

Rike Exner

Intro to Web3 frontend development

#WomenWhoCode





Hi!

I'm **Rike Exner.**

Frontend developer @IAV, Berlin

Twitter: @rikecodes

Discord: rikecodes#3756

Web: rike.dev

LinkedIn: <https://www.linkedin.com/in/rike-exner>



Vy Tran 
@tweevtran



If you wondering what's Tech Twitter been up to lately.



2:21 PM · 28 Nov, 2021

39 replies 494 shares 2.5K likes

What is Web3 frontend development?

Software development

Web Development

Web3/Blockchain development

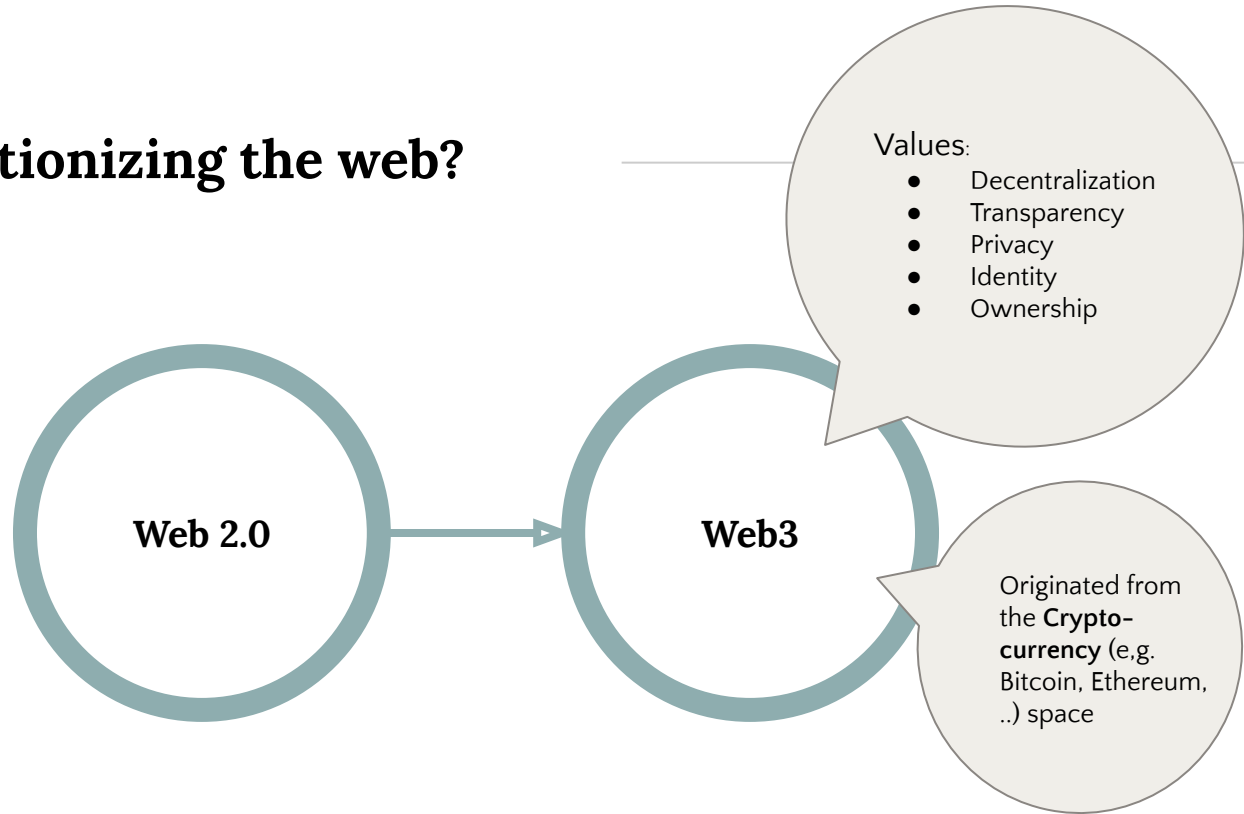
Frontend

Backend/Core





Revolutionizing the web?



what is a transaction (txn)? what is a wallet?

