol mappingstructsexample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract MappingsStructExample {
    mapping(address => uint) public balanceReceived;

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawAllMoney(address _to) public {
        uint balanceToSend = balanceReceived[msg.sender];
        balanceReceived[msg.sender] = 0;
        _to.transfer(balanceToSend);
    }
}
```

0 Search with

[vm] from: 0x5B3...eddC4 to: MappingsStructExample at 0xd91 logs: 0 hash: 0xfad...dd69e

transact to MappingsStructExample.withdrawAllMoney pending...

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x652 hash: 0xa38...27b75

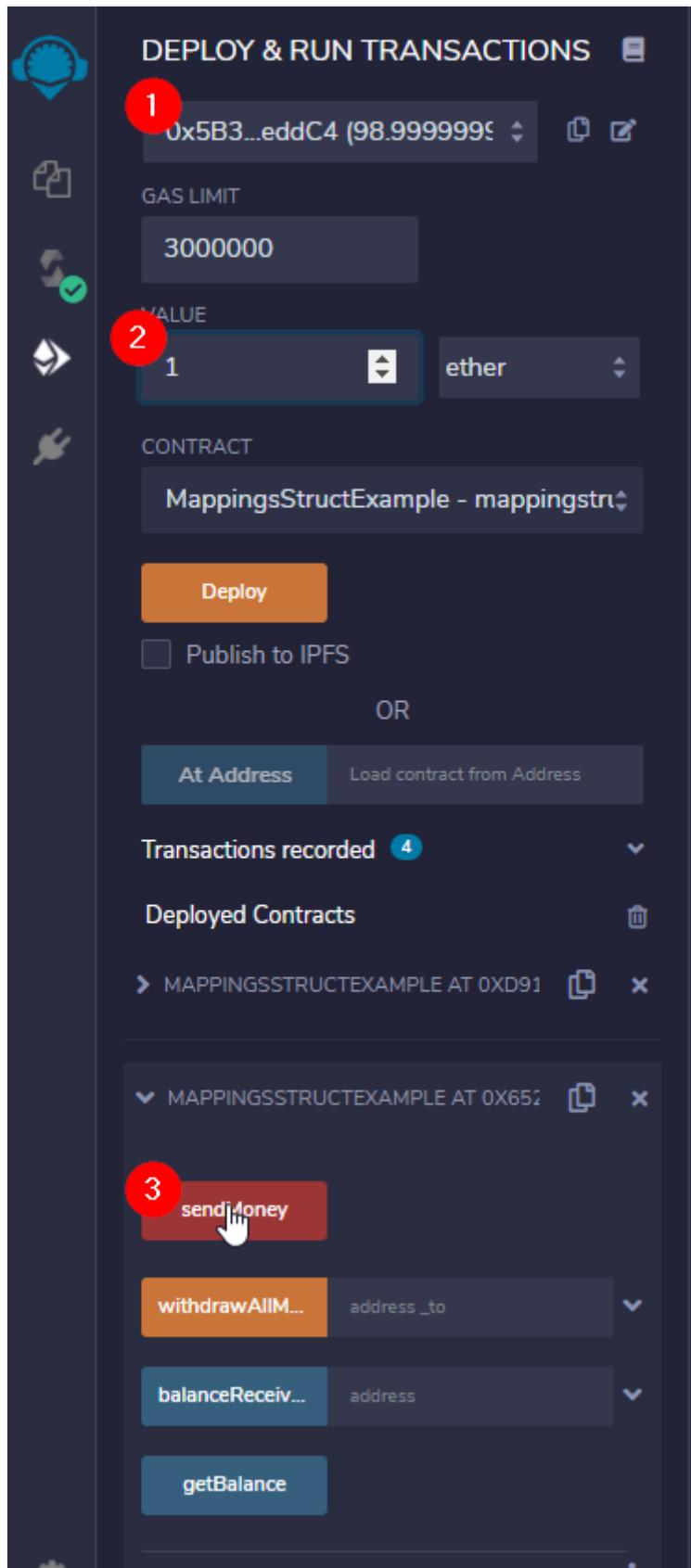
creation of MappingsStructExample pending...

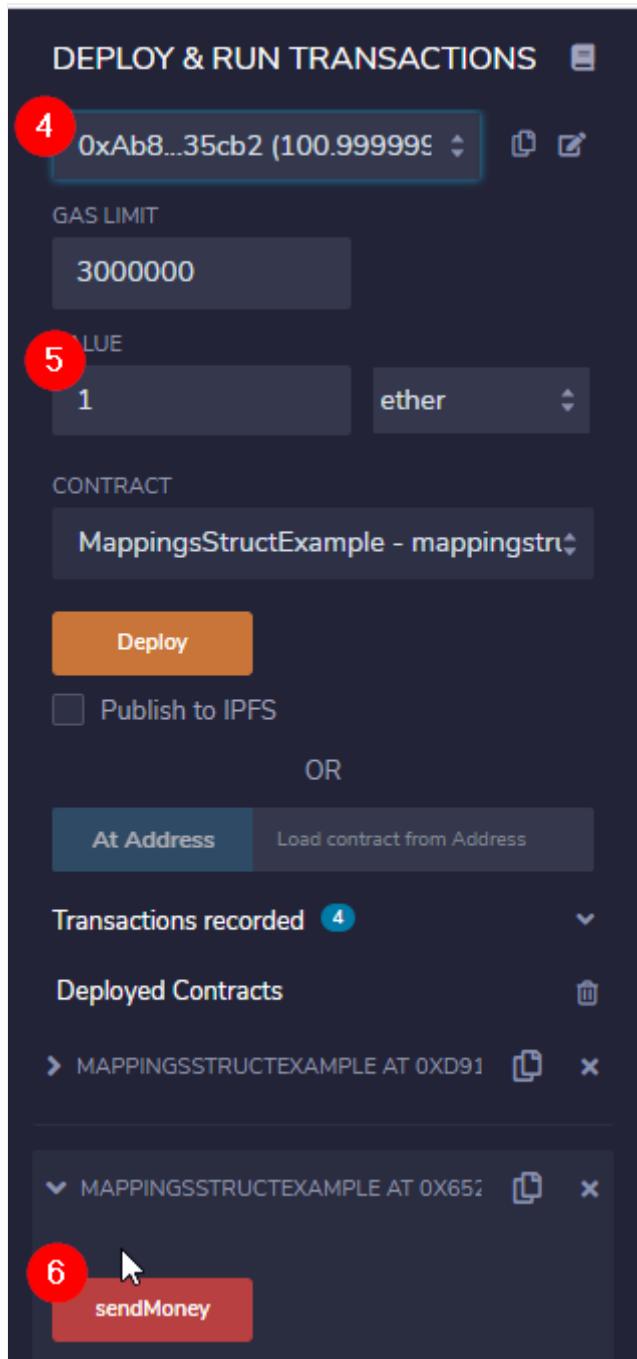
[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x652 hash: 0xa38...27b75

11.4.2 Deposit and Withdraw

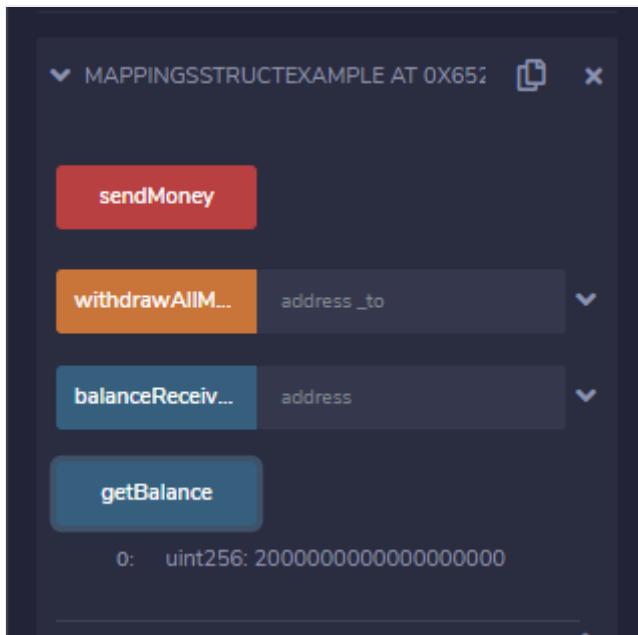
We will deposit 1 Ether from two different accounts and then withdraw Ether again:

1. select **Account#1** from the Accounts Dropdown
2. Value: 1 Ether
3. Hit the "sendMoney" button
4. Select **Account#2** from the Accounts Dropdown
5. Value: 1 Ether
6. Hit the "sendMoney" button





Now check the Balance of the Smart Contract:



You have 2 Ether ($2 * 10^{18}$ Wei) in the Smart Contract. But if you check the accounts individual amount from the mapping, then you see that each account can max withdraw 1 Ether:

DEPLOY & RUN TRANSACTIONS

0x5B3...eddC4 (97.99999999999999)

GAS LIMIT
3000000

VALUE
0 ether

CONTRACT
MappingsStructExample - mappingstruc...

OR

At Address

Transactions recorded 6

Deployed Contracts

› MAPPINGSSTRUCTEXAMPLE AT 0xD91

▼ MAPPINGSSTRUCTEXAMPLE AT 0X652

address _to

0: uint256: 10000000000000000000000000000000

0: uint256: 20000000000000000000000000000000

A sidebar on the left contains icons for account, file, settings, and help.

11.4.3 Withdraw all Money to Account#3

Let's withdraw all the funds stored in the Smart Contract to Account#3 from the Accounts Dropdown:

1. Copy the address of Account#3 (select Account#3, copy)
2. Paste the Address into the withdrawAllMoney input field, but don't hit the button yet
3. Go back to Account#1
4. Hit the "withdrawAllMoney" button with Account#3-Address in the input field
5. Select Account#2 from the Accounts Dropdown
6. Hit the "withdrawAllMoney" button again with Account#3-Address in the input field
7. Check the Balance of the Smart Contract and of Account#3

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there are several icons: a headphones icon, a file icon, a circular arrow icon with a checkmark, a diamond icon, and a wrench icon. The main area has a title 'DEPLOY & RUN TRANSACTIONS'. Below it, a dropdown menu titled 'Accounts' is open, showing a list of addresses with their ether balance. The address '0x4B2...C02db (100 ether)' is highlighted with a red circle labeled '1'. In the center, there's a section for 'Transactions recorded' with a count of 6. Below that, 'Deployed Contracts' are listed, with one named 'MAPPINGSSTRUCTEXAMPLE AT 0XD91'. At the bottom, a specific transaction is detailed: it has a red button labeled 'sendMoney' and an orange button labeled 'withdrawAllM...'. The input field for the withdrawAllMoney transaction contains the address '0x4B20993Bc481177ec7E8', which is highlighted with a red circle labeled '2'.

The screenshot shows the MetaMask extension interface. On the left is a sidebar with various icons: a blue gear, a document, a circular arrow with a checkmark, a diamond, and a hand. The main area has a dark background.

DEPLOY & RUN TRANSACTIONS

A dropdown menu is open, showing a list of addresses and their ether balances:

- 0x5B3...eddC4 (97.9999999999999734436 eth)
- 0xA8...35cb2 (99.9999999999999601739 eth)
- 0x4B2...C02db (101 ether)
- 0x787...cabAB (100 ether)
- 0x617...5E7f2 (100 ether)
- 0x17F...8c372 (100 ether)
- 0x5c6...21678 (100 ether)
- 0x03C...D1Ff7 (100 ether)
- 0x1aE...E454C (100 ether)
- 0x0A0...C70DC (100 ether)
- 0xCA3...a733c (100 ether)
- 0x147...C160C (100 ether)
- 0x4B0...4D2dB (100 ether)
- 0x583...40225 (100 ether)
- 0xD8...92148 (100 ether)

Below this, there are two buttons: "At Address" and "Load contract from Address".

Transactions recorded (7)

Deployed Contracts

- MAPPINGSSTRUCTEXAMPLE AT 0xD91
- MAPPINGSSTRUCTEXAMPLE AT 0x652

For the contract at 0xD91, there is a red button labeled "sendMoney".

For the contract at 0x652, there is an orange button labeled "withdrawAllM...".

A red circle with the number "3" is placed over the first item in the transaction history list. A red circle with the number "4" is placed over the orange "withdrawAllM..." button.

The screenshot shows the MetaMask extension interface. On the left, there's a sidebar with five icons: a gear (5), a document (6), a checkmark (7), a double arrow (8), and a plug (9). The main area has a title "DEPLOY & RUN TRANSACTIONS". Below it, a list of deployed contracts is shown, each with a copy icon and a delete icon. A red circle with the number 5 highlights the first contract entry. A red circle with the number 7 highlights the second entry. A red circle with the number 8 highlights the "Transactions recorded" section. At the bottom, there are two buttons: "sendMoney" (red) and "withdrawAllM..." (orange), followed by a recipient address "0x4B20993Bc481177ec7E8".

The screenshot shows the Truffle UI interface. On the left is a sidebar with icons for account, file, settings, and help. The main area has a title "DEPLOY & RUN TRANSACTIONS". A list of transactions is shown, with the first one highlighted in blue and circled in red with the number 5. Another transaction in the list is circled in red with the number 7. Below the list are buttons for "At Address" and "Load contract from Address". Under "Transactions recorded", there is a dropdown menu with the number 8. Under "Deployed Contracts", there are two entries: "MAPPINGSSTRUCTEXAMPLE AT 0xD91" and "MAPPINGSSTRUCTEXAMPLE AT 0x652". Each entry has a "sendMoney" button and a "withdrawAll..." button. The "withdrawAll..." button for the first entry is circled in red with the number 6.

Withdrawing all money is fun, but not very useful yet. Let's add another functionality to withdraw partial funds.

Try yourself first

Want to give it a try yourself first before you proceed?

Extend the Smart Contract and use the Mapping allow partial sending. That means, the user can specify an amount to send. The Smart Contract checks if the amount isn't larger than what the user previously deposited, deducts the amount from the users balance and sends it.

Last update: April 23, 2021

11.5 Add partial Withdrawals

Sending all funds is fun, but it isn't very useful. Sometimes, like with a Bank Account, you don't want to send out all the funds you have. You just want to send a little bit. We can do this quite easily with our new mapping.

Add the following things to the Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

contract MappingsStructExample {
    mapping(address => uint) public balanceReceived;

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds");
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }

    function withdrawAllMoney(address payable _to) public {
        uint balanceToSend = balanceReceived[msg.sender];
        balanceReceived[msg.sender] = 0;
        _to.transfer(balanceToSend);
    }
}
```

To understand what's going on here:

When someone withdraws funds, we check if the amount he wants to withdraw is smaller or equal than the amount he previously deposited. If yes, then we deduct the amount from his balance and send out the funds.

11.5.1 Deploy the new Smart Contract

Head over to the "Deploy & Run Transactions" Plugin and deploy a new Instance.



DEPLOY & RUN TRANSACTIONS

ACCOUNT

GAS LIMIT

VALUE

CONTRACT

Publish to IPFS

OR

At Address

Transactions recorded 9

Deployed Contracts

MAPPINGSSTRUCTEXAMPLE AT 0xA8...

-
- address _to
- address _to, uint256 _amount
- address
-

Low level interactions

Home

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract MappingsStructExample {
    mapping(address => uint) public balanceReceived;

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] -= _to.transfer(_amount);
    }

    function withdrawAllMoney(address payable _to) public {
        uint balanceToSend = balanceReceived[msg.sender];
        balanceReceived[msg.sender] = 0;
        _to.transfer(balanceToSend);
    }
}
```

0 listen on network

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x206...898c2 hash: 0x206...898c2

call to MappingsStructExample.getBalance

CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3 creation of MappingsStructExample pending...

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x206...898c2 hash: 0x206...898c2

11.5.2 Deposit and Withdraw 50%

We will deposit 1 Ether from Account#1 and Withdraw 50% to Account#3. I won't provide Screenshots for the first few steps, since it's exactly the same as previously:

1. select **Account#1** from the Accounts Dropdown
2. Value: 1 Ether
3. Hit the "sendMoney" button
4. Select **Account#3** from the Accounts Dropdown
5. Copy the Address And Paste it in the "withdrawMoney" Input field
6. Add "50000000000000000000" as amount
7. Switch back to Account#1 and hit the "withdrawMoney" button.
8. Check the Balance of "Account#3"

DEPLOY & RUN TRANSACTIONS

ACCOUNT

- 0xAb8...35cb2 (98.99999999999999 ether)
- 0xAb8...35cb2 (98.99999999999999011477 ether)
- 0x4B2...C02db (102.5 ether)**
- 0x787...cabab (100 ether)
- 0x617...5E7f2 (100 ether)
- 0x17F...8c372 (100 ether)
- 0x5c6...21678 (100 ether)
- 0x03C...D1Ff7 (100 ether)
- 0x1aE...E454C (100 ether)
- 0x0A0...C70DC (100 ether)
- 0xCA3...a733c (100 ether)
- 0x147...C160C (100 ether)
- 0x4B0...4D2dB (100 ether)
- 0x583...40225 (100 ether)
- 0xdD8...92148 (100 ether)

At Address

Transactions recorded 11

Deployed Contracts

MAPPINGSSTRUCTEXAMPLE AT 0xA8:

sendMoney

withdrawAllM... address _to

withdrawMoney

_to: 0x4B20993Bc481177ec7E8f57
 _amount: 50000000000000000000000000000000

Home mapping.sol mappingstructs.sol

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.3;
4
5 contract MappingsStructExample {
6
7     mapping(address => uint) public balanceReceived;
8
9     function getBalance() public view
10        return address(this).balance;
11    }
12
13    function sendMoney() public payable
14        balanceReceived[msg.sender] += msg.value;
15    }
16
17    function withdrawMoney(address payable _to)
18        require(_amount <= balanceReceived[_to]);
19        balanceReceived[_to] -= _amount;
20    }
21
22    function withdrawAllMoney(address payable _to)
23        uint balanceToSend = balanceReceived[_to];
24        balanceReceived[_to] = 0;
25        _to.transfer(balanceToSend);
26    }
27
28 }
```

0 listen on network

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0xA8 logs: 0 hash: 0xcae...17837

call to MappingsStructExample.getBalance

CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3 transact to MappingsStructExample.withdrawMoney pending

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0xA8 logs: 0 hash: 0x31b...290d3

If you are wondering why my input fields look like in the picture: There is a little down-arrow next to the input fields, so it will open an extended view.

If you followed along in this lab, then you have 102.5 Ether ($102.5 * 10^{18}$ Wei) in the Account#3. If you just started with this step, then you have 100.5 Ether in Account#3.

Now, that's all good so far. Let's add another level of complexity to it by using Structs. We define our own Datatypes so we can track single payments from users.

Last update: April 23, 2021

11.6 Add Structs

So far we relied on simple Datatypes like addresses, uint or in previous labs also Booleans. But what if that's not enough? Maybe you want to track the timestamp when a deposit happened. You want to track every single deposit from every user. You want to track how many deposits happened, and many more details.

Of course, a rule of thumb is: Do only the most necessary functions on the blockchain, and everything else off-chain. But for sake of explaining Structs, we will track every single payment in the greatest detail possible with our Smart Contract.

If you have never worked with structs, objects or classes before, then think about it like this:

While you are tracking the current balance with a mapping that maps address to uint (`mapping(address=>uint) balanceReceived`) with something schematically like `balanceReceived[THE-ADDRESS] = THE-UINT`, with a struct you would access the children with a `.` (period).

Let's say you have a struct

```
MyStruct {
    uint abc;
}

mapping(address => MyStruct) someMapping;
```

Then you would write to the mapping like this `someMapping[THE-ADDRESS].abc = THE-UINT`. Why is this useful? Let's have a look into our Smart Contract!

Add the following things to the Smart Contract:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

contract MappingsStructExample {

    struct Payment {
        uint amount;
        uint timestamp;
    }

    struct Balance {
        uint totalBalance;
        uint numPayments;
        mapping(uint => Payment) payments;
    }

    mapping(address => Balance) public balanceReceived;

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender].totalBalance += msg.value;

        Payment memory payment = Payment(msg.value, block.timestamp);
        balanceReceived[msg.sender].payments[balanceReceived[msg.sender].numPayments] = payment;
        balanceReceived[msg.sender].numPayments++;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender].totalBalance, "not enough funds");
        balanceReceived[msg.sender].totalBalance -= _amount;
        _to.transfer(_amount);
    }

    function withdrawAllMoney(address payable _to) public {
        uint balanceToSend = balanceReceived[msg.sender].totalBalance;
        balanceReceived[msg.sender].totalBalance = 0;
        _to.transfer(balanceToSend);
    }
}
```

It's getting quite extensive here, bear with me. Let's go through the Smart Contract together to understand what's going on here!

We have one struct called `Payment`, which stores the amount and the timestamp of the payment. Then we have another struct called `Balance` which stores the total balance and a mapping of all payments done.

Mapping has no Length

Mappings have no length. It's important to understand this. Arrays have a length, but, because how mappings are stored internally, they do not have a length.

Let's say you have a mapping `mapping(uint => uint) myMapping`, then all elements `myMapping[0]`, `myMapping[1]`, `myMapping[123123]`, ... are already initialized with the default value. If you map uint to uint, then you map key-type "uint" to value-type "uint".

Structs are initialized with their default value

Similar to anything else in Solidity, structs are initialized with their default value as well.

If you have a struct

```
struct Payment {
    uint amount;
    uint timestamp;
}
```

and you have a mapping `mapping(uint => Payment) myMapping`, then you can access already all possible uint keys with the default values. This would produce **no** error:

`myMapping[0].amount`, or `myMapping[123123].amount`, or `myMapping[5555].timestamp`.

Similar, you can set any value for any mapping key:

`myMapping[1].amount = 123` is perfectly fine.

So, with these two things in mind, structs allow you to define something similar like cheap class-variables.

Back to our Smart Contract.

11.6.1 Balance <-> Payment relationship

If you have a look at the two structs, then you see there is also a mapping inside:

```
struct Payment {
    uint amount;
    uint timestamp;
}

struct Balance {
    uint totalBalance;
    uint numPayments;
    mapping(uint => Payment) payments;
}

mapping(address => Balance) public balanceReceived;
```

Because mappings have no length, we can't do something like `balanceReceived.length` or `payments.length`. It's technically not possible. In order to store the length of the payments mapping, we have an additional helper variable `numPayments`.

So, if you want to the first payment for address 0x123... you could address it like this:

`balanceReceived[0x123...].payments[0].amount =` But that would mean we have static keys for the payments mapping inside the Balance struct. We actually store the keys in `numPayments`, that would mean, the current payment is in `balanceReceived[0x123...].numPayments`. If we put this together, we can do

`balanceReceived[0x123...].payments[balanceReceived[0x123...].numPayments].amount =`

Enough talking, let's give it a try!

11.6.2 Use the Smart Contract

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0xAb8...35cb2 (98.99999999999999)

GAS LIMIT
3000000

VALUE
0 ether

CONTRACT
MappingsStructExample - mappingstructs.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 12

Deployed Contracts

- MAPPINGSSTRUCTEXAMPLE AT 0X843...

sendMoney

withdrawAllM... address _to

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

Home mapping.sol mappingstructs.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

contract MappingsStructExample {
    struct Payment {
        uint amount;
        uint timestamp;
    }

    struct Balance {
        uint totalBalance;
        uint numPayments;
        mapping(uint => Payment) payments;
    }

    mapping(address => Balance) public balances;

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }

    function sendMoney() public payable {
        balanceReceived[msg.sender].totalBalance += msg.value;
        balanceReceived[msg.sender].numPayments++;
        balanceReceived[msg.sender].payments[msg.value] = Payment(msg.value, block.timestamp);
    }

    Payment memory payment = Payment(
        msg.value,
        block.timestamp
    );
}
```

0 listen on network Search with t

[call] from: 0xAb843F64d9C6d1EcF9b849Ae677dD3...

transact to MappingsStructExample.withdrawMoney pending...

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x843... logs: 0 hash: 0x31b...290d3 creation of MappingsStructExample pending...

[vm] from: 0xAb8...35cb2 to: MappingsStructExample at 0x843... logs: 0 hash: 0x31b...290d3

1. Deploy a new Instance of the Smart Contract.
2. Then deposit 1 Ether with Account#1 (1 Ether -> sendMoney button)
3. Then check the Balance



DEPLOY & RUN TRANSACTIONS

VALUE
0 ether

CONTRACT
MappingsStructExample - mappingstruc...

Deploy

Publish to IPFS

OR

At Address

Transactions recorded (1)

Deployed Contracts

- MAPPINGSSTRUCTEXAMPLE AT 0x841:

sendMoney

withdrawAllM... address _to

withdrawMoney address _to, uint256 _amount

balanceReceiv... 0x5B38Da6a701c568545dC

0: uint256: totalBalance 10000000000000000000
1: uint256: numPayments 1

getBalance

Low level interactions

CALldata

Home

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract MappingsStructExample {
    struct Payment {
        uint amount;
        uint timestamp;
    }
    struct Balance {
        uint totalBalance;
        uint numPayments;
        mapping(uint => Payment) payments;
    }
    mapping(address => Balance) public balances;
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
    function sendMoney() public payable {
        balanceReceived[msg.sender].totalBalance += msg.value;
        balanceReceived[msg.sender].numPayments++;
        balanceReceived[msg.sender].payments[msg.value] = Payment({
            amount: msg.value,
            timestamp: block.timestamp
        });
    }
    function withdrawAllMoney() public {
        require(balances[msg.sender].totalBalance > 0);
        uint amount = balances[msg.sender].totalBalance;
        balances[msg.sender].totalBalance = 0;
        msg.sender.transfer(amount);
    }
    function withdrawMoney(address to, uint amount) public {
        require(balances[msg.sender].totalBalance >= amount);
        balances[msg.sender].totalBalance -= amount;
        to.transfer(amount);
    }
}
```

listen on network Search with ...

[vm] from: 0xAB8...35cb2 to: MappingsStructExample

transact to MappingsStructExample.sendMoney pending...

[vm] from: 0x583...edc4 to: MappingsStructExample

logs: 0 hash: 0xb71...5be86

call to MappingsStructExample.balanceReceived

[call] from: 0x5B38Da6a701c568545dCfcB83FcB875

What you see is that you can retrieve the Balance easily, but you can't automatically get the "sub-mapping" (the mapping in the struct) value. You would have to write an extra getter-function, or you just use that mapping internally in your Smart Contract.

Of course, this is just an exemplary Smart Contract, but you will see later how we use mapping of mappings later in the Supply Chain Example.

On to the next Lab!

Last update: April 23, 2021

12. Exception Handling

12.1 Exception Handling: Require, Assert and Revert in Solidity

In this Lab you will learn more about Exception handling. At the end of the Lab I want you to be fully aware of when and how to use require, assert and revert.

How these behave with transactions and also between Smart Contracts.

12.1.1 What You Know At The End Of The Lab

How to validate user input with require

How to validate invariants with assert

Observe Transactions and understand the reason why they are failing

12.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. About 60 Minutes of your precious time

12.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

12.1.4 Get Started

Let's get started!

Using Solidity 0.6

In this lecture we deliberately using an older version of Solidity to demonstrate integer rollover assertion. This won't be necessary anymore with Solidity 0.8, but it is still a very common scenario in many recent DeFi Startups who are not yet using Solidity 0.8.

Last update: May 21, 2021

12.2 The Smart Contract

Let's start with this basic Smart Contract. As mentioned before, **it's written in Solidity 0.6**, not the latest version, because of the integer roll-over example that we will demonstrate.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;
contract ExceptionExample {
    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        if(_amount <= balanceReceived[msg.sender]) {
            balanceReceived[msg.sender] -= _amount;
            _to.transfer(_amount);
        }
    }
}
```

Add this Smart Contract to Remix. The correct Solidity Compiler Version should be selected automatically, but double check, to make sure everything is in order:

The screenshot shows the Remix Ethereum IDE interface. On the left, the **SOLIDITY COMPILER** panel is open, displaying compiler settings: **COMPILER** set to **0.6.12+commit.27d51765** (indicated by a red arrow), **LANGUAGE** set to **Solidity**, and **EVM VERSION** set to **compiler default**. Below these are **COMPILER CONFIGURATION** options: **Auto compile** (checked), **Enable optimization** (unchecked), and **Hide warnings** (unchecked). A large blue button at the bottom of this panel says **Compile exceptions.sol**. On the right, the **exceptions.sol** file is displayed in the editor. The code defines a **contract ExceptionExample** with two functions: **receiveMoney()** and **withdrawMoney()**. The **receiveMoney()** function adds the amount sent to the **balanceReceived** mapping. The **withdrawMoney()** function checks if the amount is less than or equal to the balance received and then transfers it to the sender. The terminal at the bottom shows a welcome message and information about accessible libraries like `web3 version 1.0.0`, `ethers.js`, `swarmgw`, and `remix`.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;
contract ExceptionExample {
    mapping(address => uint) public balanceReceived;
    function receiveMoney() public {
        balanceReceived[msg.sender] += msg.value;
    }
    function withdrawMoney(address _to) public {
        if(_amount <= balanceReceived[_to]) {
            balanceReceived[_to] -= _amount;
            _to.transfer(_amount);
        }
    }
}
```

Okay, what does the Smart Contract do? It's a simplistic wallet contract.

1. You can send Ether to the Smart Contract (via `receiveMoney`).
2. You can withdraw Ether via `withdrawMoney`.
3. The Balance of your account is stored in the `balanceReceived` mapping.

So far so good. Pay attention to the withdrawMoney function. There is a if-else control structure. If you don't have enough balance, then simply nothing happens. That is not ideal, there is no user-feedback.

Let's give it a try!

Last update: May 21, 2021

12.3 Try the Smart Contract

Deploy the Smart Contract. Head over to the "Deploy & Run Transactions" Plugin and Deploy it.

The screenshot shows the Remix IDE interface with the 'Deploy & Run Transactions' plugin active. On the left, there's a sidebar with icons for file operations. The main area has tabs for 'Home' and 'exceptions.sol'. The code editor on the right contains the following Solidity code:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.6.12;
3
4 contract ExceptionExample {
5     mapping(address => uint) public balanceReceived;
6
7     function receiveMoney() public payable {
8         balanceReceived[msg.sender] += msg.value;
9     }
10
11     function withdrawMoney(address payable _to, uint256 _amount) public {
12         if(_amount <= balanceReceived[_to]) {
13             balanceReceived[_to].transfer(_amount);
14         }
15     }
16 }
17
18 }
```

Below the code editor, there are sections for 'Transactions recorded' (1), 'Deployed Contracts' (listing 'EXCEPTIONEXAMPLE AT 0xD91...3913'), and 'Low level interactions' (listing 'receiveMoney', 'withdrawMoney', and 'balanceReceiv...'). A 'Transact' button is located at the bottom of the interactions list. The status bar at the bottom right shows 'creation of ExceptionExample pending...' and a transaction log entry: '[vm] from: 0x5B3...eddC4 to: ExceptionExample hash: 0x777...429b3'.

1. Copy your address into the withdraw money field.
2. Enter a number, like 1000, and
3. Hit withdrawMoney

You see, the transaction works without any problem. That's not good, because internally nothing happened and the user has the feedback that he just did a withdrawal for 1000 wei. Although nothing happened.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with fields for 'VALUE' (set to 0 wei), a 'CONTRACT' dropdown set to 'ExceptionExample - exceptions.sol', a 'Deploy' button, and a 'Publish to IPFS' checkbox. Below that is an 'OR' section with 'At Address' selected and a 'Transactions recorded' dropdown showing 2 entries. One entry is expanded to show a 'receiveMoney' button and a 'withdrawMoney' section with '_to: 0x5B38Da6a701c568545dCfcf' and '_amount: 1000'. On the right, a detailed transaction view is shown for a transaction that has been mined. The transaction details include: status (true), transaction hash (0x1b974585a0da666a040f3...), from (0x5B38Da6a701c568545dCfcf), to (ExceptionExample.receiveMoney), gas (3000000 gas), transaction cost (23091 gas), execution cost (1819 gas), hash (0x1b974585a0da666a040f3...), input (0x6d2...6ec18), decoded input ({}), decoded output ({}), logs ([]), and value (0 wei).

Try yourself first

Before heading to the next page, try yourself first to replace the if/else with a `require()`

Require is here for user-input validation and if it evaluates to false, it will throw an exception.

For example `require(false)` or `require(l == 0)` will throw an exception. You can optionally add an error message `require(false, "Some Error Message")`

Last update: May 21, 2021

12.4 Add a Require

Alright, now it's time to add a require instead of the if/else control structure. Did you try? Let's compare our code:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.6.12;

contract ExceptionExample {

    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "Not Enough Funds, aborting");

        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}
```

If you run this now and try to withdraw more Ether than you have, it will output an error. Repeat the steps from before:

1. Deploy a new Instance!
2. Copy your address
3. Enter an amount
4. Click on "withdrawMoney"

Observe the log window:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area is titled "DEPLOY & RUN TRANSACTIONS". Under the "CONTRACT" section, the contract name is "ExceptionExample - exceptions.sol". There's a large orange "Deploy" button. Below it is a checkbox for "Publish to IPFS". The word "OR" is centered between two options: "At Address" and "Load contract from Address". A dropdown menu shows "Transactions recorded" with a count of 4. Below that is a list of "Deployed Contracts" with one entry: "EXCEPTIONEXAMPLE AT 0xF8E...9FBE8". Under this entry, there are buttons for "receiveMoney" (red) and "withdrawMoney" (grey). The "withdrawMoney" button has fields for "_to: 0x5B38Da6a701c568545dCfcf" and "_amount: 1234". To the right of these fields is a "transact" button. On the far right, the UI displays a terminal-like window showing Solidity code and transaction logs. One log shows a successful withdrawal, and another shows a failed withdrawal attempt with a red X icon, indicating it failed due to the require statement.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;

contract ExceptionExample {
    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        balanceReceived[_to] += _amount;
        _to.transfer(_amount);
    }
}
```

How cool is that! Not only your transaction fails, which is what we want, we also get proper feedback to the user. Great! But if we have require statements, what are assert for?

Let's check it out!

Last update: May 21, 2021

12.5 Assert to check Invariants

Assert is used to check invariants. Those are states our contract or variables should never reach, ever. For example, if we decrease a value then it should never get bigger, only smaller.

Let's change a few things in our Smart Contract to add an integer roll-over bug that we can easily trigger.

Bug

This contract has an intentional limitation, which we will use to trigger a bug. To subsequently fix it.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;
contract ExceptionExample {
    mapping(address => uint64) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += uint64(msg.value);
    }

    function withdrawMoney(address payable _to, uint64 _amount) public {
        require(_amount <= balanceReceived[msg.sender], "Not Enough Funds, aborting");
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}
```

1. Deploy a new Contract Instance
2. Enter 19 **Ether** into the Value field
3. Hit "receiveMoney"

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area has tabs for "JavaScript VM" and "Solidity". The "Solidity" tab is active, displaying the source code of the `exceptions.sol` contract.

Contract Deployment:

- Step 1:** The "Deploy" button is highlighted with a red circle containing the number 1.
- Step 2:** The "Value" input field is set to 19 ether, indicated by a red circle with the number 2.
- Step 3:** The "receiveMoney" button under the deployed contract is highlighted with a red circle containing the number 3.

Contract Source Code:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;

contract ExceptionExample {
    mapping(address => uint64) public balanceReceived;

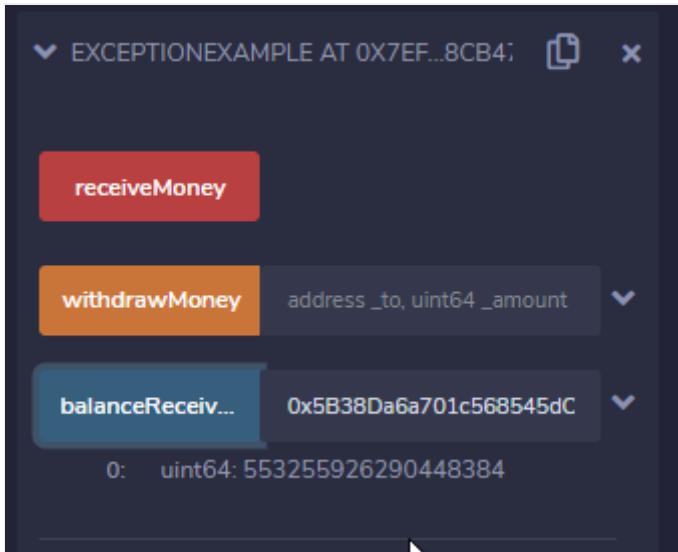
    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[_to]);
        balanceReceived[_to] -= _amount;
        _to.transfer(_amount);
    }
}
```

Deployment Log:

- [vm] from: 0x5B3...eddC4 to: ExceptionExample at 0x7EF...8CB4 hash: 0x45b...33279
- transact to ExceptionExample.receiveMoney
- [vm] from: 0x5B3...eddC4 to: ExceptionExample value: 19000000000000000000000000000000 wei data

The Transaction goes through. Let's check our balance, we should have 19000000000000000000000000000000 Wei, or?



That's only 553255926290448384 Wei, or around 0.553 Ether. Where is the rest? What happened?

We are storing the balance in an uint64. Unsigned integers go from 0 to 2^{n-1} , so that's $2^{64}-1$ or 18446744073709551615. So, it can store a max of 18.4467... Ether. We sent 19 Ether to the contract. It automatically rolls over to 0. So, we end up with $19000000000000000000 - 18446744073709551615 - 1$ (the 0 value) = 553255926290448384.

How can we fix it?

12.5.1 Add an Assert to check invariants

Asserts are here to check states of your Smart Contract that should never be violated. For example: a balance can only get bigger if we add values or get smaller if we reduce values.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;

contract ExceptionExample {
    mapping(address => uint64) public balanceReceived;

    function receiveMoney() public payable {
        assert(msg.value == uint64(msg.value));
        balanceReceived[msg.sender] += uint64(msg.value);
        assert(balanceReceived[msg.sender] >= uint64(msg.value));
    }

    function withdrawMoney(address payable _to, uint64 _amount) public {
        require(_amount <= balanceReceived[msg.sender], "Not Enough Funds, aborting");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}
```

Run it again. Try to input 20 Ether. Or also try run two transactions with 10 Ether each, so it doesn't overflow for the transaction itself, but for the second assertion, where it checks if the balance is still valid.

Last update: June 23, 2021

12.6 Understanding Try/Catch

There is a new concept in Solidity since Solidity 0.6, which allows for basic try/catch inside a Smart Contract. Before that you had to use constructs like address.call (which we will do later in the course as well). But here I quickly want to give you a hint of what's possible now with Error handling.

Let's create a Smart Contract which will always fail to execute.

Create a new file in Remix with the following content:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

contract WillThrow {
    function aFunction() public {
        require(false, "Error message");
    }
}
```

Now let's add a contract directly underneath that catches the error within a function call and instead emits an event with the error message. I am aware that we haven't covered events quite yet, so, let me quickly explain what happens. It will catch the error, and instead of unsuccessfully showing an error, it will successfully mine the transaction. During the transaction it will emit an event that shows the error message. You can see it later in the transaction logs.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

contract WillThrow {
    function aFunction() public {
        require(false, "Error message");
    }
}

contract ErrorHandling {
    event ErrorLogging(string reason);
    function catchError() public {
        WillThrow will = new WillThrow();
        try will.aFunction() {
            //here we could do something if it works
        } catch Error(string memory reason) {
            emit ErrorLogging(reason);
        }
    }
}
```

Head over to the Deploy and Run Transactions Plugin and Deploy the "ErrorHandling" Contract:

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar has icons for deploying, publishing to IPFS, and loading contracts from address. The main area is titled "DEPLOY & RUN TRANSACTIONS". It shows a gas limit of 0 ether. A dropdown menu labeled "CONTRACT" contains "ErrorHandling - trycatch.sol" (marked with a red circle 1). Below it is a "Deploy" button and a checkbox for "Publish to IPFS". An "OR" section allows deployment "At Address" or "Load contract from Address". Under "Transactions recorded", there is a list for "Deployed Contracts" with one entry: "ERRORHANDLING AT 0XDA0...42B53" (marked with a red circle 2). This entry has a "catchError" button. Below it is a "Low level interactions" section with a "Transact" button.

The right side of the interface shows the Solidity code for "trycatch.sol":

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

contract WillThrow {
    function aFunction() public {
        require(false, "Error message");
    }
}

contract ErrorHandling {
    event ErrorLogging(string reason);
    function catchError() public {
        WillThrow will = new WillThrow();
        try will.aFunction() {
            //here we could do something
        } catch Error(string memory reason) {
            emit ErrorLogging(reason);
        }
    }
}
```

At the bottom right, there is a log table with the following data:

	0	<input type="checkbox"/> listen on network
status	true	Transacted
transaction hash	0x745533ff	
from	0x5B38Da6a	

You can see in the logs on the bottom right side that the transaction is mined successfully. And when you open the transaction details you see an event with the error message:

DEPLOY & RUN TRANSACTIONS

0 ether

CONTRACT

ErrorHandling - trycatch.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 4

Deployed Contracts

ERRORHANDLING AT 0xDAO...42B53

catchError

Low level interactions

CALldata

Transact

// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

contract WillThrow {
 function aFunction() public {
 require(false, "Error message");
 }
}

[vm] from: 0x5B3...eddC4 to: ErrorHandling
hash: 0x745...8474a

status	true
transaction hash	0x745533ff...
from	0x5B38Da6...
to	ErrorHandling
gas	3000000
transaction cost	116365
execution cost	95093
hash	0x745533ff...
input	0x8b1...88
decoded input	{}
decoded output	{}
logs	[{ "from": "0xdcbe723...", "text": "ErrorLog" }]

I know, this is a lot of new stuff which we haven't covered in depth yet. The point here is that Try/Catch fits into the exception handling and this is the most simplistic example I could come up with. It's something you should've seen before going on with the rest of the course. Now back to the content!

Last update: May 21, 2021

13. Fallback Function / Constructor / View and Pure

13.1 Fallback Function, View/Pure Functions, Constructor

In this Lab you will learn more about the Fallback Functions `receive()` and `fallback()`, then more about reading functions like `view` and `pure` and also about the constructor, to set initial values of your Smart Contract.

13.1.1 What You Know At The End Of The Lab

View and Pure Functions

Receive fallback function

Fallback function

The Constructor

13.1.2 Prerequisites - You need:

1. Chrome or Firefox browser.
2. An Internet connection
3. About 60 Minutes of your precious time

13.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

13.1.4 Get Started

Let's get started!

Using Solidity 0.8

These labs were updated to Solidity 0.8 and can differ from the videos slightly. Please pay attention to the callouts where appropriate.

Last update: May 23, 2021

13.2 Contract Example

Let's start with a simple wallet-like Smart Contract that you probably know already from previous labs:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.3;

contract FunctionsExample {

    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value >= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}
```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM

ACCOUNT

0x5B3...eddC4 (100 ether)

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

FunctionsExample - FallbackExample.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 0

Deployed Contracts

Currently you have no contract instances to interact with.

Home FallbackExample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        assert(balanceReceived[msg.sender] - _amount == balanceReceived[msg.sender]);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}
```

You can use this terminal to:

- Check transactions details and start deployment
- Execute JavaScript scripts:
 - Input a script directly in the command line
 - Select a Javascript file in the file interface
 - Right click on a JavaScript file in the file interface

The following libraries are accessible:

- [web3 version 1.0.0](#)
- [ethers.js](#)
- [swarmgw](#)
- [remix \(run remix.help\(\) for more info\)](#)

13.2.1 Try the Smart Contract

What does the contract do? Let's give it a try! Head over to the Deploy and Run Transactions tab.

1. Deploy the Contract
2. Ether 1 Ether in the input field
3. Send it to the receive Money Function

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 1 ether

CONTRACT: FunctionsExample - FallbackExample.sol

1 Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded: 1

Deployed Contracts

FUNCTIONSEXAMPLE AT 0xD91...3913

3 receiveMoney

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

Home FallbackExample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;
contract FunctionsExample {
    mapping(address => uint) public balances;
    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender]);
    }
    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        assert(balanceReceived[msg.sender] - _amount == balanceReceived[_to]);
        _to.transfer(_amount);
    }
}
```

The following libraries are accessible:

- web3 version 1.0.0
- ethers.js
- swarmgw
- remix (run remix.help() for more info)

creation of FunctionsExample pending...

[vm] from: 0x5B3...eddC4 to: FunctionsExample at 0xD91...3913 hash: 0xa42...5a205

Great, you have 1 Ether in your on-chain Wallet. Let's bring it back out!

1. Copy your address
2. Enter it into the withdraw money address field
3. Enter 1 Ether in Wei (10^{18}): 1000000000000000000 into the amount field

The screenshot shows the Remix IDE interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and the Solidity code editor on the right.

Left Panel (Deployment):

- ACCOUNT:** Set to `0x5B3...eddC4 (99.9999999)`.
- GAS LIMIT:** Set to `3000000`.
- VALUE:** Set to `1 ether`.
- CONTRACT:** Set to `FunctionsExample - FallbackExample.s`.
- Deploy** button.
- Publish to IPFS.
- OR**
- At Address:** Selected tab.
- Transactions recorded:** 1 transaction.
- Deployed Contracts:** Shows `FUNCTIONSEXAMPLE AT 0xD91...3913` with methods `receiveMoney` and `withdrawMoney`.
- withdrawMoney** method parameters:
 - `_to:` `0x5B38Da6a701c568545dCfcE`
 - `_amount:` `1000000000000000000000000`
- transact** button.

Right Panel (Code Editor):

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;
contract FunctionsExample {
    mapping(address => uint) public balanceReceived;
    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender] + msg.value);
    }
    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        assert(balanceReceived[msg.sender] - _amount == balanceReceived[msg.sender] - _amount);
        _to.transfer(_amount);
    }
}
```

Bottom Right Panel (Logs):

- creation of FunctionsExample pending...
- [vm] from: `0x5B3...eddC4` to: `Functionsexample` hash: `0xa42...5a205`
- `1000000000000000000000000`

Superb, you got your Ether back out.

13.2.2 Problem

Right now, if you want to deposit Ether into the Smart Contract, you need to directly interact with the receiveMoney function. There is no way to simple send 1 Ether to the Contract address and let it deal with the rest. It would simply error out! Give it a try:

1. Enter 1 Ether into the input field
2. Hit the "transact" button within the low-level interaction section

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area has a title "DEPLOY & RUN TRANSACTIONS". A "VALUE" field is set to "1 ether". Below it, a "CONTRACT" dropdown is set to "FunctionsExample - FallbackExample.s". A large orange "Deploy" button is prominent. There's also a checkbox for "Publish to IPFS". Below the deploy section, there are tabs for "At Address" and "Load contract from Address". Under "Transactions recorded", there's a list for "FUNCTIONSEXAMPLE AT 0xD91...3913" which includes "receiveMoney", "withdrawMoney", and "balanceReceiv...". Under "Low level interactions", there's a "CALldata" section with a "Transact" button. A red circle with the number "1" is over the "VALUE" input field, and another red circle with the number "2" is over the "Transact" button.

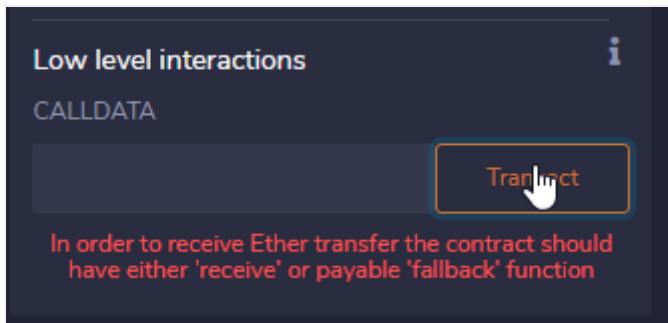
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        assert(balanceReceived[msg.sender] - _amount == balanceReceived[_to]);
        _to.transfer(_amount);
    }
}
```

You will see it will produce an error that looks like this or similar:



Let's add a low level fallback function so we can simply send Ether to our Wallet...

Last update: May 23, 2021

13.3 The Receive Fallback Function

Solidity 0.6/0.8 Update

Prior Solidity 0.6 the fallback function was simply an anonymous function that looked like this:

```
function () external {  
}
```

It's now two different functions. `receive()` to receive money and `fallback()` to just interact with the Smart Contract without receiving Ether. This example uses the updated version.

Let's add a receive fallback function to the Smart Contract so we can simply send 1 Ether to the Contract without directly interacting with a concrete function.