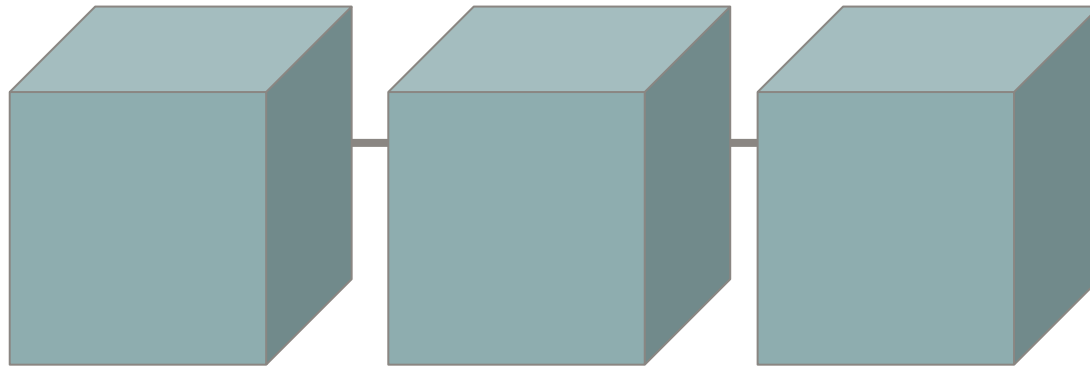


<https://qph.cf2.quoracdn.net/main-qimg-722f07a6e508377df8e286f7ee199192-lq>

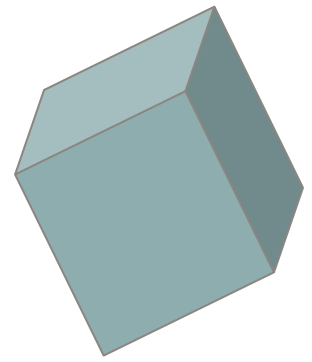
What it looks like in action..



“

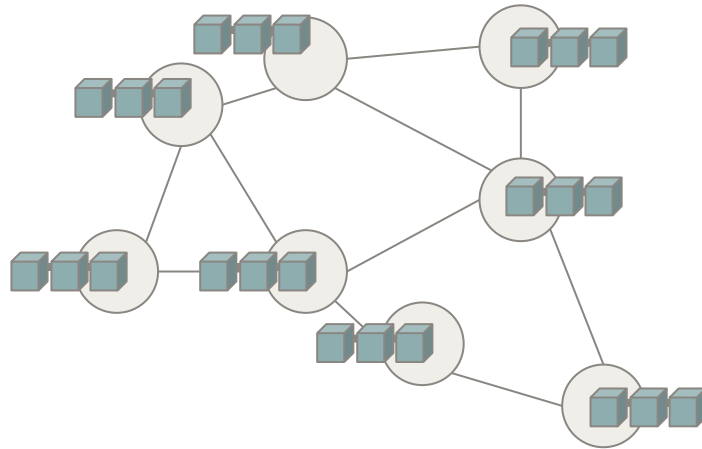


"hello block!"



a chronological/block-based **database**
full of **transactions** → the blockchain
→ the **"ledger"**

what does decentralized / distributed mean?



*all **nodes** of a Blockchain platform carry the same information
about the **ledger***





What's new with EVM and EVM-inspired Blockchains?

Bitcoin was a mere cryptocurrency transaction “ledger”. With Ethereum’s Virtual Machine, Smart Contract development was made possible.

Smart contracts are little programs, stored on the blockchain, that are able execute on any specified condition and will carry out transactions automatically.



.. behind every Ethereum address there's an account..

EOA (externally owned Account)
(e.g. "0x824A9..")

- managed by "real" people and their wallets

Contract Account
(e.g. "0x08D3..")

- managed by smart contract, a developer has coded before



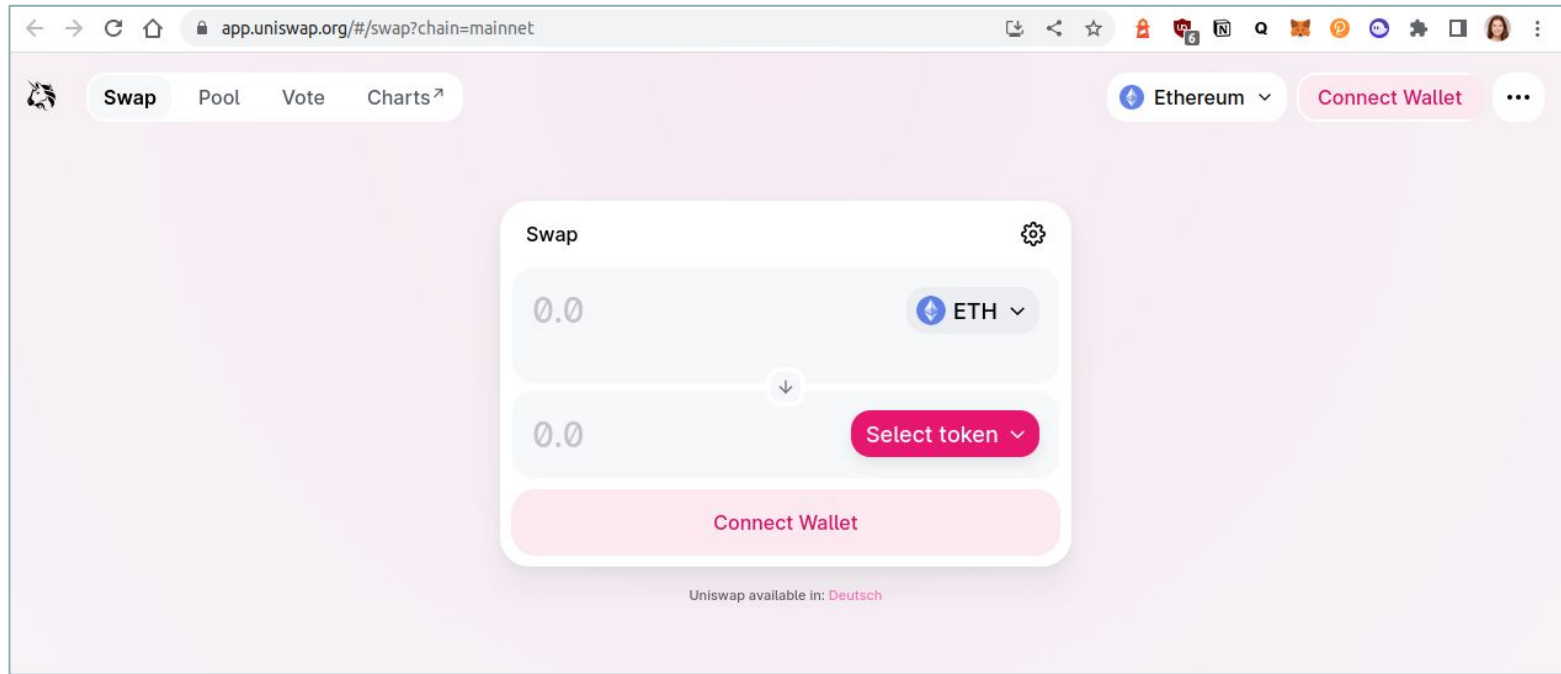
what are **decentralized Apps (dApps)?**

- A decentralized application (dApp) is an application built on a decentralized network that combines a smart contract and a frontend user interface.
- the foundation of web3 and all its use cases