```
// SPDX-License-Identifier: MIT  
  
pragma solidity 0.8.3;  
  
contract FunctionsExample {  
  
    mapping(address => uint) public balanceReceived;  
  
    function receiveMoney() public payable {  
        assert(balanceReceived[msg.sender] + msg.value >= balanceReceived[msg.sender]);  
        balanceReceived[msg.sender] += msg.value;  
    }  
  
    function withdrawMoney(address payable _to, uint _amount) public {  
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");  
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);  
        balanceReceived[msg.sender] -= _amount;  
        _to.transfer(_amount);  
    }  
  
    receive() external payable {  
        receiveMoney();  
    }  
}
```

Let's try this again:

1. Deploy a new Version
2. Enter 1 Ether in the value field
3. Hit the Transact button
4. Observe the Transaction Log, see that the transaction goes through now.

DEPLOY & RUN TRANSACTIONS

GAS LIMIT: 3000000

VALUE: 1 ether

CONTRACT: FunctionsExample - FallbackExample.sol

Deploy

Publish to IPFS

At Address | Load contract from Address

Transactions recorded: 5

Deployed Contracts: FUNCTIONSEXAMPLE AT 0xD7A...F771

receiveMoney

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

Low level interactions

CALldata

Transact

creation of FunctionsExample pending...

[vm] from: 0x5B3...eddC4 to: Functionsexample address 0xD7A...F771 hash: 0x2d9...4bc4d

transact to FunctionsExample.(receive) pending...

[vm] from: 0x5B3...eddC4 to: Functionsexample address 0xD7A...F771 logs: 0 hash: 0x8b1...5439f

Did you see it? Just works like this. It even says `.(receive)` in the transaction. That's great!

Now onto the constructor!

Last update: May 23, 2021

13.4 The Solidity Constructor

A constructor is a function that is called once during deployment of the Smart Contract. It's also called once only and can't be called again afterwards.

This lets you set specific things in your Smart Contract during deployment. For example, you could specify who deployed the Smart Contract - and then let only that person destroy the Smart Contract. Like an owner. Let's try this!

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.3;

contract FunctionsExample {

    mapping(address => uint) public balanceReceived;

    address payable public owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function destroySmartContract() public {
        require(msg.sender == owner, "You are not the owner");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value >= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }

    receive() external payable {
        receiveMoney();
    }
}
```

ENVIRONMENT

JavaScript VM

ACCOUNT

0x5B3...eddC4 (100 ether)

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

FunctionsExample - FallbackExample.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 0

Deployed Contracts

Currently you have no contract instances to interact with.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;
contract FunctionsExample {
    mapping(address => uint) public balances;
    address payable public owner;
    constructor() {
        owner = payable(msg.sender);
    }
    function destroySmartContract() public {
        require(msg.sender == owner, "Only owner can selfdestruct");
        selfdestruct(owner);
    }
    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + _value == balanceReceived[msg.sender]);
    }
    function withdrawMoney(address payable _to) public {
        require(_value <= balanceReceived[msg.sender], "Insufficient funds");
        assert(balanceReceived[msg.sender] - _value == balanceReceived[msg.sender]);
        _to.transfer(_value);
    }
}
```

0 listen on network

You can use this terminal to:

- Check transactions details and start deployment
- Execute JavaScript scripts:
 - Input a script directly in the command line interface
 - Select a Javascript file in the file browser
 - Right click on a JavaScript file in the file browser

The following libraries are accessible:

- web3 version 1.0.0
- ethers.js
- swarmgw
- remix (run remix.help() for more info)

What does it do?

1. In the constructor it sets the owner address to the address who deployed the Smart Contract
2. Upon destruction, it checks who called the function.
 1. If the owner called, then the Smart Contract is destroyed
 2. If it is not the owner, then an error is thrown

13.4.1 Running the Smart Contract

Let's try this!

1. Deploy the Smart Contract
2. Check the owner address: you should see the same account as selected!
3. Destroy the Smart Contract
4. Click on the owner address again: it's zero now, because the Smart Contract is not longer existing.

It should work fine! No error.

Before hitting Destroy:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, the 'Deploy & Run Transactions' sidebar is open, showing fields for GAS LIMIT (3000000), VALUE (0 wei), and CONTRACT (FunctionsExample - FallbackExample.sol). A red arrow points from the 'Deploy' button (marked with a red circle containing '1') to the right-hand code editor.

The code editor displays the Solidity source code for the `FunctionsExample` contract:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;
    address payable public owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function destroySmartContract() public payable {
        require(msg.sender == owner, "Only owner can call this function");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender]);
    }

    function withdrawMoney(address payable _to) public payable {
        require(_amount <= balanceReceived[msg.sender], "Insufficient funds");
        assert(balanceReceived[msg.sender] - _amount == balanceReceived[_to]);
        _to.transfer(_amount);
    }

    receive() external payable {
        receiveMoney();
    }
}
```

Below the code editor, the transaction history shows the creation of the contract and a call to its `owner` function.

Transactions recorded: 1

Deployed Contracts:

- FUNCTIONSEXAMPLE AT 0xD91...3913

Contract functions:

- destroySmart...
- receiveMoney
- withdrawMoney
- balanceReceiv...

Owner: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

After hitting Destroy:

DEPLOY & RUN TRANSACTIONS

0x5B3...eddC4 (99.9999995)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
FunctionsExample - FallbackExample.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded (1)

Deployed Contracts

FUNCTIONSEXAMPLE AT 0xD91...3913 (1)

- destroySmartContract()**
- receiveMoney()**
- withdrawMoney(address _to, uint256 _amount)**
- balanceReceived(address)**
- owner**

0: address: 0x00000000000000000000000000000000
00000000000000000000000000000000

Home FallbackExample.sol

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.3;
4
5 contract FunctionsExample {
6
7     mapping(address => uint) public balances;
8
9     address payable public owner;
10
11     constructor() {
12         owner = payable(msg.sender);
13     }
14
15     function destroySmartContract() public {
16         require(msg.sender == owner, "Only owner can call this function");
17         selfdestruct(owner);
18     }
19
20     function receiveMoney() public payable {
21         assert(balanceReceived[msg.sender] <= msg.value);
22         balanceReceived[msg.sender] += msg.value;
23     }
24
25     function withdrawMoney(address payable _to, uint256 _amount) public {
26         require(_amount <= balanceReceived[_to], "Insufficient funds");
27         assert(balanceReceived[_to] >= _amount);
28         balanceReceived[_to] -= _amount;
29         _to.transfer(_amount);
30     }
31
32     receive() external payable {
33         receiveMoney();
34     }
35 }
```

listen on network

transact to FunctionsExample.destroySmartContract

[vm] from: 0x5B3...eddC4 to: FunctionsExample address: 0x854...74728 logs: 0 hash: 0xf...

call to FunctionsExample.owner

[call] from: 0x5B38Da6a701c568545dcf...

Executing the Constructor

The constructor is executed only during deployment. There is no way to execute the constructor code afterwards.

If you look at the interface of your Smart Contract (the buttons on the left side), there is "destroySmartContract", "receiveMoney", "withdrawMoney", "balanceReceived" and "owner", but no way to call the constructor code again. This is intentional and a feature of Solidity.

But what about the require statement?

Let's deploy a new version of the Smart Contract, but this time try to call `destroySmartContract` with a different account:

1. Deploy Smart Contract
2. Select a different account from the Accounts Dropdown
3. Hit the "destroySmartContract" Button

It should give you an error.

DEPLOY & RUN TRANSACTIONS

2 0xAb8...35cb2 (99.9999999)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
FunctionsExample - FallbackExample.sol

1 Deploy Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 7

Deployed Contracts

FUNCTIONSEXAMPLE AT 0XDA0...42BE

3

destroySmart...

receiveMoney

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

owner

0: address: 0x5B38Da6a701c568545dCfcB 03FcB875f56beddC4

FallbackExample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;
    address payable public owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function destroySmartContract() public {
        require(msg.sender == owner, "Only owner can call this function");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value == balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[_to], "Insufficient funds");
        assert(balanceReceived[_to] - _amount == balanceReceived[_to]);
        _to.transfer(_amount);
    }

    receive() external payable {
        receiveMoney();
    }
}
```

0 listen on network

[vm] from: 0xAb8...35cb2 to: FunctionsExample at 0xDA0...42BE data: 0x854...74728 logs: 0 hash: 0x1234567890

transact to FunctionsExample.destroySmartContract() from initial state. Reason provided by the contract: Only owner can call this function

call to FunctionsExample.owner

CALL [call] from: 0xAb8483F64d9C6d1EcF9b8...

Do you see the different Account when you hit "owner"?

13.4.2 Summary

Why is that so powerful?

We automatically set a variable depending on who deployed the Smart Contract. You can also have constructor arguments, just like any other function. It will be executed once and once only.

Last update: May 23, 2021

13.5 View and Pure Functions in Solidity

So far, we have mostly interacted and modified state variables from our Smart Contract. For example, when we write the owner, we modify a state variable. When we update the balance, we modify a state variable (`balanceReceived`).

For this, we needed to send a transaction. That works very transparently in Remix and also looks instantaneous and completely free of charge, but in reality it isn't. Modifying the State costs gas, is a concurrent operation that requires mining and doesn't return any values.

Reading values, on the other hand, is virtually free and doesn't require mining.

There are two types of reading functions:

1. view: Accessing state variables
2. pure: Not accessing state variables

13.5.1 View Function

Let's make our owner-variable private and instead add a simple getter function function

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;
    address payable owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function getOwner() public view returns(address) {
        return owner;
    }

    function destroySmartContract() public {
        require(msg.sender == owner, "You are not the owner");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value >= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }

    receive() external payable {
        receiveMoney();
    }
}
```

DEPLOY & RUN TRANSACTIONS

0xAb8...35cb2 (99.9999999)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
FunctionsExample - FallbackExample.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 8

Deployed Contracts

FUNCTIONSEXAMPLE AT 0X652...BA59

destroySmart...

receiveMoney

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

getOwner

Low-level interactions

Home FallbackExample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;
contract FunctionsExample {
    mapping(address => uint) public balances;
    address payable owner;
    constructor() {
        owner = payable(msg.sender);
    }
    function getOwner() public view returns address {
        return owner;
    }
    function destroySmartContract() public {
        require(msg.sender == owner, "Only owner can destroy");
        selfdestruct(owner);
    }
    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] <= msg.value);
        balanceReceived[msg.sender] += msg.value;
    }
    function withdrawMoney(address payable _to) public {
        require(_amount <= balanceReceived[msg.sender]);
        assert(balanceReceived[msg.sender] >= _amount);
        _to.transfer(_amount);
    }
    receive() external payable {
        receiveMoney();
    }
}
```

0 listen on network

call to FunctionsExample.owner

CALL [call] from: 0xAb8483F64d9C6d1EcF9b8...

creation of FunctionsExample pending...

[vm] from: 0xAb8...35cb2 to: FunctionsExample.sol hash: 0xa4e...07cf0

Let's give this a try: 1. Deploy the Smart Contract 2. click on "getOwner" 3. Observe the Transaction Log

```

creation of FunctionsExample pending...

1 [vm] from: 0xAb8...35cb2 to: FunctionsExample.(constructor) value: 0 wei data: 0x608...30033
hash: 0xa4e...07cf0

call to FunctionsExample.getOwner

2 CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: FunctionsExample.getOwner()
data: 0x893...d20e8

```

You can observe that the deployment is a transaction, marked by the little green checkmark (1), while the reading operation is a call (2). In reality a transaction would need to get signed by a private key and mined by the network, while a reading operation does not need to get signed.

13.5.2 Pure Functions

So, view functions are reading functions - what are pure functions?

Pure functions are functions that are not accessing any state variables. They can call other pure functions, but not view functions.

Let's do a quick example:

```

// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balanceReceived;

    address payable owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function getOwner() public view returns(address) {
        return owner;
    }

    function convertWeiToEth(uint _amount) public pure returns(uint) {
        return _amount / 1 ether;
    }

    function destroySmartContract() public {
        require(msg.sender == owner, "You are not the owner");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + msg.value >= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}

```

```
    }
    receive() external payable {
        receiveMoney();
    }
}
```

Let's run it:

1. Deploy a new Instance
2. Run "convertWeiToEth" with 100
3. Observe the Transaction log

DEPLOY & RUN TRANSACTIONS

CONTRACT

FunctionsExample - FallbackExample.sol

1 Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 9

Deployed Contracts

FUNCTIONSEXAMPLE AT 0X417...2600

destroySmart...

receiveMoney

withdrawMoney address _to, uint256 _amount

balanceReceiv... address

convertWeiTo... 100

0: uint256: 0

getOwner

Low level interactions

CALldata

Transact

// SPDX-License-Identifier: MIT

```

pragma solidity 0.8.3;

contract FunctionsExample {
    mapping(address => uint) public balances;
    address payable owner;

    constructor() {
        owner = payable(msg.sender);
    }

    function getOwner() public view returns (address) {
        return owner;
    }

    function convertWeiToEth(uint _amount) public pure returns (uint) {
        return _amount / 1 ether;
    }

    function destroySmartContract() public {
        require(msg.sender == owner, "Only owner can selfdestruct");
        selfdestruct(owner);
    }

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] == 0);
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint256 _amount) public {
        require(_amount <= balanceReceived[msg.sender], "Insufficient funds");
        assert(balanceReceived[msg.sender] == _amount);
        _to.transfer(_amount);
    }
}
```

0 listen on network

call to FunctionsExample.convertWeiToEth

CALL [call] from: 0xAb8483F64d9C6d1EcF9b8...

call to FunctionsExample.convertWeiToEth

3 CALL [call] from: 0xAb8483F64d9C6d1EcF9b8...

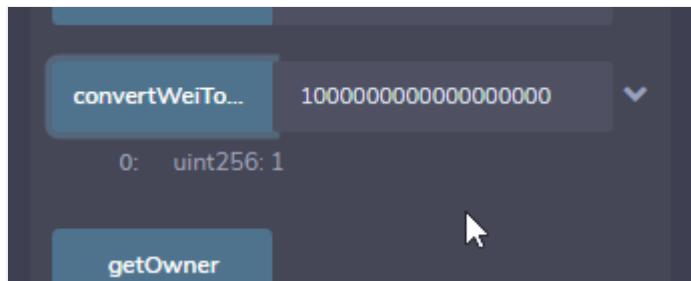
>

Do you know why the returned amount is 0? Think about it for a moment, then look into the solution!

Why the amount is 0 solution

The amount is zero, because we return an integer. We divide $100 / 1000000000000000000 = 0.0000000000000001$. Integers in Solidity are cut off (not even rounded, or always rounded down).

If you enter 1000000000000000000 into the amount field, then the returned amount is 1 (1 Ether)



Alright, that's it, there's nothing much more to say about view and pure functions at this moment. On to the next lecture!

Last update: May 23, 2021

14. LAB: Shared Wallet

14.1 Project Shared Wallet

14.1.1 Real-World Use-Case for this Project

Allowance for Children per day/week/month to be able to spend a certain amount of funds.

Employers give employees an allowance for their travel expenses.

Businesses give contractors an allowance to spend a certain budget.

14.1.2 Development Goal

Have an on-chain wallet smart contract.

This wallet contract can store funds and let users withdraw again.

- ¶ You can also give "allowance" to other, specific user-addresses.

Restrict the functions to specific user-roles (owner, user)

Re-Use existing smart contracts which are already audited to the greatest extent

14.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

14.1.4 Get Started

Let's get started by creating [Easy Smart Contract](#)

Last update: May 23, 2021

14.2 We Define the Basic Smart Contract

This is the very basic smart contract. It can receive Ether and it's possible to withdraw Ether, but all in all, not very useful quite yet. Let's see if we can improve this a bit in the next step.

Sharedwallet.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
contract SharedWallet {
    function withdrawMoney(address payable _to, uint _amount) public {
        _to.transfer(_amount);
    }
    receive() external payable {
    }
}
```

Solidity Updates

Prior Solidity 0.6 the fallback function was simply called "function() external payable" - a Function without a name. Since Solidity 0.6 there are two different functions: one called fallback and the other one called "receive". Only "receive" can receive ether. You can read more about this in [my walkthrough!](#)

Also, the code in this lab has been ported to Solidity 0.8.

The most prominent change is the removal of the SafeMath library, since Solidity 0.8 doesn't do automatic integer rollovers anymore. Read more about this in the topic about [Overflow and Underflow](#). In the lab are notes where Solidity 0.8 changes come in.

Last update: May 23, 2021

14.3 Permissions: Allow only the Owner to Withdraw Ether

In this step we restrict withdrawal to the owner of the wallet. How can we determine the owner? It's the user who deployed the smart contract.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
contract SharedWallet {
    address owner;
    constructor() {
        owner = msg.sender;
    }
    modifier onlyOwner() {
        require(msg.sender == owner, "You are not allowed");
        _;
    }
    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
        _to.transfer(_amount);
    }
    receive() external payable {
    }
}
```

Whatch out that you also add the "onlyOwner" modifier to the withdrawMoney function!

Last update: May 23, 2021

14.4 Use Re-Usable Smart Contracts from OpenZeppelin

Having the owner-logic directly in one smart contract isn't very easy to audit. Let's break it down into smaller parts and re-use existing audited smart contracts from OpenZeppelin for that. The latest OpenZeppelin contract does not have an `isOwner()` function anymore, so we have to create our own. Note that the `owner()` is a function from the `Ownable.sol` contract.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
        _to.transfer(_amount);
    }

    receive() external payable {
    }
}
```

Last update: May 23, 2021

14.5 Permissions: Add Allowances for External Roles

In this step we are adding a mapping so we can store address => uint amounts. This will be like an array that stores [0x123546...] an address, to a specific number. So, we always know how much someone can withdraw. We also add a new modifier that checks: Is it the owner itself or just someone with allowance?

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    mapping(address => uint) public allowance;

    function addAllowance(address _who, uint _amount) public onlyOwner {
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
       _;
    }

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        _to.transfer(_amount);
    }

    receive() external payable {
    }
}
```

Did you catch the bug?

Have a look at the withdrawMoney functionality and think it through!

In the next lecture we're going to improve our smart contract a little bit and avoid double spending.

Last update: May 23, 2021

14.6 Improve/Fix Allowance to avoid Double-Spending

Without reducing the allowance on withdrawal, someone can continuously withdraw the same amount over and over again. We have to reduce the allowance for everyone other than the owner.

```
function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
    allowance[_who] -= _amount;
}

function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
    require(_amount <= address(this).balance, "Contract doesn't own enough money");
    if(!isOwner()) {
        reduceAllowance(msg.sender, _amount);
    }
    _to.transfer(_amount);
}
```

Last update: May 23, 2021

14.7 Improve Smart Contract Structure

Now we know our basic functionality, we can structure the smart contract differently. To make it easier to read, we can break the functionality down into two distinct smart contracts.

Note

Note that since Allowance is Ownable, and the SharedWallet is Allowance, therefore by commutative property, SharedWallet is also Ownable.

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    mapping(address => uint) public allowance;

    function setAllowance(address _who, uint _amount) public onlyOwner {
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
       _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        allowance[_who] -= _amount;
    }
}

contract SharedWallet is Allowance {

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }

    receive() external payable {
    }
}
```

Both contracts are still in the same file, so we don't have any imports (yet). That's something for another lecture later on. Right now, the important part to understand is inheritance.

Last update: May 23, 2021

14.8 Add Events in the Allowances Smart Contract

One thing that's missing is events.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {

    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    function setAllowance(address _who, uint _amount) public onlyOwner {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
       _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount);
        allowance[_who] -= _amount;
    }
}

contract SharedWallet is Allowance {
    //...
}
```

Last update: May 23, 2021

14.9 Add Events in the SharedWallet Smart Contract

Obviously we also want to have events in our shared wallet, when someone deposits or withdraws funds:

```
contract SharedWallet is Allowance {  
  
    event MoneySent(address indexed _beneficiary, uint _amount);  
    event MoneyReceived(address indexed _from, uint _amount);  
  
    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {  
        require(_amount <= address(this).balance, "Contract doesn't own enough money");  
        if(!isOwner()) {  
            reduceAllowance(msg.sender, _amount);  
        }  
        emit MoneySent(_to, _amount);  
        _to.transfer(_amount);  
    }  
  
    receive() external payable {  
        emit MoneyReceived(msg.sender, msg.value);  
    }  
}
```

.....

Last update: May 23, 2021

14.10 Add the SafeMath Library safeguard Mathematical Operations

Directly from the [SafeMath Library source code](#):

Arithmetic operations in Solidity wrap on overflow. This can easily result in bugs, because programmers usually assume that an overflow raises an error, which is the standard behavior in high level programming languages. `SafeMath` restores this intuition by reverting the transaction when an operation overflows.

Solidity changed

In a recent update of Solidity the Integer type variables cannot overflow anymore. Read more about the following [Solidity 0.8 release notes!](#).

Add the following code only if you are using solidity < 0.8!!!

```
pragma solidity ^0.6.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/math/SafeMath.sol";

contract Allowance is Ownable {

    using SafeMath for uint;

    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    function setAllowance(address _who, uint _amount) public onlyOwner {
        //...
    }

    modifier ownerOrAllowed(uint _amount) {
        //...
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who].sub(_amount));
        allowance[_who] = allowance[_who].sub(_amount);
    }
}

contract SharedWallet is Allowance {
    //...
}
```

Last update: May 23, 2021

14.11 Remove the Renounce Ownership functionality

Now, let's remove the function to remove an owner. We simply stop this with a revert. Add the following function to the SharedWallet:

```
contract SharedWallet is Allowance {  
    //...  
  
    function renounceOwnership() public override onlyOwner {  
        revert("can't renounceOwnership here"); //not possible with this smart contract  
    }  
  
    //...  
}
```

Last update: May 23, 2021

14.12 Move the Smart Contracts into separate Files

As a last step, let's move the smart contracts into separate files and use import functionality:

Allowance.sol SharedWallet.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {
    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    function setAllowance(address _who, uint _amount) public onlyOwner {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
       _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount);
        allowance[_who] -= _amount;
    }
}

// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "./Allowance.sol";

contract SharedWallet is Allowance {
    event MoneySent(address indexed _beneficiary, uint _amount);
    event MoneyReceived(address indexed _from, uint _amount);

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);
    }

    function renounceOwnership() public override onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }

    receive() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

If you run it, then don't forget to select the correct Smart Contract from the dropdown:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "Allowance.sol";

contract SharedWallet is Allowance {
    event MoneySent(address indexed _beneficiary, uint _amount);
    event MoneyReceived(address indexed _from, uint _amount);

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
        require(_amount <= address(this).balance);
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);
    }

    function renounceOwnership() public override onlyOwner {
        revert("can't renounceOwnership here");
    }

    receive() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

Last update: May 23, 2021

14.13 Finishing Words

14.13.1 Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE:

<https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: May 23, 2021

15. LAB: Supply Chain Project

15.1 Project Supply Chain

The screenshot displays a web-based application interface for managing supply chain elements. At the top, a large title "Simply Payment/Supply Chain Exam..." is visible. Below it, a section titled "Items" contains a sub-section "Add Element". A form is present with fields for "Cost" (set to 0) and "Item Name" (set to exampleitem1). A "Create" button is partially visible. Below the form is a terminal window showing the output of an npm build command:

```
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.
```

15.1.1 Real-World Use-Case for this Project

Can be part of a supply-chain solution

Automated Dispatch upon payment

Payment collection without middlemen

15.1.2 Development-Goal

Showcase Event-Triggers

Understand the low-level function address.call.value()

Understand the Workflow with Truffle

Understand Unit Testing with Truffle

Understand Events in HTML

15.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

15.1.4 Get Started

Let's get started by creating `ItemManager`

Last update: April 17, 2021

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.

Last update: April 17, 2021

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 {

    }
}
```

Solidity Changes

Note that `call.value(msg.value)(abi.encodeWithSignature("triggerPayment(uint256)", index))`, due to changes at Solidity version 6.4 it is recommended it should be changed to `call{value:msg.value}(abi.encodeWithSignature("triggerPayment(uint256)", index))`.

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

```
pragma solidity ^0.6.0;

import "./Item.sol";

contract ItemManager {
    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 {
        Item item = new Item(this, _priceInWei, index);
        items[index] = 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.

Last update: April 17, 2021

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:

Ownable.sol

```
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 {
    //...
    }
}
```

Last update: April 17, 2021

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

thoma> npm install -g truffle
C:\Users\thoma\AppData\Roaming\npm\truffle -> C:\Users\thoma\AppData\Roaming\npm\node\_modules\truffle\bin\truffle

> truffle@5.1.8 postinstall C:\Users\thoma\AppData\Roaming\npm\node\_modules\truffle
> node ./scripts/postinstall.js

- Fetching solc version list from solc-bin. Attempt #1
+ truffle@5.1.8
updated 1 package in 9.586s
thoma> ■
```

Hint: I am working here with version 5.1.8 of Truffle. If you want to follow the exact same version then type in `npm install -g truffle@5.1.8`

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

```
mkdir s06-eventtrigger
cd s06-eventtrigger
ls
```

```
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:

```
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
```

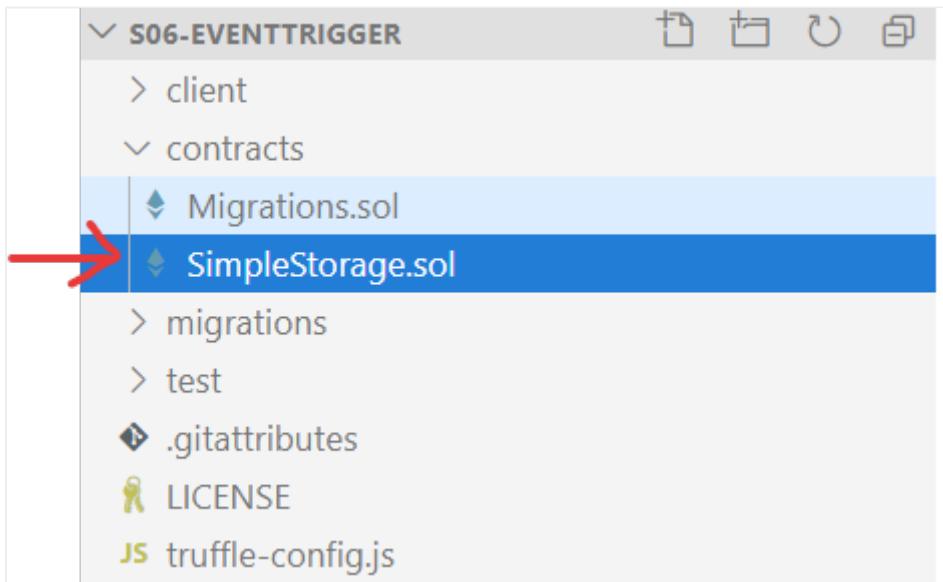
Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d----	1/11/2020 10:41 AM		client
d----	1/11/2020 10:41 AM		contracts
d----	1/11/2020 10:41 AM		migrations
d----	1/11/2020 10:41 AM		test
-a----	1/11/2020 10:41 AM	33	.gitattributes
-a----	1/11/2020 10:41 AM	1075	LICENSE
-a----	1/11/2020 10:41 AM	297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

```
s06-eventtrigger> █
```

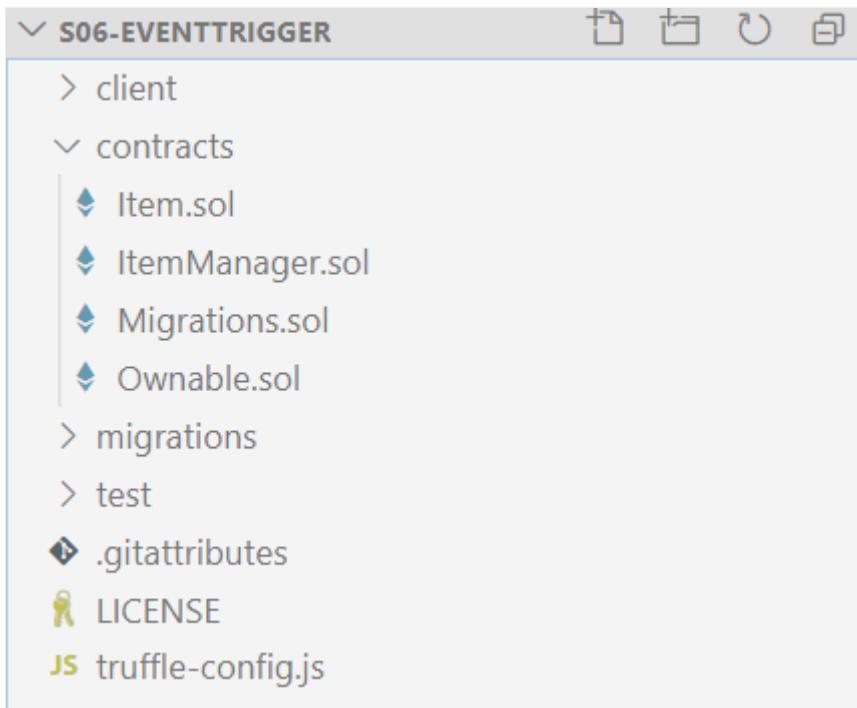
Last update: April 17, 2021

15.6 Add Contracts

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"
    }
  }
};
```

Compiler Versions

You can choose the version to be an exact version like "0.6.4", or you could add the "^" to specify all versions above greater or equal to 0.6.0, which means, it will download the latest 0.6.x version.

Run the truffle develop console to check if everything is alright and can be migrated. On the terminal/powershell run

```
truffle develop
```

and then simply type in

```
migrate
```

```
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: 'development'
```

Last update: April 17, 2021

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 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.returnValue._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

```
npm start
```

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

The screenshot shows a web application titled "Simply Payment/Supply Chain Exam". The main heading is "Items". Below it is a sub-section titled "Add Element". There is a form with two input fields: "Cost: 0" and "Item Name: exampleitem1". A "Create" button is located to the right of the item name field. The entire application is styled with dark colors and bold text.

```
➤ 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.
```

See an Error?

If you see an error message that the network wasn't found or the contract wasn't found under the address provided -- don't worry: Follow along in the next step where you change the network in MetaMask! As long as there is no error in our terminal and it says "Compiled successfully" you're good to go!

Last update: April 17, 2021

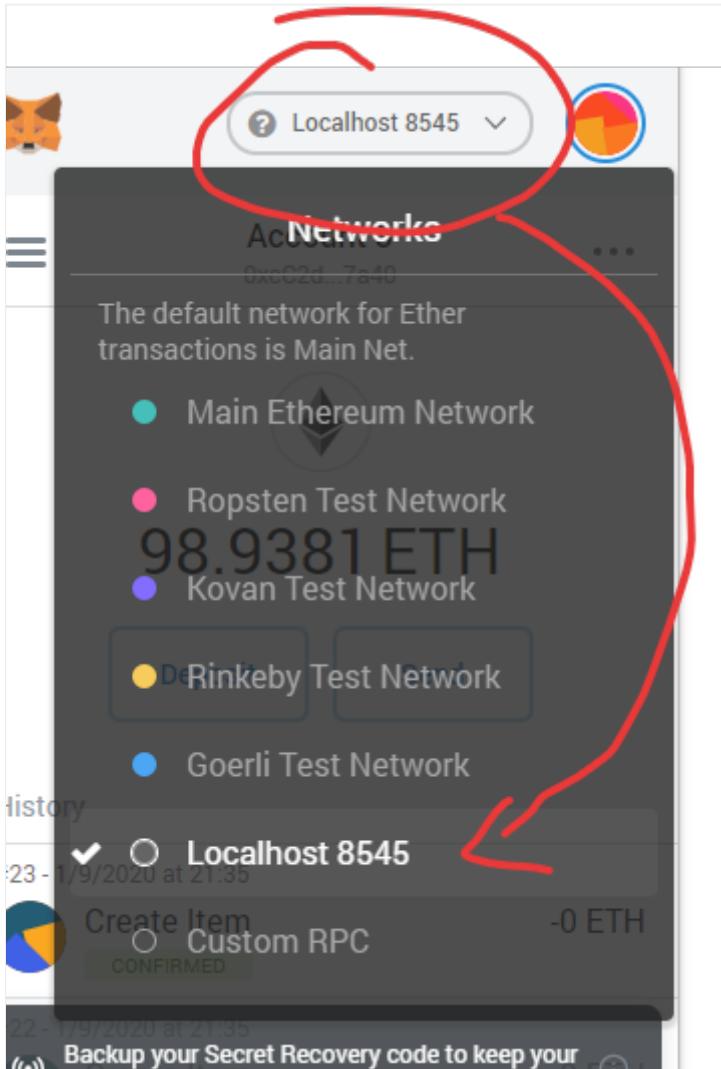
15.8 Connect with MetaMask

15.8.1 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.

15.8.2 Steps to follow

First, connect with MetaMask to the right network:



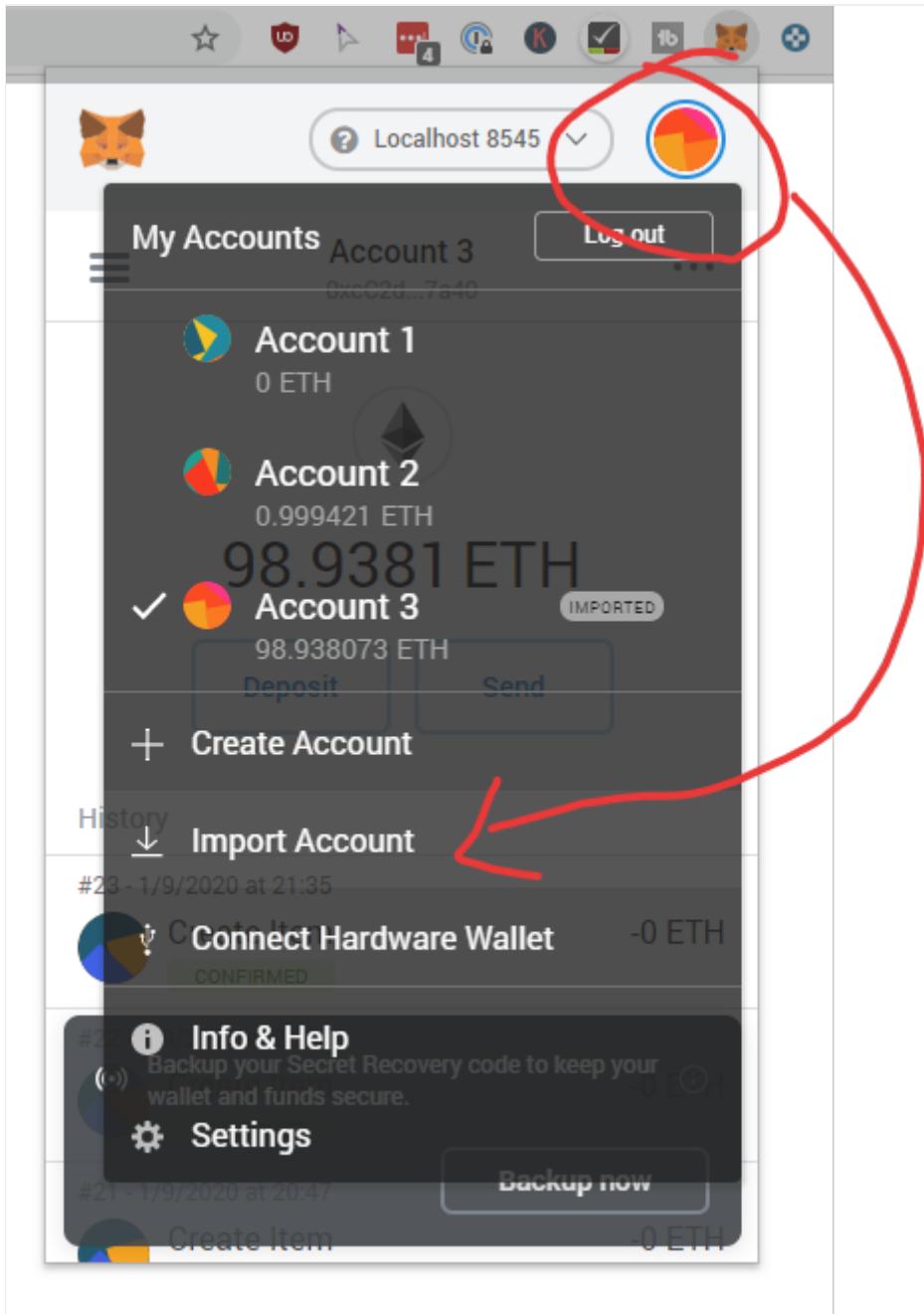
When we migrate the smart contracts with Truffle Developer console, then the first account in the truffle developer console is the "owner". So, either we disable MetaMask in the Browser to interact with the app or we add in the private key from truffle developer console to MetaMask.

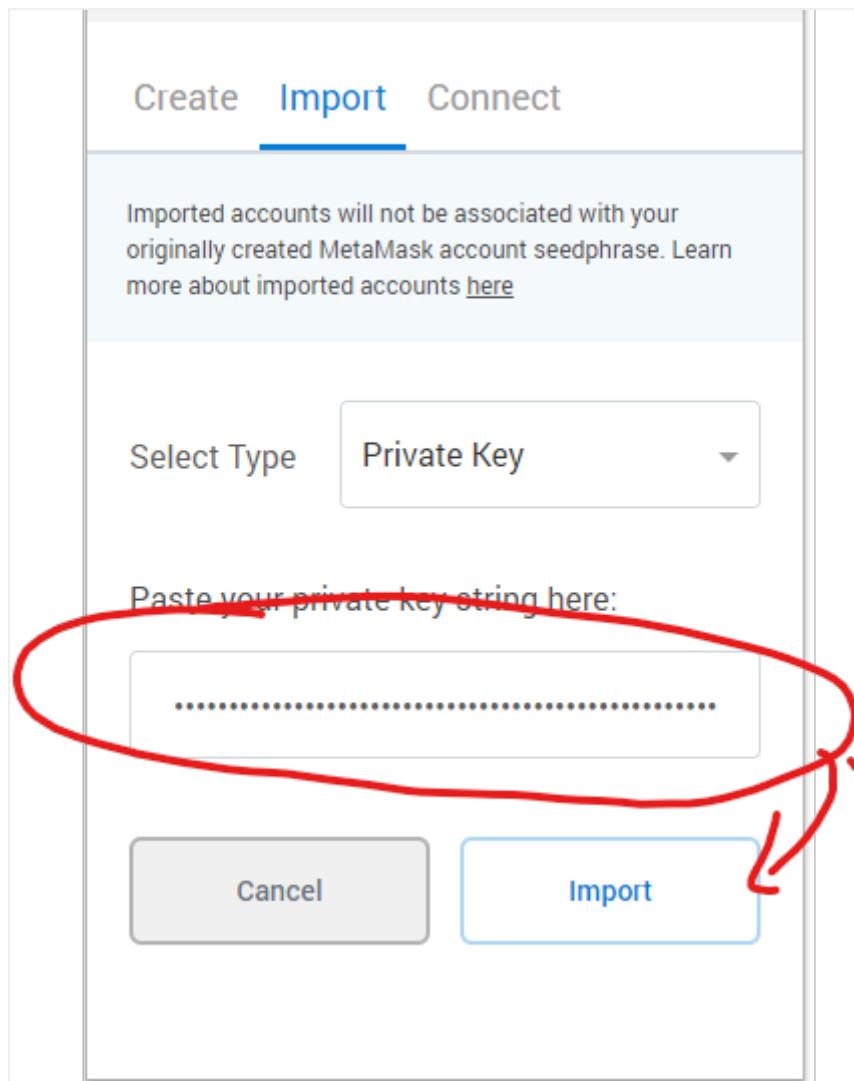
In the Terminal/Powershell where Truffle Developer Console is running scroll to the private keys on top:

Private Keys:

(0) 2a9ed36cdb66f81093a82443c2b9f237f3534ef75f4f044fa6ebd76d5d05f1
(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.

localhost:3000 says
Send 3425 Wei to 0x56da29C90CD9FaAB2567EA2077a7823aA229cDe5

OK

Add Items

Cost in Wei: 3425 Item Identifier: example_1 Create new Item

Last update: April 17, 2021

15.9 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:

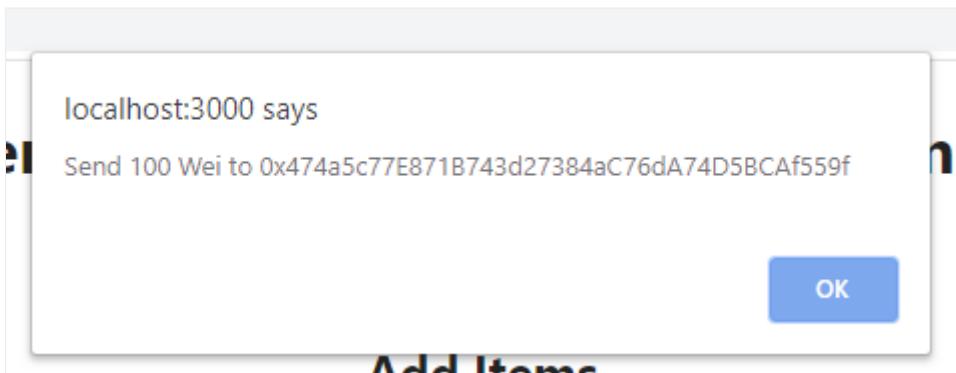
```
listenToPaymentEvent = () => {
  let self = this;
  this.itemManager.events.SupplyChainStep().on("data", async function(evt) {
    if(evt.returnValue._step == 1) {
      let item = await self.itemManager.methods.items(evt.returnValue._itemIndex).call();
      console.log(item);
      alert("Item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

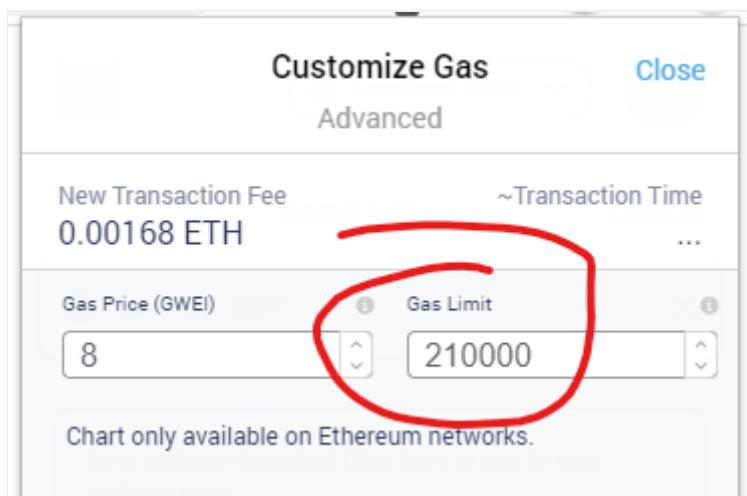
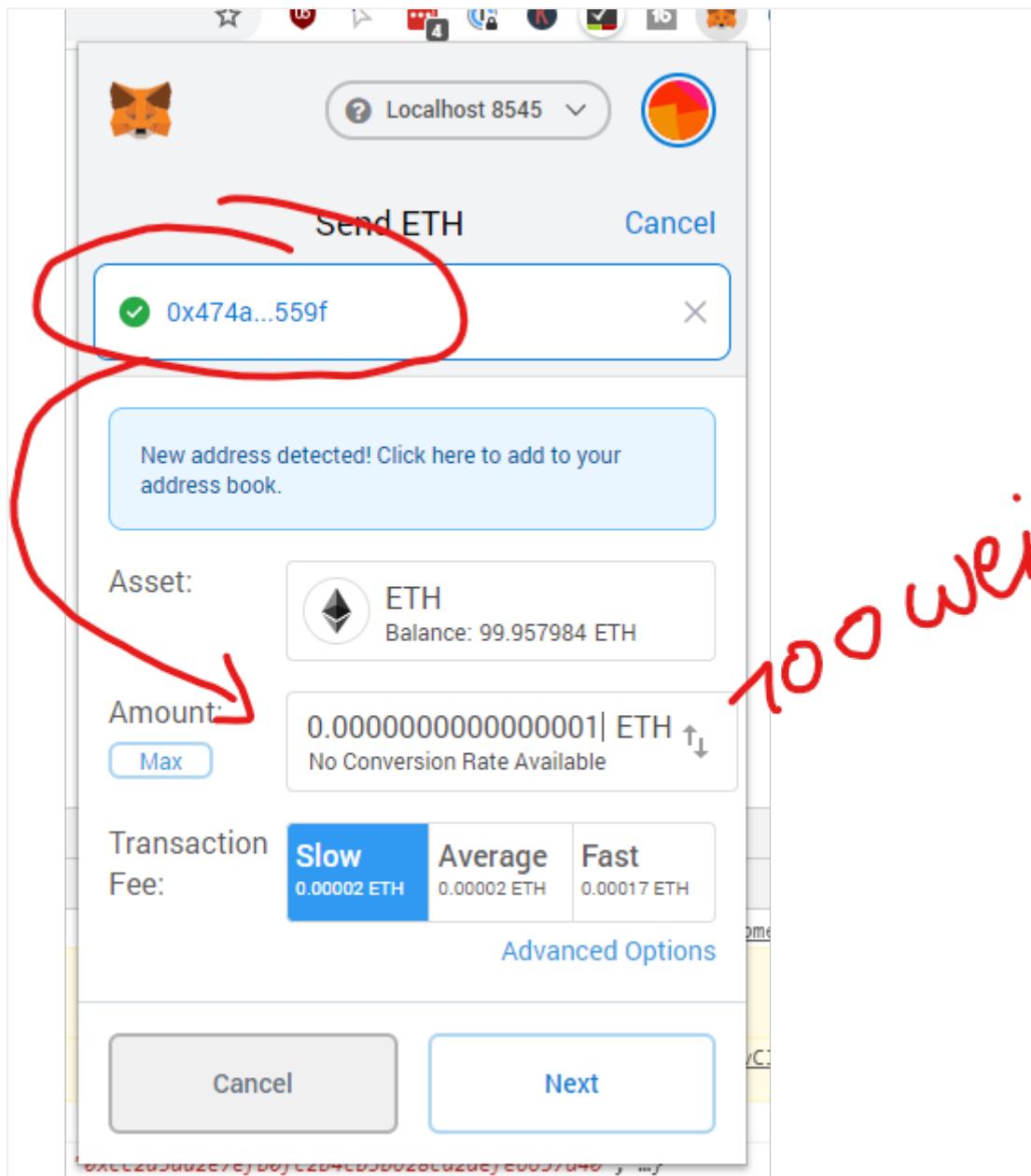
And call this function when we initialize the app in "componentDidMount":

```
//...
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:

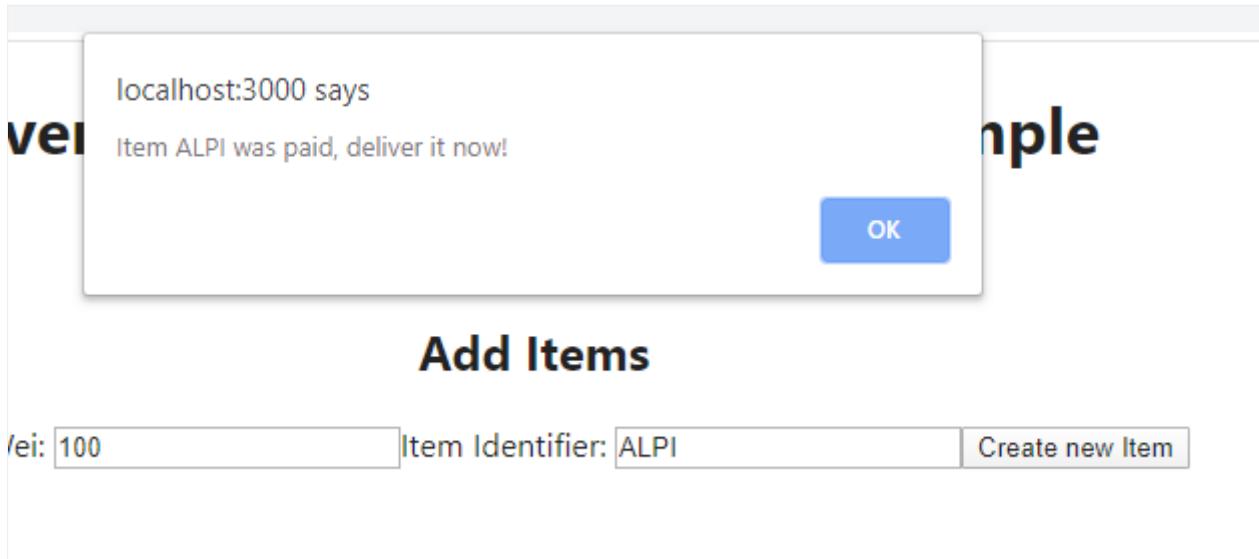




Take the address, give it to someone telling them to send 100 wei (0.0000000000000001 Ether) and a bit more gas to the specified address. You can do this either via MetaMask or via the truffle console:

```
web3.eth.sendTransaction({to: "ITEM_ADDRESS", value: 100, from: accounts[1], gas: 2000000});
```

Then a popup should appear on the website



Last update: April 17, 2021

15.10 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 truffle-contract 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:

```
truffle test
```

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.

Last update: April 17, 2021

15.11 Congratulations

Congratulations, LAB is completed

From the Course "Ethereum Blockchain Developer -- Build Projects in Solidity"

FULL COURSE: <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

Last update: April 17, 2021

16. LAB: ERC20 Token Sale

16.1 Project Tokenization Overview

The screenshot shows a web page for a token sale. At the top, it says "Capuccino Token for StarDucks". Below that is a button labeled "Enable your account". A text input field contains the address "0x123", and a button next to it says "Add Address to Whitelist". Below this is a section titled "Buy Cappucino-Tokens". It asks to "Send Ether to this address: 0x18d9d5d60c6063a63Bd424D59dbbcBE0DA233BC6". It shows the user has "You have: 0" tokens, and there is a button labeled "Buy more tokens".

16.1.1 Real-World Use-Case for this Project

Tokenization of any Assets as fungible Tokens (ERC20)

Creation of Bonus Programs, Vouchers, etc.

Creation of a new crypto currency

Creation of a Payment-layer on top of Ethereum

16.1.2 Development-Goal

Understand truffle-config json file

Understand deployment of dApps

Understand Tokenization using Open-Zeppelin Smart Contracts

Deeper dive into Unit-Testing

16.1.3 Videos

Full Video Walkthrough <https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96>

16.1.4 Get Started

Let's get started by [Truffle](#)

Last update: April 17, 2021

16.2 Truffle Initialization

Let's install Truffle and initialize a new project.

16.2.1 Truffle Installation

Before we get started, let's make sure we have the latest version of truffle installed:

```
console
npm install -g truffle
```

16.2.2 Project Initialization

Then create a new folder, "cd" into it and unbox the react-box from truffle:

```
console
mkdir s06_tokenization
cd s06_tokenization
truffle unbox react
```

Having troubles?

If you are experiencing troubles with truffle and the error has things like "npm ERR! gyp info using node@16.3.0 | darwin | arm64" then have a look at your node version. Truffle has troubles with node version 16 - install node 14 instead.

If truffle installs fine, but you are experiencing troubles running truffle, because something with "remote signed execution policy" comes up in PowerShell then run `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser` in a PowerShell window.

Another thing that can cause errors is directories with spaces. Make sure you are working from a directory without spaces

For anything else google the error message, it should bring you a solution, and if it doesn't reach out in the course Q&A, we're on standby to help out.

Running in WSL2?: One person solved it by first initializing a new project in the folder with `npm init -y`

Macbook (pro) M1 Users

Some people reported problems with Macbook M1 and Truffle developer console. Quote from one of the Students: "This program works only on node 10, 12 or 14. And in case you are lucky ones and got Macbook pro M1 then you wont be able to run 'Truffle developer or truffle develop' command with node 14. If you downgrade to node 12 and run 'truffle develop' command then you would get " wasm assemble issue ... memory over run error" ... so better you keep node 14.17.0 and run the program on Ganache ... as truffle support on node 14 version does not run the truffle develop command successfully."

And leave only the scaffolding for a blank new project:

1. Remove all contracts in the "/contracts" folder, except the Migrations.sol file.
2. Remove all the tests in the "/tests" folder.
3. Remove all migrations except the 01_intial_migration.js.

Now, let's get started with the solidity part!

Last update: June 23, 2021

16.3 ERC20 Smart Contract

The first smart contract is the Token. We don't invent it ourselves, we take the ERC20 Smart Contract from OpenZeppelin. In this version open-zeppelin v3, with Solidity 0.6 smart contracts:

16.3.1 Installation

In the console type:

```
npm install --save @openzeppelin/contracts@v3.0.0
```

Update from the Video

Note we will be using the v3.0.0 of openzeppelin contracts, instead of v3.0.0-beta.0

16.3.2 Possible Scenario

Let's think about a possible work-scenario. We will create a Token which let's you redeem a coffee at your favorite Coffee Store: StarDucks Coffee from Duckburg. It will look slightly different from other ERC20 tokens, since a coffee is hardly divisible, but still transferrable. Let's create our Token.

16.3.3 Adding the Token

Add a "MyToken.sol" file to the "/contracts" folder:

```
/contracts/MyToken.sol

pragma solidity >=0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("StarDucks Capu-Token", "SCT") public {
        _mint(msg.sender, initialSupply);
        _setupDecimals(0);
    }
}
```

Note

The ERC20Detailed has been deprecated and combined into the ERC20 contract default. The new ERC20 contract has a constructor with arguments for the name and symbol of the token. It has a name "StarDucks Capu-Token", a symbol "SCT". The new ERC20 contract has a default decimal points of 18, we can change it to decimal points of 0 in the constructor by calling the `setupDecimals(uint8 decimals)` function in the ERC20 contract, with 0 as the argument.

Last update: April 17, 2021

16.4 Migration and Compilation

Let's see if we can deploy the smart contract to a developer-only test blockchain.

16.4.1 Configuring Migrations

Add in a migration in "migrations/2_deploy_contracts.js":

```
migrations/2_deploy_contracts.js
```

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

module.exports = async function(deployer) {
    await deployer.deploy(MyToken, 1000000000);
};
```

Also lock in the compiler version in truffle-config.js:

```
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"
    }
}
};
```

16.4.2 Running Migrations

Try to run this in the truffle developer console:

```
truffle develop
```

and then simply type in

```
migrate
```

Last update: April 17, 2021

16.5 Unit Test ERC20

To test the token, we want to change our usual setup a bit. Let's use chai's expect to test the transfer of tokens from the owner to another account.

16.5.1 Install Chai

First we need some additional npm packages:

```
npm install --save chai chai-bn chai-as-promised
```

Then create our test in the tests folder. Create a new file called /tests/MyToken.test.js:

```
/tests/MyToken.test.js

const Token = artifacts.require("MyToken");

var chai = require("chai");

const BN = web3.utils.BN;
const chaiBN = require('chai-bn')(BN);
chai.use(chaiBN);

var chaiAsPromised = require("chai-as-promised");
chai.use(chaiAsPromised);

const expect = chai.expect;

contract("Token Test", async accounts => {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  it("All tokens should be in my account", async () => {
    let instance = await Token.deployed();
    let totalSupply = await instance.totalSupply();
    //old style:
    //let balance = await instance.balanceOf.call(initialHolder);
    //assert.equal(balance.valueOf(), 0, "Account 1 has a balance");
    //condensed, easier readable style:
    await expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply);
  });
});
```

Note

The next step would be testing, but the truffle version I was using has problems with the internal developer network from truffle. See this screenshot:

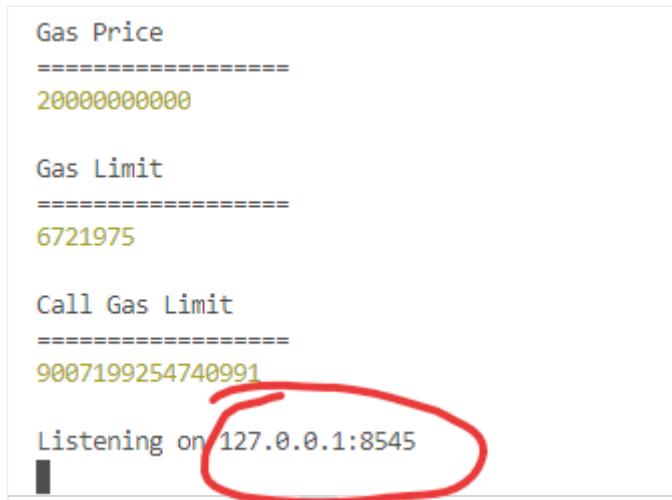
```
TypeError [ERR_INVALID_REPL_INPUT]: Listeners for `uncaughtException` cannot be used in the REPL
  at process.<anonymous> (repl.js:227:15)
  at process.emit (events.js:215:7)
  at process.emit (C:\Users\thoma\AppData\Roaming\npm\node_modules\truffle\build\webpack:\node_modules\truffle\lib\runner\index.js:11:1)
  at _addListener (events.js:236:14)
  at process.addListener (events.js:284:10)
  at Runner.run (C:\Users\thoma\AppData\Roaming\npm\node_modules\truffle\node_modules\mocha\lib\runner.js:11:1)
  at Mocha.run (C:\Users\thoma\AppData\Roaming\npm\node_modules\truffle\node_modules\mocha\lib\mocha.js:11:1)
  at C:\Users\thoma\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\core\lib\test.js:11:1)
  at new Promise (<anonymous>)
  at Object.run (C:\Users\thoma\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\core\lib\test.js:11:1)
  at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

So, in order to make this work, I had to use Ganache + Truffle.

1. Open Ganache
2. Adjust the truffle-config.js file
3. Run the tests

16.5.2 Open Ganache GUI or Ganache-cli

If you want to strictly stay on the command line, then install ganache-cli (npm install -g ganache-cli) and run it from the command line. Mind the PORT number, which we need in the next step!



```
Gas Price
=====
20000000000

Gas Limit
=====
6721975

Call Gas Limit
=====
9007199254740991

Listening on 127.0.0.1:8545
```

Ganache CLI Output

Otherwise roll with the GUI version of Ganache, which runs on Port 7545 usually (but double-check!)

Ganache						
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS	
CURRENT BLOCK 0	GAS PRICE 200000000000	GAS LIMIT 6721975	HARDFORK PETERSBURG	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING
MNEMONIC <small>?</small>	nasty brush upper leisure pioneer social swallow father saddle side mixture method					
ADDRESS 0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6	BALANCE 100.00 ETH					
ADDRESS 0xEddC05d9B2CB4D3965ee9d01C0c47960FAAbf273	BALANCE 100.00 ETH					
ADDRESS 0xD152FdAea0F5c39bD38Ce6c29312af14B93F40E3	BALANCE 100.00 ETH					
ADDRESS 0xf2Af5a487b989EcFd4F6310E2A964c1649C0CD43	BALANCE 100.00 ETH					
ADDRESS 0x4Fff1eD88e721140729fd3C179c807984bc70dfe	BALANCE 100.00 ETH					
ADDRESS 0xc0BBc1Eb8F66b494B08Ec843e12742c351499fDa	BALANCE 100.00 ETH					
ADDRESS 0x4AC266A48d587e541b6f6520246Ec1Fd94B9F6bd	BALANCE 100.00 ETH					

Ganache UI Output

16.5.3 Adjust the truffle-config.js file

If you are running Ganache-GUI then adjust the truffle-config.js file, so that the default development network is going to use the right host and port.

Also make sure the solc-version is the correct one and lock it in, if you haven't already:

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: {
  development: {
    port: 7545,
    network_id: "*",
    host: "127.0.0.1"
  }
},
compilers: {
  solc: {
    version: "^0.6.0",
  }
}
};
```

16.5.4 Test the Smart Contract

By going to your console, to the root folder of the project, and typing in `truffle test`, you will call the Truffle testing suite. If all goes well it will give you this output:

```
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL

> Compiling .\contracts\KycContract.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\MyToken.sol
> Compiling .\contracts\MyTokenSale.sol
> Artifacts written to C:\Users\thoma\AppData\Local\Temp\test-2020121-3016-435ywg.1c9vd
> Compiled successfully using:
  - solc: 0.6.1+commit.e6f7d5a4.Emscripten clang

Contract: Token Test
  ✓ All tokens should be in my account

1 passing (116ms)
```

Truffle Test Output

16.5.5 Add more Tests to your Token-Test

Add some more tests to make sure everything works as expected.

```
// add this into the contract token test:
it("I can send tokens from Account 1 to Account 2", async () => {
  const sendTokens = 1;
  let instance = await Token.deployed();
  let totalSupply = await instance.totalSupply();
  await expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply);
  await expect(instance.transfer(recipient, sendTokens)).to.eventually.be.fulfilled;
  await expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply.sub(new BN(sendTokens)));
  await expect(instance.balanceOf(recipient)).to.eventually.be.a.bignumber.equal(new BN(sendTokens));
});
```

```

it("It's not possible to send more tokens than account 1 has", async () => {
  let instance = await Token.deployed();
  let balanceOfAccount = await instance.balanceOf(initialHolder);
  await expect(instance.transfer(recipient, new BN(balanceOfAccount+1))).to.eventually.be.rejected;

  //check if the balance is still the same
  await expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(balanceOfAccount);

});

//...

```

And run it again:

```
> Compiled successfully using:
- solc: 0.6.1+commit.e6f7d5a4.Emscripten.clang
```

Contract: Token Test

- ✓ All tokens should be in my account
- ✓ I can send tokens from Account 1 to Account 2 (39ms)
- ✓ It's not possible to send more tokens than account 1 has (157ms)

3 passing (328ms)

Truffle Test Output chai-as-promised

Last update: June 24, 2021

16.6 Add Crowdsale Contracts

Here we do two things:

1. We adapt the old Crowdsale Smart Contract from Open-Zeppelin to be Solidity 0.6 compliant
2. We write our own Crowdsale on Top of it

Note

With OpenZeppelin approaching Solidity 0.6 the Crowdsale contracts were removed. Some people are inclined to add a "mintToken" functionality or something like that to the Token Smart Contract itself, but that would be bad design. We should add a separate Crowdsale Contract that handles token distribution.

Let's modify the Crowdsale Contract from Open-Zeppelin 2.5 to be available for Solidity 0.6:

Copy the contents from this file <https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/crowdsale/Crowdsale.sol> to /contracts/Crowdsale.sol and change a few lines to make it project compliant.

1. The pragma line and the import statements
2. The fallback function
3. The virtual Keyword

16.6.1 The Pragma line and the Import Statements

If we copy the smart contract out of another repository instead of just using it, then we have to adjust the import statements. Replace the existing ones with this:

```
contracts/Crowdsale.sol

pragma solidity ^0.6.0;

import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import "@openzeppelin/contracts/Context.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
```

16.6.2 The Fallback Function for Solidity 0.6

The unnamed fallback function is gone. We need to replace the function() with a receive function, since it will be possible to send Ether directly to our smart contract without really interacting with it.

Replace the "fallback ()" function with this one:

```
receive () external payable {
    buyTokens(_msgSender());
}
```

16.6.3 The Virtual Keyword in Solidity 0.6

If you want to override functions in Solidity 0.6 then the base smart-contract must define all functions as virtual to be overwritten. In the Crowdsale we must add the virtual keyword to functions that are potentially overwritten:

```
function _preValidatePurchase(address beneficiary, uint256 weiAmount) internal view virtual {
    require(beneficiary != address(0), "Crowdsale: beneficiary is the zero address");
    require(weiAmount != 0, "Crowdsale: weiAmount is 0");
    this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
```

```

    }

    function _postValidatePurchase(address beneficiary, uint256 weiAmount) internal view virtual {
        // solhint-disable-previous-line no-empty-blocks
    }

    function _deliverTokens(address beneficiary, uint256 tokenAmount) internal virtual {
        _token.safeTransfer(beneficiary, tokenAmount);
    }

    function _processPurchase(address beneficiary, uint256 tokenAmount) internal virtual {
        _deliverTokens(beneficiary, tokenAmount);
    }

    function _updatePurchasingState(address beneficiary, uint256 weiAmount) internal virtual {
        // solhint-disable-previous-line no-empty-blocks
    }

    function _getTokenAmount(uint256 weiAmount) internal view virtual returns (uint256) {
        return weiAmount.mul(_rate);
    }
}

```

16.6.4 Create your own Crowdsale Contract

Add in a contracts/MyTokenSale.sol file with the following content:

contracts/MyTokenSale.sol

```

pragma solidity ^0.6.0;

import "./Crowdsale.sol";

contract MyTokenSale is Crowdsale {

    KyCContract kyc;
    constructor(
        uint256 rate,    // rate in TKNbits
        address payable wallet,
        IERC20 token
    )
        Crowdsale(rate, wallet, token)
    public
    {
    }

}

```

16.6.5 Adopt the Migration for the Crowdsale Contract

In order for our crowdsale smart contract to work, we must send all the money to the contract. This is done on the migrations stage in our truffle installation:

The problem is now that the Test is failing. Let's change the standard Truffle-Test-Suite to the openzeppelin test suite:

migrations/2_deploy_contracts.js

```

var MyToken = artifacts.require("./MyToken.sol");
var MyTokenSales = artifacts.require("./MyTokenSale.sol");

module.exports = async function(deployer) {
let addr = await web3.eth.getAccounts();
await deployer.deploy(MyToken, 1000000000);
await deployer.deploy(MyTokenSales, 1, addr[0], MyToken.address);
let tokenInstance = await MyToken.deployed();
await tokenInstance.transfer(MyTokenSales.address, 1000000000);

};

```

Perfect, now that we have that covered, let's have a look at the unit tests again.

Last update: April 17, 2021

16.7 Change the UnitTests To Support TokenSales

The truffle tests setup is not really suitable if you want to test specific scenarios which are not covered by the migration files. After migrating a smart contract, it usually ends up in a specific state. So testing the Token Smart Contract in this way wouldn't be possible anymore. You would have to test the whole token-sale, but that's something not what we want.

We could also integrate the openzeppelin test environment. It's blazing fast and comes with an internal blockchain for testing. But it has one large drawback: It only let's you use the internal blockchain, it's not configurable so it would use an outside blockchain. That's why I would still opt to use the Truffle Environment.

We just have to make a small change:

Update /tests/MyToken.test.js

```
//... chai token setup

contract("Token Test", function(accounts) {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  beforeEach(async () => {
    this.myToken = await Token.new(1000);
  });

  it("All tokens should be in my account", async () => {
    //let instance = await Token.deployed();
    let instance = this.myToken;
    let totalSupply = await instance.totalSupply();
    //... more content
  });

  it("I can send tokens from Account 1 to Account 2", async () => {
    const sendTokens = 1;
    let instance = this.myToken;
    let totalSupply = await instance.totalSupply();
    //... more content
  });

  it("It's not possible to send more tokens than account 1 has", async () => {
    let instance = this.myToken;
    //... more content
  });
});
```

Now open your Terminal and test the smart contract:

```
Contract: Token Test
  ✓ All tokens should be in my account
  ✓ I can send tokens from Account 1 to Account 2
  ✓ It's not possible to send more tokens than account 1 has

  3 passing (688ms)

s06 - tokenization> [ ]
```

16.7.1 Add in a Central Configuration with DotEnv

One of the larger problems is that we now have a constant for the migrations-file and a constant in our test -- the amount of tokens that are created. It would be better to have this constant through an environment file.

Install Dot-Env:

```
npm install --save dotenv
```

Then create a new file .env in your root directory of the project with the following content:

```
.env
```

```
INITIAL_TOKENS = 10000000
```

Then change the migrations file to:

```
migrations/2_deploy_contracts.js
```

```
var MyToken = artifacts.require("./MyToken.sol");
var MyTokenSales = artifacts.require("./MyTokenSale.sol");
require('dotenv').config({path: '../.env'});

module.exports = async function(deployer) {
let addr = await web3.eth.getAccounts();
await deployer.deploy(MyToken, process.env.INITIAL_TOKENS);
await deployer.deploy(MyTokenSales, 1, addr[0], MyToken.address);
let tokenInstance = await MyToken.deployed();
await tokenInstance.transfer(MyTokenSales.address, process.env.INITIAL_TOKENS);

};
```

Update also the tests file:

```
Update /tests/MyToken.test.js
```

```
const Token = artifacts.require("MyToken");

// Rest of the code ...

require('dotenv').config({path: '../.env'});

contract("Token Test", function(accounts) {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  beforeEach(async () => {
    this.myToken = await Token.new(process.env.INITIAL_TOKENS);
  });

//Rest of the Code ...
```

Now run the tests again and make sure everything still works as expected! All the tests, as well as the migration itself have one single point of truth. That is the .env file.

Last update: April 17, 2021

16.8 Crowdsale Unit-Test

Now, let's test our Crowdsale Token. Create a new file in /tests/MyTokenSale.test.js:

/tests/MyTokenSale.test.js

```
const Token = artifacts.require("MyToken");
const TokenSale = artifacts.require("MyTokenSale");

var chai = require("chai");
const expect = chai.expect;

const BN = web3.utils.BN;
const chaiBN = require('chai-bn')(BN);
chai.use(chaiBN);

var chaiAsPromised = require("chai-as-promised");
chai.use(chaiAsPromised);

contract("TokenSale", async function(accounts) {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  it("there shouldnt be any coins in my account", async () => {
    let instance = await Token.deployed();
    expect(instance.balanceOf.call(initialHolder)).to.eventually.be.a.bignumber.equal(new BN(0));
  });
});
```

If you run this, it will give you an error:

```
4) Contract: TokenSale
  there shouldnt be any coins in my account:
AssertionError: expected { _events: {},
emit: [Function: emit],
on: [Function: on],
once: [Function: once],
off: [Function: removeListener],
listeners: [Function: listeners],
addListener: [Function: on],
removeListener: [Function: removeListener],
removeAllListeners: [Function: removeAllListeners] } to be an instance of BN or string
```

Problem is: this won't work out of the box for two reasons.

1. The shared Chai setup and
2. The missing return statements in the previous smart contract.

16.8.1 General Setup for Chai and Chai-as-Promised

A note on the Videos and truffle-assertions

In the videos I am mentioning that you need to return the `expect()....`. An attentive student asked where to find more about this, as it seems to be undocumented.

This is where I believe it comes from: If you look at [Chai-As-Promised](#) then "should.eventually.be" will return a promise, which means the testing framework needs to be informed about a pending promise. This is the actual example on their website:

```
return doSomethingAsync().should.eventually.equal("foo");
```

Having said that, I am not 100% convinced that it's necessary (anymore) since it also works without the return in most cases.

In the meantime I found another wrapper which I can wholeheartedly recommend: Truffle-Assertions. So, as an alternative (or in addition), check out <https://github.com/rkalis/truffle-assertions>, they are easy to use and cover pretty much anything you will probably come across to test for in Solidity.

Create a new file in tests/chaisetup.js with the following content:

tests/chaisetup.js

```
"use strict";
var chai = require("chai");
const expect = chai.expect;

const BN = web3.utils.BN;
const chaiBN = require('chai-bn')(BN);
chai.use(chaiBN);

var chaiAsPromised = require("chai-as-promised");
chai.use(chaiAsPromised);
module.exports = chai;
```

Then update the tests/Token.test.js file. Mind the "return" keywords:

Update tests/Token.test.js

```
const Token = artifacts.require("MyToken");

const chai = require("./chaisetup.js");
const BN = web3.utils.BN;
const expect = chai.expect;

require('dotenv').config({path: '../.env'});

contract("Token Test", function(accounts) {
// rest of the code...
  it("All tokens should be in my account", async () => {
// rest of the code...
    return expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply);
  });
  it("I can send tokens from Account 1 to Account 2", async () => {
// rest of the code...
    return expect(instance.balanceOf(recipient)).to.eventually.be.a.bignumber.equal(new BN(sendTokens));
  });
  it("It's not possible to send more tokens than account 1 has", async () => {
    return expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(balanceOfAccount);
  });
});
```

And then fix the TokenSale.test.js:

Update /tests/TokenSale.test.js

```
const Token = artifacts.require("MyToken");
const TokenSale = artifacts.require("MyTokenSale");

const chai = require("./chaisetup.js");
const BN = web3.utils.BN;
const expect = chai.expect;

contract("TokenSale", async function(accounts) {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  it("there shouldnt be any coins in my account", async () => {
    let instance = await Token.deployed();
    return expect(instance.balanceOf.call(initialHolder)).to.eventually.be.a.bignumber.equal(new BN(0));
  });
});
```

Run the tests:

```
Contract: Token Test
✓ All tokens should be in my account (66ms)
✓ I can send tokens from Account 1 to Account 2 (90ms)
✓ It's not possible to send more tokens than account 1 has (65ms)

Contract: TokenSale
✓ there shouldnt be any coins in my account (69ms)

4 passing (887ms)
```

16.8.2 Add more Unit-Tests for actually purchasing a Token

In the tests/TokenSale.test.js add the following:

Add in tests/TokenSale.test.js

```
//other code in test

it("all coins should be in the tokensale smart contract", async () => {
  let instance = await Token.deployed();
  let balance = await instance.balanceOf.call(TokenSale.address);
  let totalSupply = await instance.totalSupply.call();
  return expect(balance).to.be.a.bignumber.equal(totalSupply);
});

it("should be possible to buy one token by simply sending ether to the smart contract", async () => {
  let tokenInstance = await Token.deployed();
  let tokenSaleInstance = await TokenSale.deployed();
  let balanceBeforeAccount = await tokenInstance.balanceOf.call(recipient);

  await expect(tokenSaleInstance.sendTransaction({from: recipient, value: web3.utils.toWei("1", "wei"})).to.be.fulfilled;
  return expect(balanceBeforeAccount + 1).to.be.bignumber.equal(await tokenInstance.balanceOf.call(recipient));
});
```

Run the tests and it should work:

Contract: Token Test

- ✓ All tokens should be in my account (39ms)
- ✓ I can send tokens from Account 1 to Account 2 (84ms)
- ✓ It's not possible to send more tokens than account 1 has (69ms)

Contract: TokenSale

- ✓ there shouldnt be any coins in my account (58ms)
- ✓ all coins should be in the tokensale smart contract (43ms)
- ✓ should be possible to buy one token by simply sending ether to the smart contract (351ms)

6 passing (1s)

s06 - tokenization> █

Errors?

If you are running into troubles, unexpected errors, try to restart Ganache!

In the next step we model some sort of Know-Your-Customer Whitelisting Smart Contract. This can be a mockup for a larger KYC solution. But in our case, it will just whitelist addresses by the admin of the system.

Last update: June 24, 2021

16.9 Add in a Kyc Mockup

KYC, or "know your customer", is necessary for many different applications nowadays. In its simplest form, it's just a whitelist, where, based on some criteria, someone gets the permission to do something.

Let's add a simple whitelist, or KYC functionality.

16.9.1 The KYC Smart Contract

First, we're going to add a KYC Smart Contract which handles the white-listing. In contracts/KycContract.sol add the following content.

contracts/KycContract.sol

```
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/access/Ownable.sol";

contract KycContract is Ownable {
    mapping(address => bool) allowed;

    function setKycCompleted(address _addr) public onlyOwner {
        allowed[_addr] = true;
    }

    function setKycRevoked(address _addr) public onlyOwner {
        allowed[_addr] = false;
    }

    function kycCompleted(address _addr) public view returns(bool) {
        return allowed[_addr];
    }
}
```

And in our TokenSale.sol we have to check -- before the actual sale -- if the user is whitelisted. Change the contracts/MyTokenSale.sol to:

contracts/MyTokenSale.sol

```
pragma solidity ^0.6.0;

import "./Crowdsale.sol";
import "./KycContract.sol";

contract MyTokenSale is Crowdsale {

    KycContract kyc;
    constructor(
        uint256 rate, // rate in TKNbits
        address payable wallet,
        IERC20 token,
        KycContract _kyc
    )
        Crowdsale(rate, wallet, token)
    public
    {
        kyc = _kyc;
    }

    function _preValidatePurchase(address beneficiary, uint256 weiAmount) internal view override {
        super._preValidatePurchase(beneficiary, weiAmount);
        require(kyc.kycCompleted(beneficiary), "KYC not completed yet, aborting");
    }
}
```

And now we also have to change the migration obviously, or else it won't work:

migrations/02_deploy_contracts.js

```
var MyToken = artifacts.require("./MyToken.sol");
var MyTokenSales = artifacts.require("./MyTokenSale.sol");
var KycContract = artifacts.require("./KycContract.sol");
require('dotenv').config({path: '../.env'});

module.exports = async function(deployer) {
  let addr = await web3.eth.getAccounts();
  await deployer.deploy(MyToken, process.env.INITIAL_TOKENS);
  await deployer.deploy(KycContract);
  await deployer.deploy(MyTokenSales, 1, addr[0], MyToken.address, KycContract.address);
  let tokenInstance = await MyToken.deployed();
  await tokenInstance.transfer(MyTokenSales.address, process.env.INITIAL_TOKENS);

};

}
```

Now let's change also the Unit-Tests to reflect this:

tests/MyTokenSale.test.js

```
const Token = artifacts.require("MyToken");
const TokenSale = artifacts.require("MyTokenSale");
const KycContract = artifacts.require("KycContract");

const chai = require("./chaisetup.js");
const BN = web3.utils.BN;
const expect = chai.expect;

contract("TokenSale", async function(accounts) {
  const [ initialHolder, recipient, anotherAccount ] = accounts;

  //the rest of the code here

  it("should be possible to buy one token by simply sending ether to the smart contract", async () => {
    let tokenInstance = await Token.deployed();
    let tokenSaleInstance = await TokenSale.deployed();
    let balanceBeforeAccount = await tokenInstance.balanceOf.call(recipient);
    await expect(tokenSaleInstance.sendTransaction({from: recipient, value: web3.utils.toWei("1", "wei"})).to.be.rejected;
    await expect(balanceBeforeAccount).to.be.bignumber.equal(await tokenInstance.balanceOf.call(recipient));

    let kycInstance = await KycContract.deployed();
    await kycInstance.setKycCompleted(recipient);
    await expect(tokenSaleInstance.sendTransaction({from: recipient, value: web3.utils.toWei("1", "wei"})).to.be.fulfilled;
    return expect(balanceBeforeAccount + 1).to.be.bignumber.equal(await tokenInstance.balanceOf.call(recipient));
  });
});
```

Last update: June 24, 2021

16.10 Frontend: Load Contracts to React

Now it's time to load the contracts into the frontend. We haven't touched it yet, so let's get started by modifying the App.js file.

Let's modify the client/App.js file and add in the right contracts to import:

```
import React, { Component } from "react";
import MyToken from "./contracts/MyToken.json";
import MyTokenSale from "./contracts/MyTokenSale.json";
import KycContract from "./contracts/KycContract.json";
import getWeb3 from "./getWeb3";

import "./App.css";

class App extends Component {
//more code ...
```

MetaMask / Ganache Problems (or Changes)

Two problems here:

1. There is a problem with `this.web3.eth.net.getId()` using MetaMask. See [deprecated features](#) from MetaMask. Instead we need to use the ChainId. The Code below is already fixed!
2. Ganache has sometimes the wrong ChainId <https://github.com/trufflesuite/ganache-core/issues/575>. This means, at the time of writing this, we need to use ganache-cli with custom network id and chain id. Start ganache cli with `ganache-cli --networkId 1337 --chainId 1337`

Then change the state variable, as well as the componentDidMount function to load all the Smart Contracts using web3.js:

```
state = { 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.
    //this.networkId = await this.web3.eth.getId(); <- this doesn't work with MetaMask anymore
    this.networkId = await this.web3.eth.getChainId();

    this.myToken = new this.web3.eth.Contract(
      MyToken.abi,
      MyToken.networks[this.networkId] && MyToken.networks[this.networkId].address,
    );

    this.myTokenSale = new this.web3.eth.Contract(
      MyTokenSale.abi,
      MyTokenSale.networks[this.networkId] && MyTokenSale.networks[this.networkId].address,
    );
    this.kycContract = new this.web3.eth.Contract(
      KycContract.abi,
      KycContract.networks[this.networkId] && KycContract.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.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);
  }
};
```

Finally change the bottom part to listen for "loaded" instead of web3:

```
render() {
  if (!this.state.loaded) {
```

```
    return <div>Loading Web3, accounts, and contract...</div>;
}

return (
  <div className="App">
    <h1>Good to Go!</h1>
    <p>Your Truffle Box is installed and ready.</p>
    <h2>Smart Contract Example</h2>
    <p>
      If your contracts compiled and migrated successfully, below will show
      a stored value of 5 (by default).
    </p>
    <p>
      Try changing the value stored on <strong>line 40</strong> of App.js.
    </p>
    <div>The stored value is: {this.state.storageValue}</div>
  </div>
);
}
```

Start the development server and see if there are any obvious errors being thrown:

```
cd client && npm run start
```

First, if installed, MetaMask should ask you if you want to connect:

React App

localhost:3000

Loading Web3, accounts, and contract...

Network

Name	Status
bundle.js	200
0.chunk.js	200
main.chunk.js	200
info?t=1582638538364	200
manifest.json	200
favicon.ico	200
websocket	101
loan192.pnn	200

8 requests | 880 KB transferred | 3.8 MB resources

What's New

Debug why a cookie was blocked
Click a resource in the Network panel and go to the updated Cookies tab.

View cookie values
Click a row in the Cookies pane in the Application panel to see the cookie's value.

Simulate prefers-color-scheme and prefers-reduced-motion preferences
Open the Rendering tab to force your site into dark or light mode or set motion preferences.

MetaMask Notification

Ropsten Test Network

Connect Request

React App localhost

Account 3

React App would like to connect to your account

This site is requesting access to view your current account address. Always make sure you trust the sites you interact with.

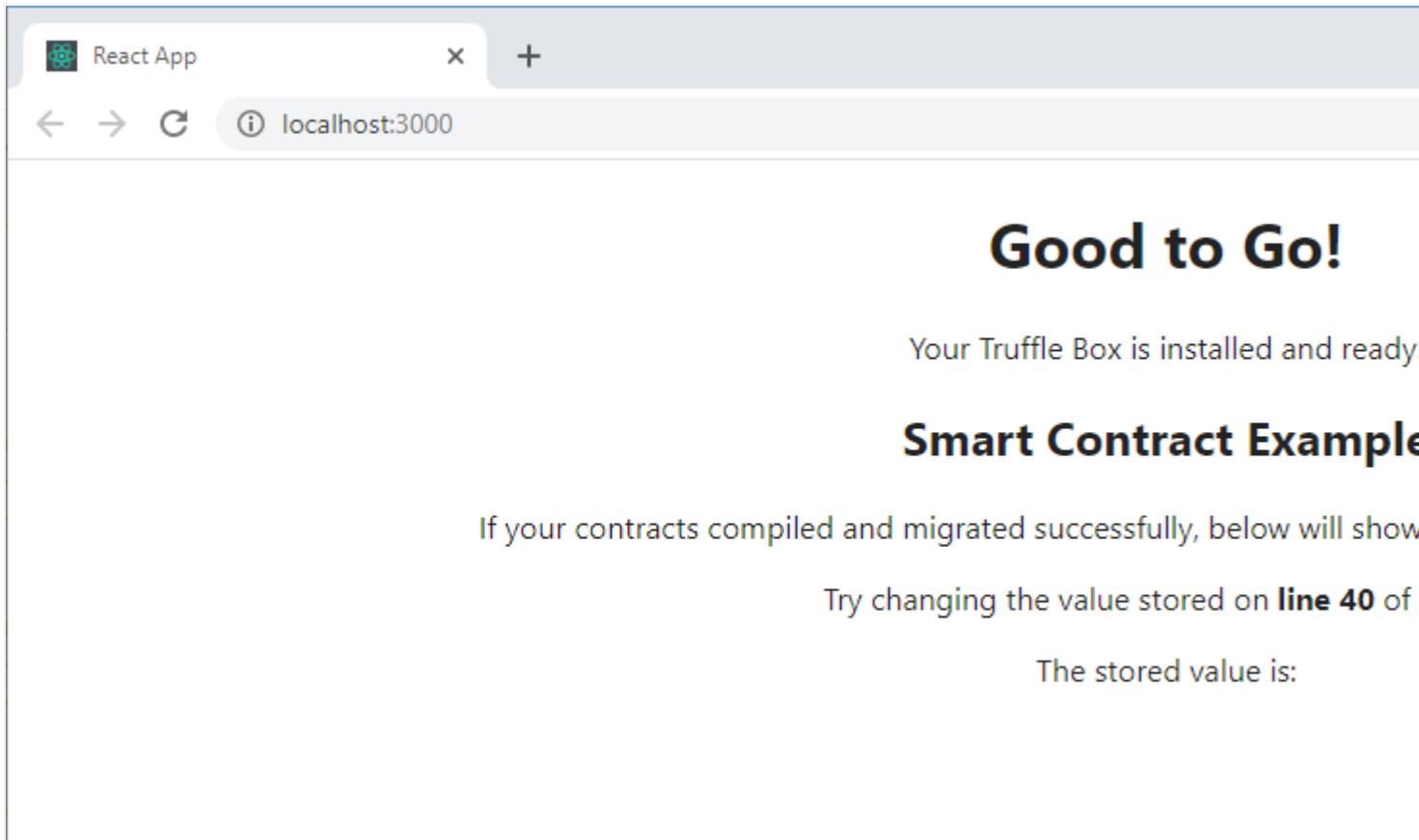
Learn more.

MetaMask Secure

Cancel Connect

315/447 Copyright © 2016 - 2021 Thomas Wiesner

If you say "Connect" then you should be able to see this page:



Let's start by developing our KYC Rules...

Last update: April 17, 2021

16.11 Update KYC

First, re-model the Frontend part:

```
render() {
  if (!this.state.loaded) {
    return <div>Loading Web3, accounts, and contract...</div>;
  }
  return (
    <div className="App">
      <h1>Capuccino Token for StarDucks</h1>

      <h2>Enable your account</h2>
      Address to allow: <input type="text" name="kycAddress" value={this.state.kycAddress} onChange={this.handleInputChange} />
      <button type="button" onClick={this.handleKycSubmit}>Add Address to Whitelist</button>
    </div>
  );
}
```

Secondly, add the functions "handleInputChange" and "handleKycSubmit":

```
handleInputChange = (event) => {
  const target = event.target;
  const value = target.type === "checkbox" ? target.checked : target.value;
  const name = target.name;
  this.setState({
    [name]: value
  });
}

handleKycSubmit = async () => {
  const {kycAddress} = this.state;
  await this.kycContract.methods.setKycCompleted(kycAddress).send({from: this.accounts[0]});
  alert("Account "+kycAddress+" is now whitelisted");
}
```

Also don't forget to change the state on the top of your App.js to modify the KYC Whitelist:

```
class App extends Component {
  state = { loaded: false, kycAddress: "0x123" };

  componentDidMount = async () => {
```

Finally, it should look like this:

The screenshot shows a browser window with the title "React App" and the URL "localhost:3000". The main content of the page is a heading "Capuccino Token for Sta..." followed by a button "Enable your account". Below the button is a text input field containing the address "0x123" and a button "Add Address". At the bottom of the page, the browser's developer tools are visible, specifically the Console tab. A warning message is displayed in the console:

```
Download the React DevTools for a better development experience: https://fb.me/react-devtools
⚠️ ▶️ MetaMask: MetaMask will soon stop reloading pages on network change.
If you rely upon this behavior, add a 'networkChanged' event handler to trigger the reload manually: https://ethereum.on\(eventName%2Ccallback\)
Set 'ethereum.autoRefreshOnNetworkChange' to 'false' to silence this warning: https://metamask.github.io/metamask-networkchange
```

The problem is now, your accounts to deploy the smart contract is in ganache, the account to interact with the dApp is in MetaMask. These are two different sets of private keys. We have two options:

1. Import the private key from Ganache into MetaMask (we did this before)
2. Use MetaMask Accounts to deploy the smart contract in Ganache (hence making the MetaMask account the "admin" account)
3. But first we need Ether in our MetaMask account. Therefore: First transfer Ether from Ganache-Accounts to MetaMask Accounts

Last update: April 17, 2021

16.12 Deploy Smart Contracts With MetaMask

One of the problems is during deployment of the smart contracts, we use Ganache Accounts

In order to transfer Ether from an Account in Ganache to an account in MetaMask, we have to start a transaction. The easiest way to do this is to use the truffle console to transfer ether from one of the networks defined in the truffle-config.js file to another account. In your project root in a terminal enter:

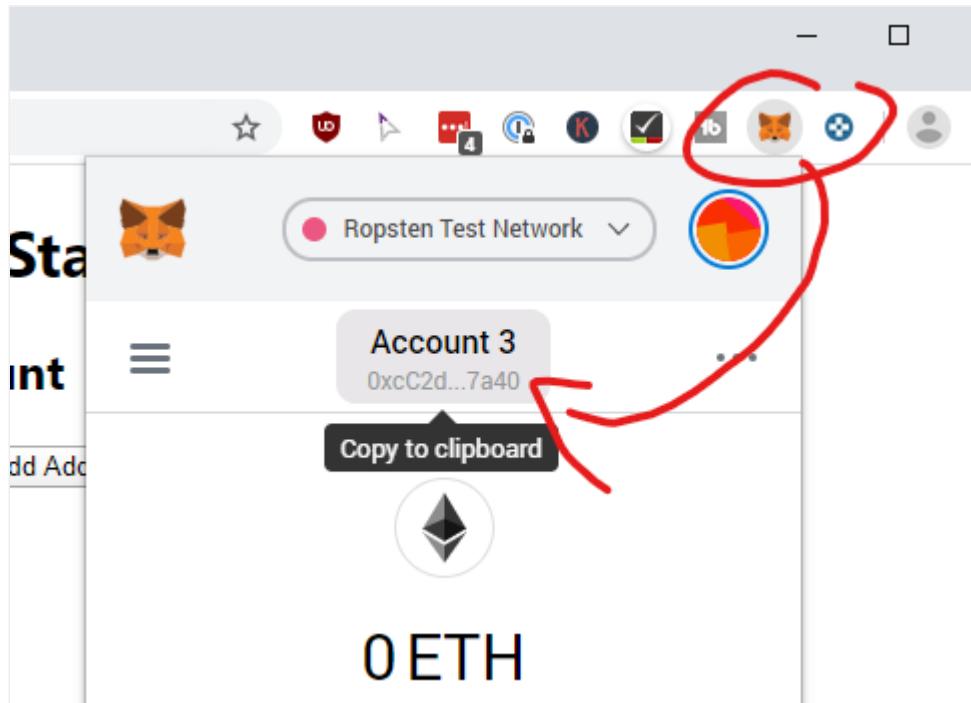
```
truffle console --network development
```

Then a new truffle console should pop up. You should be able to list the accounts by simply typing in "accounts":

```
s06 - tokenization> truffle console --network development
truffle(development)> accounts
[
  '0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6',
  '0xEddC05d9B2CB4D3965ee9d01C0c47960FAAbf273',
  '0xD152FdAea0F5c39bD38Ce6c29312af14B93F40E3',
  '0xf2Af5a487b989EcFd4F6310E2A964c1649C0CD43',
  '0x4FFF1eD88e721140729fd3C179c807984bc70dfe',
  '0xc0BBc1Eb8F66b494B08Ec843e12742c351499fDa',
  '0x4AC266A48d587e541b6f6520246Ec1Fd94B9F6bd',
  '0x0e369aEDD00dC0bcc3fa3ffF63A46fa5fDD55A4c',
  '0x34C027923c8c3cE277E2aE8b0eadc4b7f486804c',
  '0xAeC5b09f722d36622F811631493E60f7ce6dB4c4'
]
truffle(development)>
```

These are the same accounts as in Ganache. You are connected to your node via RPC. The node is Ganache. You can send off transactions using the private keys behind these accounts. Ganache will sign them.

We have to send a transaction from these accounts to MetaMask. Copy the account in MetaMask:

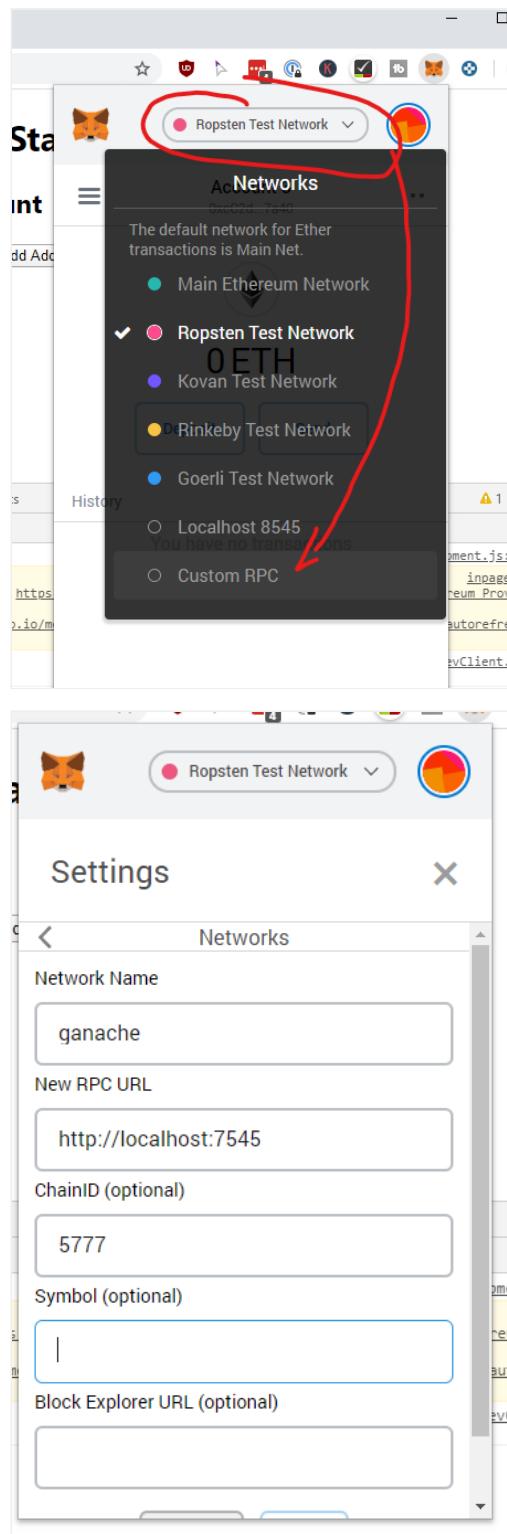


Type in:

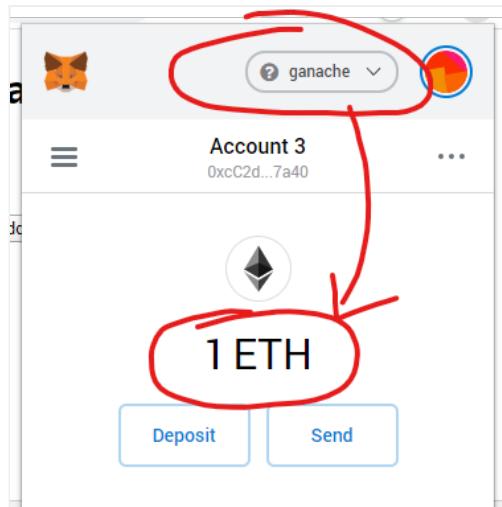
```
web3.eth.sendTransaction({from: accounts[0], to:"PASTE_ACCOUNT_FROM_METAMASK", value: web3.utils.toWei("1","ether")})
```

don't forget the quotes around the account! It should return a transaction object:

And your account in MetaMask should have now 1 Ether, if connected to the right network. Connect MetaMask to Ganache first:



Hit Save.



16.12.1 Add HDWalletProvider and the Mnemonic to Truffle and modify truffle-config.js

The first step is to add the HDWalletProvider to truffle. On the command line type in:

```
npm install --save @truffle/hdwallet-provider
```

The next step is to add the hdwallet provider and the mnemonic from MetaMask to the truffle-config.js in a secure way. The best suited place for the mnemonic would be the .env file, which should never be shared!

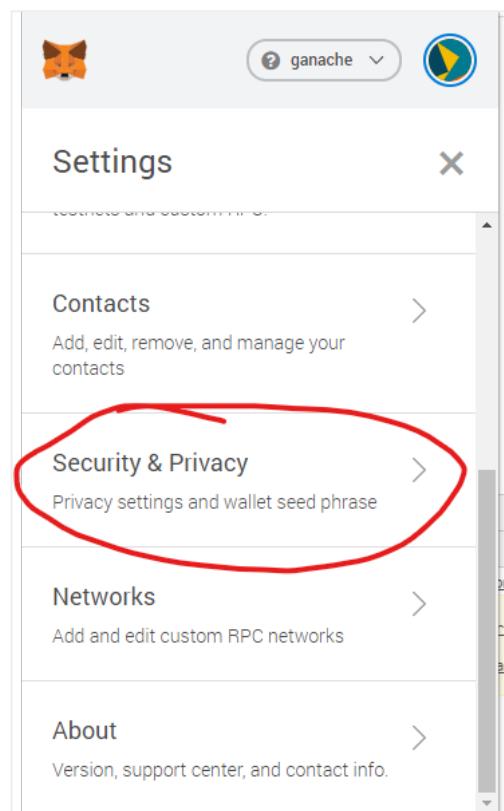
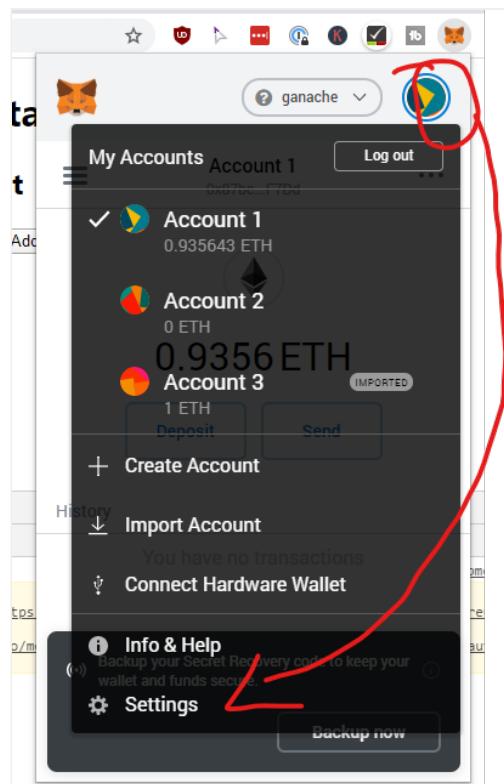
Let's start with the HDWalletProvider. Open the truffle-config.js file and add these parts:

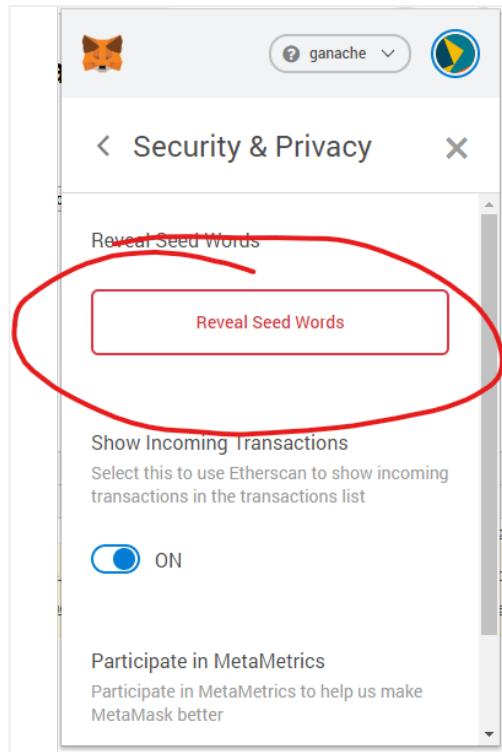
truffle-config.js

```
const path = require("path");
require('dotenv').config({path: './.env'});
const HDWalletProvider = require("@truffle/hdwallet-provider");
const MetaMaskAccountIndex = 0;

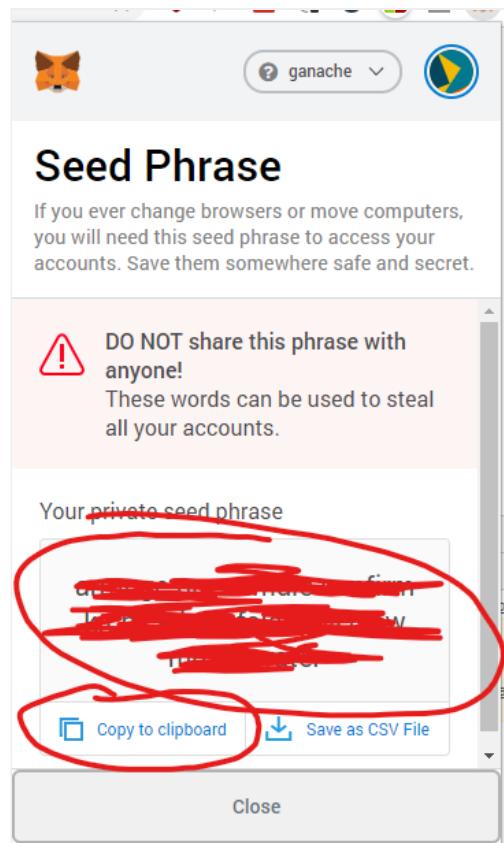
module.exports = {
// See <http://truffleframework.com/docs/advanced/configuration>
// to customize your Truffle configuration!
contracts_build_directory: path.join(__dirname, "client/src/contracts"),
networks: {
  development: {
    port: 7545,
    network_id: "*",
    host: "127.0.0.1"
  },
  ganache_local: {
    provider: function() {
      return new HDWalletProvider(process.env.MNEMONIC, "http://127.0.0.1:7545", MetaMaskAccountIndex )
    },
    network_id: 5777
  }
},
compilers: {
  solc: {
    version: "0.6.1",
  }
}
};
```

And add the Mnemonic from MetaMask to the .env file:





Copy the Mnemonic and add it to the env-file:



```
.env
1 INITIAL_TOKENS=10000000
2 MNEMONIC=
```

A red arrow points from the text 'MNEMONIC=' in the .env file up towards the Ganache 'Copy to clipboard' button in the screenshot above.