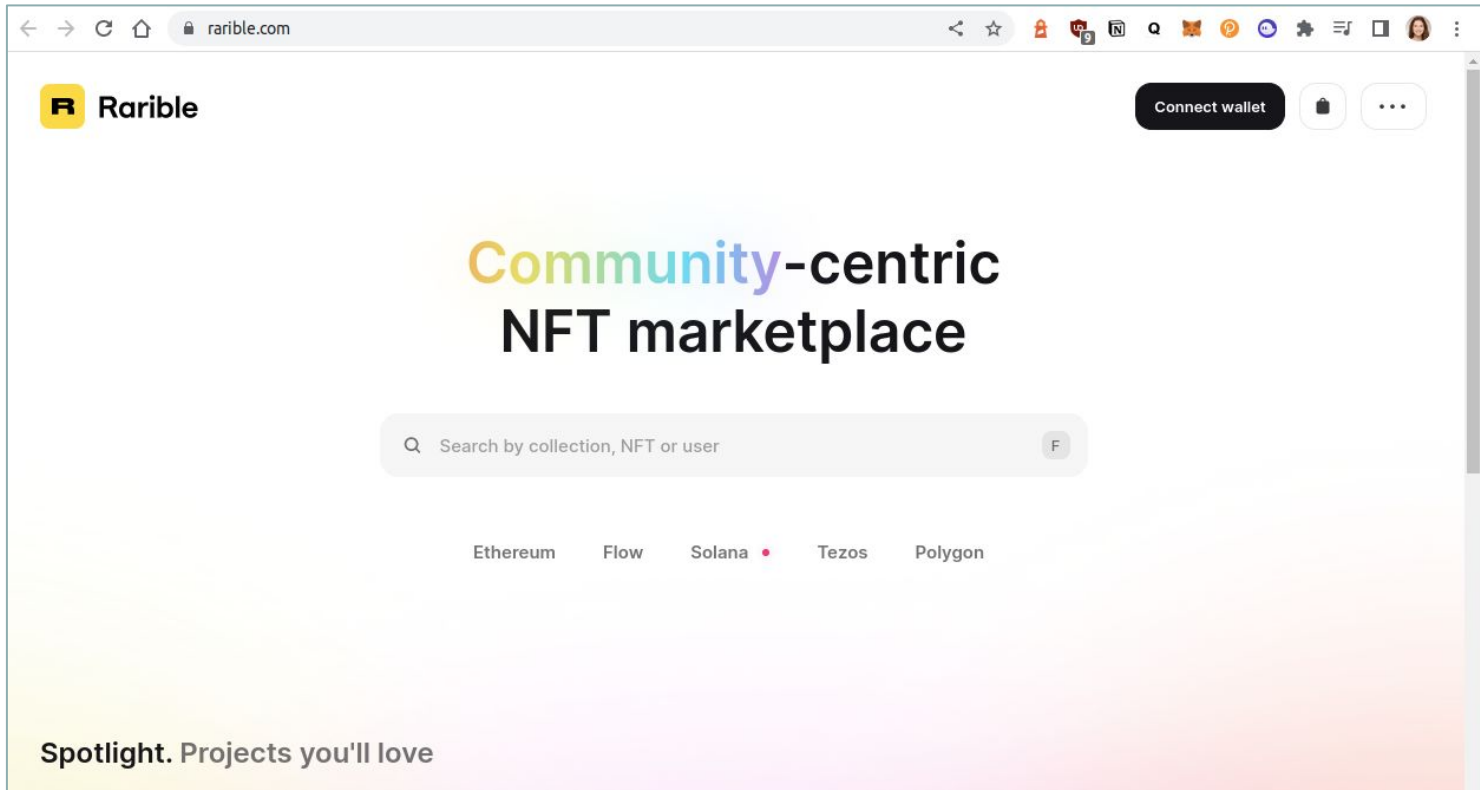


Web3 Use Cases

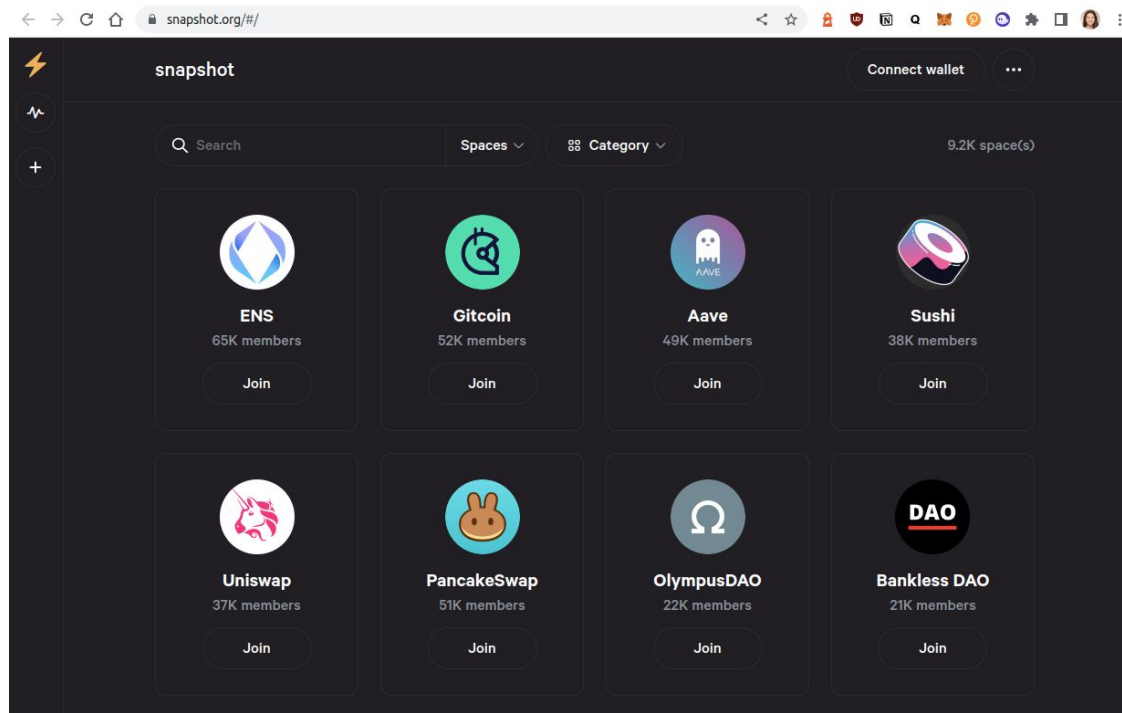
- ◉ Decentralized Finance (DEFI)
- ◉ Tokenization, NFTs