Then run the migrations again with the right network and see how your smart contracts are deployed:

```
truffle migrate --network ganache_local
```

It should come up as these migrations:

Replacing 'MyTokenSale'

```

> transaction hash: 0x0d7ef4cca2edb03b9b574d4f8c66c196de6ca04c3b02641354b7a2122576f079
> Blocks: 0 Seconds: 0
> contract address: 0x89416e57C05DF790F80E46d32cBe719F7DDF8aCd
> block number: 17
> block timestamp: 1582654868
> account: 0x87bc6aE16286b1D848E0ac25E7205554671aF7Dd
> balance: 0.93691332
> gas used: 941369
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.01882738 ETH

> Saving migration to chain.
> Saving artifacts
----->
> Total cost: 0.05847164 ETH

```

Summary

```

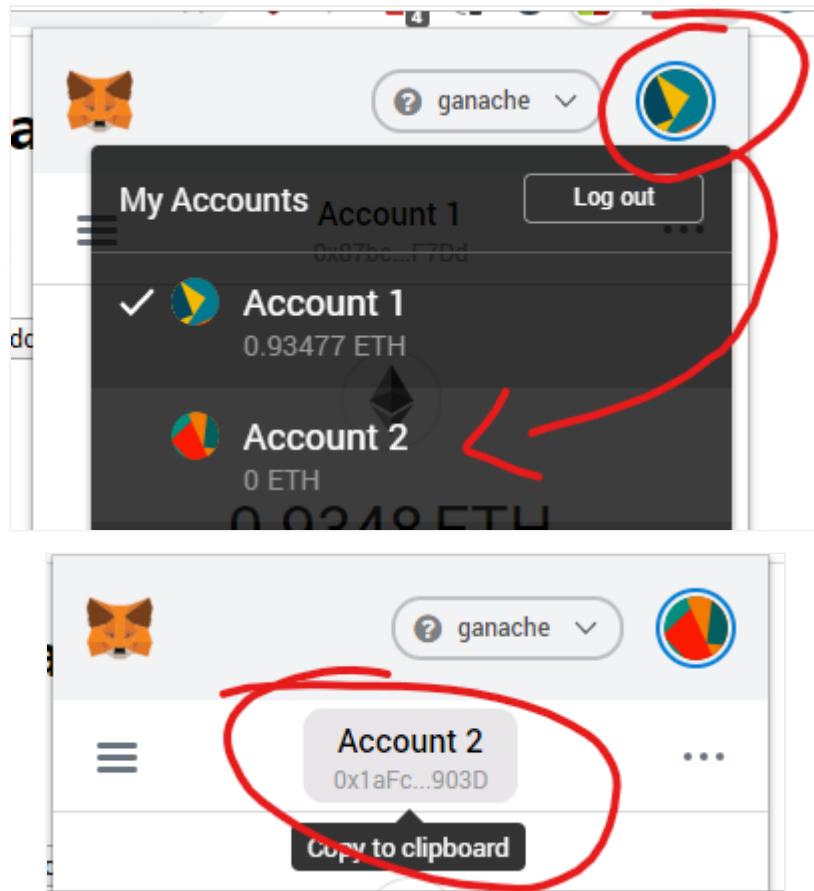
=====>
> Total deployments: 4
> Final cost: 0.06224666 ETH

```

16.12.2 Use the KycContract from your DApp using MetaMask

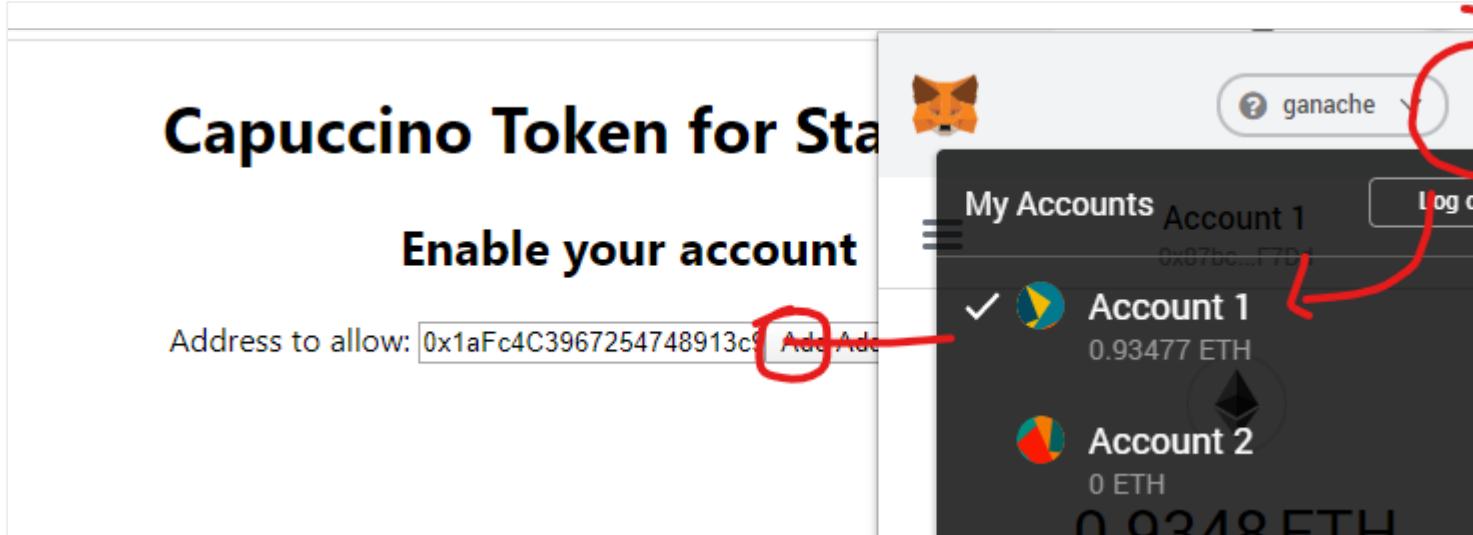
This was the groundwork. Next up is to actually white list an account. We could use another account in MetaMask to whitelist it.

Copy one of your accounts in MetaMask (other than your account#1) to whitelist:



Now, paste this account into the account-field in your new HTML UI:

But before sending off the transaction, make sure you switch back to Account #1 (the account that created the smart contract from truffle migrate):



You should see a popup to confirm the transaction and then an alert box, that tells you that your account is now whitelisted:

React App × +

localhost:3000

Capuccino Token for Sta

Enable your account

Address to allow: Add Address

MetaMask Notification

Account 1 → 0xD647

CONTRACT INTERACTION

0

DETAILS DATA

GAS FEE 0.0013

No Conversion Rate Available

AMOUNT + GAS FEE

TOTAL 0.0013

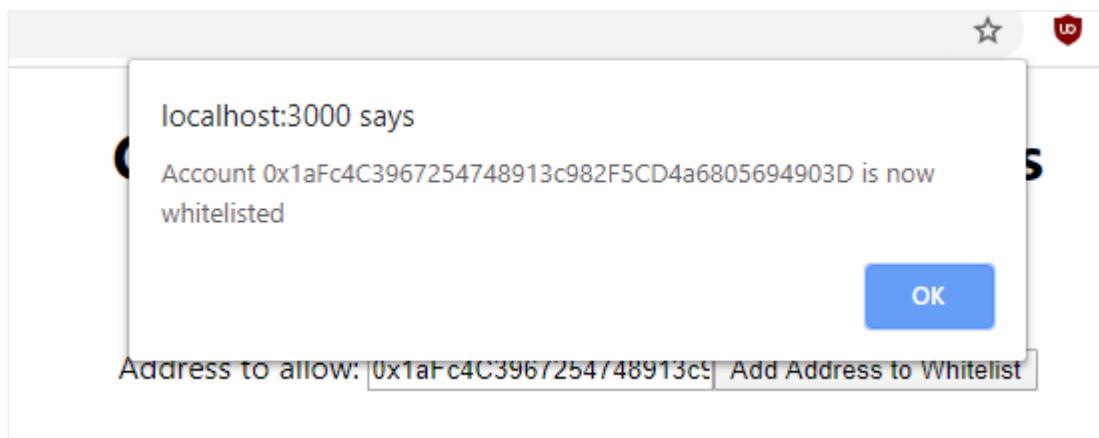
No Conversion Rate Available

Console What's New X

Highlights from the Chrome 79 update

Debug why a cookie was blocked

Reject Confirm

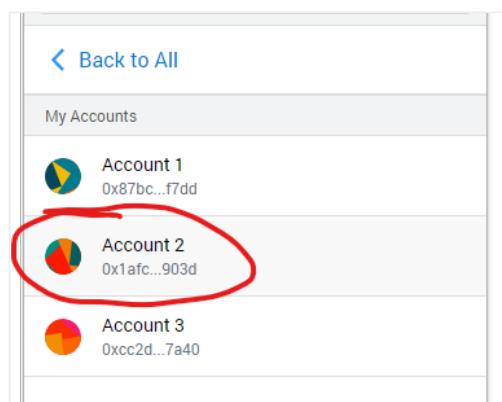
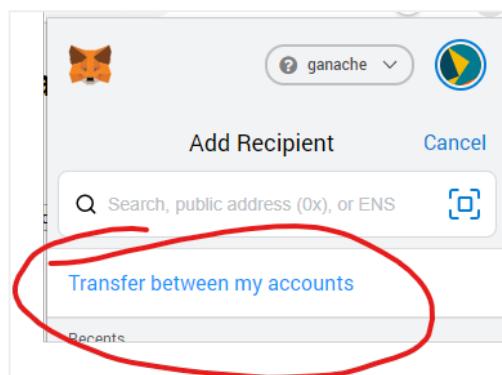
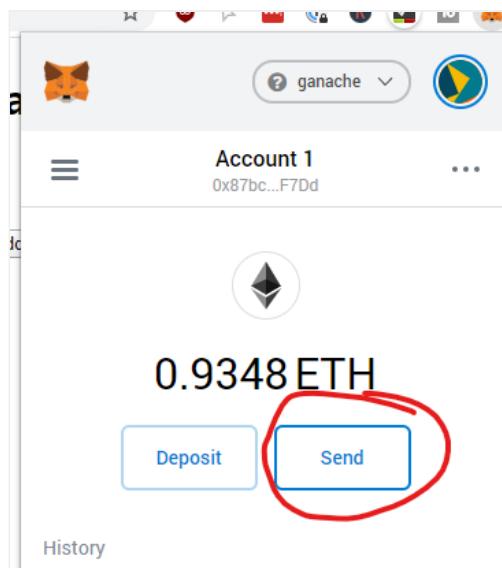


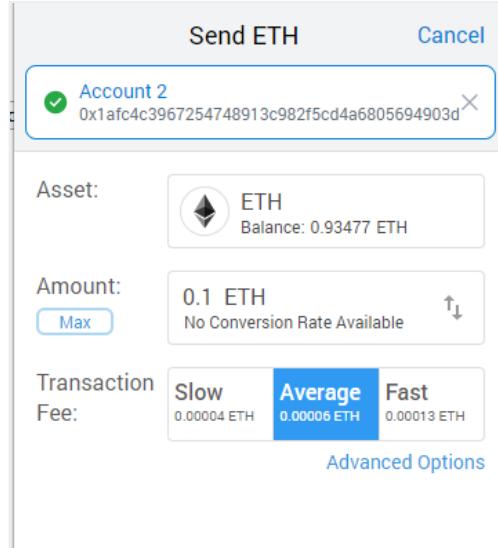
Account #2 is now whitelisted. But how to purchase Tokens?

Last update: April 17, 2021

16.13 Buy Coffee Tokens

To actually purchase any tokens, we must first get some ether into the account. Every token has a price, so, let's first send Ether from Account #1 to Account #2:





Enter 0.1 Ether, Hit confirm and wait for the 0.1 to arrive in Account#2. Using Ganache, it should take no longer than 5 seconds.

Now the interesting part. How to get Tokens?

First, we have to send Wei (or Ether) to the right address. Let's display the address inside the UI.

```
render() {
  if (!this.state.loaded) {
    return <div>Loading Web3, accounts, and contract...</div>;
  }
  return (
    <div className="App">
      <h1>Capuccino Token for StarDucks</h1>

      <h2>Enable your account</h2>
      Address to allow: <input type="text" name="kycAddress" value={this.state.kycAddress} onChange={this.handleInputChange} />
      <button type="button" onClick={this.handleKycSubmit}>Add Address to Whitelist</button>
      <h2>Buy Cappuccino-Tokens</h2>
      <p>Send Ether to this address: {this.state.tokenSaleAddress}</p>
    </div>
  );
}
```

And also add the variable to the state:

```
class App extends Component {
  state = { loaded: false, kycAddress: "0x123", tokenSaleAddress: "" };

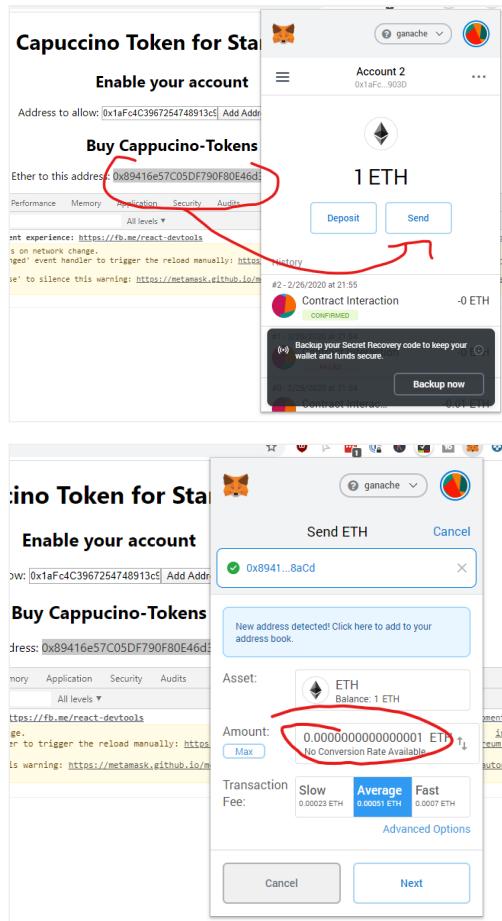
  componentDidMount = async () => {
```

And in componentDidMount write the tokenSaleAddress:

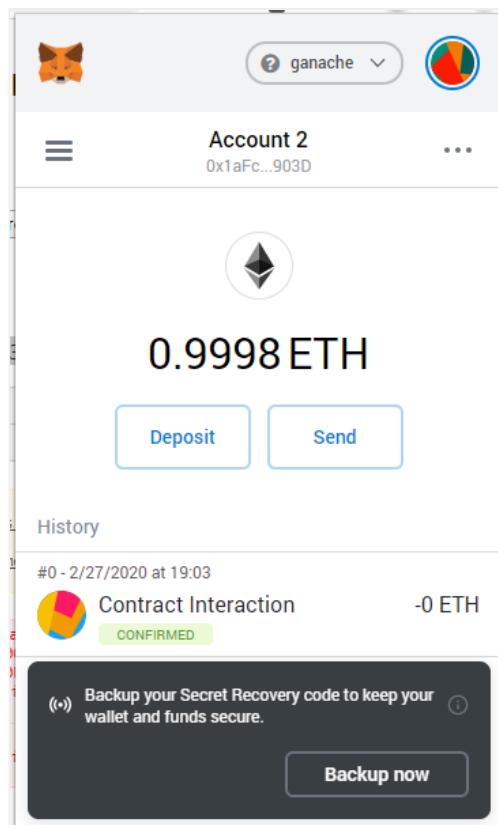
```
this.kycContract = new this.web3.eth.Contract(
  KycContract.abi,
  KycContract.networks[this.networkId] && KycContract.networks[this.networkId].address,
);

// Set web3, accounts, and contract to the state, and then proceed with an
// example of interacting with the contract's methods.
this.setState({ loaded:true, tokenSaleAddress: this.myTokenSale._address });
} catch (error) {
```

Then simply send 1 Wei from your account. Initially, we set 1 Wei equals 1 Token, we might want to change that later on, but for testing it's okay:



1 Ether = 10^{18} Wei, so 1 Wei = 0.0000000000000001 Ether. A tool I use to convert is [Eth Converter](#)



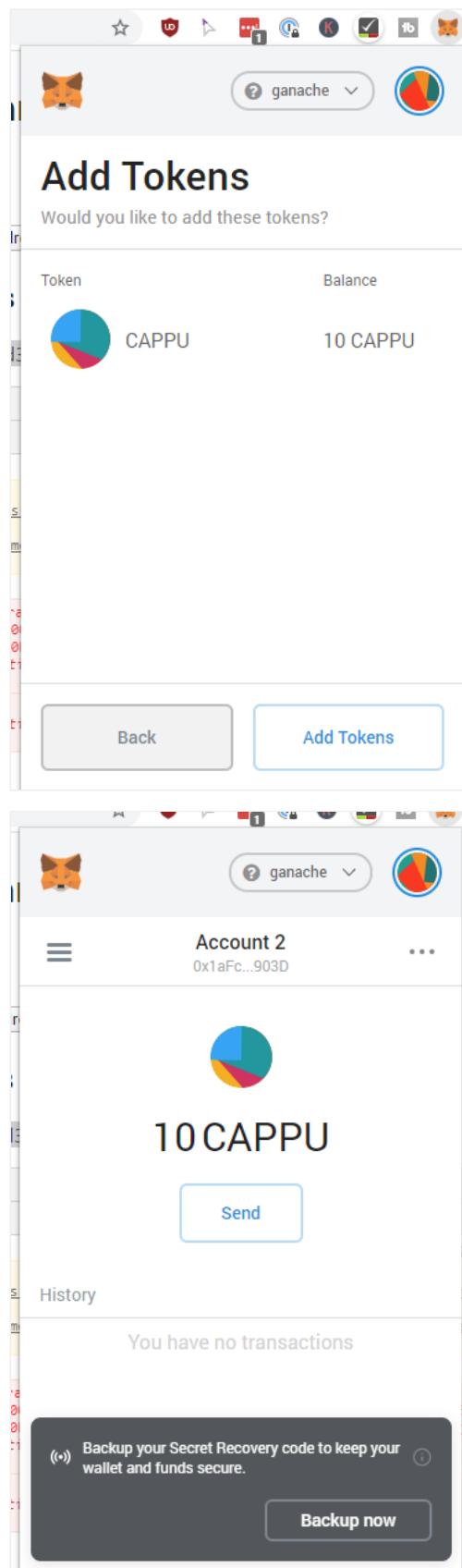
16.13.1 How to Display the Tokens within MetaMask?

We need to open MetaMask and add a custom Token to our UI. Follow the following pictures to add the Token:

The image consists of two screenshots of the MetaMask extension. The top screenshot shows the main dashboard. At the top, there is a network selection dropdown showing 'Rinkeby Test Network'. Below it, the account status is 'Not connected' and the account address is 'Account 1 0x5AD2...053E'. The balance is displayed as '0 ETH' with a small ETH icon. Below the balance are three buttons: 'Buy', 'Send', and 'Swap'. Underneath these buttons are two tabs: 'Assets' (which is underlined in blue) and 'Activity'. In the 'Assets' section, there is a card for '0 ETH' with a red oval drawn around the 'Add Token' button below it. The bottom screenshot shows the 'Add Tokens' dialog. At the top, there is a network selection dropdown showing 'ganache'. Below it, there are two tabs: 'Search' and 'Custom Token' (which is underlined in blue). The dialog has four input fields: 'Token Contract Address' (an empty input field), 'Token Symbol' (an empty input field), 'Decimals of Precision' (a field containing '0'), and a 'Next' button at the bottom right. A 'Cancel' button is also present.

You need the Token-Address, not the TokenSaleAddress. You can either print the address to the UI or copy it directly from the json file in the client/contracts/MyToken.json file.

Add in the Token-Address from the Token and the Symbol "CAPPU", then click next. You should see your token appear in MetaMask for your account:



You could also send one CAPPU token to your other account, directly through MetaMask!

16.13.2 How to Buy and Display the Tokens Amount on the Website

Let's also add in the Tokens amount on the website, as well as a method to buy directly tokens via the website, without calculating yourself how much you want to buy.

```
render() {
  if (!this.state.loaded) {
    return <div>Loading Web3, accounts, and contract...</div>;
  }
  return (
    <div className="App">
      <h1>Capuccino Token for StarDucks</h1>

      <h2>Enable your account</h2>
      Address to allow: <input type="text" name="kycAddress" value={this.state.kycAddress} onChange={this.handleInputChange} />
      <button type="button" onClick={this.handleKycSubmit}>Add Address to Whitelist</button>
      <h2>Buy Cappuccino-Tokens</h2>
      <p>Send Ether to this address: {this.state.tokenSaleAddress}</p>
      <p>You have: {this.state.userTokens}</p>
      <button type="button" onClick={this.handleBuyToken}>Buy more tokens</button>
    </div>
  );
}
```

Then add in first the function to handleBuyToken:

```
handleBuyToken = async () => {
  await this.myToken.methods.buyTokens(this.accounts[0]).send({from: this.accounts[0], value: 1});
}
```

And also update the userTokens...

Add the state:

```
state = { loaded: false, kycAddress: "0x123", tokenSaleAddress: "", userTokens: 0 };
```

And add both, a function to update the userTokens, as well as an event-listener that updates the variable upon purchase:

```
updateUserTokens = async() => {
  let userTokens = await this.myToken.methods.balanceOf(this.accounts[0]).call();
  this.setState({userTokens: userTokens});
}

listenToTokenTransfer = async() => {
  this.myToken.events.Transfer({to: this.accounts[0]}).on("data", this.updateUserTokens);
}
```

The last step is to call these functions at the appropriate place in the code. Add/Change this in the componentDidMount function:

```
this.kycContract = new this.web3.eth.Contract(
  KycContract.abi,
  KycContract.networks[this.networkId] && KycContract.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.listenToTokenTransfer();
this.setState({ loaded:true, tokenSaleAddress: this.myTokenSale._address }, this.updateUserTokens);
} 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);
}
```

Let's give it a try:

Capuccino Token for StarDucks

Enable your account

Address to allow: 0x123

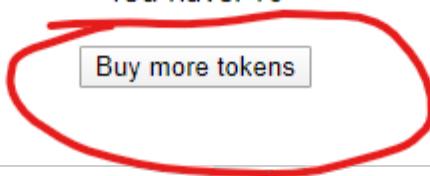
[Add Address to Whitelist](#)

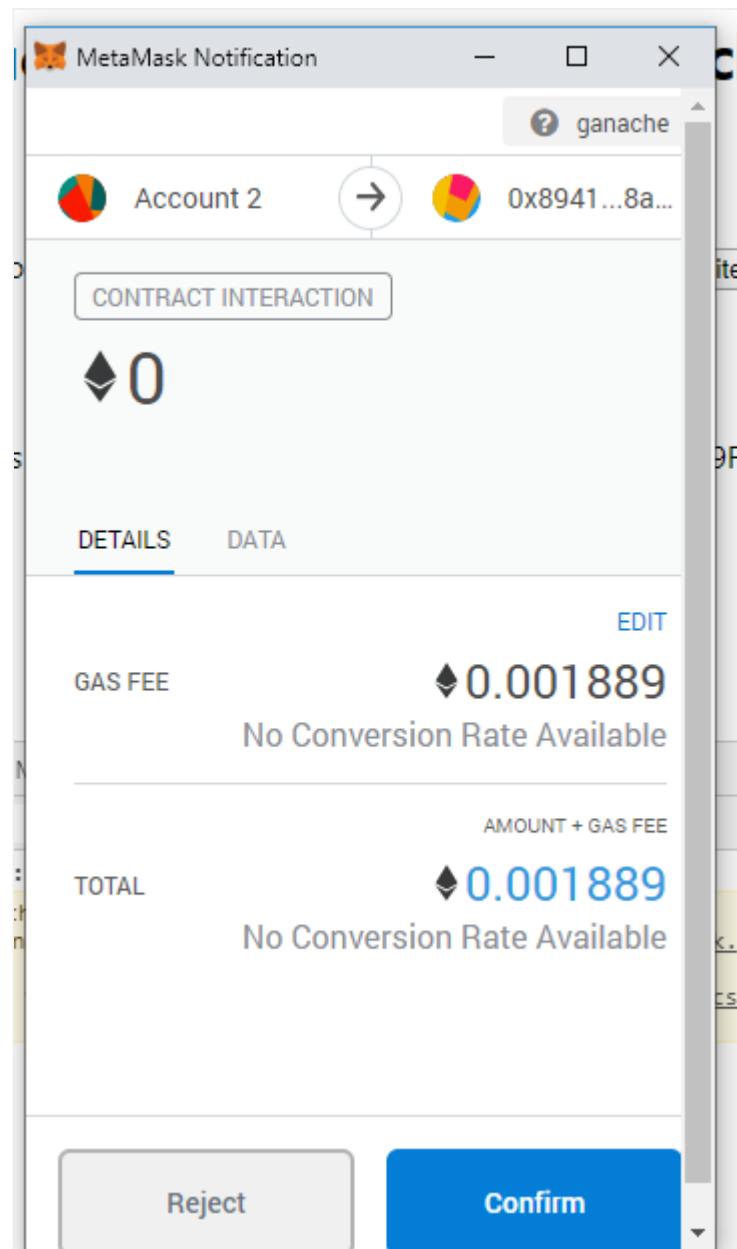
Buy Cappucino-Tokens

Send Ether to this address: 0x89416e57C05DF790F80E46d32cBe719F7DDF8aCd

You have: 10

[Buy more tokens](#)





The screenshot shows a web page with the following content:

- no Token for Site**
- Enable your account**
- A text input field containing `: 0x123` with an **Add** button next to it.
- Buy Cappuccino-Tokens**
- Address: `0x89416e57C05DF790F80E`
- You have: 11** (This text is circled in red.)
- Buy more tokens**

Last update: April 17, 2021

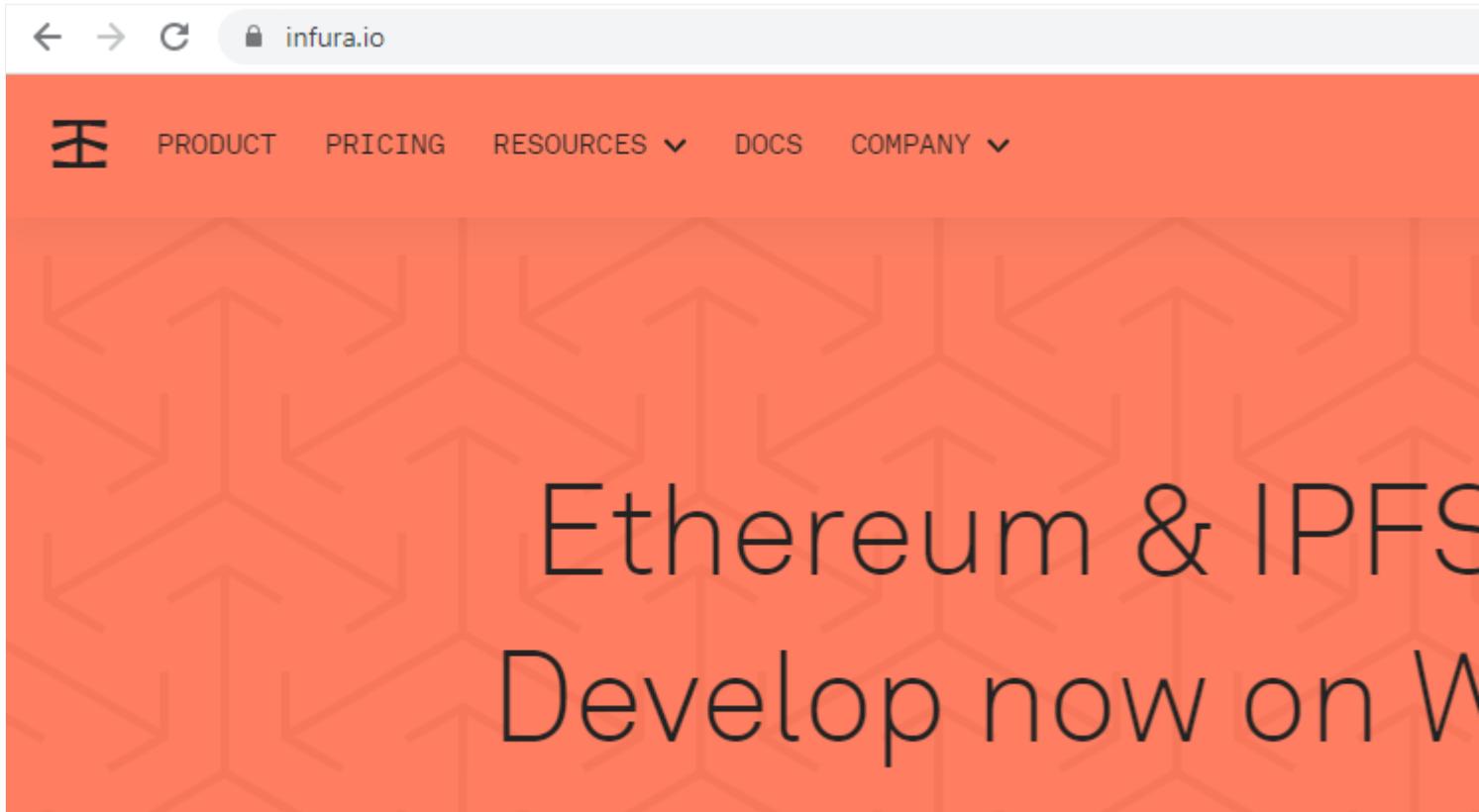
16.14 Deployment with Infura

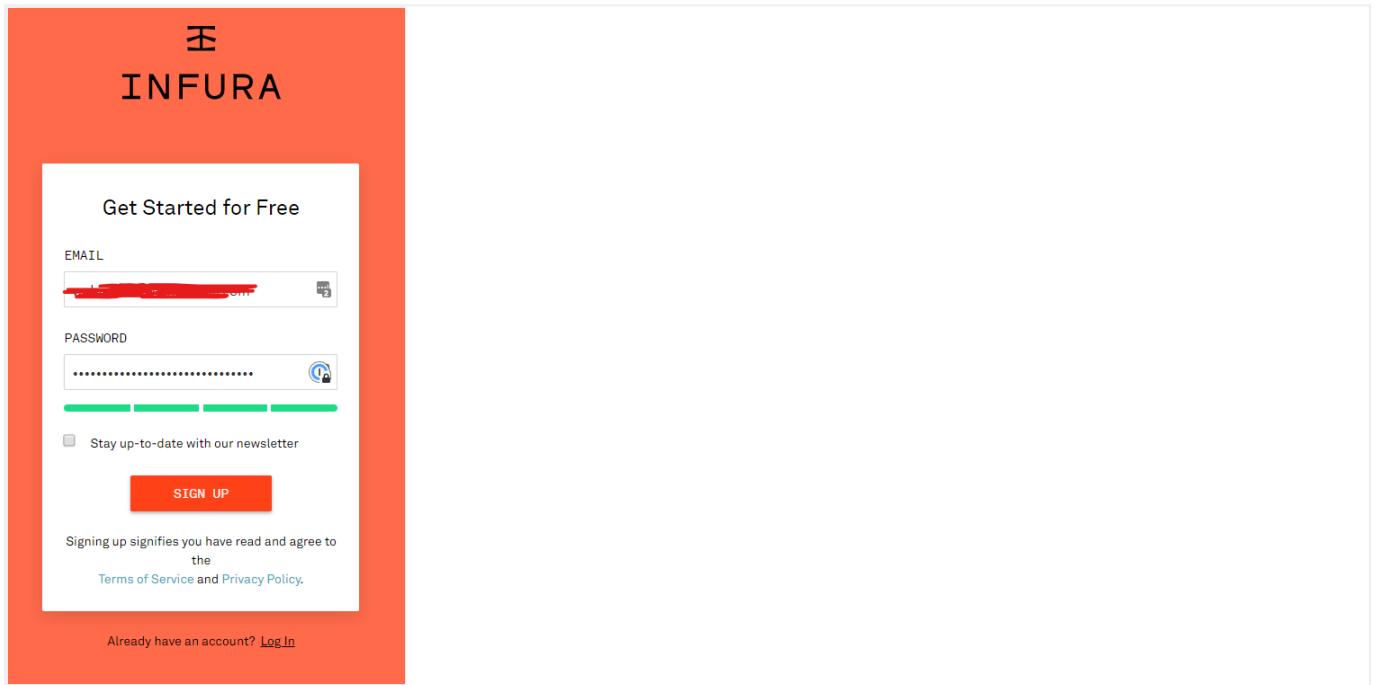
In this step we are deploying our token into the Test-Network, either Görli or Ropsten. We do this without setting up our own Blockchain Node. We use a hosted node with Infura.

Because our setup is already so well prepared, it's extremely easy to do so.

16.14.1 Signup with Infura

First thing is to signup with Infura. Go to <https://infura.io> and signup with your email address.





Signup with Infura

Confirm the Mail address and enter the dashboard.

16.14.2 Create a new Infura Project

First you need to create a new Infura Project

The screenshot shows the Infura.io dashboard with a red hand-drawn style callout pointing from the bottom right towards the 'CREATE NEW PROJECT' button. The dashboard features a header with a logo and 'DASHBOARD' text, a central promotional banner for 'INFURA+' with a character icon, and a section titled 'YOUR PROJECTS' showing '3 remaining'. A large orange button labeled 'CREATE NEW PROJECT' is prominent on the right. Below it, there's a network icon and text encouraging users to start their first project by setting up a project ID, secret, and endpoints.

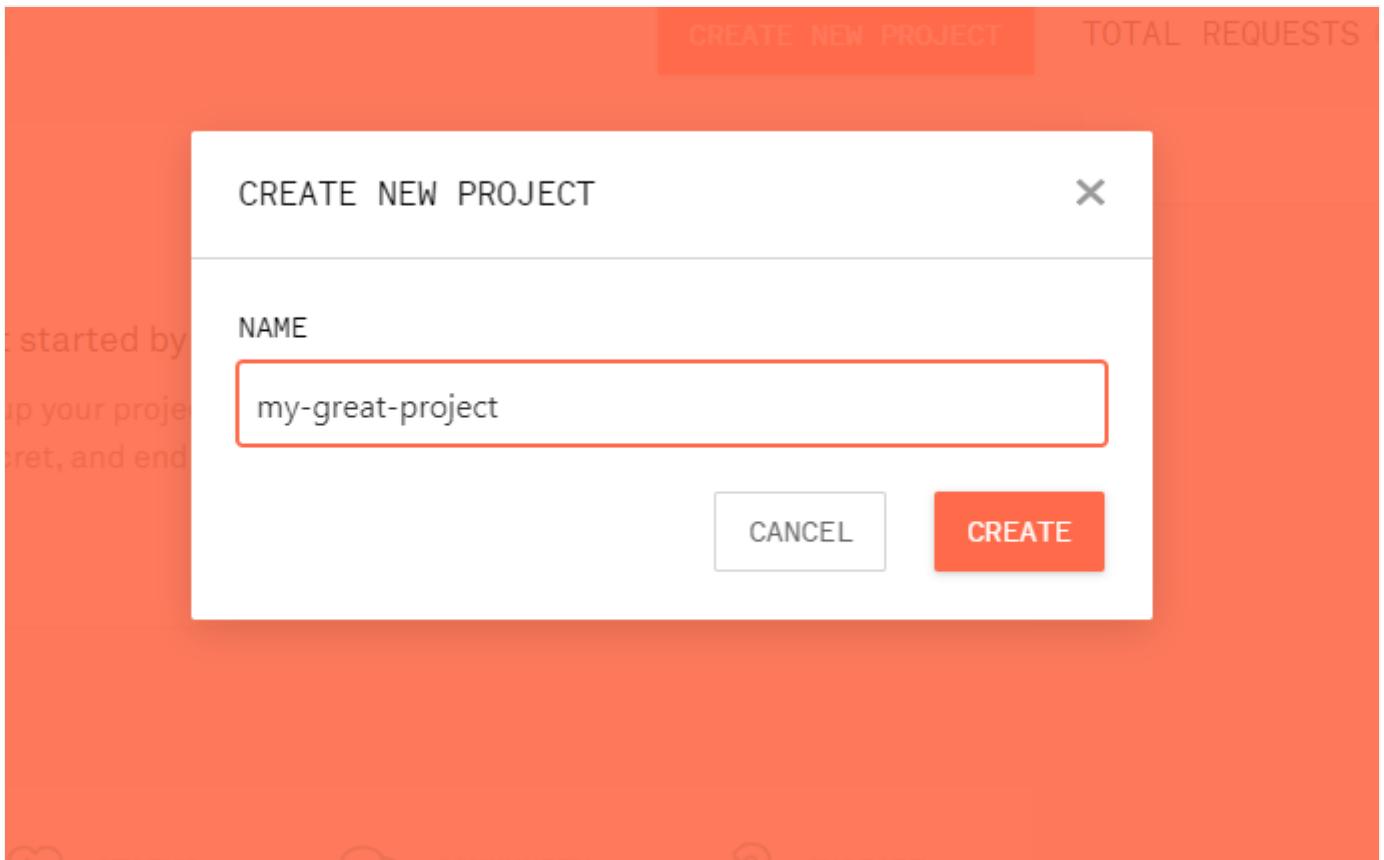
YOUR PROJECTS 3 remaining

CREATE NEW PROJECT

Get started by creating your first project

Setup your project to generate your project ID, secret, and endpoints as well as to whitelist contracts.

Give it a name:



And hit "CREATE".

View the Project:

A screenshot of the Infura 'YOUR PROJECTS' page. The page header says 'YOUR PROJECTS' and '2 remaining'. There is a list of projects with one item: 'MY-GREAT-PROJECT' (Created on Feb 27, 2020). To the right of the list are two buttons: 'VIEW PROJECT' and 'VIEW STATS'. A red circle highlights the 'VIEW PROJECT' button.

This is the important ID you will need:

PROJECT ID
7f63b0deb8e7425daafbc0fba88ea811

PROJECT SECRET ⓘ
40731f0bb1e446f788f1cf8e92d1f8ee

ENDPOINT MAINNET ⚙️

mainnet.infura.io/v3/7f63b0deb8e7425daafbc0fba88ea811 ⚙️

16.14.3 Update the truffle-config.json File

Now let's update the truffle-config.json file so we can deploy using the nodes from Infura. Add a new network:

```
networks: {
  development: {
    port: 7545,
    network_id: "*",
    host: "127.0.0.1"
  },
  ganache_local: {
    provider: function() {
      return new HDWalletProvider(process.env.MNEMONIC, "http://127.0.0.1:7545", MetaMaskAccountIndex)
    },
    network_id: 5777
  },
  ropsten_infura: {
    provider: function() {
      return new HDWalletProvider(process.env.MNEMONIC, "https://ropsten.infura.io/v3/YOUR_INFURA_ID", MetaMaskAccountIndex)
    },
    network_id: 3
  },
  goerli_infura: {
    provider: function() {
      return new HDWalletProvider(process.env.MNEMONIC, "https://goerli.infura.io/v3/YOUR_INFURA_ID", MetaMaskAccountIndex)
    },
    network_id: 5
  }
},
compilers: {
  solc: {

```

Where it says "YOUR_INFURA_ID" enter the ID from your own Infura Dashboard please! That's it. Let's run this!

16.14.4 Run the Migrations

The last part is to run the migrations. At the very beginning of the course we got some test-ether in our MetaMask. You should still have them. Just run the Migrations and see if it works:

```
truffle migrate --network ropsten_infura
```

and watch the output -- this might take a while:

```

> transaction hash: 0x501f819d158d06d54476ebfbba7b6afdf6af2f9bc288c0e22c56cc3a0f6e0118
> Blocks: 1
> contract address: 0x89416e57C05DF790F80E46d32cBe719F7DDF8aCd
> block number: 7415251
> block timestamp: 1582832729
> account: 0x87bc6aE16286b1D848E0ac25E7205554671aF70d
> balance: 0.76279102
> gas used: 521896
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.01843792 ETH

Deploying 'MyTokenSale'
-----
> transaction hash: 0x043d59b01b001836286d55d5d5001f39730d3d875e9dfb673ebc43e5dc0c655
> Blocks: 2
> contract address: 0x18d9d5d60c6063a638d424D59dbbcBE0DA233BC6
> block number: 7415255
> block timestamp: 1582832750
> account: 0x87bc6aE16286b1D848E0ac25E7205554671aF70d
> balance: 0.74745244
> gas used: 766929
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.01533858 ETH

.: Saving migration to chain.

```

Migration is Running

```

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.04799292 ETH

```

Summary

```

=====
> Total deployments: 4
> Final cost: 0.05128074 ETH

```

Migration is Finished

And then open your Browser Window again and switch MetaMask to the Ropsten (or Görli) network, depending on which one you deployed. You already can see that you have no tokens there, but also the address of the TokenSale contract changed:

Capuccino Token for StarDucks**Enable your account**Address to allow: 0x123 **Buy Cappuccino-Tokens**

Send Ether to this address: 0x18d9d5d60c6063a638d424D59dbbcBE0DA233BC6

You have: 0

Deployed on Ropsten

Capuccino Token for StarDucks**Enable your account**Address to allow: 0x123 **Buy Cappuccino-Tokens**

Send Ether to this address: 0x89416e57C05DF790F80E46d32cBe719F7DDF8aCd

You have: 11

Deployed on Ganache

You could go ahead and whitelist another account now and get some tokens. You can also implement a spending token facility, for actually burning tokens once the cappuccino is bought. Before we do that, let's change the whole crowdsale!

Last update: April 17, 2021

17. LEARN: Proxies and Upgrades

17.1 Upgrade Smart Contract and Smart Contract Proxies

This is the lab you've been looking for if you want a tutorial style guide for Smart Contract Upgrades and Proxy Patterns. That is the thing where storage and/or Smart Contract addresses don't change. You only change the logic of the contract itself.

|

17.1.1 Real-World Use-Case for this Project

❖ Iteratively Release new Features

Understand the possibilities for Bug-Fixing

Pick the right Architecture for your Project

Avoid Scammers

Make Auditors life easier

17.1.2 Development-Goal

Understand Storage Collisions

Deep Dive Into Storage Patterns

⌚ Understand All Standards for Proxies

Understand the CREATE2 Op-Code

17.1.3 What's in it for you?

At the end of this I want you to know really all about upgradeable Smart Contracts as of Q1/2021.

First I want to discuss the different standards. Then I want to do a hands-on deep-dive into OpenZeppelin OS with the Proxy pattern. Lastly I want to discuss Metamorphosis Smart Contracts which can be re-deployed to the same address using CREATE2.

Let's do this!

Last update: April 17, 2021

17.2 Introduction

One thing the Blockchain is very often connected to is the immutability of data. For long time it was "Once it's deployed, it cannot be altered". That is still true for historical transaction information. But it is not true for Smart Contract storage and addresses.

17.2.1 The Main Reasons For Upgrades

Opinions about this are split in half. A lot of users very much love to have the opportunity of upgradeable Smart Contracts. The others absolutely hate the fact that Smart Contracts are not immutable anymore.

But why would you want to do Smart Contract upgrades in the first place?

The reasons are a diverse mix between Bug-Fixing and Feature-Adding. Sometimes it is updating logic. Sometimes it is combined with a decentralized governance. But hey, sometimes it is also just scamming people into getting their Money.



You will see later how easy it is to fool even seasoned Solidity developers into thinking a Smart Contract is secure.

Enough of the introduction. Let's talk about facts and examples. And how you can detect if you're getting scammed or not.

Last update: April 17, 2021

17.3 The Problematic Smart Contract

One of the main problems with Solidity is that the storage for normal Smart Contracts is bound to the Smart Contract Address. If you deploy a new version you also start with an empty storage.

We can easily try this, and you probably know it already. That's the very basic stuff to get started with, but we need to start somewhere, so why not with a simple Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract LostStorage {
    address public myAddress;

    function setAddress(address _address) public {
        myAddress = _address;
    }
}
```

Deployed in Remix, we can set an address into the variable `myAddress`. Nothing new here.

The screenshot shows the Truffle UI interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and the code editor on the right.

Left Panel (Deployment):

- ENVIRONMENT:** Set to "JavaScript VM".
- ACCOUNT:** Address 0x5B3...eddC4 (99.99999999999999).
- GAS LIMIT:** Set to 3000000.
- VALUE:** Set to 0 wei.
- CONTRACT:** Selected "LostStorage - browser/Upgrades/01-lost-storage.sol".
- Buttons:** "Deploy" (orange), "Publish to IPFS" (checkbox), and "OR" (text).
- Transactions recorded:** 2 entries.
- Deployed Contracts:** A list showing "LOSTSTORAGE AT 0xD91...39138 (MEM)" with a "setAddress" button and address 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4.

Right Panel (Code Editor):

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract LostStorage {
    address public myAddress;

    function setAddress(address _address) public {
        myAddress = _address;
    }
}
```

Bottom Panel (Logs):

- 0 transactions recorded.
- 0 pending transactions.
- 0 errors.
- Search bar: "Search with..."
- Logs:
 - [vm] from: 0x5B3...eddC4 to: LostStorage.(constructor)
 - transact to LostStorage.setAddress pending ...
 - [vm] from: 0x5B3...eddC4 to: LostStorage.setAddress(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)

The important part is this: When we re-deploy the Smart Contract, the Smart Contract not only gets a new address, but also the storage is empty again.

The screenshot shows the Truffle UI interface for managing deployed contracts. It displays two separate contracts, each with its own storage state and interaction options.

Contract 1 (Top):

- Storage:** setAddress (orange button) → Value: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
- Function:** myAddress (blue button)
- Output:** 0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
- Low level interactions:** CALldata (grey button) with a "Transact" button.

Contract 2 (Bottom):

- Storage:** setAddress (orange button) → Value: address _address
- Function:** myAddress (blue button)
- Output:** 0: address: 0x00000000000000000000000000000000
- Low level interactions:** CALldata (grey button) with a "Transact" button.

We end up with two Smart Contracts on - two different addresses - with different storage.

So, we're tackling one problem after the other one.

Last update: April 17, 2021

17.4 Overview of Standards for Smart Contract Upgrades

Here are the different standards that emerged for Smart Contract Upgrades:

1. We're trying the [Eternal Storage Pattern](#)

Actually it was initially proposed by [Elena Dimitrova on this Blog](#)

2. We expand with Proxies where it all started (apparently):

the [upgradeable.sol gist](#) from Nick Johnson, Lead developer of ENS & Ethereum Foundation alum.

3. [EIP-897: ERC DelegateProxy](#)

Created 2018-02-21 by Jorge Izquierdo and Manuel Araoz

4. [EIP-1822: Universal Upgradeable Proxy Standard \(UUPS\)](#)

Created 2019-03-04 by Gabriel Barros and Patrick Gallagher

5. [EIP-1967: Standard Proxy Storage Slots](#)

Created 2019-04-24 by Santiago Palladino That's OpenZeppelin is using.

6. [EIP-1538: Transparent Contract Standard](#) Created 2018-10-31 by Nick Mudge

7. [EIP-2535: Diamond Standard](#) Created 2020-02-22 by Nick Mudge

8. Not really a standard, but I think [Metamorphic Smart Contracts](#) should be covered as well. Those are Smart Contracts that get re-deployed to the same address with different logic using [EIP-1014 CREATE2](#). It's said to be [wild magic](#) in Ethereum.

Simplified Contracts

For me it is important to understand the essence of what's going on under the hood. I will therefore reduce the Smart Contract examples to its absolute necessity for the architectural explanation.

There is no ownership, no control, no governance, just barebones the theory behind the Storage Patterns.

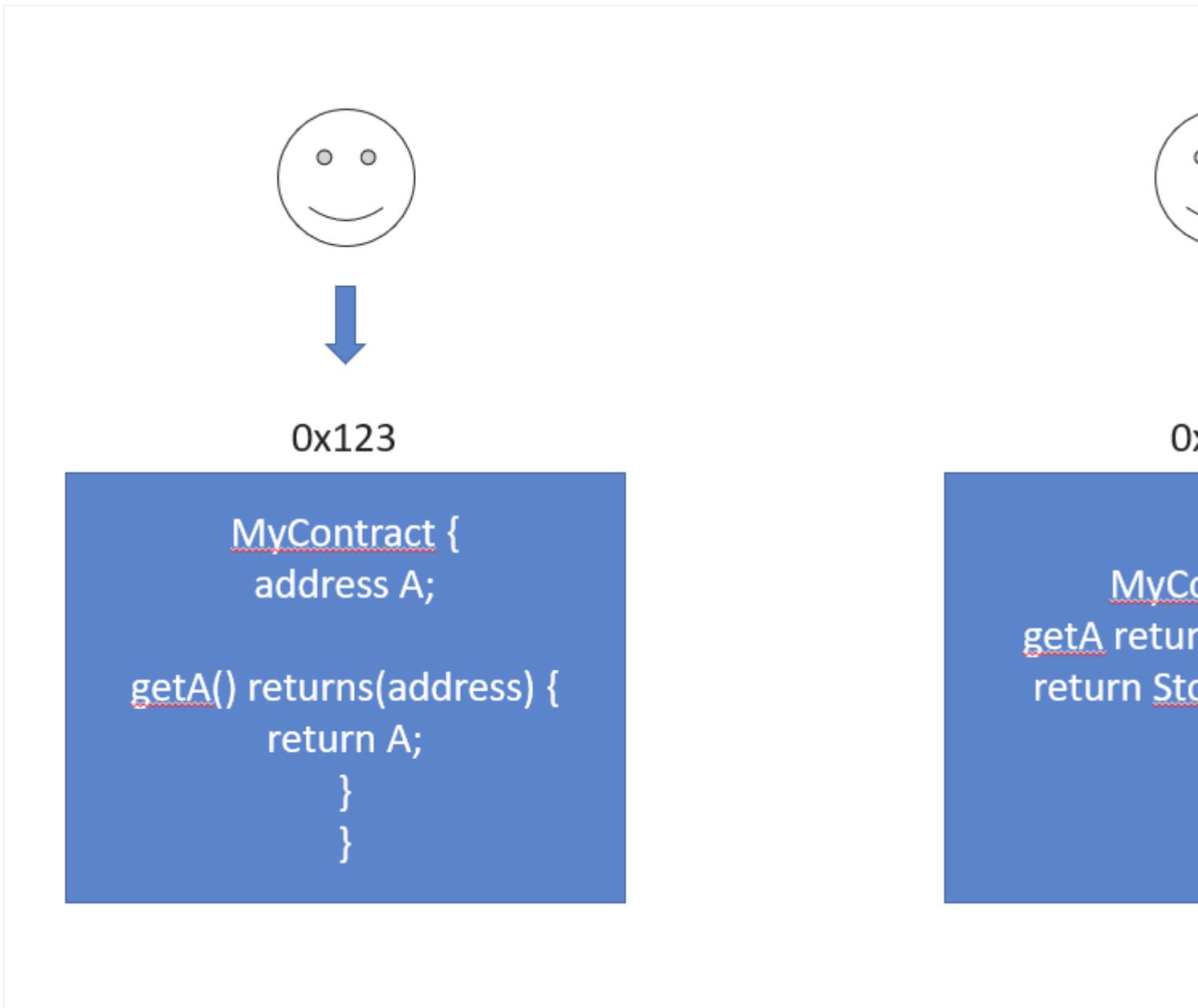
If you need a full blown solution that works out of the box, checkout OpenZeppelin.

Last update: April 17, 2021

17.5 Eternal Storage *without Proxy*

The first thing to tackle is the loss of data during re-deployment. What comes to mind is to separate logic from storage. The question is how are we doing that?

We go from the left side of this graphics to the right side.



In the Eternal Storage pattern, we move the storage with setters and getters to a separate Smart Contract and let only read/write the logic Smart Contract from it.

This can be a Smart Contract which deals with exactly the variables you need, or you generalize by variable types. Let me show you what I mean by that in the example below.

For sake of simplicity, I will closely take what Elena Dimitrova was using in her [Example](#). But I will greatly simplify this and boil it down to the essence. The Smart Contracts are not therefore remotely complete, but show the most important part to understand what's going on under the hood.

I've ported them to Solidity 0.8.1. Just fyi.

It could look like this:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

contract EternalStorage{
    mapping(bytes32 => uint) UIntStorage;

    function getUIntValue(bytes32 record) public view returns (uint){
        return UIntStorage[record];
    }

    function setUIntValue(bytes32 record, uint value) public {
        UIntStorage[record] = value;
    }

    mapping(bytes32 => bool) BooleanStorage;

    function getBooleanValue(bytes32 record) public view returns (bool){
        return BooleanStorage[record];
    }

    function setBooleanValue(bytes32 record, bool value) public {
        BooleanStorage[record] = value;
    }
}

library ballotLib {

    function getNumberOfVotes(address _eternalStorage) public view returns (uint256) {
        return EternalStorage(_eternalStorage).getUIntValue(keccak256('votes'));
    }

    function setVoteCount(address _eternalStorage, uint _voteCount) public {
        EternalStorage(_eternalStorage).setUIntValue(keccak256('votes'), _voteCount);
    }
}

contract Ballot {
    using ballotLib for address;
    address eternalStorage;

    constructor(address _eternalStorage) {
        eternalStorage = _eternalStorage;
    }

    function getNumberOfVotes() public view returns(uint) {
        return eternalStorage.getNumberOfVotes();
    }

    function vote() public {
        eternalStorage.setVoteCount(eternalStorage.getNumberOfVotes() + 1);
    }
}
```

This is a simple voting Smart Contract. You call `vote()` and increase a number - pretty basic business logic. Under the hood is the magic.

First we need to deploy the Eternal Storage. This contract remains a constant and isn't changed at all.

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" and "01-lost-storage.sol".

ENVIRONMENT: Set to "JavaScript VM".

ACCOUNT: Address 0x5B3...eddC4 (100 ether).

GAS LIMIT: Set to 3000000.

VALUE: Set to 0 wei.

CONTRACT: Selected "EternalStorage - browser/Upgrades/02".

Buttons: "Deploy" (highlighted with a red circle labeled 1) and "Publish to IPFS" (highlighted with a red circle labeled 2).

OR

At Address: "Load contract from Address".

Transactions recorded: 0.

Deployed Contracts: A list with a delete icon.

Message: "Currently you have no contract instances to interact with."

Code Editor: Shows the Solidity code for the EternalStorage contract.

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.2;
4
5
6 contract EternalStorage{
7
8     mapping(bytes32 => uint) UIntStorage;
9
10    function getUIntValue(bytes32 record) public
11        returns UIntStorage[record];
12    }
13
14    function setUIntValue(bytes32 record, uint value)
15    {
16        UIntStorage[record] = value;
17    }
18
19
20    mapping(bytes32 => bool) BooleanStorage;
21
22    function getBooleanValue(bytes32 record) public
23        returns BooleanStorage[record];
24    }
25
26    function setBooleanValue(bytes32 record, bool value)
27    {
28        BooleanStorage[record] = value;
29    }
30

```

Help: Includes instructions for deploying via command line, file explorer, or right-clicking on a JavaScript file. It also lists accessible libraries: web3 version 1.0.0, ethers.js, swargmw, and remix (run remix.help() for more info).

Then we deploy the Ballot Smart Contract, which will take the library and the Ballot Contract to do the actual logic.

The screenshot shows the Truffle UI interface for deploying a Solidity contract. The left sidebar contains environment settings like 'JavaScript VM', account '0x5B3...eddC4 (99.9999999)', gas limit '3000000', value '0 wei', and a list of deployed contracts ('ETERNALSTORAGE AT 0xD91...', 'BALLOT AT 0xF8E...'). The main area shows the Solidity code for the 'EternalStorage' contract:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.2;

contract EternalStorage {

    mapping(bytes32 => uint) UIntStorage;

    function getUIntValue(bytes32 record) public
        returns UIntStorage[record];
    }

    function setUIntValue(bytes32 record, uint value)
    {
        UIntStorage[record] = value;
    }

    mapping(bytes32 => bool) BooleanStorage;

    function getBooleanValue(bytes32 record) public
        returns BooleanStorage[record];
    }

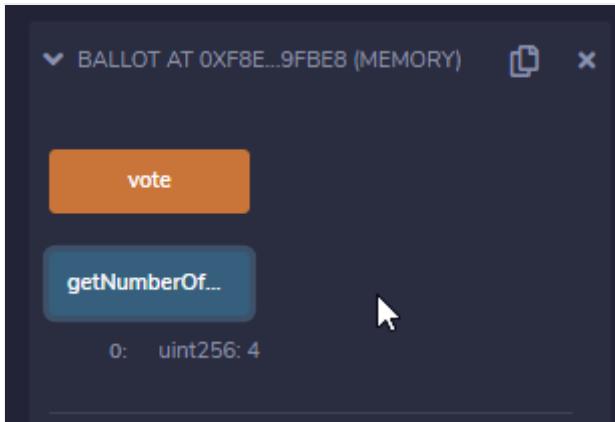
    function setBooleanValue(bytes32 record, bool value)
    {
        BooleanStorage[record] = value;
    }
}
```

Below the code, there are deployment options: 'Deploy' (button highlighted with red circle 1), 'Publish to IPFS' (checkbox highlighted with red circle 2), and 'At Address'. A red arrow points from the 'Deploy' button to the clipboard message at the bottom right.

On the right, the transaction status shows a pending creation of a Ballot contract with hash 0xa64...25fff8. A blue message box at the bottom right says 'Copied value to clipboard.'

Under the hood, a library does a `delegatecall`, which executes the libraries code in the context of the Ballot Smart Contract. If you were to use `msg.sender` in the library, then it has the same value as in the Ballot Smart Contract itself.

Let's test this by voting a few times in the new Ballot Instance:



Let's say we found a bug, because everyone can vote as many times as they want. We fix it and re-deploy only the Ballot Smart Contract (neglecting that the old version still runs and that there is no way to stop it without extra code).

Replace everything with the following code. Highlighted are the actual changes:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract EternalStorage{
    mapping(bytes32 => uint) UIntStorage;

    function getUIntValue(bytes32 record) public view returns (uint){
        return UIntStorage[record];
    }

    function setUIntValue(bytes32 record, uint value) public
    {
        UIntStorage[record] = value;
    }

    mapping(bytes32 => bool) BooleanStorage;

    function getBooleanValue(bytes32 record) public view returns (bool){
        return BooleanStorage[record];
    }

    function setBooleanValue(bytes32 record, bool value) public
    {
        BooleanStorage[record] = value;
    }
}

library ballotLib {

    function getNumberOfVotes(address _eternalStorage) public view returns (uint256) {
        return EternalStorage(_eternalStorage).getUIntValue(keccak256('votes'));
    }

    function getUserHasVoted(address _eternalStorage) public view returns(bool) {
        return EternalStorage(_eternalStorage).getBooleanValue(keccak256(abi.encodePacked("voted",msg.sender)));
    }

    function setUserHasVoted(address _eternalStorage) public {
        EternalStorage(_eternalStorage).setBooleanValue(keccak256(abi.encodePacked("voted",msg.sender)), true);
    }

    function setVoteCount(address _eternalStorage, uint _voteCount) public {
        EternalStorage(_eternalStorage).setUIntValue(keccak256('votes'), _voteCount);
    }
}

contract Ballot {
    using ballotLib for address;
    address eternalStorage;

    constructor(address _eternalStorage) {
        eternalStorage = _eternalStorage;
    }

    function getNumberOfVotes() public view returns(uint) {
        return eternalStorage.getNumberOfVotes();
    }
}
```

```

    }

    function vote() public {
        require(eternalStorage.getUserHasVoted() == false, "ERR_USER_ALREADY_VOTED");
        eternalStorage.setUserHasVoted();
        eternalStorage.setVoteCount(eternalStorage.getNumberOfVotes() + 1);
    }
}

```

You see, only the Library changed. The Storage is exactly the same as before. But how to deploy the update?

Re-Deploy the "Ballot" Smart Contract and give it the address of the Storage Contract. That's all.

The screenshot shows the Truffle UI interface divided into two main sections: the left side for deploying transactions and the right side for viewing the deployed contracts' code and logs.

Left Side (Deploy & Run Transactions):

- Gas:** 0 wei
- Contract:** Ballot - browser/Upgrades/02-ethernal... (highlighted with a red circle)
- Deploy:** Orange button with the value 0xd9145CCE52D386f254911 (highlighted with a red circle)
- Publish to IPFS:** Checkbox (unchecked)
- OR**
- At Address:** Button (selected)
- Transactions recorded:** 11
- Deployed Contracts:** List of deployed contracts:
 - ETERNALSTORAGE AT 0XD91...39138 (with a copy icon and delete icon)
 - BALLOT AT 0XF8E...9FBE8 (MEMORY) (with a copy icon and delete icon)
 - BALLOT AT 0XD2A...FD005 (MEMORY) (with a copy icon and delete icon)
- Interactions:** A section showing interactions for the BALLOT contract at 0XD2A...FD005 (MEMORY):
 - vote:** Orange button (highlighted with a red circle)
 - getNumberOf...:** Blue button
 - Output: 0: uint256: 5
- Low level interactions:** A section for interacting with the contract at the low level.
- Transact:** Orange button

```

37    }
38
39    function getUserHasVoted(address _eternalStorage) public view returns (bool) {
40        return EternalStorage(_eternalStorage).getUserHasVoted();
41    }
42
43    function setUserHasVoted(address _eternalStorage) public {
44        EternalStorage(_eternalStorage).setBoolean(true);
45    }
46
47    function setVoteCount(address _eternalStorage) public {
48        EternalStorage(_eternalStorage).setUIntValue(1);
49    }
50
51
52    contract Ballot {
53        using ballotLib for address;
54        address eternalStorage;
55
56        constructor(address _eternalStorage) {
57            eternalStorage = _eternalStorage;
58        }
59
60        function getNumberOfVotes() public view returns (uint) {
61            return eternalStorage.getNumberOfVotes();
62        }
63
64        function vote() public {
65            require(eternalStorage.getUserHasVoted() == false, "ERR_USER_ALREADY_VOTED");
66            eternalStorage.setUserHasVoted();
67            eternalStorage.setVoteCount(eternalStorage.getNumberOfVotes() + 1);
68        }
69    }
70 }

```

Right Side (Contract View):

- Home:** Home tab selected
- 01-lost-storage.sol:** Contract file tab
- 02-ethernal-storage.sol:** Contract file tab
- Logs:**
 - transact to Ballot.vote pending ... (highlighted with a red circle)
 - [vm] from: 0x5B3...eddC4 to: Ballot.vote() 0xd9145CCE52D386f254911 hash: 0x60d...4fe32 (highlighted with a red circle)
 - transact to Ballot.vote errored: VM error: revert. reason: revert: revert. Reason provided by the contract: "ERR_USER_ALREADY_VOTED"

The Storage Contract hasn't changed at all, we don't even need to redeploy it. Just use the one that already exists! You see then that you can vote one last time - so we flag your account, then you get an error (3) in the screenshot.

The original Storage Smart Contract from Elena has a couple more variable types of course, as uint and boolean would not be enough.

While it sounds good, this has some advantages and some disadvantages.

Relatively easy to understand: It doesn't involve any assembly magic at all. If you come from traditional software development, these patterns should look fairly familiar.

Would also work without Libraries, just a Storage Smart Contract running under its own address.

Eliminates the Storage Migration after Contract Updates.

Address of Contracts change - this can also be good for transparency reasons. E.g. you run an online service and fees change for new signups.

Quite difficult access pattern for variables.

Doesn't work out of the box for existing Smart Contracts like Tokens etc.

It is simple, but a very viable solution - depending on the use case. Sometimes, especially with Smart Contracts, simpler is better. If you want a real-world example of this, checkout the Smart Contracts [MorpherState](#) and [MorpherToken](#). They are linked together simply with getters and setters, but have the same effect. They are easy to audit and it's very easy to grasp what's going on under the hood in terms of data storage and retrieval.

Many other project use a proxy pattern where the address of the upgraded Smart Contract stays constant.

That's what we're talking next!

Last update: April 17, 2021

17.6 The First Proxy Contract

The first proxy that was ever proposed (to my best knowledge), came from Nick Johnson. If you don't know him, he's founder and lead dev at the ENS (Ethereum Name Service). Also, make sure to checkout his twitter, he's quite active. And he's always ahead of time, literally: he's from New Zealand - GMT+13.

The proxy looks like [this](#). I believe it was written for Sol 0.4.0 (or alike), since later Solidity version would require function visibility specifiers and an actual `pragma` line.

So, here is a copy of the same Smart Contract ported to Solidity 0.8.1 and stripped of any comments and the replace-method made public so that we can actually replace Smart Contracts. Again, it's a simplified version without any governance or control, simply showing the upgrade architecture:

```
// SPDX-License-Identifier: No-Idea!

pragma solidity 0.8.1;

abstract contract Upgradeable {
    mapping(bytes4 => uint32) _sizes;
    address _dest;

    function initialize() virtual public;

    function replace(address target) public {
        _dest = target;
        target.delegatecall(abi.encodeWithSelector(bytes4(keccak256("initialize()"))));
    }
}

contract Dispatcher is Upgradeable {

    constructor(address target) {
        replace(target);
    }

    function initialize() override public{
        // Should only be called by on target contracts, not on the dispatcher
        assert(false);
    }

    fallback() external {
        bytes4 sig;
        assembly { sig := calldataload(0) }
        uint len = _sizes[sig];
        address target = _dest;

        assembly {
            // return _dest.delegatecall(msg.data)
            calldatacopy(0x0, 0x0, calldatasize())
            let result := delegatecall(sub(gas(), 10000), target, 0x0, calldatasize(), 0, len)
            return(0, len) //we throw away any return data
        }
    }
}

contract Example is Upgradeable {
    uint _value;

    function initialize() override public {
        _sizes[bytes4(keccak256("getUint()"))] = 32;
    }

    function getUint() public view returns (uint) {
        return _value;
    }

    function setUint(uint value) public {
        _value = value;
    }
}
```

So, what's going on here? Before we try the contract, let me quickly explain the assembly in the fallback function.

What happens is basically a `delegatecall` to the Example Smart Contract. What's a delegate call anyways?

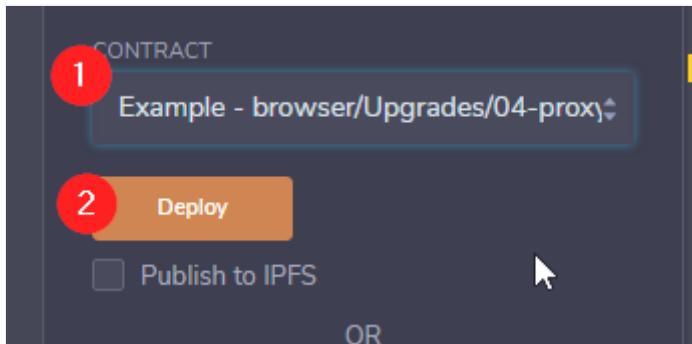
Delegatecall from the Solidity Docs

There exists a special variant of a message call, named `delegatecall` which is identical to a message call apart from the fact that the code at the target address is executed in the context of the calling contract and `msg.sender` and `msg.value` do not change their values.

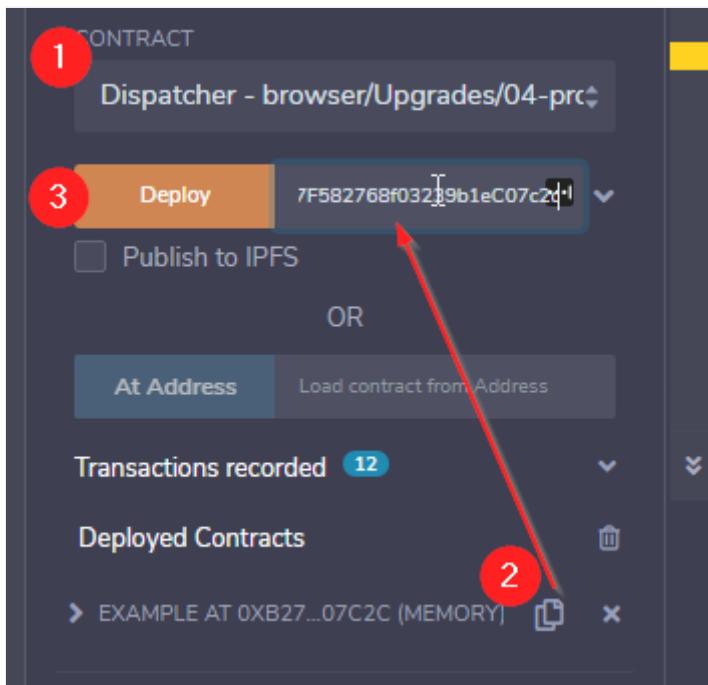
If that doesn't tell you much: Instead of running the code of the target contract on the target contracts address, we're running the code of the target contract on the contract that called the target. WOOW! Complicated sentence.

Let's play around and you see where this is going:

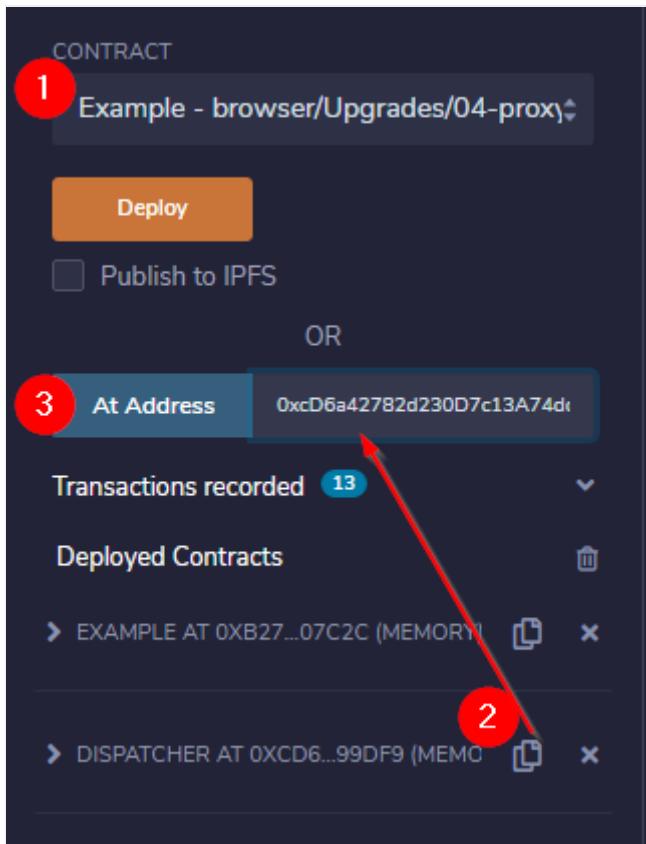
1. Deploy Example
2. Deploy the Dispatcher using the Example address as the Dispatchers constructor argument.
3. Tell Remix that the Example Contract is now running on the Dispatcher address.



then deploy the dispatcher:



then use the Example on the Dispatchers address:



Storage Pointer

Attention: This implementation only works, because the Upgradeable contract has the target address on storage slot 0. If you're interested why the other implementations use `mload(0x40)` and what happens here with the storage pointers, then checkout the following [guide](#) from OpenZeppelin, which explains this quite elegantly.

In the Example-via-Dispatcher Contract, set a uint and get a uint. Voilà, variables are stored correctly, although our Dispatcher doesn't know any `setUint` or `getUint` functions. It also doesn't inherit from Example.

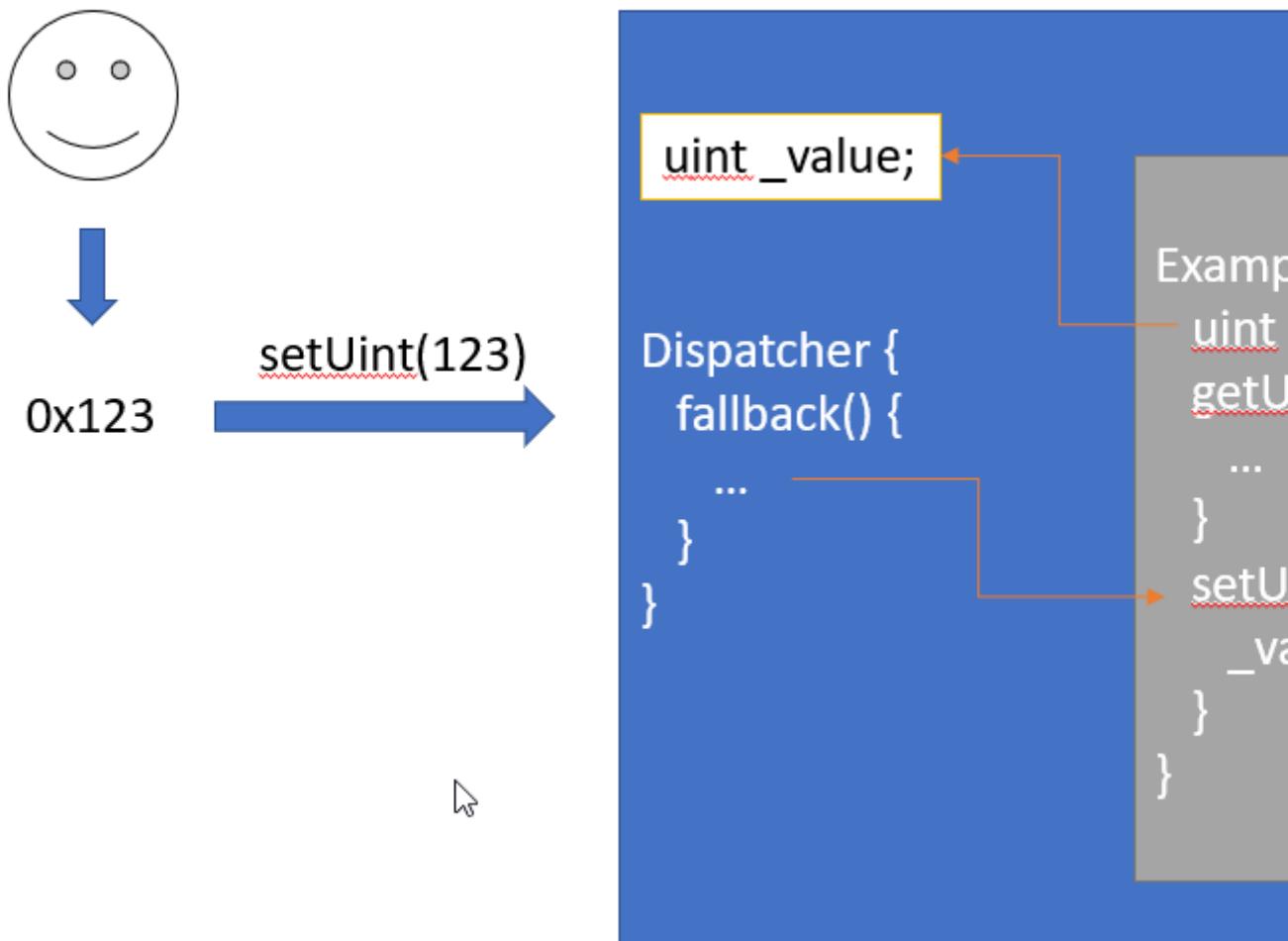
The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has a title "DEPLOY & RUN TRANSACTIONS". Below it are buttons for "Deploy" (orange), "Publish to IPFS" (grey), and "At Address" (blue). The "At Address" field contains the address "0xD6a42782d230D7c13A74d". Under "Transactions recorded", there are three entries: "EXAMPLE AT 0XB27...07C2C (MEMORY)", "DISPATCHER AT 0XCD6...99DF9 (MEMO)", and "EXAMPLE AT 0XCD6...99DF9 (MEMORY)". Each entry has a "copy" icon and a "x" icon. The third entry is expanded, showing buttons for "initialize", "replace", and "setUint". The "setUint" button is highlighted with a red circle containing the number "1". Below it is a "getUint" button with a red circle containing the number "2". The output shows "0: uint256: 555". Under "Low level interactions", there's a "CALLDATA" tab and a "Transact" button. On the right, the Solidity code for "01-lost-storage.sol" is displayed. The code defines an abstract contract "Upgradeable" and a concrete contract "Dispatcher" that delegates calls to a target contract. Transaction logs at the bottom show interactions with the dispatcher contract.

```
// SPDX-License-Identifier: No-Idea!
pragma solidity 0.8.1;
abstract contract Upgradeable {
    mapping(bytes4 => uint32) _sizes;
    address _dest;
    function initialize() virtual public;
}
contract Dispatcher is Upgradeable {
    constructor(address target) {
        replace(target);
    }
    function initialize() override public{
        // Should only be called by on target
        assert(false);
    }
    fallback() external {
        bytes4 sig;
        assembly { sig := calldataload(0) }
        uint len = _sizes[sig];
        address target = _dest;
        assembly {

```

Pretty cool!

This will essentially use the Dispatcher as a storage, but use the logic stored on the Example contract to control what happens. Instead of the Dispatcher "talking to" the Example contract, we're now moving the code of the Example contract into the scope of the Dispatcher and executing it there - changing the Dispatchers storage. That is a huge difference to before with the EternalStorage pattern.



The op-code `delegatecall` will "move" the Example contract into the Dispatcher and use the Dispatcher's storage.

It's a great example of a first proxy implementation. Especially, considering it was early days for Solidity development, that was quite forward thinking!

Let's say we want to upgrade our Smart Contract returning 2* the uint value from `getUint()`:

```

//... more code
contract Example is Upgradeable {
  uint _value;

  function initialize() override public {
    _sizes[bytes4(keccak256("getUint()"))] = 32;
  }

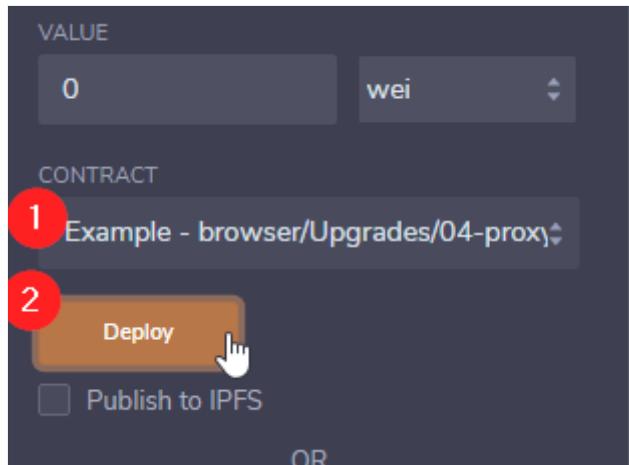
  function getUint() public view returns (uint) {
    return _value*2;
  }

  function setUint(uint value) public {
    _value = value;
  }
}

```

That's how you can upgrade your logic contract using the `replace` method:

1. Update the Example Contract, for example return 2* the value in `getUint()`
2. Deploy the Example Contract
3. Copy the Example Contract address
4. Call `replace` in the Dispatcher with the new Example Contract address



then call replace:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" and "01-lost-storage.sol".

Deployment Options:

- "Deploy" button
- "Publish to IPFS" checkbox
- "At Address" dropdown set to "0xD6a42782d230D7c13A74dC"
- "Transactions recorded" dropdown set to "15"
- "Deployed Contracts" section listing:
 - EXAMPLE AT 0XB27...07C2C (MEMORY)
 - DISPATCHER AT 0XCD6...99DF9 (MEMORY)
- "initialize" button
- "replace" button (highlighted with a red circle and number 2)
- "Low level interactions" section with a "Transact" button (highlighted with a red circle and number 1)

Code Editor (01-lost-storage.sol):

```

24 // Should only be called by on target contract
25 assert(false);
26 }
27
28 fallback() external {
29   bytes4 sig;
30   assembly { sig := calldataload(0) }
31   uint len = _sizes[sig];
32   address target = _dest;
33
34 assembly {
35   // return _dest.delegatecall(msg.data)
36   calldatacopy(0x0, 0x0, calldatasize());
37   let result := delegatecall(sub(gas()), target, 0, len)
38   return(0, len) //we throw away any return value
39 }
40 }
41
42 contract Example is Upgradeable {
43   uint _value;
44
45   function initialize() override public {
46     _sizes[bytes4(keccak256("getUint()"))] =
47   }
48
49   function getUint() public view returns (uint) {
50     return _value*2;
51   }
52
53   function setUint(uint value) public {
54     _value = value;
55   }
56 }
57

```

Logs:

- CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875 data: 0x000...267a4
- creation of Example pending...
- [vm] from: 0x5B38Da6a701c568545dCfcB03FcB875 hash: 0: Copied value to clipboard.

You can still use the old instance, it will return now 2* the value.

The screenshot shows the Truffle UI interface with two contracts loaded:

- 01-lost-storage.sol** (Deployed at `0xCD6...990F9 (MEMORY)`):


```

24 // Should only be called by on target contract
25 assert(false);
26
27
28 fallback() external {
29     bytes4 sig;
30     assembly { sig := calldataload(0) }
31     uint len = _sizes[sig];
32     address target = _dest;
33
34     assembly {
35         // return _dest.delegatecall(msg.data)
36         calldatcopy(0x0, 0x0, calldatasize())
37         let result := delegatecall(sub(gas(), len), target, len)
38         return(0, len) //we throw away any return value
39     }
40 }
41
42 contract Example is Upgradeable {
43     uint _value;
44
45     function initialize() override public {
46         _sizes[bytes4(keccak256("getUInt()"))] = 32;
47     }
48
49     function getUInt() public view returns (uint) {
50         return _value*2;
51     }
52
53     function setUInt(uint value) public {
54         _value = value;
55     }
56 }
```
- 02-ethernal-storage.sol** (Deployed at `0x9d803e140330758a8FD0`):


```

DEPLOY & RUN TRANSACTIONS
DISPATCHER AT 0xCD6...990F9 (MEMORY)
Low level interactions
CALldata
Transact
EXAMPLE AT 0x9d803e140330758a8FD0 (MEMORY)
Low level interactions
CALldata
Transact
```

The UI displays various interaction buttons like `initialize`, `replace`, and `call`. The `call` button for the `getUInt` function on the `Example` contract is highlighted. The result of the call is shown as `0: uint256: 1110`.

The terminal window on the right shows transaction logs:

```

[vm] from: 0x583...addC4 to: Dispatcher.replace
data: 0xcab...a5692 logs: 0 hash: 0x7bc...93d13
call to Example.getUInt
[call] from: 0x58380da6a781c568545dCfcB83Fc8875...
data: 0x000...267a4
```

Obviously there's a lot going on under the hood. And this is not the end of the whole story, but it's the beginning of how Proxies work internally.

It has a great disadvantage though: You need to extend from the Upgradeable Smart Contract in all Contracts that are using the Dispatcher, otherwise you will get Storage collisions.

But what are Storage Collisions anyways?

Last update: April 17, 2021

17.7 Understanding Storage and Storage Collisions

Let's try a set of two Smart Contracts, where one is a proxy pattern. You'll see in a second how the storage clashes in the proxy with the first variable.

Copy and paste this contract to Remix:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract LostStorage {
    address public myAddress;
    uint public myUint;

    function setAddress(address _address) public {
        myAddress = _address;
    }

    function setMyUint(uint _uint) public {
        myUint = _uint;
    }
}

contract ProxyClash {
    address public otherContractAddress;

    constructor(address _otherContract) {
        otherContractAddress = _otherContract;
    }

    function setOtherAddress(address _otherContract) public {
        otherContractAddress = _otherContract;
    }

    fallback() external {
        address _impl = otherContractAddress;

        assembly {
            let ptr := mload(0x40)
            calldatadcopy(ptr, 0, calldatasize())
            let result := delegatecall(gas(), _impl, ptr, calldatasize(), 0, 0)
            let size := returndatasize()
            returndatadcopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

This fallback function looks slightly more complicated than the previous one, but it does essentially the same thing. Here it also can return values and throw exceptions if there were any in the target contract. A lot happened since Solidity 0.4 and 0.8...

One **major** difference is that the LostStorage is not inheriting the Proxy. So, internally they have separated storage layout and both start from storage slot 0.

What are we going to do with this?

1. Deploy the LostStorage Contract
2. Deploy the Proxy, setting the LostStorage contract address as the constructor argument
3. Tell Remix that the LostStorage is running on the Proxy address
4. Call `myAddress()`. It surprisingly returns a non-zero address. BAM! Collision.

The screenshot shows the Truffle UI interface with two main panes. The left pane, titled "DEPLOY & RUN TRANSACTIONS", displays the deployment of two contracts: "LOSTSTORAGE AT 0XD91..." and "PROXYCLASH AT 0XD8B...". The right pane shows the Solidity source code for "01-lost-storage.sol" which defines a proxy contract "ProxyClash" that inherits from "LostStorage". The UI also shows transaction logs and a search bar.

```

11   function setMyUint(uint _uint) public {
12     myUint = _uint;
13   }
14 }
15
16
17
18 contract ProxyClash {
19   address public otherContractAddress;
20
21   constructor(address _otherContract) {
22     otherContractAddress = _otherContract;
23   }
24
25   function setOtherAddress(address _otherContract)
26     otherContractAddress = _otherContract;
27   }
28
29 fallback() external {
30   address _impl = otherContractAddress;
31
32   assembly {
33     let ptr := mload(0x40)
34     calldatadcopy(ptr, 0, calldatasize())
35     let result := delegatecall(gas(), _impl, pt,
36                               calldatasize())
37     let size := returndatasize()
38     returndatadcopy(ptr, 0, size)
39
40     switch result
41     case 0 { revert(ptr, size) }
42     default { return(ptr, size) }
43   }
44 }

```

That is exactly why we do inheritance with a Storage Contract, so that the Solidity compiler knows where the Storage slots are used. And we will later see that there's an elegant solution around that.

Let's start with the first EIP for Proxies!

Last update: April 17, 2021

17.8 EIP-897: The first *real* Proxy

In order to avoid having two variables taking the same storage slot, we need all contracts to be aware of the additional storage necessary for the proxy. In other words: If we had a separate Smart Contract that does only the storage for the Proxy, then we can use this as a base contract for our `LostStorage`, which is now not lost anymore.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract ProxyStorage {
    address public otherContractAddress;

    function setOtherAddressStorage(address _otherContract) internal {
        otherContractAddress = _otherContract;
    }
}

contract NotLostStorage is ProxyStorage {
    address public myAddress;
    uint public myUint;

    function setAddress(address _address) public {
        myAddress = _address;
    }

    function setMyUint(uint _uint) public {
        myUint = _uint;
    }
}

contract ProxyNoMoreClash is ProxyStorage {

    constructor(address _otherContract) {
        setOtherAddress(_otherContract);
    }

    function setOtherAddress(address _otherContract) public {
        super.setOtherAddressStorage(_otherContract);
    }

    /**
     * @dev Fallback function allowing to perform a delegatecall to the given implementation.
     * This function will return whatever the implementation call returns
     */
    fallback() payable external {
        address _impl = otherContractAddress;

        assembly {
            let ptr := mload(0x40)
            calldatadcopy(ptr, 0, calldatasize())
            let result := delegatecall(gas(), _impl, ptr, calldatasize(), 0, 0)
            let size := returndatasize()
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

If you have a look at the [EIP-897](#), then you'll see it references an implementation from [aragonOS](#) and [zeppelinOS](#). Under the hood it is this sample-implementation here. They just add more bang like ownership so that only the admin can do upgrades etc. In its essence: that's it. Period.

Let's give it a try?

Same pattern as above:

1. Deploy the `NoLostStorage`
2. Deploy the Proxy with the `NoLostStorage` address as the constructor argument
3. Tell Remix that the `NoLostStorage` contract is running on the Proxy
4. Call `myAddress()` - it's zero now, and you can set it to whatever you want.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, publishing to IPFS, and managing contracts. The main area has tabs for 'DEPLOY & RUN TRANSACTIONS' and 'CONTRACT'. Under 'CONTRACT', it shows 'NotLostStorage - browser/Upgrades/0'. Below that is a 'Deploy' button and a checkbox for 'Publish to IPFS'. A section labeled 'OR' shows 'At Address' and the address '0xD7ACd2a9FD159E69Bb102A'. Below this, a dropdown says 'Transactions recorded' with a count of 2. A list of 'Deployed Contracts' shows three entries: 1. NOTLOSTSTORAGE AT 0XF8E...9FBE8, 2. PROXYNOMORECLASH AT 0XD7A...F77, and 3. NOTLOSTSTORAGE AT 0XD7A...F771B. Each entry has a clipboard icon and a delete icon. Numbered circles (1, 2, 3) highlight these three contracts. To the right, a code editor shows the Solidity code for 'ProxyNoMoreClash' which inherits from 'ProxyStorage'. A red arrow points from the 'myAddress' slot in the UI to the corresponding storage slot in the code. On the far right, a transaction history shows a call to 'NotLostStorage.myAddress'.

```

24
25
26
27
28 contract ProxyNoMoreClash is ProxyStorage {
29
30     constructor(address _otherContract) {
31         setOtherAddress(_otherContract);
32     }
33
34     function setOtherAddress(address _otherContr
35         super.setOtherAddressStorage(_otherContr
36     }
37
38     /**
39     * @dev Fallback function allowing to perform a
40     * This function will return whatever the imple
41     */
42     fallback() payable external {
43         address _impl = otherContractAddress;
44
45         assembly {
46             let ptr := mload(0x40)
47             calldatadcopy(ptr, 0, calldatasize())
48             let result := delegatecall(gas(), _impl, p
49             let size := returndatasize()
50             returndatadcopy(ptr, 0, size)
51
52             switch result
53             case 0 { revert(ptr, size) }
54             default { return(ptr, size) }
55         }
56     }
57 }

```

As the ProxyStorage contract is inherited by both, the NoLostStorage and the Proxy, the compiler will know that it can't just start again from storage slot 0. You will not overwrite the storage slot anymore.

But there are also some downsides to this.

The Downside: Contract Modifications

While this solution sounds pretty cool at first, there is an obvious downside to this approach! All upgradeable Smart Contracts have to extend ProxyStorage for this to work.

If you develop all your Smart Contracts yourself, then you can probably add in the ProxyStorage Smart Contract to all your Smart Contracts, but as soon as you go standardized - maybe with Smart Contract packages from OpenZeppelin ., then it becomes increasingly harder.

So, what if there was another way to avoid those storage collisions?

Last update: April 17, 2021

17.9 EIP-1822: Proxies without Storage Collision without common Storage Contracts

Welcome [EIP-1822: Universal Upgradeable Proxy Standard \(UUPS\)](#). A clever solution without the need for a common Storage Smart Contract to let the compiler know which storage slots to use.

So, instead this methods just simply uses a pseudo-random storage slot to store the address of the logic contract.

Before I show you the example, the two important lines are these ones:

```
sstore(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7, contractLogic)
```

and

```
let contractLogic := sload(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7)
```

So, in assembly you can store some variable to a specific storage slot and then load it again from that slot. In this case the EIP-1822 uses the `keccak256("PROXIABLE") = "0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7"` which results in the storage slot. It's not 100% random, but random enough so that there's no collision happening. Under normal circumstances at least. You can deep dive into the [Layout of Storage Variables in Solidity](#) then you'll see that there is little chance to create a collision.