- ◉ Decentralized Autonomous Organizations (DAOs)
- ◉ Blockchain Games
- ◉ Metaverse



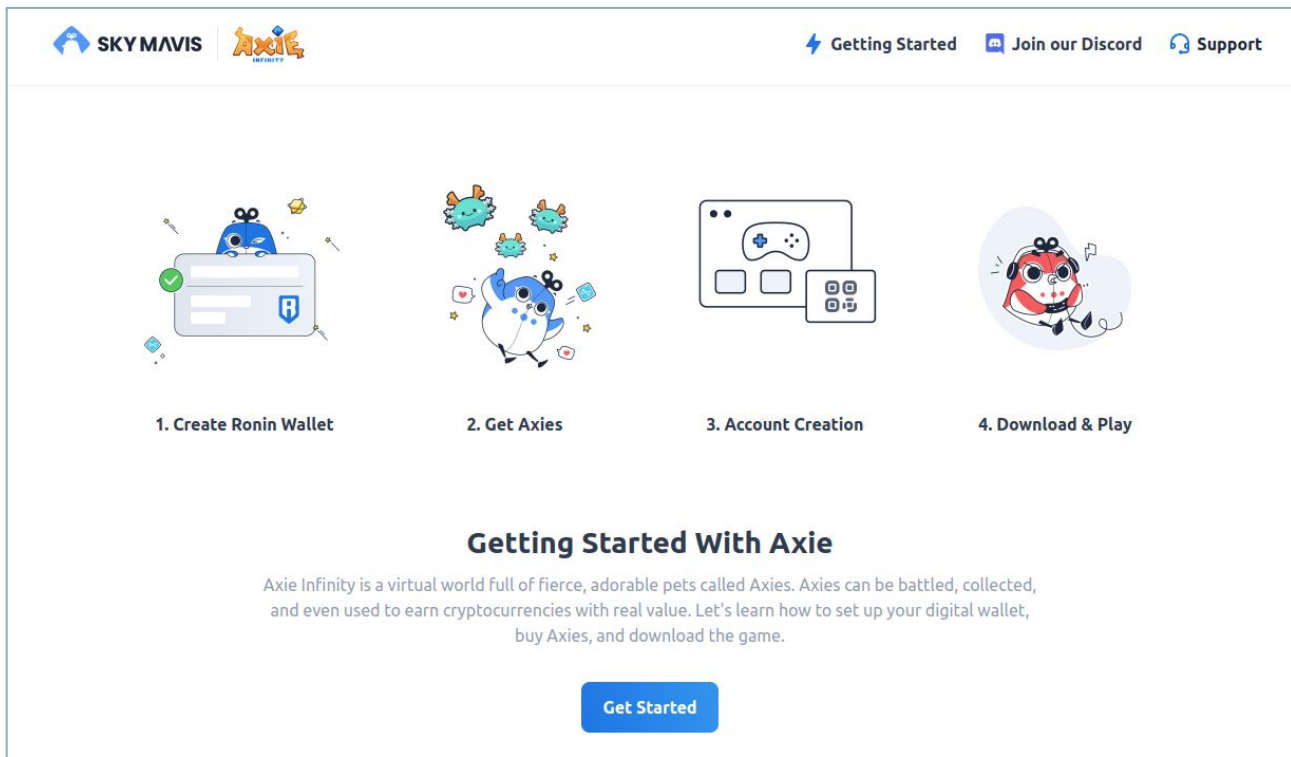
DEFI dApp “Uniswap”