The full example using EIP-1822 could look like this:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

contract Proxy {
    // Code position in storage is keccak256("PROXIABLE") = "0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7"
    constructor(bytes memory constructData, address contractLogic) {
        // save the code address
        assembly { // solium-disable-line
            sstore(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7, contractLogic)
        }
        (bool success, bytes memory result) = contractLogic.delegatecall(constructData); // solium-disable-line
        require(success, "Construction failed");
    }

    fallback() external payable {
        assembly { // solium-disable-line
            let contractLogic := sload(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7)
            calldatadcopy(0x0, 0x0, calldatasize())
            let success := delegatecall(sub(gas()), 10000, contractLogic, 0x0, calldatasize(), 0, 0)
            let retSz := returndatasize()
            returnndatacopy(0, 0, retSz)
            switch success
            case 0 {
                revert(0, retSz)
            }
            default {
                return(0, retSz)
            }
        }
    }
}

contract Proxiable {
    // Code position in storage is keccak256("PROXIABLE") = "0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7"

    function updateCodeAddress(address newAddress) internal {
        require(
            bytes32(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7) == Proxiable(newAddress).proxiableUUID(),
            "Not compatible"
        );
        assembly { // solium-disable-line
            sstore(0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7, newAddress)
        }
    }

    function proxiableUUID() public pure returns (bytes32) {
        return 0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7;
    }
}

contract MyContract {

    address public owner;
    uint public myUint;
```

```

function constructor1() public {
    require(owner == address(0), "Already initialized");
    owner = msg.sender;
}

function increment() public {
    //require(msg.sender == owner, "Only the owner can increment"); //someone forgot to uncomment this
    myUint++;
}

contract MyFinalContract is MyContract, Proxiable {

    function updateCode(address newCode) onlyOwner public {
        updateCodeAddress(newCode);
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner is allowed to perform this action");
        _;
    }
}

```

1. Deploy the `MyFinalContract`
2. Deploy the Proxy, argument is the `MyFinalContract` Address and as a calldata the `bytes4(keccak256("constructor1()))`. This can be done with `web3.utils.sha3('constructor1()').substring(0,10)` in the Remix Console. See picture below.
3. Then simply tell Remix that `MyFinalContract` is running on the address of the Proxy Contract. As you did before.

DEPLOY & RUN TRANSACTIONS

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: Proxy - browser/Upgrades/01-lost-sto

DEPLOY

CONSTRUCTDATA: 0x473be604

CONTRACTLOGIC: 0x7EF2e0048f5bAe1e046f6E

transact

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

MYFINALCONTRACT AT 0X7EF...8CB47

Home 01-lost-storage.sol

```

45 }
46
47 function proxiableUUID() public pure returns
48     return 0xc5f16f0fcc639fa48a6947836d9850F
49 }
50 }
51
52 contract MyContract {
53
54     address public owner;
55     uint public myUint;
56
57 function constructor1() public {
58     require(owner == address(0), "Already in");
59     owner = msg.sender;
60 }
61
62 function increment() public {
63     //require(msg.sender == owner, "Only the
64     myUint++;
65 }
66
67
68 contract MyFinalContract is MyContract, Proxiable
69
70 function updateCode(address newCode) onlyOwner
71     updateCodeAddress(newCode);
72 }
73
74 modifier onlyOwner() {
75     require(msg.sender == owner, "Only owner
76     _;
77 }
78 }
```

creation of MyFinalContract pending...

[vm] from: 0x5B3...eddC4 to: MyFinalContract.(0x7EF2e0048f5bAe1e046f6E)

> web3.utils.sha3('constructor1()').substring(0,10)

As you can see, if you follow the steps, the Contract is now aware of the logic from the MyFinalContract - which can inherit any contract, neglecting any Storage inheritance, because it can actually start from storage slot 0.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running, and monitoring transactions. Below it, a list of deployed contracts is shown:

- 1 MYFINALCONTRACT AT 0X7EF...8CB47**: Shows storage slots: myUint (0: uint256: 0) and owner (0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4).
- 2 PROXY AT 0XDA0...42B53 (MEMORY)**: Shows storage slot: proxiableUUID (0: address: 0xc5f16f0fcc639fa48a6947836d9850f).
- 3 MYFINALCONTRACT AT 0XDA0...42B53**: Shows storage slot: proxiableUUID (0: address: 0xc5f16f0fcc639fa48a6947836d9850f).

On the right, the Solidity code for the contracts is displayed:

```

45 }
46
47 function proxiableUUID() public pure returns (address)
48     return 0xc5f16f0fcc639fa48a6947836d9850f;
49 }
50
51 contract MyContract {
52
53     address public owner;
54     uint public myUint;
55
56     function constructor1() public {
57         require(owner == address(0), "Already initialized");
58         owner = msg.sender;
59     }
60
61     function increment() public {
62         //require(msg.sender == owner, "Only the owner can increment");
63         myUint++;
64     }
65 }
66
67 contract MyFinalContract is MyContract, ProxiableStorage {
68
69     function updateCode(address newCode) onlyOwner {
70         updateCodeAddress(newCode);
71     }
72
73     modifier onlyOwner() {
74         require(msg.sender == owner, "Only owner can call this function");
75         _;
76     }
77 }
78

```

Below the code, there are logs and transaction details:

- CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4**
- call to MyFinalContract.owner**
- CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4**
- > web3.utils.sha3('constructor1()').substring(0,10)**

If there's a change: Deploy a new version of the `MyFinalContract` then update the Proxy with the new address.

Removal of Variables is impossible

One very important thing to note is that you can't remove or mix variables that were defined earlier. The problem is that they still reside in a specific storage slot in the Proxy contract (pulled in the scope of the logic contract).

If you remove a variable, then the Solidity compiler will simply assume that the next variable is on the place of the previous one. Your storage will clash again.

It's already a pretty good implementation! The only problem here is that the storage slot isn't really standardized. That means, you can pretty much choose any storage slot you want to store the logic contract address.

For block explorers that makes it very hard to act upon and show information to the user.

Welcome EIP-1967...

Last update: April 17, 2021

17.10 EIP-1967 Standard Proxy Storage Slots

This EIP standardizes how proxies store the logic contract address. Having this makes it easier for outside services, such as block explorers, to show the correct information to the end-user. Etherscan [added support for this](#) end of 2019. The EIP-1967 also adds a few more gimmicks to the overall pattern!

What's the main function? The storage slot, obviously. While in EIP-1822 it was somewhat keccak256("PROXYABLE") - or anything of your choice really - in EIP-1967 it is well defined:

```
Storage slot 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc (obtained as
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1) ) is reserved for the logic contract.
```

But EIP-1967 also adds beacon contracts and storage for the actual admin functionality.

17.10.1 The Beacon Contract

The idea behind the beacon contract is re-usability. If you have several proxies pointing to the same logic contract address then, every time you want to update the logic contract, you'd have to update all proxies. As this can become gas intensive, it would make more sense to have a beacon contract that returns the address of the logic contract for all proxies.

So, if you use beacons, you are having another layer of Smart Contract in between that returns the address of the actual logic contract.

A really good sample implementation is used by OpenZeppelin [in their Repository](#).

As this is basically the same functionality as EIP-1822, just with a clear defined namespace, I'll refer at this point to the examples we did in the previous explanation.

Instead of repeating the same experiment as before, I want to talk about another pattern: The diamond storage pattern.

Last update: April 17, 2021

17.11 EIP-1538: Transparent Contract Standard

Now it's time to talk about two things: [EIP-1538: Transparent Contract Standard](#) and [EIP-2535: Diamond Standard](#). EIP-2535 replaces EIP-1538 and both were created by Nick Mudge, so we will briefly touch on the idea of EIP-1538 without going into too much detail, just to understand what's going on.

This was the first implementation, which does something very clever: Instead of defining a logic contract as a whole, it basically extracts the functions of logic contracts and sets an address for it.

This way you can have as many logic contracts as you want, and update functions incrementally.

A sample implementation can be seen [in the repository](#).

I will explain how it works, but not further dive into EIP-1538, because it was withdrawn and superseded by EIP-2535.

From [the test case](#), you see it all revolves around "MyTransparentContract", which also contains the fallback function that does the delegatecall. It gets the address of ERC1538Delegate, which contains functionality to map function-signatures (bytes4) to addresses.

Later, the fallback function in MyTransparentContract will use lookups to determine which function signature runs on which address and does the `delegatecall` from within the MyTransparentContract.

It needs quite a bit of setup: For example for an ERC20 Token, you would need to give it all function signatures that are running on the ERC20 address and add it to the mappings through MyTransparentContracts `updateContract` which is the logic used from ERC1538Delegate.

It's quite complex to understand and, at least in my opinion, does solve only one thing: You can get around the 24KB maximum contract size limitation.

I might be wrong, but I don't see gas savings by adding functions atomically, because to upgrade a function I would still need to deploy the whole contract first. I do understand that you can [get around](#) this to some extend, by providing virtual functions, but not enough to count as atomic updates for my understanding. So, for example, if you deploy a mintable ERC20 contract and you want to change the mint-function somehow, you would still need to re-deploy the whole ERC20 contract with all the functions the mint function depends on, including the new mint function, to change it.

But it seems Nick Mudge came up with a better solution. So, let's talk about Diamonds here...

Last update: April 17, 2021

17.12 EIP-2535: Diamond Standard

The Diamond Standard is an improvement over EIP-1538. It has the same idea: To map single functions for a delegatecall to addresses, instead of proxying a whole contract through.

The important part of the Diamond Standard is the way storage works. Unlike the unstructured storage pattern that OpenZeppelin uses, the Diamond Storage is putting a single `struct` to a specific storage slot.

Function wise it looks like this, given from the EIP Page:

```
// A contract that implements diamond storage.
library LibA {

    // This struct contains state variables we care about.
    struct DiamondStorage {
        address owner;
        bytes32 dataA;
    }

    // Returns the struct from a specified position in contract storage
    // ds is short for DiamondStorage
    function diamondStorage() internal pure returns(DiamondStorage storage ds) {
        // Specifies a random position from a hash of a string
        bytes32 storagePosition = keccak256("diamond.storage.LibA")
        // Set the position of our struct in contract storage
        assembly {ds.slot := storagePosition}
    }
}

// Our facet uses the diamond storage defined above.
contract FacetA {

    function setDataA(bytes32 _dataA) external {
        LibA.DiamondStorage storage ds = LibA.diamondStorage();
        require(ds.owner == msg.sender, "Must be owner.");
        ds.dataA = _dataA
    }

    function getDataA() external view returns (bytes32) {
        return LibDiamond.diamondStorage().dataA
    }
}
```

Having this, you can have as many LibXYZ and FacetXYZ as you want, they are always in a separate storage slot as a whole, because of the whole `struct`. To completely understand it, this is stored in the Proxy contract that does the delegatecall, not in the Faucet itself.

That's why you can share storage across other faucets. Every storage slot is defined manually (`keccak256("diamond.storage.LibXYZ")`).

17.12.1 The Proxy Contract

In the "Diamond Standard" everything revolves around the Diamond terms. The idea is quite visually cutting a Diamond to add functions (or mapping of addresses to functions and vice versa).

The function to add Facets and functions is called "diamondCut".

The functionality to view what functions a Facet has is called "Loupe": It returns the function signatures and addresses and everything else you might want to know about a Facet.

There is not one way to implement this functionality. Nick went ahead and created three different ways to do a reference implementation, which can be seen on [his repository](#).

First, checkout how the Smart Contracts are deployed in [the migration file](#). This reveals that deploying the `Diamond` contract already gives the addresses and function selectors of the `DiamondCutFacet` and the `DiamondLoupeFacet`. Essentially making them part of the Diamond Proxy.

If you checkout [the test-case](#), then you see exactly that the first test cases are getting back address<->signature mapping and checking that these were really set in the Diamond proxy. Line 121 is where the Test1Facet and then later Test2Facet functions are added.

17.12.2 Giving It A Try

First, we need to clone the repository:

```
git clone https://github.com/mudgen/diamond-1.git
```

then we start ganache-cli (download it with `npm install -g ganache-cli` if you don't have it), in a second terminal window:

```
ganache-cli
```

then we simply run the tests and have a look what happens

```
truffle test
```

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

> Compiling .\contracts\interfaces\IERC173.sol
> Compiling .\contracts\libraries\LibDiamond.sol
> Compiling .\contracts\interfaces\IDiamondCut.sol
> Compiling .\contracts\interfaces\IDiamondLoupe.sol
> Compiling .\contracts\interfaces\IERC165.sol
> Compiling .\contracts\interfaces\IERC173.sol
> Compiling .\contracts\libraries\LibDiamond.sol
> Artifacts written to C:\Users\thoma\AppData\Local\Temp\test--20568-TAsM4CAcG5KF
> Compiled successfully using:
  - solc: 0.7.6+commit.7338295f.Emscripten clang

Contract: Cache bug test
  ✓ should not exhibit the cache bug (75ms)

Contract: DiamondTest
  ✓ should have three facets -- call to facetAddresses function (56ms)
  ✓ facets should have the right function selectors -- call to facetFunctionSelectors function (124ms)
  ✓ selectors should be associated to facets correctly -- multiple calls to facetAddress function (16ms)
  ✓ should get all the facets and function selectors of the diamond -- call to facets function (54ms)
  ✓ should add test1 functions (433ms)
  ✓ should test function call (43ms)
  ✓ should replace test1 function (117ms)
  ✓ should add test2 functions (383ms)
  ✓ should remove some test2 functions (219ms)
  ✓ should remove some test1 functions (269ms)
  ✓ remove all functions and facets accept diamondCut and facets (775ms)
  ✓ add most functions and facets (795ms)

13 passing (4s)

diamond-1> []

```

What you can observe is that the diamondCut interface is only available through the library and called in the Diamond contract in the constructor. If you were to remove the complete update functionality, you can simply remove the diamondCut function.

Let's add a new file "FacetA.sol" in the contracts/facets folder with a bugfixed version of the content given above to write a simple variable and add it to the Diamond in the test case!

`contracts/facets/FacetA.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

library LibA {

// This struct contains state variables we care about.
struct DiamondStorage {
    address owner;
    bytes32 dataA;
}

// Returns the struct from a specified position in contract storage
// ds is short for DiamondStorage
function diamondStorage() internal pure returns(DiamondStorage storage ds) {
    // Specifies a random position from a hash of a string
    bytes32 storagePosition = keccak256("diamond.storage.LibA");
    // Set the position of our struct in contract storage
    assembly {
        ds.slot := storagePosition
    }
}

// Our facet uses the diamond storage defined above.
contract FacetA {

function setDataA(bytes32 _dataA) external {
    LibA.DiamondStorage storage ds = LibA.diamondStorage();
    ds.dataA = _dataA;
}

function getDataA() external view returns (bytes32) {
    return LibA.diamondStorage().dataA;
}
}
```

Let's also adapt our migrations file:

`/migrations/03_faceta.js`

```
const FacetA = artifacts.require('Test2Facet')

module.exports = function (deployer, network, accounts) {
deployer.deploy(FacetA)
}
```

If you paid attention so far, then you'll see the code, as it is right now, isn't very secure because anyone in any facet can retrieve keccak256("diamond.storage.LibA"); and overwrite the storage slot.

Add the following unit-test:

```
/test/facetA.test.js

/* eslint-disable prefer-const */
/* global contract artifacts web3 before it assert */

const Diamond = artifacts.require('Diamond')
const DiamondCutFacet = artifacts.require('DiamondCutFacet')
const DiamondLoupeFacet = artifacts.require('DiamondLoupeFacet')
const OwnershipFacet = artifacts.require('OwnershipFacet')
const FacetA = artifacts.require('FacetA')
const FacetCutAction = {
  Add: 0,
  Replace: 1,
  Remove: 2
}

const zeroAddress = '0x0000000000000000000000000000000000000000000000000000000000';

function getSelectors (contract) {
  const selectors = contract.abi.reduce((acc, val) => {
    if (val.type === 'function') {
      acc.push(val.signature)
    }
    return acc
  }, [])
}
return selectors
}

contract('FacetA Test', async (accounts) => {

it('should add FacetA functions', async () => {
  let facetA = await FacetA.deployed();
  let selectors = getSelectors(facetA);
  let addresses = [];
  addresses.push(facetA.address);
  let diamond = await Diamond.deployed();
  let diamondCutFacet = await DiamondCutFacet.at(diamond.address);
  await diamondCutFacet.diamondCut([[facetA.address, FacetCutAction.Add, selectors]], zeroAddress, '0x');

  let diamondLoupeFacet = await DiamondLoupeFacet.at(diamond.address);
  result = await diamondLoupeFacet.facetFunctionSelectors(addresses[0]);
  assert.sameMembers(result, selectors)
})

it('should test function call', async () => {
  let diamond = await Diamond.deployed();
  let facetAViaDiamond = await FacetA.at(diamond.address);
  const dataToStore = '0xabcdef';
  await facetAViaDiamond.setDataA(dataToStore);
  let dataA = await facetAViaDiamond.getDataA();
  assert.equal(dataA, web3.eth.abi.encodeParameter('bytes32', dataToStore));
})
})
})
```

If you run the test with `truffle test test/facetA.test.js` then you'll see that it adds the functions from `FacetA.sol` to the `Diamond`. In the second test case it stores a value and retrieves it again.

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
=====
Compiling your contracts...
=====
> Compiling .\contracts\Diamond.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\facets\DiamondCutFacet.sol
> Compiling .\contracts\facets\DiamondLoupeFacet.sol
> Compiling .\contracts\facets\FacetA.sol
> Compiling .\contracts\facets\OwnershipFacet.sol
> Compiling .\contracts\facets\Test1Facet.sol
> Compiling .\contracts\facets\Test2Facet.sol
> Compiling .\contracts\interfaces\IDiamondCut.sol
> Compiling .\contracts\interfaces\IDiamondLoupe.sol
> Compiling .\contracts\interfaces\IERC165.sol
> Compiling .\contracts\interfaces\IERC173.sol
> Compiling .\contracts\libraries\LibDiamond.sol
> Compiling .\contracts\interfaces\IDiamondCut.sol
> Compiling .\contracts\interfaces\IDiamondLoupe.sol
> Compiling .\contracts\interfaces\IERC165.sol
> Compiling .\contracts\interfaces\IERC173.sol
> Compiling .\contracts\libraries\LibDiamond.sol
> Artifacts written to C:\Users\thoma\AppData\Local\Temp\test--9528-iG3AUKtfI9It
> Compiled successfully using:
  - solc: 0.7.6+commit.7338295f.Emscripten clang
Contract: FacetA Test
  ✓ should add FacetA functions (183ms)
  ✓ should test function call (100ms)

2 passing (390ms)
diamond-1>

```

17.12.3 Pros and Cons

On the plus side, this is an interesting concept for circumventing very large Smart Contracts limits and gradually updating your Contracts. It definitely is in its infancy and should be investigated further.

I was hoping you could get a framework that let's you break up your Smart Contracts into smaller parts and deploy and update each one of them separately. It does that, somehow, but it also doesn't, since Facets still need a complete picture of internally used functions and signatures.

All in all, I believe Nick is on a good way to get there. There are, however, a few major drawbacks which need makes it unusable for us:

- The proxy could be a central point of entry to a larger ecosystem of Smart Contracts. Unfortunately, larger systems often make use of inheritance quite heavily and therefore you have to be extremely careful with adding functions to the Diamond proxy. Also function signatures could easily collide for two different parts of the system with the same name.
- Every Smart Contract in the System needs adoption for the Diamond Storage, unless you use only one single facet that uses unstructured storage. Simply adding the OpenZeppelin ERC20 or ERC777 tokens wouldn't be advised, as they would start writing to the Diamond Contract storage slot 0.
- Sharing storage between facets is dangerous. It puts a lot of liability on the admin.
- Adding functions to the Diamond via diamondCut is quite cumbersome. I do understand that there are other techniques where the facets bring their own configuration - which is much better, like in [this blog post](#).
- Adding functions to the Diamond via DiamondCut could become quite gas heavy. Adding the two functions for our FacetA Contract costs 109316. That's \$20. Extra.

Alright, now we come to the last part of this article. Wild Magic with CREATE2...

Last update: April 17, 2021

17.13 Metamorphosis Smart Contracts using CREATE2

So far, all of the Smart Contracts are linking a Proxy to another Smart Contract through `delegatecall`. So the proxy address stays constant and all calls are forwarded from (or executed in scope of) the Proxy.

What if there was a way to replace a Smart Contract all-together?

Turns out, there is! It's called "Metamorphosis Smart Contracts" and feels a bit like this:



With this solution you deploy a Smart Contract that deploys a Smart Contract that replaces its own bytecode with another Smart Contract. So, like Jim talks to Scotty to beam stuff around. Let's see how that works.

Very low level

Attention, we're going very low level here now. It's super advanced stuff, it might take some time to fully grasp the full details of what we're doing here. I will try my best to go as detailed as possible on the underlaying architecture.

17.13.1 How CREATE2 works - A Primer

A quick primer on how CREATE2 works. CREATE2 is an assembly op-code for Solidity to create a Smart Contract on a specific address. CREATE2 has a cool advantage: This address is known in advance.

The address of Smart Contracts is normally created by taking the `deployersAddress` and the nonce. The nonce is ever increasing, but with CREATE2 there's no nonce, instead a salt. The salt can be defined by the user.

So, you can know the address of a Smart Contract in advance. CREATE2 has the following specification:

```
keccak256(0xff ++ deployersAddr ++ salt ++ keccak256(bytecode))[12:]
```

1. 0xFF, a constant
2. the address of the deployer, so the Smart Contracts address that sends the CREATE2
3. A random salt
4. And the hashed bytecode that will be deployed on that particular address

this will give you the address where the new Smart Contract is deployed.

Let's try this:

First, we need a factory contract that deploys contracts:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

contract Factory {
    event Deployed(address _addr);
    function deploy(uint salt, bytes calldata bytecode) public {
        bytes memory implInitCode = bytecode;
        address addr;
        assembly {
            let encoded_data := add(0x20, implInitCode) // load initialization code.
            let encoded_size := mload(implInitCode) // load init code's length.
            addr := create2(0, encoded_data, encoded_size, salt)
        }
        emit Deployed(addr);
    }
}
```

That's hopefully fairly straight forward: When a new contract is deployed we emit the address as event.

And then we can use this to deploy other smart contracts. The address at which the Smart Contracts get deployed is deterministic. That's what [EIP-1014](#) says.

```
keccak256( 0xff ++ address ++ salt ++ keccak256(init_code))[12:]
```

[Miguel Mota](#) did a great job in writing a single function that computes the address for CREATE2. But we're not using this, we do it step by step!

First, let's deploy the following Smart Contract with the Factory. Add it into the existing file.

```
contract NoConstructor {
    uint public myUint = 5;
}
```

Then head over to the Solidity Compiler, copy the Bytecode from the Web3-create. Make sure you selected the correct Contract:

The screenshot shows the Solidity Compiler interface with three numbered steps highlighted:

- Step 1:** Shows the compiler version set to "0.8.1+commit.df193b15". A red circle with the number 1 is on the "Include nightly builds" checkbox.
- Step 2:** Shows the "Contract" dropdown menu with "Factory (05-create2-example.sol)" selected. A red circle with the number 2 is on the "NoConstructor (05-create2-example.sol)" option.
- Step 3:** Shows the "Compilation Details" section with "ABI" and "Bytecode" buttons. A red circle with the number 3 is on the "Compilation Details" button.

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract Factory {
6     event Deployed(address _addr);
7     function deploy(uint salt, bytes calldata bytecode,
8         bytes memory implInitCode = bytecode;
9         address addr;
10        assembly {
11            let encoded_data := add(0x20, implInitCode);
12            let encoded_size := mload(implInitCode);
13            addr := create2(0, encoded_data, encoded_size);
14        }
15        emit Deployed(addr);
16    }
17 }
18
19 contract NoConstructor {
20     uint public myUint = 5;
21 }
22
23

```

Checkout the popup:

NoConstructor

```
{ "generatedSources": [], "linkReferences": {}, "object": "6080604052600E"
```

ABI ?

WEB3DEPLOY ?

```
var noconstructorContract = new web3.eth.Contract([{"inputs":[],"name":"myUint","type":"uint256"}]);
var noconstructor = noconstructorContract.deploy({
    data: '0x6080604052600560005534801561001557600080fd5b50',
    arguments: []
}).send({
    from: web3.eth.accounts[0],
    gas: '4700000'
}, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
        console.log('Contract mined at address: ' + contract.address);
    }
})
```

METADATAHASH ?

FUNCTIONHASHES ?

Close

Then head over to the Deploy tab, deploy the Factory first and then use the bytecode to deploy the NoConstructor Contract with Create2.

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0x5B3...eddC4 (99.9999999 : 0.00)

GAS LIMIT
6000000

VALUE
0 wei

CONTRACT
Factory - browser/Upgrades/05-create;

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded (0)

Deployed Contracts

FACTORY AT 0XDA0...42B53 (MEMORY) [copy] [x]

deploy

salt: 1

bytecode: 06064736f6c63430008010033

[] **transact**

Low level interactions

CALldata

Transact

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract Factory {
6     event Deployed(address _addr);
7     function deploy(uint salt, bytes calldata bytecode, bytes memory implInitCode) external {
8         address addr = create2(0, encoded_data, encoded_size);
9         assembly {
10             let encoded_data := add(@x20, implInitCode);
11             let encoded_size := mload(implInitCode);
12             addr := create2(0, encoded_data, encoded_size);
13         }
14         emit Deployed(addr);
15     }
16 }
17
18 contract NoConstructor {
19     uint public myUint = 5;
20 }
21
22
23

```

The salt is currently a number, you can start with any number, I am starting with 1. It's used to determine the final contracts address. The bytecode is simply the bytecode we copied from before. Hit "transact" and open the Transaction details. It should show you the address of your newly deployed NoConstructor contract via the Factory contract:

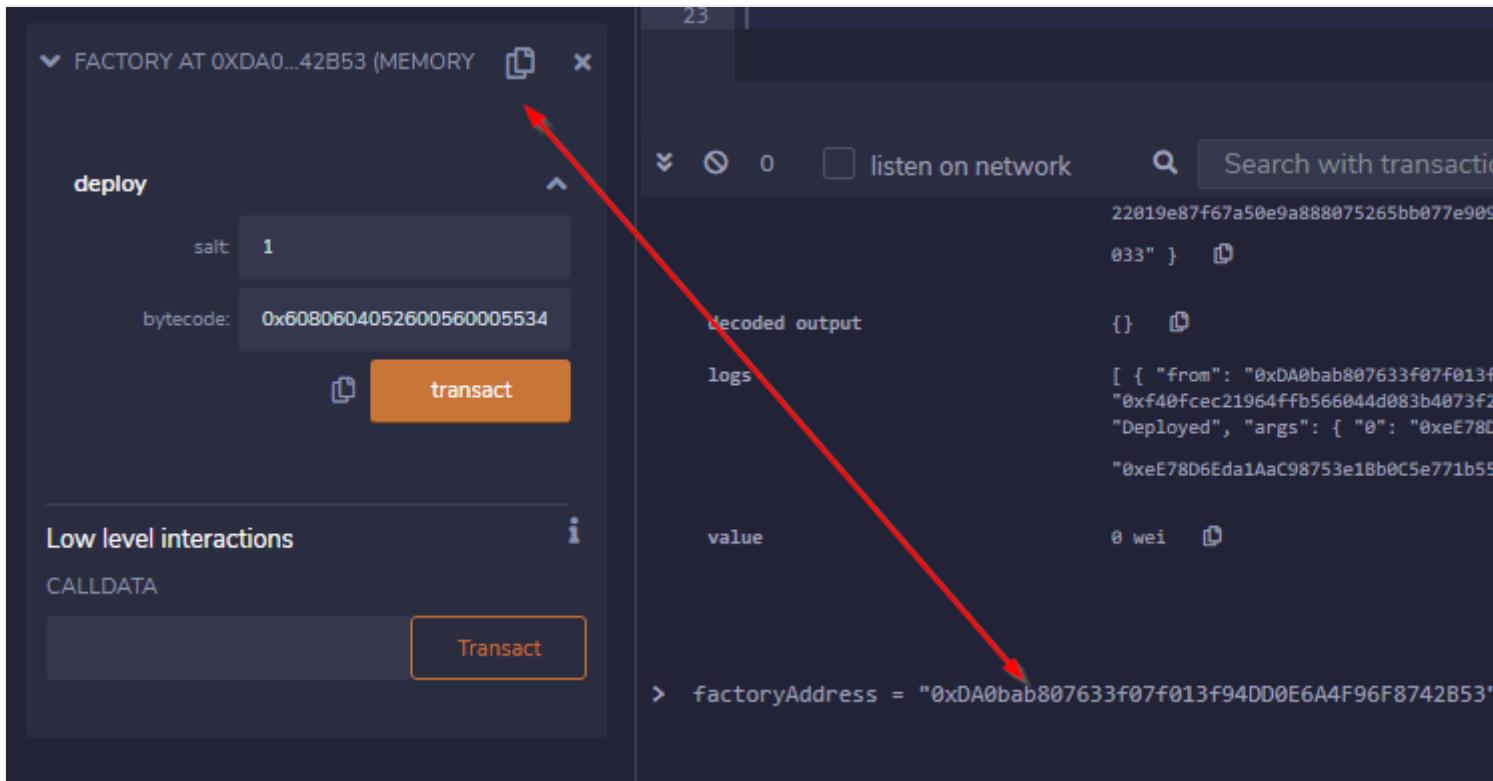
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract Factory {
    event Deployed(address _addr);
    function deploy(uint salt, bytes calldata bytecode, bytes memory implInitCode) external {
        address addr;
        assembly {
            let encoded_data := add(0x20, implInitCode)
            let encoded_size := mload(implInitCode)
            addr := create2(0, encoded_data, encoded_size)
        }
        emit Deployed(addr);
    }
}

contract NoConstructor {
    uint public myUint = 5;
}
```

How to calculate this address in advance? Very easy! We can do this directly in the console of Remix:

```
factoryAddress = "ENTER_FACTORY_ADDRESS"
bytecode = "0x6080604052600560005534801561001557600080fd5b5060b3806100246000396000f3fe6080604052348015600f57600080fd5b506004361060285760003560e01c806306540
salt = 1;
"0x" + web3.utils.sha3('0xff' + factoryAddress.slice(2) + web3.eth.abi.encodeParameter('uint256',salt).slice(2).toString() + web3.utils.sha3(bytecode).slice(2))
```



Pretty much copy and paste one line after the other. The result should be the same address as was emitted by the Factory Smart Contract:

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract Factory {
6     event Deployed(address _addr);
7     function deploy(uint salt, bytes calldata bytecode,
8         bytes memory implInitCode = bytecode;
9         address addr;
10        assembly {
11            let encoded_data := add(0x20, implInitCode);
12            let encoded_size := mload(implInitCode);
13            addr := create2(0, encoded_data, encoded_size);
14        }
15    }

```

logs

```
[ { "from": "0xDA0bab807633f07f013f94DD0E6A4F96F8742B53", "topics": [ "0xf40fc...44d", "Deployed", "args": { "0": "0xE78D6Eda1AaC98753e1Bb0c5e771b55af1102fd" } ], "data": "0x608060405260056000553480156100157600080fd5b5060b380003560e01c806306540f7e14602d5754600080fd5b6033604756525050565b6000602082019050606d000830184604d565b92915bb077e909763324a0aae35530f139e047b40e06064736f6c6343" }
```

value

wei

0 wei

1

0xd246a0502618d24145be74fc...5b315e

0xee78d6eda1aac98753e1bb0c5e771b55af1102fd

17.13.2 CREATE2 with Constructor Argument

How does it work with a Constructor? A little bit different. Essentially the data that the constructor gets as argument needs to be attached to the init-bytecode. Appended. Let's run an example.

Add this to the already existing file:

```

contract WithConstructor {
    address public owner;

    constructor(address _owner) {
        owner = _owner;
    }
}

```

So, if you want to deploy this Smart Contract then you need to add a properly encoded address at the end of it. How to encode the address?

First, copy the address from the address dropdown. Then type in the console `web3.eth.abi.encodeParameter('address', "THE_ADDRESS")`

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area has sections for ENVIRONMENT (set to JavaScript VM), ACCOUNT (set to 0x5B3...eddC4 (100 ether)), GAS LIMIT (set to 3000000), and VALUE (set to 0 wei). Below these, under CONTRACT, is a dropdown menu showing 'Factory - browser/Upgrades/05-create'. A large orange 'Deploy' button is prominent. To the right of the deploy button is a code editor containing Solidity code for a Factory contract. The code includes a constructor that takes a salt and bytecode, and uses the `create2` function to deploy a new contract with a specific address. A red arrow points from the account dropdown to the code editor. Below the code editor, there are buttons for publishing to IPFS and a search bar. At the bottom, a message says 'Currently you have no contract instances to interact with.'

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract Factory {
6     event Deployed(address _addr);
7     function deploy(uint salt, bytes calldata bytecode, bytes memory implInitCode) payable {
8         address addr;
9         assembly {
10             let encoded_data := add(0x20, implInitCode)
11             let encoded_size := mload(implInitCode)
12             addr := create2(0, encoded_data, encoded_size, salt)
13             emit Deployed(addr);
14         }
15     }
16 }
17
18
19 contract NoConstructor {
20     uint public myUint = 5;
21 }
22
23 contract WithConstructor {
24     address public owner;
25
26 constructor(address _owner) {
27     owner = _owner;
28 }
29 }

```

Then copy the output, but remove the starting "0x" and append it to the bytecode that you are deploying using the Factory contract.

In my case I am deploying the following bytecode + address:

`0x608060405234801561001057600080fd5b506040516102043803806102048339818101604052810190610032919061008d565b806000806101000a81548173ffff`

So, now interact with the Smart Contract:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area has fields for 'VALUE' (0 wei) and a 'CONTRACT' dropdown menu where 'WithConstructor - browser/Upgrades/WithConstructor' is selected. A red circle labeled '1' is placed over this dropdown. Below it is a 'Deploy' button and a checkbox for 'Publish to IPFS'. An 'OR' section follows, with a 'At Address' input set to '0x62cF9075968FAeBC03E9e41'. A red arrow labeled '2' points from this address input to the 'owner' field of the deployed 'WITHCONSTRUCTOR' contract. The deployed contract list shows 'FACTORY AT 0xD91...39138 (MEMORY)' and 'WITHCONSTRUCTOR AT 0x62c...21B3'. The 'WITHCONSTRUCTOR' entry is expanded, showing its state with 'owner' set to '0: address: 0x5B38Da6a701c568545dCfcB 03FcB875f56beddC4'. A red circle labeled '3' is placed over the 'owner' field in the expanded contract details. On the right, the Solidity code for the contracts is displayed:

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract Factory {
6     event Deployed(address _addr);
7     function deploy(uint salt, bytes calldata bytecode, bytes memory implInitCode = bytecode, address addr) assembly {
8         let encoded_data := add(0x20, implInitCode)
9         let encoded_size := mload(implInitCode)
10        addr := create2(0, encoded_data, encoded_size, salt)
11        emit Deployed(addr);
12    }
13 }
14
15 contract NoConstructor {
16     uint public myUint = 5;
17 }
18
19 contract WithConstructor {
20     address public owner;
21     constructor(address _owner) {
22         owner = _owner;
23     }
24 }
25
26
27
28
29

```

It should output the address that was given in the constructor.

Well, great, now you know how to deploy Smart Contracts using a CREATE2 op-code. The problem is, you can't change the bytecode, because the hash of the bytecode is used to create the new contract address, right?

WRONG! (I think you knew that there was a way...)

17.13.3 Overwriting Smart Contracts

SELFDESTRUCT Removal

The overwrite function needs to selfdestruct a Smart Contract to work. This might be removed in [upcoming Protocol Upgrades](#)

The idea is to deploy a smart contract that, upon deployment, replaces its own bytecode with a different bytecode. So, the bytecode you run through CREATE2 is always the same, and that calls back to the Factory and replaces itself during deployment.

Clever, right?!

And dangerous!

Let's give it a try. The full example can be found here, <https://github.com/0age/metamorphic>, I am running a minimal example here for you to understand what's going on under the hood!

Create a new file in Remix and add the following Smart Contracts:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

contract Factory {
    mapping (address => address) _implementations;

    event Deployed(address _addr);

    function deploy(uint salt, bytes calldata bytecode) public {
        bytes memory implInitCode = bytecode;

        // assign the initialization code for the metamorphic contract.
        bytes memory metamorphicCode = (
            hex"5860208158601c335a63aaf10f428752fa158151803b80938091923cf3"
        );

        // determine the address of the metamorphic contract.
        address metamorphicContractAddress = _getMetamorphicContractAddress(salt, metamorphicCode);

        // declare a variable for the address of the implementation contract.
        address implementationContract;

        // load implementation init code and length, then deploy via CREATE.
        /* solhint-disable no-inline-assembly */
        assembly {
            let encoded_data := add(0x20, implInitCode) // load initialization code.
            let encoded_size := mload(implInitCode) // load init code's length.
            implementationContract := create( // call CREATE with 3 arguments.
                0, // do not forward any endowment.
                encoded_data, // pass in initialization code.
                encoded_size // pass in init code's length.
            )
        } /* solhint-enable no-inline-assembly */

        // first we deploy the code we want to deploy on a separate address
        // store the implementation to be retrieved by the metamorphic contract.
        _implementations[metamorphicContractAddress] = implementationContract;
    }

    address addr;
    assembly {
        let encoded_data := add(0x20, metamorphicCode) // load initialization code.
        let encoded_size := mload(metamorphicCode) // load init code's length.
        addr := create2(0, encoded_data, encoded_size, salt)
    }

    require(
        addr == metamorphicContractAddress,
        "Failed to deploy the new metamorphic contract."
    );
    emit Deployed(addr);
}

/**
 * @dev Internal view function for calculating a metamorphic contract address
 * given a particular salt.
 */
function _getMetamorphicContractAddress()
```

```

        uint256 salt,
        bytes memory metamorphicCode
    ) internal view returns (address) {
        // determine the address of the metamorphic contract.
        return address(
            uint160(
                keccak256(
                    keccak256(
                        abi.encodePacked(
                            hex"ff",
                            address(this),
                            salt,
                            keccak256(
                                abi.encodePacked(
                                    metamorphicCode
                                )
                            )
                        )
                    )
                )
            )
        );
    }

    //those two functions are getting called by the metamorphic Contract
    function getImplementation() external view returns (address implementation) {
        return _implementations[msg.sender];
    }
}

contract Test1 {
    uint public myUint;

    function setUint(uint _myUint) public {
        myUint = _myUint;
    }

    function killme() public {
        selfdestruct(payable(msg.sender));
    }
}

contract Test2 {
    uint public myUint;

    function setUint(uint _myUint) public {
        myUint = 2*_myUint;
    }

    function killme() public {
        selfdestruct(payable(msg.sender));
    }
}

```

What does it do? 1. It deploys a contract that does only two things: 1. Call back the msg.sender and inquire an address. 1. Copy the bytecode running on that address over its own bytecode

That's it. If you look through the code then that's exactly what it does.

How do you use it?

1. Deploy the Factory
2. use Test1 bytecode with salt=1 to deploy the Test1.
3. Tell Remix that Test1 runs on the address of the Metamorphic contract
4. Set the "myUint" to whatever value you want, it works
5. Kill Test1
6. Deploy Test2 bytecode using the same salt=1
7. It will deploy a different bytecode to the same address!!!
8. Get comfortable that setUint now doubles the input amount.
9. Imagine what this does with a Token Contract you thought it safe to use.

Now imagine for a moment that this is a token contract. Or a new shiny DeFi Project. Imagine people start investing, and suddenly the contract logic changes. All the trust you put into Blockchain is lost. How to avoid getting scammed here? Glad you are asking: First look for a selfdestruct functionality. If it has one, then it's necessary to follow the whole chain of deployers and see if one used the create2 opcode. If yes, then further investigate what they deployed. If it's a Metamorphic Smart Contract, then you know that something fishy is going on...

Alright, that's it all together and I am not aware of any other method to upgrade Smart Contracts. Let's do a quick re-cap.

Last update: April 17, 2021

17.14 Conclusion

In this lab you learned to use all available methods to upgrade Smart Contracts. From an audit perspective, it's always better to use a simpler method. I am personally a big fan of KISS (keep it simple stupid), although it sometimes means that it doesn't look elegant.

I think the Diamond Storage is a very interesting way to "deconstruct" a Smart Contract into smaller parts and plug them back together in a Proxy contract. At this point I would not choose the architecture, because it adds a new layer of complexity to an ecosystem that often manages large amounts of money.

Knowing what's happening under the hood, if I'd start a new larger project from scratch, I'd use [OpenZeppelin Plugins](#) now, if upgradeability is necessary to keep an address constant.

If I don't need a constant address, I'd probably go with either the Eternal Storage pattern or something even simpler. It's easier to audit, easier to grasp, and less error prone.

I hope this lab helped you to choose the right pattern for your project.

Last update: April 17, 2021

18. LEARN: ERC721 Tokens with OpenZeppelin Presets

18.1 ERC721 with Physical Asset Shipping Part 1

A Project that explores how to ship real, physical products, which are connected to ERC721 NFTs.

In this part we are using the OpenZeppelin ERC721PresetMinterPauserAutoId.

In the next part we are reducing it to a more low-level functionality and discuss gas costs and potential savings, as well as the shipping module.

In the final part we are building a small Web3 UI around this.

18.1.1 Video Walkthrough



18.1.2 Background

Beginning of 2021 we (the Team behind the Ethereum Blockchain Course) had an intriguing idea: What if we can auction tokens like traditional NFTs, but have a physical component to it? [Project Aishtisi](#) was born. This page contains the technical lessons learned and a full rundown of the Smart Contract architecture we're using for the project.

18.1.3 Content

This is the lab you've been looking for if you want a tutorial style guide for ERC721 Smart Contracts which contain a Supply Chain component. All following industry best practices.

18.1.4 Real-World Use-Case for this Project

ERC721 Token generation

Auction of Real World Items

Understand the Supply Chain Component

Applicable for Oil, Wine, and all kinds of Commodities

18.1.5 Development-Goal

Deep Dive Into ERC721 and MetaData

⌚ Understand how tokens are auctioned on Opensea or Mintable

Understand the OpenZeppelin ERC721 Contracts

Let's get started!

Last update: August 19, 2021

18.2 The Project Idea

Before directly diving into the Smart Contract, this time I want to briefly talk about the underlying idea. The Smart Contracts will be just the tool to make the idea happen. But first we need a strategy how we actually deliver tokens.

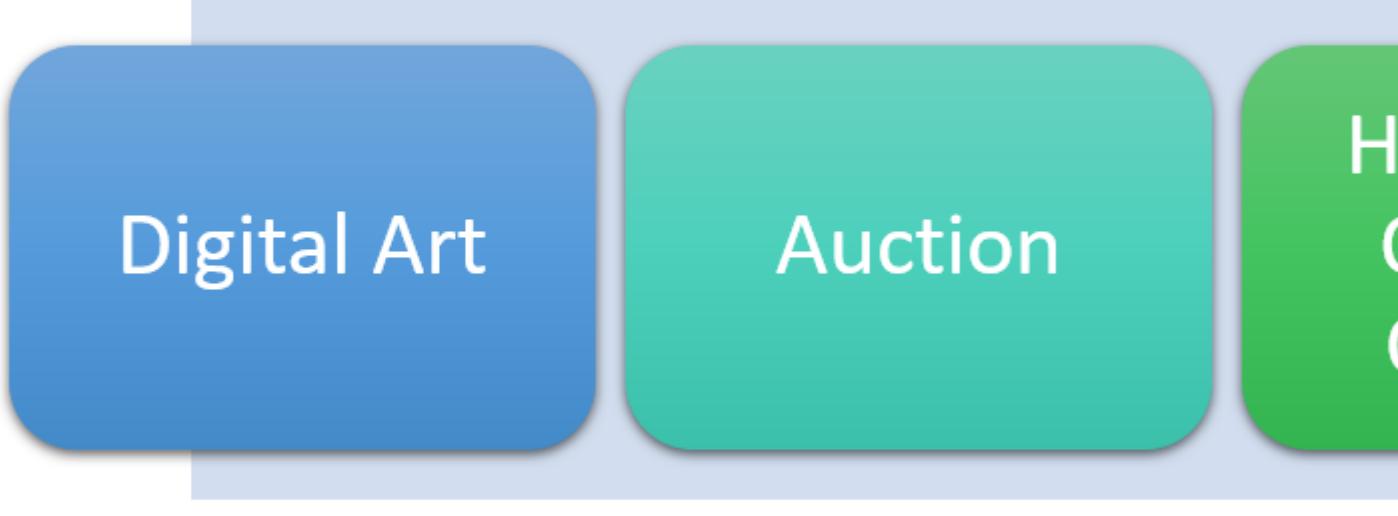
For our project, we are using olive oil as the product that gets shipped. We are experimenting with a small batch where we collaborate with Artists around the world, who create digital art, which then gets printed as labels for the bottles:

We have a few components that play together:

1. A digital Art piece
2. A physical olive oil (that doesn't exist yet!)
3. A physical label
4. A customized video message of the harvest

The Process to get the art + oil to the end-customer is supposedly a simple 4 step process:

1. Some artist creates a digital art piece
2. We mint an ERC721 NFT and auction it
3. We harvest the olives and press the olive oil
4. And then we stick the label on the bottle and ship the olive oil to the NFT holder



Digital Art

Auction

What can possibly go wrong? ...

18.2.1 Physical<->Digital Twin Challenges

Each one of these steps have their unique challenges. To give you a brief rundown:

1. Pay to Play - Gas Costs: Creating the initial ERC721 contract requires about 4 million gas, which is \$250 to \$500 in Ether just to deploy the contract the first time at 40 Gwei. Lazy Minting is not an option in this case, more on that later.
2. 10 Artists: That means 10 deliverables that need to be on time, and it also means 10 different opinions on how branding/ packaging/bottles should look like.
3. Authenticity of the Owner: Who to ship to? That needs to be a trustable address.
4. The video isn't known upfront before the auction
5. It is a variable time when the harvest occurs. It's nature after all, we can't predict in May at which day exactly in October the olives will be half-ripe.

Before we write the code, let's dive into the mechanics of Authenticity first. Then write the code and see if we can bring down the gas costs.

18.2.2 Authenticity and Shipping Address

The difference between ERC20 and ERC721 tokens is that every token in the ERC721 contract is unique. It's like deploying many ERC20 contracts, just under one address. ERC20 Tokens can be sent from one address to another. This is a very fundamental property why ERC20 tokens are so popular.

NFTs are usually ERC-721 contracts and should adhere to the ERC-721 interface. This means, they should also contain a function called `function transferFrom(address from, address to, uint256 tokenId) external;`. With this function you can transfer a token from one address to another address, just like with ERC20 tokens. With ERC721 you need to add an additional tokenId, because every token is unique. But the functionality is essentially the same.

That causes a problem when we want to ship the physical product that is represented by a token. There is a very real way to game the system.

How to Game the System

Let's say Mr.X has the token and wants the bottle to ship to New York. We're shipping the bottle to New York. While the bottle is on the way, he sells the token to Mr.Y in Russia. Mr.X gets the bottle, Mr.Y gets no physical product. That can happen intentional, but also unintentional.

Generate Token

Transfer
Freely until
Oct 1st

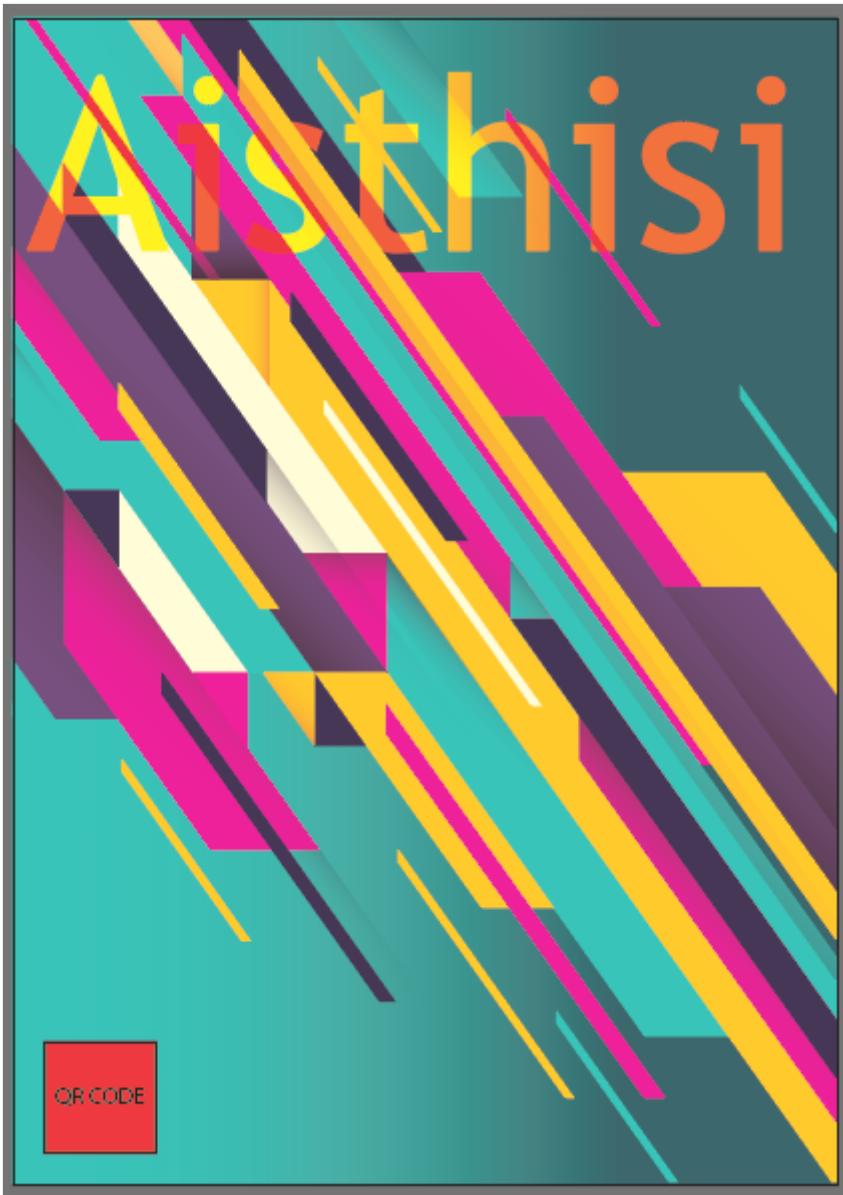
Ship Bottle
with unlock-
QR Code

This is why we will introduce a novel locking mechanism that requires the physical product to unlock the token transfer again.

Time-Based Locking Mechanism

The token will lock itself automatically on Oct 1st, the month in which the harvest takes place. The token owner can then send us the shipping address in a secure way (more on that later). Then we harvest the olives, produce the oil and ship the bottle. On the bottle there will be a QR code to unlock the token.

This can be done only by the token owner, by nobody else.



The QR code will either be directly in the label or an additional flyer added to the packaging. We will provide a simple website to scan the QR code later and generate a simple transaction to unlock the token, which should work from every wallet - MetaMask, Fortmatic, Portis, etc... without problems.

With this in mind, let's start to create a simple version of our ERC721 Token.

Last update: June 23, 2021

18.3 Truffle Setup

Let's setup Truffle and import the OpenZeppelin Token Library. We are using Truffle 5.3.7 and OpenZeppelin 4.1.0 for this project.

18.3.1 ERC721 Token Prototype

For our token, we will be using Truffle and the OpenZeppelin Contracts. The first step is therefore to initialize a new Truffle Project and add in the OpenZeppelin contracts.

If you haven't installed truffle globally yet, then install truffle first:

```
npm install -g truffle
```

Then create a new folder, open VSCode (or PowerShell, or a Terminal), init a new Truffle project and install OpenZeppelin Contracts:

```
truffle init  
npm install @openzeppelin/contracts
```

File Edit Selection View Go Run Terminal Help oiltoken-writeup - Visual Studio Code

EXPLORER

- OPEN EDITORS
- OILTOKEN-WRITEUP
 - > contracts
 - > migrations
 - > node_modules
 - > test
 - { } package-lock.json
 - JS truffle-config.js

PROBLEMS

```
- Fetching solc version list from solc-bin@^0.4.14
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents): Not compatible with your operating system (Darwin) ("os": "darwin", "arch": "any") (current: {"os": "win32", "arch": "x64"})

+ truffle@5.3.7
added 1 package from 1 contributor, removed 1 package in 1.13s
oiltoken-writeup> truffle init
```

Starting init...

```
=====
> Copying project files to C:\101Project\oiltoken-writeup
```

Init successful, sweet!

```
oiltoken-writeup> npm install @openzeppelin/truffle-upgradeable
npm WARN saveError ENOENT: no such file or directory, open 'C:\101Project\oiltoken-writeup\package.json'
npm notice created a lockfile as package-lock.json
npm WARN enoent ENOENT: no such file or directory, open 'C:\101Project\oiltoken-writeup\package.json'
npm WARN oiltoken-writeup No description
npm WARN oiltoken-writeup No repository
npm WARN oiltoken-writeup No README data
npm WARN oiltoken-writeup No license data

+ @openzeppelin/truffle-upgradeable@1.0.0
added 1 package from 1 contributor
found 0 vulnerabilities
```

oiltoken-writeup>

Last update: June 23, 2021

18.4 OpenZeppelin Preset Contracts

Let's add a simple ERC721 Token Preset.

18.4.1 Why use OpenZeppelin Contracts

From the [OpenZeppelin Github Page](#): A library for secure smart contract development. Build on a solid foundation of community-vetted code.

They come with a preset, so we can easily do an ERC721 Contract that is mintable, pausable, burnable and supports all functions we possibly need to be ERC721 compliant.

For this first implementation we will be using <https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/presets/ERC721PresetMinterPauserAutoId.sol>

18.4.2 Creating the ERC721 Token

Create a new file in `/contracts/AisthisiToken.sol` with the following content:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

import "@openzeppelin/contracts/token/ERC721/presets/ERC721PresetMinterPauserAutoId.sol";

contract AisthisiToken is ERC721PresetMinterPauserAutoId {

    constructor() ERC721PresetMinterPauserAutoId("AisthisiToken", "AIS", "https://aisthisi.art/metadata/") {}

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        return string(abi.encodePacked(super.tokenURI(tokenId), ".json"));
    }
}
```

This is a normal ERC721 Contract based on the preset that OpenZeppelin gives us. It might look overwhelming at this point, but there is nothing special to it - let's look what happens under the hood. If we open the [ERC721PresetMinterPauserAutoId](#) Contract then you see that it actually does a few things in the constructor ([here](#) is the documentation of the preset):

1. It sets up a new ERC721 Token Contract with the name and the symbol, in this case "AisthisiToken" and "AIS".
2. It sets the tokenURI base url to "https://aisthisi.art/metadata/"
3. It will grant the deployer pausing rights and minting rights.
4. Additionally, and completely optional, but we decided to add ".json" to the url for the metadata file automatically, hence modified the `tokenURI` function.

```
constructor(string memory name, string memory symbol, string memory baseTokenURI) ERC721(name, symbol) {
    _baseTokenURI = baseTokenURI;

    _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());

    _setupRole(MINTER_ROLE, _msgSender());
    _setupRole(PAUSER_ROLE, _msgSender());
}
```

Let's give this a try in Truffle.

18.4.3 Adding Migrations to Truffle

Create a migrations file to `migrations/2_token_migration.js` with the following content

```
const AisthisiToken = artifacts.require("AisthisiToken");

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

It should look like this now:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure:
 - OPEN EDITORS:** Getting Started (highlighted), AishtisiToken.sol contracts, JS 2_token_migration.js migrations (highlighted).
 - OILTOKEN-WRITEUP:** contracts (AishtisiToken.sol, Migrations.sol), migrations (1_initial_migration.js, 2_token_migration.js).
 - node_modules\@openzeppelin\contracts (highlighted).
 - test, package-lock.json, truffle-config.js.
- Code Editor (Top Right):** Displays the content of 2_token_migration.js:


```
1 const Aisthisitoken = artifacts.require("AishtisiToken");
2
3 module.exports = function (deployer) {
4   deployer.deploy(Aisthisitoken);
5 };
6
```
- Terminal (Bottom Right):** Shows a Windows PowerShell session:


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

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

Loading personal and system profiles took 1078ms
oiltoken-writeup> 
```
- Bottom Status Bar:** Shows icons for close, minimize, maximize, and full screen.

In the truffle-config.js file, edit the compiler settings to:

```
// Configure your compilers
compilers: {
  solc: {
    version: "0.8.3", // Fetch exact version from solc-bin (default: truffle's version)
    // docker: true, // Use "0.5.1" you've installed locally with docker (default: false)
    //settings: { // See the solidity docs for advice about optimization and evmVersion
    // optimizer: {
```

```
//   enabled: true,  
//   runs: 200  
// },  
// evmVersion: "byzantium"  
}  
},  
,
```

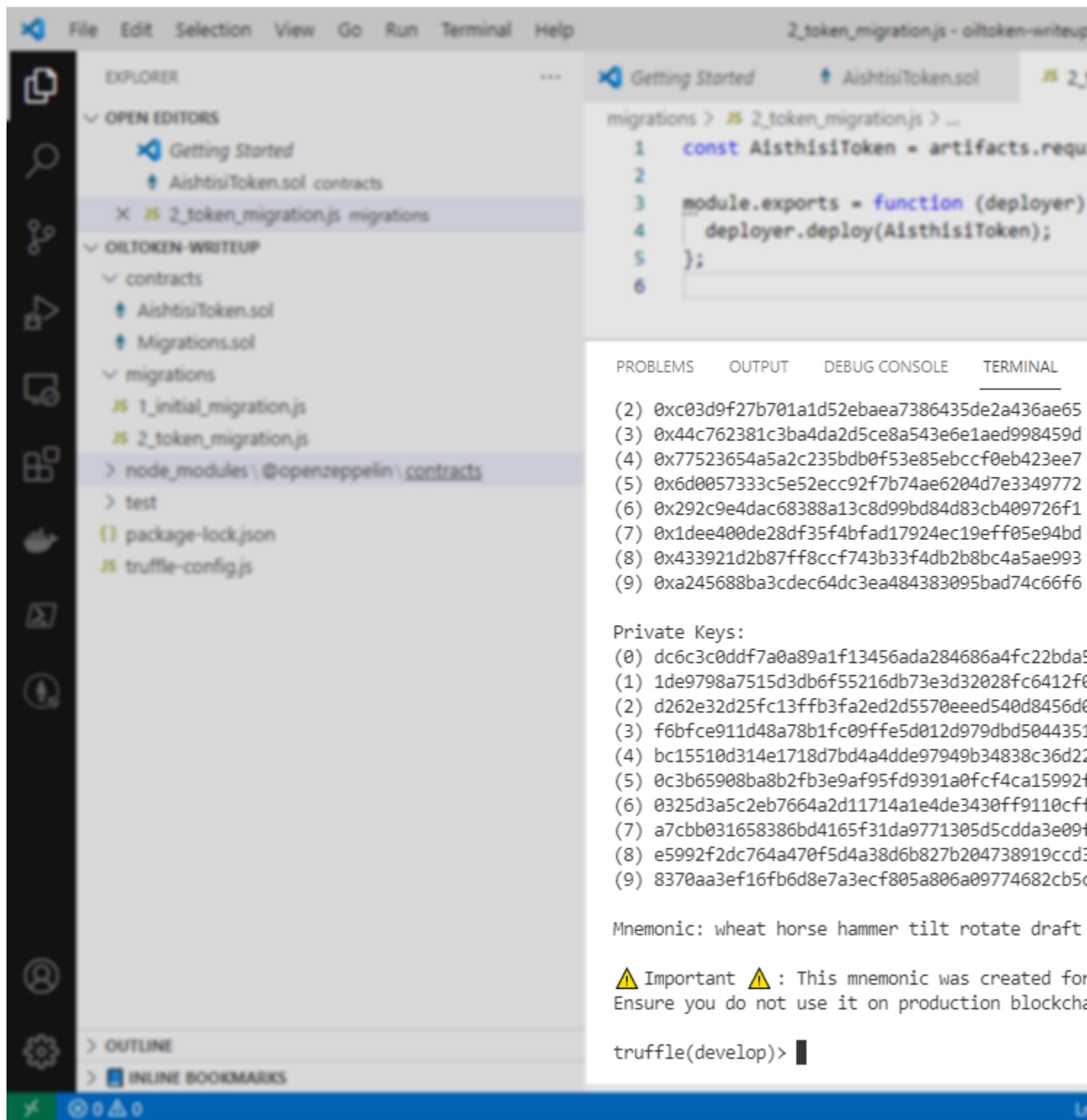
We will later have a look how we can use the optimizer and other techniques to save gas costs.

18.4.4 Deploying the Token to the Truffle Development Network

Then simply run the truffle development console

```
truffle development
```

This should open the Truffle Development console:



```

File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
  Getting Started
  AishtisiToken.sol contracts
  2_token_migration.js migrations
OILTOKEN-WRITEUP
  contracts
    AishtisiToken.sol
    Migrations.sol
  migrations
    1_initial_migration.js
    2_token_migration.js
    node_modules\@openzeppelin\contracts
    test
    package-lock.json
    truffle-config.js
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
migrations > 2_token_migration.js > ...
1 const AishtisiToken = artifacts.require("AishtisiToken");
2
3 module.exports = function (deployer) {
4   deployer.deploy(AishtisiToken);
5 };
6
(2) 0xc03d9f27b701a1d52ebaea7386435de2a436ae65
(3) 0x44c762381c3ba4da2d5ce8a543e6e1aed998459d
(4) 0x77523654a5a2c235bdb0f53e85ebccf0eb423ee7
(5) 0x6d0057333c5e52ecc92f7b74ae6204d7e3349772
(6) 0x292c9e4dac68388a13c8d99bd84d83cb409726f1
(7) 0x1dee400de28df35f4bfad17924ec19eff05e94bd
(8) 0x433921d2b87ff8ccf743b33f4db2b8bc4a5ae993
(9) 0xa245688ba3cdec64dc3ea484383095bad74c66f6
Private Keys:
(0) dc6c3c0ddf7a0a89a1f13456ada284686a4fc22bda5
(1) 1de9798a7515d3db6f55216db73e3d32028fc6412f0
(2) d262e32d25fc13ffb3fa2ed2d5570eed540d8456d0
(3) f6bfce911d48a78b1fc09ffe5d012d979fdb5044351
(4) bc15510d314e1718d7bd4a4dde97949b34838c36d22
(5) 0c3b65908ba8b2fb3e9af95fd9391a0fcf4ca159921
(6) 0325d3a5c2eb7664a2d11714a1e4de3430ff9110cff
(7) a7ccb031658386bd4165f31da9771305d5cdda3e091
(8) e5992f2dc764a470f5d4a38d6b827b204738919cccd3
(9) 8370aa3ef16fb6d8e7a3ecf805a806a09774682cb5c
Mnemonic: wheat horse hammer tilt rotate draft
⚠️ Important ⚠️ : This mnemonic was created for
Ensure you do not use it on production blockchain
truffle(develop)>

```

Now lets just migrate the token and then mint one token:

migrate

will deploy the tokens to this test-chain.

18.4.5 Testing the Token on the Truffle Development Console

Then you can simply type in

```
let token = await AisthisToken.deployed();
token.mint(accounts[0]);
```

This will mint one token to your address.

That also means we can freely send it around!

```
token.transferFrom(accounts[0], accounts[1], 0);
```

Will transfer the token from account 0 to account #1.

Let's check the balance:

```
(await token.balanceOf(accounts[1])).toString()
```

Should output 1:

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure under "OPEN EDITORS" and "OILTOKEN-WRITEUP".
- Editor:** Displays the Solidity code for the `balanceOf` function in `ERC721.sol`.
- Bottom Right:** Truffle Development Console tab bar with PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL.
- Bottom:** Truffle Development Console output showing a transaction log and a prompt for the next command.

```

56      /**
57       * @dev See {IERC721-balanceOf}.
58     */
59
60     function balanceOf(address owner)
61     require(owner != address(0));
62     return balances[owner];
63   }
64
65   /**
66    * @dev See {IERC721-ownerOf}.
67    */
68
69   function ownerOf(address tokenAddress) external view returns (address owner) {
70     require(tokenAddress != address(0));
71     owner = ownerOf[tokenAddress];
72   }
73
74   event Transfer(
75     address indexed from,
76     address indexed to,
77     uint256 indexed tokenId
78   );
79
80   function safeTransferFrom(
81     address from,
82     address to,
83     uint256 tokenId
84   ) public virtual override {
85     require(_isApprovedOrOwner(_msgSender(), tokenId));
86     _safeTransfer(from, to, tokenId);
87   }
88
89   function safeTransferFrom(
90     address from,
91     address to,
92     uint256 tokenId,
93     bytes memory data
94   ) public virtual override {
95     require(_isApprovedOrOwner(_msgSender(), tokenId));
96     _safeTransferWithData(from, to, tokenId, data);
97   }
98
99   function transferFrom(
100    address from,
101    address to,
102    uint256 tokenId
103  ) public virtual override {
104    require(_isApprovedOrOwner(_msgSender(), tokenId));
105    _transfer(from, to, tokenId);
106  }
107
108  function approve(
109    address spender,
110    uint256 tokenId
111  ) public virtual {
112    require(spender != address(0));
113    require(tokenId != 0);
114    _approve(_msgSender(), spender, tokenId);
115  }
116
117  function setApprovalForAll(
118    address operator,
119    bool approved
120  ) public virtual {
121    require(operator != address(0));
122    _setApprovalForAll(_msgSender(), operator, approved);
123  }
124
125  function transfer(
126    address to,
127    uint256 tokenId
128  ) public virtual {
129    require(to != address(0));
130    require(tokenId != 0);
131    _transfer(_msgSender(), to, tokenId);
132  }
133
134  function _transfer(
135    address from,
136    address to,
137    uint256 tokenId
138  ) internal virtual {
139    require(from != address(0));
140    require(to != address(0));
141    require(tokenId != 0);
142    require(_isApprovedOrOwner(_msgSender(), tokenId));
143    require(!_exists(tokenId));
144    _balances[from] -= 1;
145    _balances[to] += 1;
146    _owners[tokenId] = to;
147    emit Transfer(from, to, tokenId);
148  }
149
150  function _safeTransfer(
151    address from,
152    address to,
153    uint256 tokenId
154  ) internal virtual {
155    _transfer(from, to, tokenId);
156    require(_checkOnERC721Received(from, to, tokenId, ""));
157  }
158
159  function _safeTransferWithData(
160    address from,
161    address to,
162    uint256 tokenId,
163    bytes memory data
164  ) internal virtual {
165    _transfer(from, to, tokenId);
166    require(_checkOnERC721Received(from, to, tokenId, data));
167  }
168
169  function _exists(uint256 tokenId) internal view returns (bool) {
170    return _balances[tokenId] > 0;
171  }
172
173  function _isApprovedOrOwner(
174    address spender,
175    uint256 tokenId
176  ) internal view returns (bool) {
177    require(spender != address(0));
178    require(tokenId != 0);
179    return spender == _ownerOf[tokenId] || spender == _msgSender();
180  }
181
182  function _ownerOf(uint256 tokenId) internal view returns (address) {
183    return _owners[tokenId];
184  }
185
186  function _setApprovalForAll(
187    address operator,
188    bool approved
189  ) internal virtual {
190    require(operator != address(0));
191    _operatorApprovals[_msgSender()][operator] = approved;
192    emit ApprovalForAll(_msgSender(), operator, approved);
193  }
194
195  function _approve(
196    address owner,
197    address spender,
198    uint256 tokenId
199  ) internal virtual {
200    require(owner != address(0));
201    require(spender != address(0));
202    require(tokenId != 0);
203    require(_isApprovedOrOwner(_msgSender(), tokenId));
204    _operatorApprovals[owner][spender] = true;
205    emit Approval(owner, spender, tokenId);
206  }
207
208  function _checkOnERC721Received(
209    address from,
210    address to,
211    uint256 tokenId,
212    bytes memory data
213  ) internal returns (bool) {
214    if (to.isContract()) {
215      try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, data) returns (bytes4 retval) {
216        return (retval == IERC721Receiver(to).onERC721Received.selector);
217      } catch Error(string memory reason) {
218        revert("ERC721: recipient rejected update");
219      }
220    }
221    return true;
222  }
223
224  function _burn(
225    address owner,
226    uint256 tokenId
227  ) internal virtual {
228    require(owner != address(0));
229    require(_isApprovedOrOwner(_msgSender(), tokenId));
230    require(!_exists(tokenId));
231    _balances[owner] += 1;
232    _owners[tokenId] = address(0);
233    emit Transfer(owner, address(0), tokenId);
234  }
235
236  function _burn(
237    uint256 tokenId
238  ) internal virtual {
239    _burn(_msgSender(), tokenId);
240  }
241
242  function _mint(
243    address to,
244    uint256 tokenId
245  ) internal virtual {
246    require(to != address(0));
247    require(!_exists(tokenId));
248    require(_checkOnERC721Received(address(0), to, tokenId, ""));
249    _balances[to] += 1;
250    _owners[tokenId] = to;
251    emit Transfer(address(0), to, tokenId);
252  }
253
254  function _mint(
255    address to,
256    uint256 tokenId,
257    bytes memory data
258  ) internal virtual {
259    require(to != address(0));
260    require(!_exists(tokenId));
261    require(_checkOnERC721Received(address(0), to, tokenId, data));
262    _balances[to] += 1;
263    _owners[tokenId] = to;
264    emit Transfer(address(0), to, tokenId);
265  }
266
267  function _burnFrom(
268    address owner,
269    uint256 tokenId
270  ) internal virtual {
271    require(owner != address(0));
272    require(_isApprovedOrOwner(_msgSender(), tokenId));
273    require(!_exists(tokenId));
274    _balances[owner] -= 1;
275    _owners[tokenId] = address(0);
276    emit Transfer(owner, address(0), tokenId);
277  }
278
279  function _burnFrom(
280    address owner,
281    uint256 tokenId,
282    bytes memory data
283  ) internal virtual {
284    require(owner != address(0));
285    require(_isApprovedOrOwner(_msgSender(), tokenId));
286    require(!_exists(tokenId));
287    _balances[owner] -= 1;
288    _owners[tokenId] = address(0);
289    emit Transfer(owner, address(0), tokenId);
290  }
291
292  function _safeTransfer(
293    address from,
294    address to,
295    uint256 tokenId
296  ) internal virtual {
297    _burnFrom(from, tokenId);
298    _safeTransfer(from, to, tokenId);
299  }
300
301  function _safeTransfer(
302    address from,
303    address to,
304    uint256 tokenId,
305    bytes memory data
306  ) internal virtual {
307    _burnFrom(from, tokenId);
308    _safeTransferWithData(from, to, tokenId, data);
309  }
310
311  function _burn(
312    address from,
313    uint256 tokenId
314  ) internal virtual {
315    _burnFrom(from, tokenId);
316  }
317
318  function _burn(
319    address from,
320    uint256 tokenId,
321    bytes memory data
322  ) internal virtual {
323    _burnFrom(from, tokenId);
324  }
325
326  function _safeTransfer(
327    address from,
328    address to,
329    uint256 tokenId,
330    bytes memory data
331  ) internal virtual {
332    _burnFrom(from, tokenId);
333    _safeTransferWithData(from, to, tokenId, data);
334  }
335
336  function _safeTransferWithData(
337    address from,
338    address to,
339    uint256 tokenId,
340    bytes memory data
341  ) internal virtual {
342    _burnFrom(from, tokenId);
343    require(_checkOnERC721Received(from, to, tokenId, data));
344  }
345
346  function _burn(
347    uint256 tokenId
348  ) internal virtual {
349    _burn(_msgSender(), tokenId);
350  }
351
352  function _burn(
353    uint256 tokenId,
354    bytes memory data
355  ) internal virtual {
356    _burn(_msgSender(), tokenId);
357  }
358
359  function _burn(
360    address owner,
361    uint256 tokenId,
362    bytes memory data
363  ) internal virtual {
364    _burnFrom(owner, tokenId);
365  }
366
367  function _burn(
368    address owner,
369    uint256 tokenId,
370    address operator
371  ) internal virtual {
372    _burnFrom(owner, tokenId);
373  }
374
375  function _burn(
376    address owner,
377    uint256 tokenId,
378    address operator,
379    bytes memory data
380  ) internal virtual {
381    _burnFrom(owner, tokenId);
382  }
383
384  function _burn(
385    address owner,
386    uint256 tokenId,
387    address operator,
388    bytes memory data,
389    bytes memory extraData
390  ) internal virtual {
391    _burnFrom(owner, tokenId);
392  }
393
394  function _burn(
395    address owner,
396    uint256 tokenId,
397    address operator,
398    bytes memory data,
399    bytes memory extraData,
400    bytes memory extraData2
401  ) internal virtual {
402    _burnFrom(owner, tokenId);
403  }
404
405  function _burn(
406    address owner,
407    uint256 tokenId,
408    address operator,
409    bytes memory data,
410    bytes memory extraData,
411    bytes memory extraData2,
412    bytes memory extraData3
413  ) internal virtual {
414    _burnFrom(owner, tokenId);
415  }
416
417  function _burn(
418    address owner,
419    uint256 tokenId,
420    address operator,
421    bytes memory data,
422    bytes memory extraData,
423    bytes memory extraData2,
424    bytes memory extraData3,
425    bytes memory extraData4
426  ) internal virtual {
427    _burnFrom(owner, tokenId);
428  }
429
430  function _burn(
431    address owner,
432    uint256 tokenId,
433    address operator,
434    bytes memory data,
435    bytes memory extraData,
436    bytes memory extraData2,
437    bytes memory extraData3,
438    bytes memory extraData4,
439    bytes memory extraData5
440  ) internal virtual {
441    _burnFrom(owner, tokenId);
442  }
443
444  function _burn(
445    address owner,
446    uint256 tokenId,
447    address operator,
448    bytes memory data,
449    bytes memory extraData,
450    bytes memory extraData2,
451    bytes memory extraData3,
452    bytes memory extraData4,
453    bytes memory extraData5,
454    bytes memory extraData6
455  ) internal virtual {
456    _burnFrom(owner, tokenId);
457  }
458
459  function _burn(
460    address owner,
461    uint256 tokenId,
462    address operator,
463    bytes memory data,
464    bytes memory extraData,
465    bytes memory extraData2,
466    bytes memory extraData3,
467    bytes memory extraData4,
468    bytes memory extraData5,
469    bytes memory extraData6,
470    bytes memory extraData7
471  ) internal virtual {
472    _burnFrom(owner, tokenId);
473  }
474
475  function _burn(
476    address owner,
477    uint256 tokenId,
478    address operator,
479    bytes memory data,
480    bytes memory extraData,
481    bytes memory extraData2,
482    bytes memory extraData3,
483    bytes memory extraData4,
484    bytes memory extraData5,
485    bytes memory extraData6,
486    bytes memory extraData7,
487    bytes memory extraData8
488  ) internal virtual {
489    _burnFrom(owner, tokenId);
490  }
491
492  function _burn(
493    address owner,
494    uint256 tokenId,
495    address operator,
496    bytes memory data,
497    bytes memory extraData,
498    bytes memory extraData2,
499    bytes memory extraData3,
500    bytes memory extraData4,
501    bytes memory extraData5,
502    bytes memory extraData6,
503    bytes memory extraData7,
504    bytes memory extraData8,
505    bytes memory extraData9
506  ) internal virtual {
507    _burnFrom(owner, tokenId);
508  }
509
510  function _burn(
511    address owner,
512    uint256 tokenId,
513    address operator,
514    bytes memory data,
515    bytes memory extraData,
516    bytes memory extraData2,
517    bytes memory extraData3,
518    bytes memory extraData4,
519    bytes memory extraData5,
520    bytes memory extraData6,
521    bytes memory extraData7,
522    bytes memory extraData8,
523    bytes memory extraData9,
524    bytes memory extraData10
525  ) internal virtual {
526    _burnFrom(owner, tokenId);
527  }
528
529  function _burn(
530    address owner,
531    uint256 tokenId,
532    address operator,
533    bytes memory data,
534    bytes memory extraData,
535    bytes memory extraData2,
536    bytes memory extraData3,
537    bytes memory extraData4,
538    bytes memory extraData5,
539    bytes memory extraData6,
540    bytes memory extraData7,
541    bytes memory extraData8,
542    bytes memory extraData9,
543    bytes memory extraData10,
544    bytes memory extraData11
545  ) internal virtual {
546    _burnFrom(owner, tokenId);
547  }
548
549  function _burn(
550    address owner,
551    uint256 tokenId,
552    address operator,
553    bytes memory data,
554    bytes memory extraData,
555    bytes memory extraData2,
556    bytes memory extraData3,
557    bytes memory extraData4,
558    bytes memory extraData5,
559    bytes memory extraData6,
560    bytes memory extraData7,
561    bytes memory extraData8,
562    bytes memory extraData9,
563    bytes memory extraData10,
564    bytes memory extraData11,
565    bytes memory extraData12
566  ) internal virtual {
567    _burnFrom(owner, tokenId);
568  }
569
570  function _burn(
571    address owner,
572    uint256 tokenId,
573    address operator,
574    bytes memory data,
575    bytes memory extraData,
576    bytes memory extraData2,
577    bytes memory extraData3,
578    bytes memory extraData4,
579    bytes memory extraData5,
580    bytes memory extraData6,
581    bytes memory extraData7,
582    bytes memory extraData8,
583    bytes memory extraData9,
584    bytes memory extraData10,
585    bytes memory extraData11,
586    bytes memory extraData12,
587    bytes memory extraData13
588  ) internal virtual {
589    _burnFrom(owner, tokenId);
590  }
591
592  function _burn(
593    address owner,
594    uint256 tokenId,
595    address operator,
596    bytes memory data,
597    bytes memory extraData,
598    bytes memory extraData2,
599    bytes memory extraData3,
600    bytes memory extraData4,
601    bytes memory extraData5,
602    bytes memory extraData6,
603    bytes memory extraData7,
604    bytes memory extraData8,
605    bytes memory extraData9,
606    bytes memory extraData10,
607    bytes memory extraData11,
608    bytes memory extraData12,
609    bytes memory extraData13,
610    bytes memory extraData14
611  ) internal virtual {
612    _burnFrom(owner, tokenId);
613  }
614
615  function _burn(
616    address owner,
617    uint256 tokenId,
618    address operator,
619    bytes memory data,
620    bytes memory extraData,
621    bytes memory extraData2,
622    bytes memory extraData3,
623    bytes memory extraData4,
624    bytes memory extraData5,
625    bytes memory extraData6,
626    bytes memory extraData7,
627    bytes memory extraData8,
628    bytes memory extraData9,
629    bytes memory extraData10,
630    bytes memory extraData11,
631    bytes memory extraData12,
632    bytes memory extraData13,
633    bytes memory extraData14,
634    bytes memory extraData15
635  ) internal virtual {
636    _burnFrom(owner, tokenId);
637  }
638
639  function _burn(
640    address owner,
641    uint256 tokenId,
642    address operator,
643    bytes memory data,
644    bytes memory extraData,
645    bytes memory extraData2,
646    bytes memory extraData3,
647    bytes memory extraData4,
648    bytes memory extraData5,
649    bytes memory extraData6,
650    bytes memory extraData7,
651    bytes memory extraData8,
652    bytes memory extraData9,
653    bytes memory extraData10,
654    bytes memory extraData11,
655    bytes memory extraData12,
656    bytes memory extraData13,
657    bytes memory extraData14,
658    bytes memory extraData15,
659    bytes memory extraData16
660  ) internal virtual {
661    _burnFrom(owner, tokenId);
662  }
663
664  function _burn(
665    address owner,
666    uint256 tokenId,
667    address operator,
668    bytes memory data,
669    bytes memory extraData,
670    bytes memory extraData2,
671    bytes memory extraData3,
672    bytes memory extraData4,
673    bytes memory extraData5,
674    bytes memory extraData6,
675    bytes memory extraData7,
676    bytes memory extraData8,
677    bytes memory extraData9,
678    bytes memory extraData10,
679    bytes memory extraData11,
680    bytes memory extraData12,
681    bytes memory extraData13,
682    bytes memory extraData14,
683    bytes memory extraData15,
684    bytes memory extraData16,
685    bytes memory extraData17
686  ) internal virtual {
687    _burnFrom(owner, tokenId);
688  }
689
690  function _burn(
691    address owner,
692    uint256 tokenId,
693    address operator,
694    bytes memory data,
695    bytes memory extraData,
696    bytes memory extraData2,
697    bytes memory extraData3,
698    bytes memory extraData4,
699    bytes memory extraData5,
700    bytes memory extraData6,
701    bytes memory extraData7,
702    bytes memory extraData8,
703    bytes memory extraData9,
704    bytes memory extraData10,
705    bytes memory extraData11,
706    bytes memory extraData12,
707    bytes memory extraData13,
708    bytes memory extraData14,
709    bytes memory extraData15,
710    bytes memory extraData16,
711    bytes memory extraData17,
712    bytes memory extraData18
713  ) internal virtual {
714    _burnFrom(owner, tokenId);
715  }
716
717  function _burn(
718    address owner,
719    uint256 tokenId,
720    address operator,
721    bytes memory data,
722    bytes memory extraData,
723    bytes memory extraData2,
724    bytes memory extraData3,
725    bytes memory extraData4,
726    bytes memory extraData5,
727    bytes memory extraData6,
728    bytes memory extraData7,
729    bytes memory extraData8,
730    bytes memory extraData9,
731    bytes memory extraData10,
732    bytes memory extraData11,
733    bytes memory extraData12,
734    bytes memory extraData13,
735    bytes memory extraData14,
736    bytes memory extraData15,
737    bytes memory extraData16,
738    bytes memory extraData17,
739    bytes memory extraData18,
740    bytes memory extraData19
741  ) internal virtual {
742    _burnFrom(owner, tokenId);
743  }
744
745  function _burn(
746    address owner,
747    uint256 tokenId,
748    address operator,
749    bytes memory data,
750    bytes memory extraData,
751    bytes memory extraData2,
752    bytes memory extraData3,
753    bytes memory extraData4,
754    bytes memory extraData5,
755    bytes memory extraData6,
756    bytes memory extraData7,
757    bytes memory extraData8,
758    bytes memory extraData9,
759    bytes memory extraData10,
760    bytes memory extraData11,
761    bytes memory extraData12,
762    bytes memory extraData13,
763    bytes memory extraData14,
764    bytes memory extraData15,
765    bytes memory extraData16,
766    bytes memory extraData17,
767    bytes memory extraData18,
768    bytes memory extraData19,
769    bytes memory extraData20
770  ) internal virtual {
771    _burnFrom(owner, tokenId);
772  }
773
774  function _burn(
775    address owner,
776    uint256 tokenId,
777    address operator,
778    bytes memory data,
779    bytes memory extraData,
780    bytes memory extraData2,
781    bytes memory extraData3,
782    bytes memory extraData4,
783    bytes memory extraData5,
784    bytes memory extraData6,
785    bytes memory extraData7,
786    bytes memory extraData8,
787    bytes memory extraData9,
788    bytes memory extraData10,
789    bytes memory extraData11,
790    bytes memory extraData12,
791    bytes memory extraData13,
792    bytes memory extraData14,
793    bytes memory extraData15,
794    bytes memory extraData16,
795    bytes memory extraData17,
796    bytes memory extraData18,
797    bytes memory extraData19,
798    bytes memory extraData20,
799    bytes memory extraData21
800  ) internal virtual {
801    _burnFrom(owner, tokenId);
802  }
803
804  function _burn(
805    address owner,
806    uint256 tokenId,
807    address operator,
808    bytes memory data,
809    bytes memory extraData,
810    bytes memory extraData2,
811    bytes memory extraData3,
812    bytes memory extraData4,
813    bytes memory extraData5,
814    bytes memory extraData6,
815    bytes memory extraData7,
816    bytes memory extraData8,
817    bytes memory extraData9,
818    bytes memory extraData10,
819    bytes memory extraData11,
820    bytes memory extraData12,
821    bytes memory extraData13,
822    bytes memory extraData14,
823    bytes memory extraData15,
824    bytes memory extraData16,
825    bytes memory extraData17,
826    bytes memory extraData18,
827    bytes memory extraData19,
828    bytes memory extraData20,
829    bytes memory extraData21,
830    bytes memory extraData22
831  ) internal virtual {
832    _burnFrom(owner, tokenId);
833  }
834
835  function _burn(
836    address owner,
837    uint256 tokenId,
838    address operator,
839    bytes memory data,
840    bytes memory extraData,
841    bytes memory extraData2,
842    bytes memory extraData3,
843    bytes memory extraData4,
844    bytes memory extraData5,
845    bytes memory extraData6,
846    bytes memory extraData7,
847    bytes memory extraData8,
848    bytes memory extraData9,
849    bytes memory extraData10,
850    bytes memory extraData11,
851    bytes memory extraData12,
852    bytes memory extraData13,
853    bytes memory extraData14,
854    bytes memory extraData15,
855    bytes memory extraData16,
856    bytes memory extraData17,
857    bytes memory extraData18,
858    bytes memory extraData19,
859    bytes memory extraData20,
860    bytes memory extraData21,
861    bytes memory extraData22,
862    bytes memory extraData23
863  ) internal virtual {
864    _burnFrom(owner, tokenId);
865  }
866
867  function _burn(
868    address owner,
869    uint256 tokenId,
870    address operator,
871    bytes memory data,
872    bytes memory extraData,
873    bytes memory extraData2,
874    bytes memory extraData3,
875    bytes memory extraData4,
876    bytes memory extraData5,
877    bytes memory extraData6,
878    bytes memory extraData7,
879    bytes memory extraData8,
880    bytes memory extraData9,
881    bytes memory extraData10,
882    bytes memory extraData11,
883    bytes memory extraData12,
884    bytes memory extraData13,
885    bytes memory extraData14,
886    bytes memory extraData15,
887    bytes memory extraData16,
888    bytes memory extraData17,
889    bytes memory extraData18,
890    bytes memory extraData19,
891    bytes memory extraData20,
892    bytes memory extraData21,
893    bytes memory extraData22,
894    bytes memory extraData23,
895    bytes memory extraData24
896  ) internal virtual {
897    _burnFrom(owner, tokenId);
898  }
899
900  function _burn(
901    address owner,
902    uint256 tokenId,
903    address operator,
904    bytes memory data,
905    bytes memory extraData,
906    bytes memory extraData2,
907    bytes memory extraData3,
908    bytes memory extraData4,
909    bytes memory extraData5,
910    bytes memory extraData6,
911    bytes memory extraData7,
912    bytes memory extraData8,
913    bytes memory extraData9,
914    bytes memory extraData10,
915    bytes memory extraData11,
916    bytes memory extraData12,
917    bytes memory extraData13,
918    bytes memory extraData14,
919    bytes memory extraData15,
920    bytes memory extraData16,
921    bytes memory extraData17,
922    bytes memory extraData18,
923    bytes memory extraData19,
924    bytes memory extraData20,
925    bytes memory extraData21,
926    bytes memory extraData22,
927    bytes memory extraData23,
928    bytes memory extraData24,
929    bytes memory extraData25
930  ) internal virtual {
931    _burnFrom(owner, tokenId);
932  }
933
934  function _burn(
935    address owner,
936    uint256 tokenId,
937    address operator,
938    bytes memory data,
939    bytes memory extraData,
940    bytes memory extraData2,
941    bytes memory extraData3,
942    bytes memory extraData4,
943    bytes memory extraData5,
944    bytes memory extraData6,
945    bytes memory extraData7,
946    bytes memory extraData8,
947    bytes memory extraData9,
948    bytes memory extraData10,
949    bytes memory extraData11,
950    bytes memory extraData12,
951    bytes memory extraData13,
952    bytes memory extraData14,
953    bytes memory extraData15,
954    bytes memory extraData16,
955    bytes memory extraData17,
956    bytes memory extraData18,
957    bytes memory extraData19,
958    bytes memory extraData20,
959    bytes memory extraData21,
960    bytes memory extraData22,
961    bytes memory extraData23,
962    bytes memory extraData24,
963    bytes memory extraData25,
964    bytes memory extraData26
965  ) internal virtual {
966    _burnFrom(owner, tokenId);
967  }
968
969  function _burn(
970    address owner,
971    uint256 tokenId,
972    address operator,
973    bytes memory data,
974    bytes memory extraData,
975    bytes memory extraData2,
976    bytes memory extraData3,
977    bytes memory extraData4,
978    bytes memory extraData5,
979    bytes memory extraData6,
980    bytes memory extraData7,
981    bytes memory extraData8,
982    bytes memory extraData9,
983    bytes memory extraData10,
984    bytes memory extraData11,
985    bytes memory extraData12,
986    bytes memory extraData13,
987    bytes memory extraData14,
988    bytes memory extraData15,
989    bytes memory extraData16,
990    bytes memory extraData17,
991    bytes memory extraData18,
992    bytes memory extraData19,
993    bytes memory extraData20,
994    bytes memory extraData21,
995    bytes memory extraData22,
996    bytes memory extraData23,
997    bytes memory extraData24,
998    bytes memory extraData25,
999    bytes memory extraData26,
1000   bytes memory extraData27
1001 ) internal virtual {
1002   _burnFrom(owner, tokenId);
1003 }
1004
1005  function _burn(
1006    address owner,
1007    uint256 tokenId,
1008    address operator,
1009    bytes memory data,
1010    bytes memory extraData,
1011    bytes memory extraData2,
1012    bytes memory extraData3,
1013    bytes memory extraData4,
1014    bytes memory extraData5,
1015    bytes memory extraData6,
1016    bytes memory extraData7,
1017    bytes memory extraData8,
1018    bytes memory extraData9,
1019    bytes memory extraData10,
1020    bytes memory extraData11,
1021    bytes memory extraData12,
1022    bytes memory extraData13,
1023    bytes memory extraData14,
1024    bytes memory extraData15,
1025    bytes memory extraData16,
1026    bytes memory extraData17,
1027    bytes memory extraData18,
1028    bytes memory extraData19,
1029    bytes memory extraData20,
1030    bytes memory extraData21,
1031    bytes memory extraData22,
1032    bytes memory extraData23,
1033    bytes memory extraData24,
1034    bytes memory extraData25,
1035    bytes memory extraData26,
1036    bytes memory extraData27,
1037    bytes memory extraData28
1038  ) internal virtual {
1039    _burnFrom(owner, tokenId);
1040  }
1041
1042  function _burn(
1043    address owner,
1044    uint256 tokenId,
1045    address operator,
1046    bytes memory data,
1047    bytes memory extraData,
1048    bytes memory extraData2,
1049    bytes memory extraData3,
1050    bytes memory extraData4,
1051    bytes memory extraData5,
1052    bytes memory extraData6,
1053    bytes memory extraData7,
1054    bytes memory extraData8,
1055    bytes memory extraData9,
1056    bytes memory extraData10,
1057    bytes memory extraData11,
1058    bytes memory extraData12,
1059    bytes memory extraData13,
1060    bytes memory extraData14,
1061    bytes memory extraData15,
1062    bytes memory extraData16,
1063    bytes memory extraData17,
1064    bytes memory extraData18,
1065    bytes memory extraData19,
1066    bytes memory extraData20,
1067    bytes memory extraData21,
1068    bytes memory extraData22,
1069    bytes memory extraData23,
1070    bytes memory extraData24,
1071    bytes memory extraData25,
1072    bytes memory extraData26,
1073    bytes memory extraData27,
1074    bytes memory extraData28,
1075    bytes memory extraData29
1076  ) internal virtual {
1077    _burnFrom(owner, tokenId);
1078  }
1079
1080  function _burn(
1081    address owner,
1082    uint256 tokenId,
1083    address operator,
1084    bytes memory data,
1085    bytes memory extraData,
1086    bytes memory extraData2,
1087    bytes memory extraData3,
1088    bytes memory extraData4,
1089    bytes memory extraData5,
1090    bytes memory extraData6,
1091    bytes memory extraData7,
1092    bytes memory extraData8,
1093    bytes memory extraData9,
1094    bytes memory extraData10,
1095    bytes memory extraData11,
1096    bytes memory extraData12,
1097    bytes memory extraData13,
1098    bytes memory extraData14,
1099    bytes memory extraData15,
1100    bytes memory extraData16,
1101    bytes memory extraData17,
1102    bytes memory extraData18,
1103    bytes memory extraData19,
1104    bytes memory extraData20,
1105    bytes memory extraData21,
1106    bytes memory extraData22,
1107    bytes memory extraData23,
1108    bytes memory extraData24,
1109    bytes memory extraData25,
1110    bytes memory extraData26,
1111    bytes memory extraData27,
1112    bytes memory extraData28,
1113    bytes memory extraData29,
1114    bytes memory extraData30
1115  ) internal virtual {
1116    _burnFrom(owner, tokenId);
1117  }
1118
1119  function _burn(
1120    address owner,
1121    uint256 tokenId,
1122    address operator,
1123    bytes memory data,
1124    bytes memory extraData,
1125    bytes memory extraData2,
1126    bytes memory extraData3,
1127    bytes memory extraData4,
1128    bytes memory extraData5,
1129    bytes memory extraData6,
1130    bytes memory extraData7,
1131    bytes memory extraData8,
1132    bytes memory extraData9,
1133    bytes memory extraData10,
1134    bytes memory extraData11,
1135    bytes memory extraData12,
1136    bytes memory extraData13,
1137    bytes memory extraData14,
1138    bytes memory extraData15,
1139    bytes memory extraData16,
1140    bytes memory extraData17,
1141    bytes memory extraData18,
1142    bytes memory extraData19,
1143    bytes memory extraData20,
1144    bytes memory extraData21,
1145    bytes memory extraData22,
1146    bytes memory extraData23,
1147    bytes memory extraData24,
1148    bytes memory extraData25,
1149    bytes memory extraData26,
1150    bytes memory extraData27,
1151    bytes memory extraData28,
1152    bytes memory extraData29,
1153    bytes memory extraData30,
1154    bytes memory extraData31
1155  ) internal virtual {
1156    _burnFrom(owner, tokenId);
1157  }
1158
1159  function _burn(
1160    address owner,
1161    uint256 tokenId,
1162    address operator,
1163    bytes memory data,
1164    bytes memory extraData,
1165    bytes memory extraData2,
1166    bytes memory extraData3,
1167    bytes memory extraData4,
1168    bytes memory extraData5,
1169    bytes memory extraData6,
1170    bytes memory extraData7,
1171    bytes memory extraData8,
1172    bytes memory extraData9,
1173    bytes memory extraData10,
1174    bytes memory extraData11,
1175    bytes memory extraData12,
1176    bytes memory extraData13,
1177    bytes memory extraData14,
1178    bytes memory extraData15,
1179    bytes memory extraData16,
1180    bytes memory extraData17,
1181    bytes memory extraData18,
1182    bytes memory extraData19,
1183    bytes memory extraData20,
1184    bytes memory extraData21,
1185    bytes memory extraData22,
1186    bytes memory extraData23,
1187    bytes memory extraData24,
1188    bytes memory extraData25,
1189    bytes memory extraData26,
1190    bytes memory extraData27,
1191    bytes memory extraData28,
1192    bytes memory extraData29,
1193    bytes memory extraData30,
1194    bytes memory extraData31,
1195    bytes memory extraData32
1196  ) internal virtual {
1197    _burnFrom(owner, tokenId);
1198  }
1199
1200  function _burn(
1201    address owner,
1
```

18.5 Token Time Based Locking

The problem with this Smart Contract is that the transfer is always possible. That means, if we ship the actual physical product to someone, he can just send the NFT somewhere else in the meantime. There is no proof that the product ever received correctly. It is possible to send the NFT to someone else, that person now knowing if the product was already shipped - a property we want to have on-chain.

Our Token just doesn't have the Supply Chain component yet built in.

Let's add a locking mechanism to the Smart Contract.

We need several things for this to work:

1. A timestamp for each token from when this token isn't transferrable anymore. This can be potentially be set on minting.
2. Some unlock code, which is hashed twice (e.g. `hash(hash(code))`), so on the blockchain the original passphrase is never revealed, we just send `hash(code)`. More on that later.
3. A variable if the token was unlocked already or not.
4. A hook into the transfer function to only allow token transfer of tokens which are not yet locked or which have been successfully unlocked

Let's have a look at how we can do this in code with the OpenZeppelin Preset.

18.5.1 Modifying the Smart Contract

And here is the final contract for a first try:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

import "@openzeppelin/contracts/token/ERC721/presets/ERC721PresetMinterPauserAutoId.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract AishtisiToken is ERC721PresetMinterPauserAutoId {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    mapping (uint => uint) public tokenLockedFromTimestamp;
    mapping (uint => bytes32) public tokenUnlockCodeHashes;
    mapping (uint => bool) public tokenUnlocked;

    event TokenUnlocked(uint tokenId, address unlockerAddress);

    constructor() ERC721PresetMinterPauserAutoId("AishtisiToken", "AIS", "https://aishtisi.art/metadata/") {}

    function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal override {
        require(tokenLockedFromTimestamp[tokenId] > block.timestamp || tokenUnlocked[tokenId], "AishtisiToken: Token locked");
        super._beforeTokenTransfer(from, to, tokenId);
    }

    function unlockToken(bytes32 unlockHash, uint256 tokenId) public {
        require(msg.sender == ownerOf(tokenId), "AishtisiToken: Only the Owner can unlock the Token"); //not 100% sure about that one yet
        require(keccak256(abi.encode(unlockHash)) == tokenUnlockCodeHashes[tokenId], "AishtisiToken: Unlock Code Incorrect");
        tokenUnlocked[tokenId] = true;
        emit TokenUnlocked(tokenId, msg.sender);
    }

    /**
     * This one is the mint function that sets the unlock code, then calls the parent mint
     */
    function mint(address to, uint lockedFromTimestamp, bytes32 unlockHash) public {
        tokenLockedFromTimestamp[_tokenIds.current()] = lockedFromTimestamp;
        tokenUnlockCodeHashes[_tokenIds.current()] = unlockHash;
        _tokenIds.increment();
        super.mint(to);
    }

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        return string(abi.encodePacked(super.tokenURI(tokenId), ".json"));
    }
}
```

18.5.2 Testing it in the Truffle Development Console

Let's give it a try again: 1. Open `truffle develop` 2. Deploy the contracts to this test-network with `migrate` 3. Then mint a new token which locks 3 seconds in the future:

```
let token = await AisthisiToken.deployed()
let txMint = await token.mint(accounts[1], Math.round(Date.now()/1000)+3, web3.utils.sha3(web3.utils.sha3('mySecretHash')))

txMint.receipt.status
```

That should return true

The screenshot shows the Visual Studio Code interface with the Truffle Development Console extension installed. The Explorer sidebar on the left lists project files: AishtisiToken.sol, contracts, ERC721.sol, 2_token_migration.js, migrations, build, contracts, Migrations.sol, 1_initial_migration.js, 2_token_migration.js, node_modules/@openzeppelin/contracts, test, package-lock.json, and truffle-config.js. The main editor area displays the Solidity code for AishtisiToken.sol, specifically the unlockToken function. The terminal output at the bottom right shows the deployment of two contracts, the cost of 0.1015441 ETH, and a summary of the deployment process. The Truffle shell prompt is visible at the bottom.

```

AishtisiToken.sol - oiltoken-writeup - V
File Edit Selection View Go Run Terminal Help
AishtisiToken.sol - oiltoken-writeup - V
EXPLORER ...
OPEN EDITORS ...
contracts > AishtisiToken.sol > AishtisiToken > 2_token_migration.js
24 function unlockToken(bytes32 unlock)
25     require(msg.sender == ownerOf(tokenId));
26     require(keccak256(abi.encode(
27         tokenUnlocked[tokenId] = true));
28     emit TokenUnlocked(tokenId);
29 }
30 /**
31 * This one is the mint function
32 */
33

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.09660602 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.1015441 ETH

- Blocks: 0 Seconds: 0
- Saving migration to chain.
- Blocks: 0 Seconds: 0
- Saving migration to chain.

truffle(develop)> let token = await AishtisiToken
undefined
truffle(develop)> let txMint = await token.mint(
3(web3.utils.sha3('mySecretHash')));
undefined
truffle(develop)> txMint.receipt.status
true
truffle(develop)>

```

Let's wait 3 seconds and then try to send the token from account[1] to account[2]:

```
let txTransfer = await token.transferFrom(accounts[1], accounts[2], 0, {from: accounts[1]});
```

This will output an error:

```
Error: Returned error: VM Exception while processing transaction: revert AishtisiToken: Token locked -- Reason given:  
AishtisiToken: Token locked.
```

The screenshot shows the Visual Studio Code (VS Code) interface with the Truffle Development Console integrated. The left sidebar (Explorer) lists project files: AishtisiToken.sol, contracts, ERC721.sol, node_modules/@openzeppelin/contracts, 2_token_migration.js, migrations, and truffle-config.js. Below these are sections for OILTOKEN-WRITEUP, build, contracts, migrations, node_modules/@openzeppelin/contracts, test, package-lock.json, and truffle-config.js. The main area has tabs for AishtisiToken.sol, ERC721.sol, and 2_token_migration.js. The AishtisiToken.sol tab is active, showing the Solidity code for the token contract. The terminal window at the bottom shows the command truffle(develop)> let txTransfer = await token.unlockToken(); followed by an error message: Error: Returned error: VM Exception while processing transaction: revert AishtisiToken: Token locked. The stack trace for this error is also visible in the terminal.

```
AishtisiToken.sol - oiltoken-writeup - V
File Edit Selection View Go Run Terminal Help
AishtisiToken.sol - oiltoken-writeup - V
contracts > AishtisiToken.sol > AishtisiToken > AishtisiToken.sol
16     ftrace | funcSig
17     constructor() ERC721PresetMinterPauser
18
19     ftrace | funcSig
20     function _beforeTokenTransfer(address
21         require(tokenLockedFromTimes > now);
22         super._beforeTokenTransfer(f
23
24     ftrace | funcSig
25     function unlockToken(bytes32 unl
26         require(msg.sender == ownerOf(unl
         require(keccak256(abi.encode(
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
truffle(develop)> let txTransfer = await token.unlockToken();
]);
Uncaught:
Error: Returned error: VM Exception while processing transaction: revert AishtisiToken: Token locked.
    at evalmachine.<anonymous>:1:15
    at evalmachine.<anonymous>:2:49
    at sigintHandlersWrap (vm.js:273:12)
    at Script.runInContext (vm.js:140:14)
    at runScript (C:\Users\thoma\AppData\Roaming\npm\truffle\lib\console.js:270:1)
    at Console.interpret (C:\Users\thoma\AppData\Roaming\npm\truffle\lib\console.js:285:1)
    at bound (domain.js:413:15)
    at REPLServer.runBound [as eval] (domain.js:417:12)
    at REPLServer.onLine (repl.js:817:18)
    at REPLServer.emit (events.js:315:20)
    at REPLServer.EventEmitter.emit (domain.js:417:12)
    at REPLServer.Interface._onLine (readline.js:391:10)
    at REPLServer.Interface._line (readline.js:415:10)
    at REPLServer.Interface._ttyWrite (readline.js:442:10)
    data: {
        '0x18e47e0ad55482623ce624fbaeca68239749b1d

```

Let's unlock the token then. We have to do it from the current token holder address, because only the current NFT token holder can unlock the token:

```
let txUnlock = await token.unlockToken(web3.utils.sha3('mySecretHash'), 0, {from: accounts[1]})

txUnlock.receipt.status
```

That should return true:

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:**
 - OPEN EDITORS:** AishtisiToken.sol contracts, ERC721.sol node_modules\@openzeppelin\contracts..., JS 2_token_migration.js migrations, JS truffle-config.js.
 - OILTOKEN-WRITEUP:** build, contracts (AishtisiToken.sol, Migrations.sol), migrations (1_initial_migration.js, 2_token_migration.js), node_modules\@openzeppelin\contracts, test, package-lock.json, truffle-config.js.
- Terminal:**

```
16      ftrace | funcSig
17      constructor() ERC721PresetMinter
18
19      ftrace | funcSig
20      function _beforeTokenTransfer(ad
21          require(tokenLockedFromTimes)
22          super._beforeTokenTransfer(f
23
24      ftrace | funcSig
25      function unlockToken(bytes32 unl
26          require(msg.sender == ownerO
              require(keccak256(abi.encode
```

```
es\\truffle\\build\\webpack:\\node_modules\\web
+
    at XMLHttpRequestEventTarget.dispatchEvent
\\truffle\\build\\webpack:\\node_modules\\xhr2-
    at XMLHttpRequest.exports.modules.9967
\\Roaming\\npm\\node_modules\\truffle\\build\\w
.js:208:1)\n' +
    at XMLHttpRequest.exports.modules.9967
Data\\Roaming\\npm\\node_modules\\truffle\\buil
uest.js:318:1)\n' +
    at IncomingMessage.<anonymous> (C:\\Us
uild\\webpack:\\node_modules\\xhr2-cookies\\di
    at IncomingMessage.emit (events.js:327)
    at IncomingMessage.EventEmitter.emit (emitt
    at endReadableNT (internal/streams/readable
    at processTicksAndRejections (internal/ti
```

```
ts[1]))
```

```
truffle(develop)> let txUnlock = await token.un
truffle(develop)> txUnlock.receipt.status
```

```
true
```

```
truffle(develop)>
```

Now it should be possible to send tokens again:

```
let txTransfer2 = await token.transferFrom(accounts[1], accounts[2], 0, {from: accounts[1]});  
txTransfer2.receipt.status
```

should return true.

```
truffle(develop)> let txTransfer2 = await token.transferFrom(accounts[1], accounts[2], 0, {from: accounts[1]});  
undefined  
truffle(develop)> txTransfer2.receipt.status  
true  
truffle(develop)> █
```

The current owner of token #0 is account #2:

```
await token.ownerOf(0)  
accounts[2]
```

Should be equal:

```
truffle(develop)> await token.ownerOf(0);  
'0xc03d9F27B701a1d52ebAEA7386435De2a436ae65'  
truffle(develop)> accounts[2]  
'0xc03d9F27B701a1d52ebAEA7386435De2a436ae65'  
truffle(develop)> █
```

Now, let's exit the truffle developer console with `.exit` so you are on a regular terminal inside the project folder.

Last update: June 23, 2021

18.6 Unit Testing the ERC721 Token

As always, we need to add Unit Tests. Without going too much into the unit-testing details. The tests are doing essentially the same as we did manually here and will come in handy in the next part, where we are optimizing for gas cost and where we customize the Token. Essentially we don't need the burn and pause functionality at all, and we can also make it a bit more lightweight.

18.6.1 Install truffle-assertions

On a regular terminal (exit the truffle developer console if you are still in there), we need to add another npm package.

Install truffle assertions:

```
npm install truffle-assertions
```

18.6.2 Adding a Unit Test

Then add the following unit tests to `/test/lockUnlocking.test.js`

```
const AishtisiToken = artifacts.require('AishtisiToken');
const truffleAssert = require('truffle-assertions');

let correctUnlockCode = web3.utils.sha3('test'); //test is the password
let timestampLockedFrom = Math.round(Date.now() / 1000) + 3; //lock it in 3 seconds to test unlock
let unlockCodeHash = web3.utils.sha3(correctUnlockCode); //double hashed

contract('AishtisiToken: test mint and lock', (accounts) => {
    const [deployerAddress, tokenHolderOneAddress, tokenHolderTwoAddress] = accounts;

    it('is possible to mint tokens for the minter role', async () => {
        let token = await AishtisiToken.deployed();

        await token.mint(tokenHolderOneAddress, timestampLockedFrom, unlockCodeHash); //minting works
        await truffleAssert.fails(token.transferFrom(deployerAddress, tokenHolderOneAddress, 0)); //transferring for others doesn't work

        //but transferring in general works
        await truffleAssert.passes(
            token.transferFrom(tokenHolderOneAddress, tokenHolderTwoAddress, 0, { from: tokenHolderOneAddress })
        );
    });

    it('is not possible to transfer locked tokens', async () => {
        //unless we wait 4 seconds and the token will be locked
        let token = await AishtisiToken.deployed();
        await new Promise((res) => {
            setTimeout(res, 4000);
        });
        await truffleAssert.fails(
            token.transferFrom(tokenHolderTwoAddress, tokenHolderOneAddress, 0, { from: tokenHolderTwoAddress }),
            truffleAssert.ErrorType.REVERT,
            'AishtisiToken: Token locked',
        );
    });

    it('is not possible to unlock tokens for anybody else than the token holder', async () => {
        let token = await AishtisiToken.deployed();
        await truffleAssert.fails(
            token.unlockToken(correctUnlockCode, 0, { from: deployerAddress }),
            truffleAssert.ErrorType.REVERT,
            'AishtisiToken: Only the Owner can unlock the Token',
        );
    });

    it('is not possible to unlock tokens without the correct unlock code', async () => {
        let token = await AishtisiToken.deployed();
        let wrongUnlockCode = web3.utils.sha3('Santa Lucia');
        await truffleAssert.fails(
            token.unlockToken(wrongUnlockCode, 0, { from: tokenHolderTwoAddress }),
            truffleAssert.ErrorType.REVERT,
            'AishtisiToken: Unlock Code Incorrect',
        );
    });

    it('is possible to unlock the token and transfer it again', async () => {
        let token = await AishtisiToken.deployed();
        await truffleAssert.passes(token.unlockToken(correctUnlockCode, 0, { from: tokenHolderTwoAddress }));
        await truffleAssert.passes(

```

```

    token.transferFrom(tokenHolderTwoAddress, deployerAddress, 0, { from: tokenHolderTwoAddress }),
);
let tokenOwner = await token.ownerOf(0);
assert.equal(tokenOwner, deployerAddress, 'The Owner is not the correct address');
});

it('is possible to retrieve the correct token URI', async () => {
  let token = await AisthisIToken.deployed();
  let metadata = await token.tokenURI(0);
  assert.equal('https://aisthis.i.art/metadata/0.json', metadata);
})
});

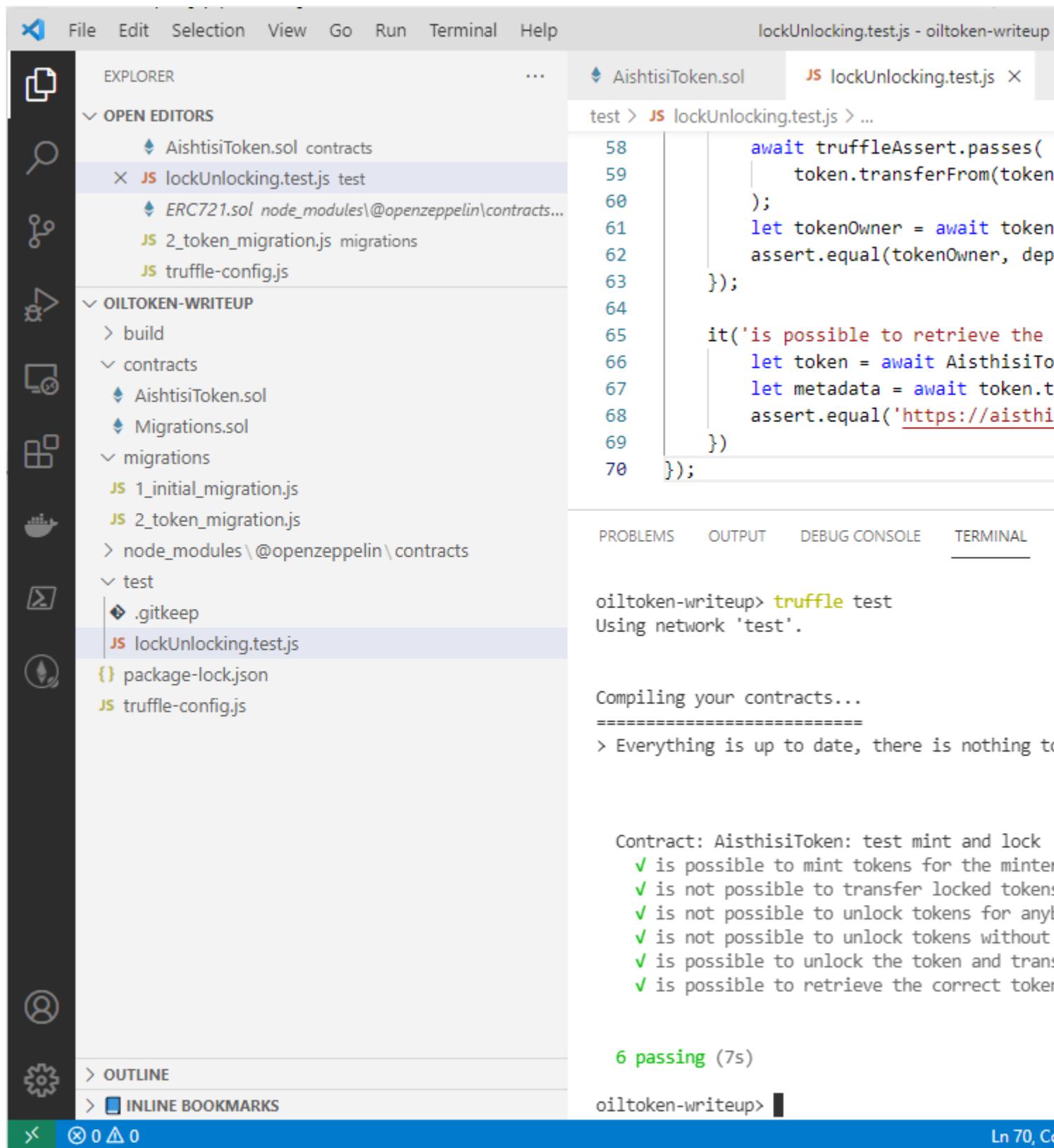
```

So, what's inside the tests?

- is possible to mint tokens for the minter role tests that only an account with the "minter" role can mint tokens
- is not possible to transfer locked tokens tests that locked tokens can really not be transferred
- is not possible to unlock tokens for anybody else than the token holder tests that only the NFT holder can actually unlock the tokens
- is not possible to unlock tokens without the correct unlock code tests that you can not unlock the token with an incorrect code
- is possible to unlock the token and transfer it again tests that you can unlock the token with the correct code and then send it to someone else again
- is possible to retrieve the correct token URI tests that we are actually getting the URI back that we expect

18.6.3 Running the Unit Test

Then simply run `truffle test`:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like AishtisiToken.sol, lockUnlocking.test.js, and various migration scripts.
- Editor (Top Right):** Displays the code for lockUnlocking.test.js, which includes assertions for token transfer, metadata retrieval, and migration execution.
- Terminal (Bottom Right):** Shows the command "truffle test" being run, and the output indicating that 6 tests passed in 7 seconds.

```

58     await truffleAssert.passes(
59         | token.transferFrom(tokenOwner,
60             );
61         let tokenOwner = await tokenOwner;
62         assert.equal(tokenOwner, deployment);
63     );
64
65     it('is possible to retrieve the',
66         | let token = await AishtisiToken();
67         let metadata = await token.metadata();
68         assert.equal(metadata.url, 'https://aisthisi');
69     );
70 });

```

```

oiltoken-writeup> truffle test
Using network 'test'.

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: AishtisiToken: test mint and lock
  ✓ is possible to mint tokens for the minter
  ✓ is not possible to transfer locked tokens
  ✓ is not possible to unlock tokens for anyone
  ✓ is not possible to unlock tokens without
  ✓ is possible to unlock the token and transfer
  ✓ is possible to retrieve the correct token

```

```

6 passing (7s)
oiltoken-writeup>

```

I personally like going through tests, because they tell me what the Author of Smart Contracts had in mind. They are usually quite easy to read. I hope ours are easy enough to understand what's going on under the hood.

18.7 Final Words and Next Part

This time we introduced a novel locking mechanism. Once locked and unlocked, it give proof that a physical item was shipped and received. It needs to be unlocked by the token holder to allow further transfer.

In the next part I want to tackle a few Gas optimizations. So stay tuned.

In the meantime follow us on <https://aisthisi.art>

Last update: June 23, 2021

19. LEARN: ERC721 Token Secondary Sales Royalties

19.1 ERC721 Part 2: Secondary Sales Royalties

For project [Project Aisthisi](#) we are exploring the connection between digital art, blockchain and olive oil. In other words: ERC721 Tokens with a supply chain component attached to it.

In the [previous part 1](#) we were setting up a basic token with some locking functionality using the ERC721PresetMinterPauserAutoId. In this part we are reducing it to the most necessary functionality and add in Secondary Sales Royalties for Opensea, Rarible and Mintable.

In the final part we are building a small Web3 UI around this.



19.1.1 Background

Beginning of 2021 we (the Team behind the Ethereum Blockchain Course) had an intriguing idea: What if we can auction tokens like traditional NFTs, but have a physical component to it? [Project Aishtisi](#) was born. This page contains the technical lessons learned and a full rundown of the Smart Contract architecture we're using for the project.

19.1.2 Content

This is the lab you've been looking for if you want a tutorial style guide for low(er) level ERC721 Smart Contracts which contain a Supply Chain component. All following industry best practices.

This is for educational purposes only, do your own Audit when used in Production.

19.1.3 Real-World Use-Case for this Project

ERC721 Token generation

Secondary Sales Royalties on Opensea, Rarible and Mintable (ERC2981 NFT Royalty Standard)

Understanding the costs for Deployment of ERC721 Tokens

19.1.4 Development-Goal

Deep Dive Into ERC721

⊕ Understand how tokens are auctioned on Opensea or Mintable

No Liability - Educational Only!

Platforms change all the time and it is a very volatile marketplace. This example is for educational purposes only. Consult an expert before deploying tokens in production and use on your own risk!

These were our learnings when we deployed the [Aisthisi Token](#).

If you're ok with that, then let's get started!

Last update: January 3, 2022

19.2 The Token Improvement

In the previous part we were developing our simple token based on a preset from OpenZeppelin. On top of it we developed a lockable component, but we did this with the ERC721PresetMinterPauserAutoId.

In this tutorial you will learn one specific component, which is the secondary royalties. To do that, we will start from scratch with a simple ERC721 Token from OpenZeppelin. It's not a direct continuation of the previous part, it's an ERC721 Token with minimal requirements from scratch.

19.2.1 Prerequisite

To get started, you need:

[Node](#) and [Truffle](#) installed (best globally).

```
npm install -g truffle
```

19.2.2 Empty Truffle Project with OpenZeppelin ERC721 Token

Open a new empty directory in your favorite IDE (here it is Visual Studio Code). And then open a terminal and type in:

```
truffle init  
npm init -y  
npm install @openzeppelin/contracts
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Left Sidebar:**
 - EXPLORER:** Shows the project structure under "ERC721_ROYALTIES".
 - OPEN EDITORS:** Shows the "Welcome" file.
 - OUTLINE:**
 - INLINE BOOKMARKS:**
 - Bottom Status Bar:** Shows a green icon with a white X, a blue icon with a white circle containing a number, and a white icon with a blue triangle.
- Center Panel:** A "Welcome" tab is open, displaying options: Start, New File..., Open File..., Open Folder..., and Run a Command... .
- Bottom Panel:** Shows the terminal output for an "npm install" command:


```
"name": "erc721_royalties",
"version": "1.0.0",
"description": "",
"main": "truffle-config.js",
"directories": {
  "test": "test"
},
"dependencies": {},
"devDependencies": {},
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}

erc721_royalties> npm install @openzeppelin/contracts
npm WARN erc721_royalties@1.0.0 No description
npm WARN erc721_royalties@1.0.0 No repository field.

+ @openzeppelin/contracts@4.2.0
updated 1 package and audited 1 package in 0.597s
found 0 vulnerabilities

erc721_royalties> 
```

Great, now we can get started with the actual token!

Last update: August 19, 2021

19.3 Minimal ERC721 Token

If you look at the official [ERC721 Standard](#) then you see there are a few mandatory and a few optional functionalities the smart contract needs to implement.

Luckily OpenZeppelin has us covered for most functions directly out of the box.

Let's have a look at a minimal example using the audited functions from OpenZeppelin.

Create a new file "MinimalERC721.sol" in the contracts folder and insert the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/token/ERC721/ERC721.sol";

contract MinimalERC721 is ERC721 {
    constructor() ERC721("Minimal", "MIN") {}
}
```

Before we can compile this we also have to change the truffle-config.js file. Open it and on the bottom you will find compiler settings. Set the version to 0.8.0 and enable the optimizer:

```
// Configure your compilers
compilers: {
  solc: {
    version: "0.8.4",      // Fetch exact version from solc-bin (default: truffle's version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
    settings: {            // See the solidity docs for advice about optimization and evmVersion
      optimizer: {
        enabled: true,
        runs: 200
      },
      // evmVersion: "byzantium"
    }
  },
},
```

then run `truffle compile`. It should output something like this:

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER** pane on the left:
 - OPEN EDITORS**: Shows `MinimalERC721.sol contracts` and `truffle-config.js`.
 - ERC721_ROYALTIES**: Shows a folder structure with:
 - `build`
 - `contracts` (selected): Contains `Migrations.sol` and `MinimalERC721.sol`.
 - `migrations`
 - `node_modules\@openzeppelin\contracts` (selected): Contains `access`, `build`, `finance`, `governance`, `interfaces`, `metatx`, `proxy`, `security`, and `token`.
 - `token` (selected): Contains `ERC20`, `ERC721` (selected), `extensions`, `presets`, `utils`, `ERC721.sol`, `IERC721.sol`, `IERC721Receiver.sol`, `README.adoc`, and `ERC777`.
 - `OUTLINE` and `INLINE BOOKMARKS`
- EDITOR** pane on the right:
 - File: `MinimalERC721.sol` (selected)
 - Language: `JS truffle-config.js`
 - Content area shows the Solidity code for `MinimalERC721.sol`:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MinimalERC721 is ERC721, Ownable {
    constructor() ERC721("Minimal", "MIN") {}
}
```
- TERMINAL** pane at the bottom:
 - Shows the command `truffle compile` being run.
 - Output:


```
--> /C/101Projects/erc721_royalties/contracts/MinimalERC721.sol
> Artifacts written to C:\101Projects\erc721_royalties\artifacts
> Compiled successfully using:
- solc: 0.8.4+commit.c7e474f2.Emscripten clang

erc721_royalties> truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\MinimalERC721.sol
> Artifacts written to C:\101Projects\erc721_royalties\artifacts
> Compiled successfully using:
- solc: 0.8.4+commit.c7e474f2.Emscripten clang

erc721_royalties>
```
 - Status bar at the bottom right: Ln 1, Col 27 | Spaces: 4 | UTF-8

Great!

Let's add a migration for the Token!

Last update: August 19, 2021

19.4 Adding a Migration for the ERC721 Token

To deploy the contract we need a migrations file!

In the migrations folder create a new file "2_erc721.js" and insert the following content:

```
const Token = artifacts.require("MinimalERC721");

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

Then open the truffle developer network and sample migrate the token!

```
truffle develop
```

then simply

```
migrate
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure under "OPEN EDITORS" (1 UNSAVED):
 - MinimalERC721.sol contracts
 - JS 1_initial_migration.js migrations
 - X JS 2_erc721.js migrations
 - JS truffle-config.js
- Editor:** Displays the content of the "JS 2_erc721.js" file in the "migrations" folder:


```

1 const Token = artifacts.require("MinimalERC721");
2
3 module.exports = function (deployer) {
4   deployer.deploy(Token);
5 };
6
    
```
- Terminal:** Shows the output of a Truffle migration command:


```

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.002217612 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.002217612 ETH

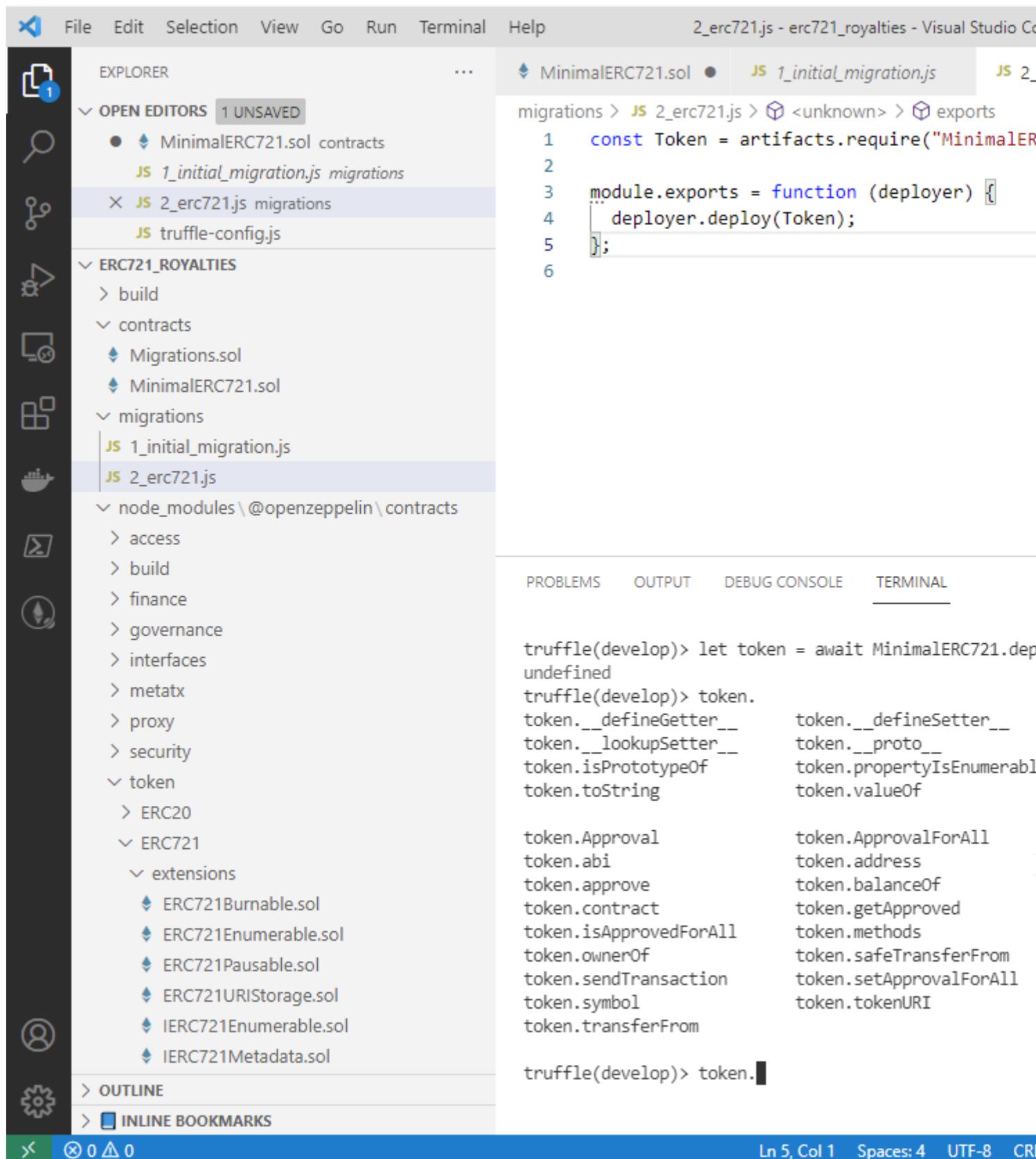
- Blocks: 0 Seconds: 0
- Saving migration to chain.

truffle(develop)>
    
```
- Status Bar:** Ln 5, Col 1 Spaces: 4 UTF-8 CR

Now you can interact with your token on a developer network!

```
let token = await MinimalERC721.deployed();
```

Then you could auto-complete by pressing the tab-key and see which functions are available:



```

File Edit Selection View Go Run Terminal Help
2_erc721.js - erc721_royalties - Visual Studio Code

EXPLORER ... MINIMALERC721.sol 1_INITIAL_MIGRATION.JS JS 2_
OPEN EDITORS 1 UNSAVED
MinimalERC721.sol contracts
JS 1_initial_migration.js migrations
X JS 2_erc721.js migrations
JS truffle-config.js
ERC721_ROYALTIES
> build
contracts
Migrations.sol
MinimalERC721.sol
migrations
JS 1_initial_migration.js
JS 2_erc721.js
node_modules\@openzeppelin\contracts
> access
> build
> finance
> governance
> interfaces
> metatx
> proxy
> security
token
> ERC20
ERC721
extensions
ERC721Burnable.sol
ERC721Enumerable.sol
ERC721Pausable.sol
ERC721URIStorage.sol
IERC721Enumerable.sol
IERC721Metadata.sol
OUTLINE
INLINE BOOKMARKS
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
truffle(develop)> let token = await MinimalERC721.deploy()
truffle(develop)> token.
token.__defineGetter__
token.__lookupSetter__
token.isPrototypeOf
token.toString
token.Approval
token.abi
token.approve
token.contract
token.isApprovedForAll
token.ownerOf
token.sendTransaction
token.symbol
token.transferFrom
truffle(develop)> token.
Ln 5, Col 1 Spaces: 4 UTF-8 CR

```

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:** The file `2_erc721.js` is open, showing a migration script for the `MinimalERC721` contract.
- Explorer:** Shows the project structure under `ERC721_ROYALTIES`, including `node_modules\@openzeppelin\contracts`.
- Terminal:** Shows Truffle development environment commands being run.
- Status Bar:** Ln 5, Col 1, Spaces: 4, UTF-8, CR.

You see that we have all the transfer and balance functionality - but there is nothing where we can actually mint a token.

We have a token, but so far you can't mint and you can't collect any royalties on any platform.

After this long prologue, let's go and add the first platform: OpenSea!

Last update: August 19, 2021

19.5 Secondary Royalties on OpenSea

Opensea is allowing the Contract Owner to set royalties on a whole token collection. That ownership is determined by querying a function called `owner()` in the smart contract.

If you ever wondered why you can't set secondary sales royalties on tokens minted by Mintable or Rarible its because you are not the contract owner - you are the token owner!

But here we're doing our own custom contract.

And the best part is: We don't have to write that functionality ourselves from scratch, because openzeppelin has everything already there. We just need to include it.

At the same time we can also add a mint functionality for the contract owner!

19.5.1 Ownable functionality

Let's add the ownable functionality first:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MinimalERC721 is ERC721, Ownable {
    constructor() ERC721("Minimal", "MIN") {}
}
```

If you do the same as before: deploy the token to the developer network and query the functions, then you see it now contains a "owner" function:

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** MinimalERC721.sol - erc721_royalties - Visual Studio Code.
- Left Sidebar:**
 - EXPLORER: Shows the project structure under "OPEN EDITORS".
 - OPEN EDITORS: Shows files like MinimalERC721.sol, 1_initial_migration.js, 2_erc721.js, and truffle-config.js.
 - ERC721_ROYALTIES: Shows files like 2_erc721.js and node_modules\@openzeppelin\contracts\access\Ownable.sol.
 - node_modules\@openzeppelin\contracts\access: Shows files like AccessControl.sol, AccessControlEnumerable.sol, Ownable.sol, and README.adoc.
 - build, finance, governance, interfaces, metatx, proxy, security, token, ERC20, ERC721, extensions, and other sub-folders.
 - OUTLINE and INLINE BOOKMARKS.
- Editor Area:**
 - File: MinimalERC721.sol
 - Language: Solidity (JS)
 - Content:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ow...
```
- Bottom Status Bar:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, Ln 10, Col 2, Spaces: 4, UTF-8.

If you are logged into Opensea and you are the owner of the collection, then Opensea lets you set a secondary sales royalty.

We still cannot mint tokens. Let's add a minting functionality, so that the owner - and only the owner - can mint new tokens!

19.5.2 Mint tokens

To mint tokens we basically need two things:

1. generate a unique token id for each token we mint
2. somehow (potentially) limit who is allowed to mint tokens. In our example only the contract owner will be allowed to mint tokens.

Let's add the following functions to the contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract MinimalERC721 is ERC721, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIdTracker;
    constructor() ERC721("Minimal", "MIN") {}

    function mint(address _to) public onlyOwner {
        super._mint(_to, _tokenIdTracker.current());
        _tokenIdTracker.increment();
    }
}
```

If you open the developer console again (`truffle develop` and then `migrate --reset` and then `let token = await MinimalERC721.deployed()`) then you see the functions which are available. You can mint tokens!

```
await token.mint(accounts[0]);
(await token.balanceOf(accounts[0])).toString()
```

File Edit Selection View Go Run Terminal Help • MinimalERC721.sol - erc721_royalties - Visual Studio Code

EXPLORER OPEN EDITORS 1 UNSAVED

- MinimalERC721.sol contracts
 - JS 2_erc721.js migrations
 - ERC721PresetMinterPauserAutoId.sol nodes
 - JS truffle-config.js
- ERC721_ROYALTIES
 - > build
 - contracts
 - Migrations.sol
 - MinimalERC721.sol
 - migrations
 - JS 1_initial_migration.js
 - JS 2_erc721.js
 - node_modules\@openzeppelin\contracts
 - access
 - AccessControl.sol
 - AccessControlEnumerable.sol
 - Ownable.sol
 - README.adoc
 - > build
 - > finance
 - > governance
 - > interfaces
 - > metatx
 - > proxy
 - > security
 - token
 - > ERC20
 - > ERC721
 - extensions
 - ERC721Burnable.sol
 - ERC721Enumerable.sol
- > OUTLINE
- > IN LINE BOOKMARKS

contracts > MinimalERC721.sol > ...

```

3     tokenId
4 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5 import "@openzeppelin/contracts/access/Ownable.sol";
6 import "@openzeppelin/contracts/utils/Counters.sol";
7
8 contract MinimalERC721 is ERC721, Ownable {
9     using Counters for Counters.Counter;
10    Counters.Counter private _tokenIdTracker;
11
12    constructor() ERC721("Minimal", "MIN") {
13        _tokenIdTracker = Counters.newCounter();
14    }
15    function mint(address _to) public override {
16        super._mint(_to, _tokenIdTracker);
17        _tokenIdTracker.increment();
18    }
19}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

transactionIndex: 0,
transactionHash: '0xa076d474247c2320cbb7f0f1d97',
blockHash: '0x4e78e461f0852dd4e6543049ecbec2c4',
blockNumber: 13,
address: '0x51a40e77820572Ec374dFC7409BbAd4C508',
type: 'mined',
id: 'log_782c0f1f',
event: 'Transfer',
args: [Result]
}
]
}
truffle(develop)> (await token.balanceOf(accounts[0]))
evalmachine.<anonymous>:2
);
} catch(e) { global.ERROR = e; throw e; } }()

```

Uncaught SyntaxError: Unexpected token ';' truffle(develop)> (await token.balanceOf(accounts[0])) '1' truffle(develop)>

Ln 17, Col 2 (570 selected) Spaces: 4 UTF-8

Awesome, you can mint tokens. You are also your own contract owner. But how does Mintable and Rarible work?

Last update: August 19, 2021

19.6 Secondary Royalties on Rarible

Rarible works by using an on-chain OrderMatcher, which internally queries the token for secondary sales royalties. This is written, in code, over here:

<https://github.com/rarible/protocol-contracts/blob/master/royalties/contracts/RoyaltiesV2.sol>

We could install the Rarible Royalties Package, but at the time of writing, the package is written for Solidity < 0.8, which makes it incompatible.

Instead, we will mimic the interface, so that the Rarible DAO gets the right information!

Create a new folder structure "@rarible/royalties/contracts" in the contracts folder. Then add the following files:

IRoyaltiesProvider.sol LibPart.sol LibRoyaltiesV2.sol RoyaltiesV2.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./LibPart.sol";
interface IRoyaltiesProvider {
    function getRoyalties(address token, uint tokenId) external returns (LibPart.Part[] memory);
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

library LibPart {
    bytes32 public constant TYPE_HASH = keccak256("Part(address account,uint96 value)");

    struct Part {
        address payable account;
        uint96 value;
    }

    function hash(Part memory part) internal pure returns (bytes32) {
        return keccak256(abi.encode(TYPE_HASH, part.account, part.value));
    }
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

library LibRoyaltiesV2 {
    /*
     * bytes4(keccak256('getRoyalties(LibAsset.AssetType)')) == 0x44c74bcc
     */
    bytes4 constant _INTERFACE_ID_ROYALTIES = 0x44c74bcc;
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./LibPart.sol";
interface RoyaltiesV2 {
    event RoyaltiesSet(uint256 tokenId, LibPart.Part[] royalties);

    function getRaribleV2Royalties(uint256 id) external view returns (LibPart.Part[] memory);
}
```

Then add another folder inside called "impl" and add the following files in there:

impl/AbstractRoyalties.sol impl/RoyaltiesV2Impl.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../LibPart.sol";

abstract contract AbstractRoyalties {
    mapping (uint256 => LibPart.Part[]) public royalties;

    function _saveRoyalties(uint256 _id, LibPart.Part[] memory _royalties) internal {
        for (uint i = 0; i < _royalties.length; i++) {
            require(_royalties[i].account != address(0x0), "Recipient should be present");
            require(_royalties[i].value != 0, "Royalty value should be positive");
            royalties[_id].push(_royalties[i]);
        }
        _onRoyaltiesSet(_id, _royalties);
    }

    function _updateAccount(uint256 _id, address _from, address _to) internal {
        uint length = royalties[_id].length;
        for(uint i = 0; i < length; i++) {
            if (royalties[_id][i].account == _from) {
                royalties[_id][i].account = payable(address(uint160(_to)));
            }
        }
    }

    function _onRoyaltiesSet(uint256 _id, LibPart.Part[] memory _royalties) virtual internal;
}

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./AbstractRoyalties.sol";
import "../RoyaltiesV2.sol";

contract RoyaltiesV2Impl is AbstractRoyalties, RoyaltiesV2 {
    function getRaribleV2Royalties(uint256 id) override external view returns (LibPart.Part[] memory) {
        return royalties[id];
    }

    function _onRoyaltiesSet(uint256 _id, LibPart.Part[] memory _royalties) override internal {
        emit RoyaltiesSet(_id, _royalties);
    }
}
```

If you add everything into the project, then it should look something like this:

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Status Bar:** 19.6 Secondary Royalties on Rarible, Ln 17, Col 2, Spaces: 4, UTF-8.
- Editor:** Shows the Solidity file `MinimalERC721.sol` with the following code snippet:


```

3 import "@openzeppelin/contracts/token/ERC721/ERC721PresetMinterPauserAutoId.sol";
4 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Royalties.sol";
6 import "@openzeppelin/contracts/utils/Counters.sol";
7
8 contract MinimalERC721 is ERC721 {
9     using Counters for Counters.Counter;
10    Counters.Counter private _tokenIdTracker;
11
12    constructor() ERC721("MinimalERC721");
13
14    function mint(address _to, uint256 _tokenId) public {
15        super._mint(_to, _tokenId);
16        _tokenIdTracker.increment();
17    }
      
```
- Explorer:** Shows the project structure:
 - OPEN EDITORS:** 1 UNSAVED (MinimalERC721.sol, 2_erc721.js, ERC721PresetMinterPauserAutoId.sol, truffle-config.js)
 - ERC721_ROYALTIES:**
 - build
 - contracts
 - @rarible\royalties\contracts
 - impl
 - AbstractRoyalties.sol
 - RoyaltiesV2Impl.sol
 - IRoyaltiesProvider.sol
 - LibPart.sol
 - LibRoyaltiesV2.sol
 - RoyaltiesV2.sol
 - Migrations.sol
 - MinimalERC721.sol
 - migrations
 - 1_initial_migration.js
 - 2_erc721.js
 - node_modules\@openzeppelin\contracts
 - access
 - AccessControl.sol
 - AccessControlEnumerable.sol
 - Ownable.sol
 - README.adoc
 - build
 - finance
 - OUTLINE
 - INLINE BOOKMARKS

- Bottom Status Bar:** Try the new cross-platform PowerShell https://www.powershell.com, Loading personal and system profiles took 86ms, erc721_royalties>

It is the same structure as in their official [Github repository](#). But their contracts are, at the time of writing, solidity <0.8. So, it mimics the official structure with the hope that the contracts are updated to sol >=0.8 at some point, but having the same structure as before.

If you have a look at the [RoyaltiesRegistry](#) then you see that on every token transfer, the Rarible Protocol will "ask" the token, if there are any royalties to be paid out. That is done by calling `getRaribleV2Royalties`. So, all our contract needs to do is expose this function. And it does it if we are extending from RoyaltiesV2Impl. Let's improve our token:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./@rarible/royalties/contracts/impl/RoyaltiesV2Impl.sol";
import "./@rarible/royalties/contracts/LibPart.sol";
import "./@rarible/royalties/contracts/LibRoyaltiesV2.sol";

contract MinimalERC721 is ERC721, Ownable, RoyaltiesV2Impl {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIdTracker;
    constructor() ERC721("Minimal", "MIN") {}

    function mint(address _to) public onlyOwner {
        super._mint(_to, _tokenIdTracker.current());
        _tokenIdTracker.increment();
    }

    function setRoyalties(uint _tokenId, address payable _royaltiesRecipientAddress, uint96 _percentageBasisPoints) public onlyOwner {
        LibPart.Part[] memory _royalties = new LibPart.Part[](1);
        _royalties[0].value = _percentageBasisPoints;
        _royalties[0].account = _royaltiesRecipientAddress;
        _saveRoyalties(_tokenId, _royalties);
    }

    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC721) returns (bool) {
        if(interfaceId == LibRoyaltiesV2._INTERFACE_ID_ROYALTIES) {
            return true;
        }
        return super.supportsInterface(interfaceId);
    }
}
```

If you open this again on the truffle developer console, you can test it:

```
truffle develop
migrate
let token = await MinimalERC721.deployed()
token.mint(accounts[0]);
token.setRoyalties(0, accounts[0], 1000)
```

Info

The Percentage is given in percentage basis points. That means, 2.5% is 250

If you now execute

```
token.getRaribleV2Royalties(0);
```

It should output that account[0] gets 10% or 1000 basis points:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
args: [Result]
}
]
}
truffle(develop)> token.get
token.getApproved          token.getPastEvents      token.getRaribleV2Royalties

truffle(develop)> token.getRaribleV2Royalties(0);
[
  [
    {
      '0x06076f85FF865bb89Fef4201f83D11D1cE9e6d8E',
      '1000',
      account: '0x06076f85FF865bb89Fef4201f83D11D1cE9e6d8E',
      value: '1000'
    }
  ]
truffle(develop)> █
```

Perfect, but what about Mintable? Let's do that next!

Last update: August 19, 2021

19.7 Secondary Royalties on Mintable (ERC2981)

Mintable builds upon ERC2981, the [NFT Royalties Standard](#).

Luckily we just have to make minor modifications to support this.

First, it needs to support the interface `bytes4 private constant _INTERFACE_ID_ERC2981 = 0x2a55205a;`

Secondly, it needs to report back royalties when queried

```
function royaltyInfo(
    uint256 _tokenId,
    uint256 _salePrice
) external view returns (
    address receiver,
    uint256 royaltyAmount
);
```

All we need to do is integrate this function. One option would be to multiply the royalties for the token with the sales price / 100.

Let's integrate it into our Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./@rareible/royalties/contracts/impl/RoyaltiesV2Impl.sol";
import "./@rareible/royalties/contracts/LibPart.sol";
import "./@rareible/royalties/contracts/LibRoyaltiesV2.sol";

contract MinimalERC721 is ERC721, Ownable, RoyaltiesV2Impl {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIdTracker;

    bytes4 private constant _INTERFACE_ID_ERC2981 = 0x2a55205a;
    constructor() ERC721("Minimal", "MIN") {}

    function mint(address _to) public onlyOwner {
        super._mint(_to, _tokenIdTracker.current());
        _tokenIdTracker.increment();
    }

    function setRoyalties(uint _tokenId, address payable _royaltiesRecipientAddress, uint96 _percentageBasisPoints) public onlyOwner {
        LibPart.Part[] memory _royalties = new LibPart.Part[](1);
        _royalties[0].value = _percentageBasisPoints;
        _royalties[0].account = _royaltiesRecipientAddress;
        _saveRoyalties(_tokenId, _royalties);
    }

    function royaltyInfo(uint256 _tokenId, uint256 _salePrice) external view returns (address receiver, uint256 royaltyAmount) {
        LibPart.Part[] memory _royalties = royalties[_tokenId];
        if(_royalties.length > 0) {
            return (_royalties[0].account, (_salePrice * _royalties[0].value)/10000);
        }
        return (address(0), 0);
    }

    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC721) returns (bool) {
        if(interfaceId == LibRoyaltiesV2._INTERFACE_ID_ROYALTIES) {
            return true;
        }
        if(interfaceId == _INTERFACE_ID_ERC2981) {
            return true;
        }
        return super.supportsInterface(interfaceId);
    }
}
```

If you test it again on the truffle developer console, you will get this result:

File Edit Selection View Go Run Terminal Help MinimalERC721.sol - erc721_royalties - Visual Studio Code

EXPLORER OPEN EDITORS

- MinimalERC721.sol contracts
 - LibPart.sol contracts\@rarible\royalties\contracts
 - RoyaltiesV2Impl.sol contracts\@rarible\royalties\contr...
- JS 2_erc721.js migrations
- JS truffle-config.js

ERC721_ROYALTIES

- > build
- contracts
 - @rarible\royalties\contracts
 - impl
 - AbstractRoyalties.sol
 - RoyaltiesV2Impl.sol
 - IRoyaltiesProvider.sol
 - LibPart.sol
 - LibRoyaltiesV2.sol
 - RoyaltiesV2.sol
 - Migrations.sol
- MinimalERC721.sol

migrations

- JS 1_initial_migration.js
- JS 2_erc721.js

node_modules\@openzeppelin\contracts

- > access
- > build
- > finance
- > governance
- > interfaces
- > metatx

OUTLINE

INLINE BOOKMARKS

MinimalERC721.sol x LibPart.sol

contracts > MinimalERC721.sol > MinimalERC721.sol

```
24 > tokenU
25 _royalties[_tokenId].value = _per
26 royalties[0].account =
27 saveRoyalties(_tokenId,
28 }
29
ftrace | funcSig
30 function royaltyInfo(uint256 _id)
31     LibPart.Part[] memory _royalties;
32     return (_royalties[0].account);
33 }
34
35
ftrace | funcSig
36 function supportsInterface(bytes4 interfaceId)
37     if(interfaceId == LibRoyaltiesV2Interface.id)
38         return true;
39     return false;
40 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
truffle(develop)> token.royaltyInfo(0, 5000)
Result {
  '0': '0x06076f85FF865bb89Fef4201f83D11D1cE',
  '1': BN {
    negative: 0,
    words: [ 500, <1 empty item> ],
    length: 1,
    red: null
  },
  receiver: '0x06076f85FF865bb89Fef4201f83D11D1cE',
  royaltyAmount: BN {
    negative: 0,
    words: [ 500, <1 empty item> ],
    length: 1,
    red: null
  }
}
```

Ln 42, Col 10 Spaces: 4 UTF-8

Unable To Test

Unfortunately there is no test-platform for mintable. So, the implementation is on a "best guess" basis at the moment.

But there is one component still missing - that's the token URI. Let's add that before concluding this tutorial!

Last update: August 19, 2021

19.8 Token URI

To make this a complete example, the last thing that's missing is the TokenURI.

To make it work, our Smart Contract needs to implement the baseURI function:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./rarible/royalties/contracts/impl/RoyaltiesV2Impl.sol";
import "./rarible/royalties/contracts/LibPart.sol";
import "./rarible/royalties/contracts/LibRoyaltiesV2.sol";

contract MinimalERC721 is ERC721, Ownable, RoyaltiesV2Impl {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIdTracker;

    bytes4 private constant _INTERFACE_ID_ERC2981 = 0x2a55205a;
    constructor() ERC721("Minimal", "MIN") {}

    function mint(address _to) public onlyOwner {
        super._mint(_to, _tokenIdTracker.current());
        _tokenIdTracker.increment();
    }

    function setRoyalties(uint _tokenId, address payable _royaltiesRecipientAddress, uint96 _percentageBasisPoints) public onlyOwner {
        LibPart.Part[] memory _royalties = new LibPart.Part[](1);
        _royalties[0].value = _percentageBasisPoints;
        _royalties[0].account = _royaltiesRecipientAddress;
        _saveRoyalties(_tokenId, _royalties);
    }

    function royaltyInfo(uint256 _tokenId, uint256 _salePrice) external view returns (address receiver, uint256 royaltyAmount) {
        LibPart.Part[] memory _royalties = royalties[_tokenId];
        if(_royalties.length > 0) {
            return (_royalties[0].account, (_salePrice * _royalties[0].value)/10000);
        }
        return (address(0), 0);
    }

    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC721) returns (bool) {
        if(interfaceId == LibRoyaltiesV2._INTERFACE_ID_ROYALTIES) {
            return true;
        }
        if(interfaceId == _INTERFACE_ID_ERC2981) {
            return true;
        }
        return super.supportsInterface(interfaceId);
    }

    function _baseURI() internal view virtual override returns (string memory) {
        return "https://mydomain/metadata/";
    }
}
```

If you test it in the developer console, it will output you a url to a metadata file. Adjust it to your own server!

```
truffle(develop)> token.tokenURI(0)
'https://mydomain/metadata/0'
truffle(develop)>
```

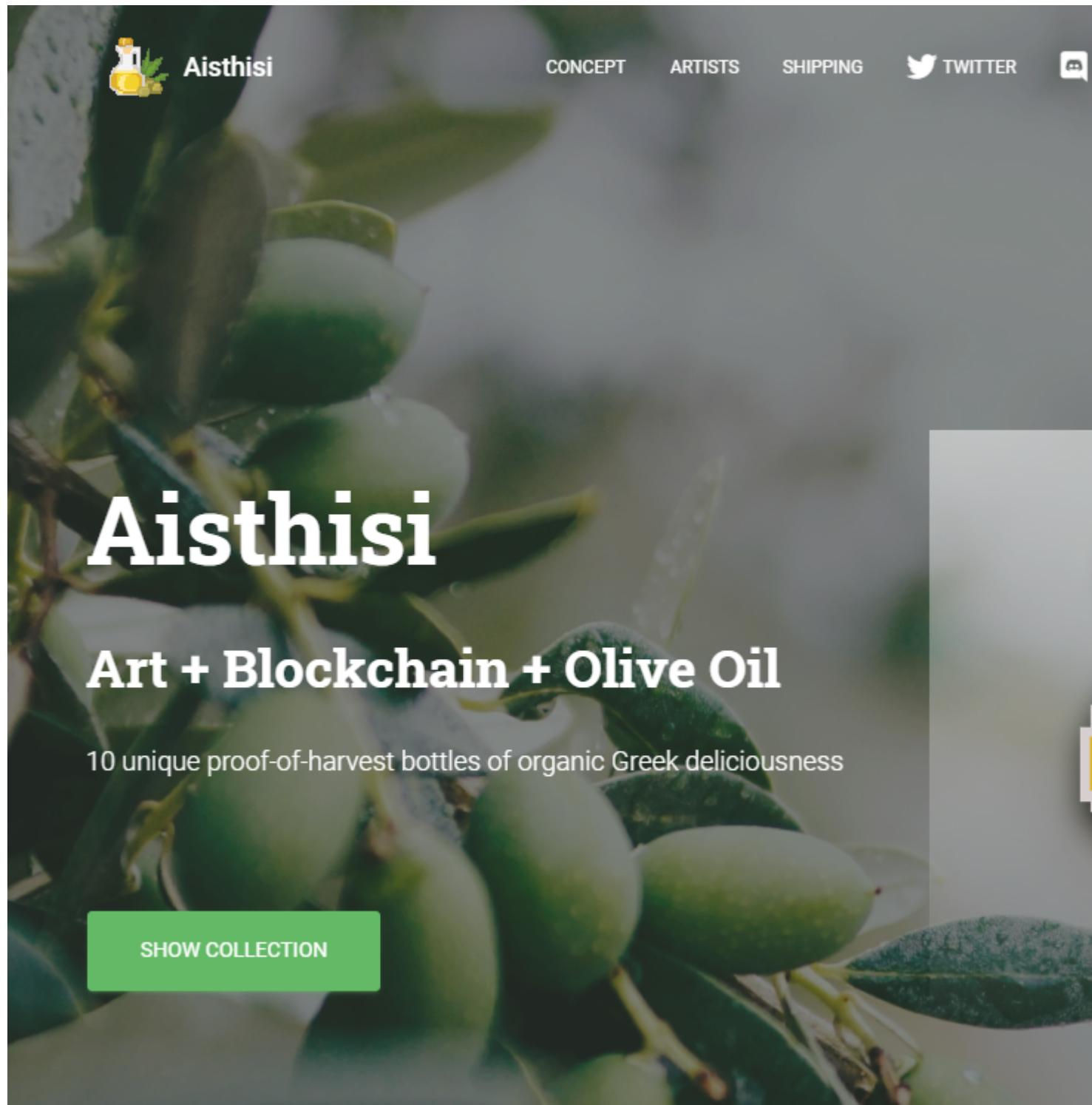
Last update: August 19, 2021

19.9 Final Words

It is not easy to get all the platforms secondary sales royalties implemented, but it is do-able.

It was one of the biggest hurdles we had with our own custom LockableToken for Project Aisthisi.

If you are interested in our Project or our Digital/Physical NFTs then follow us!



Last update: August 19, 2021