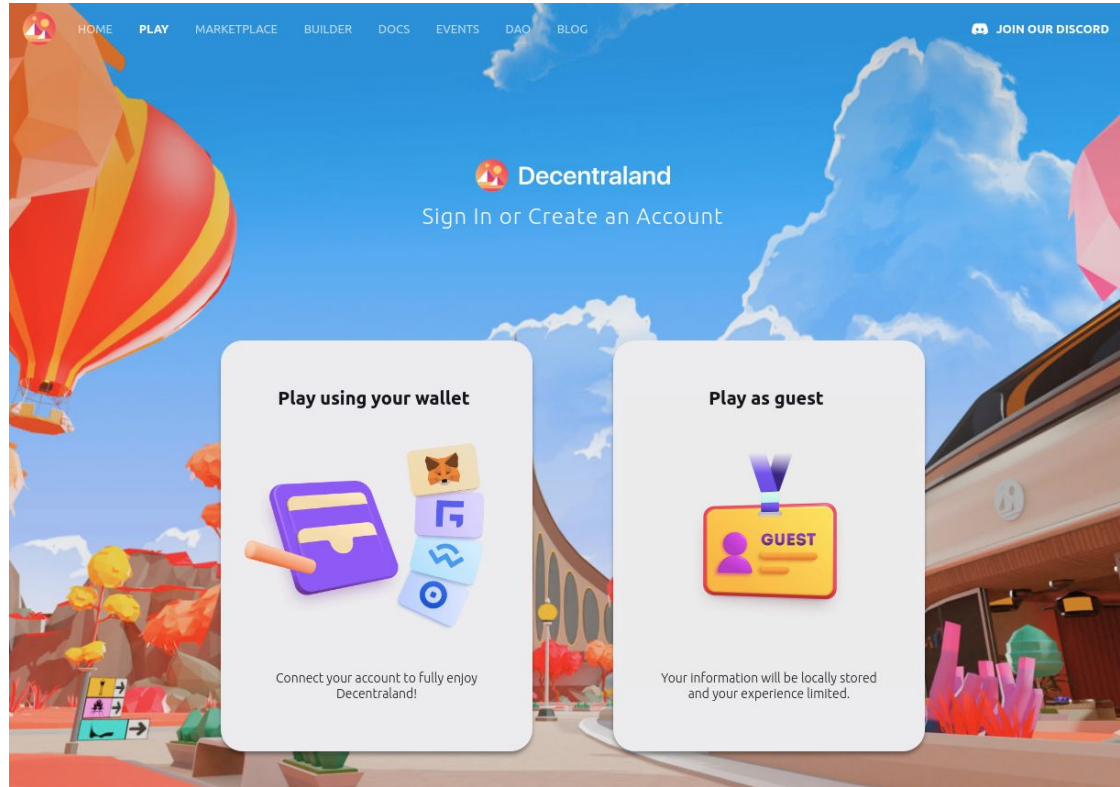
NFT dApp “Rarible”



DAO Voting dApp “Snapshot”



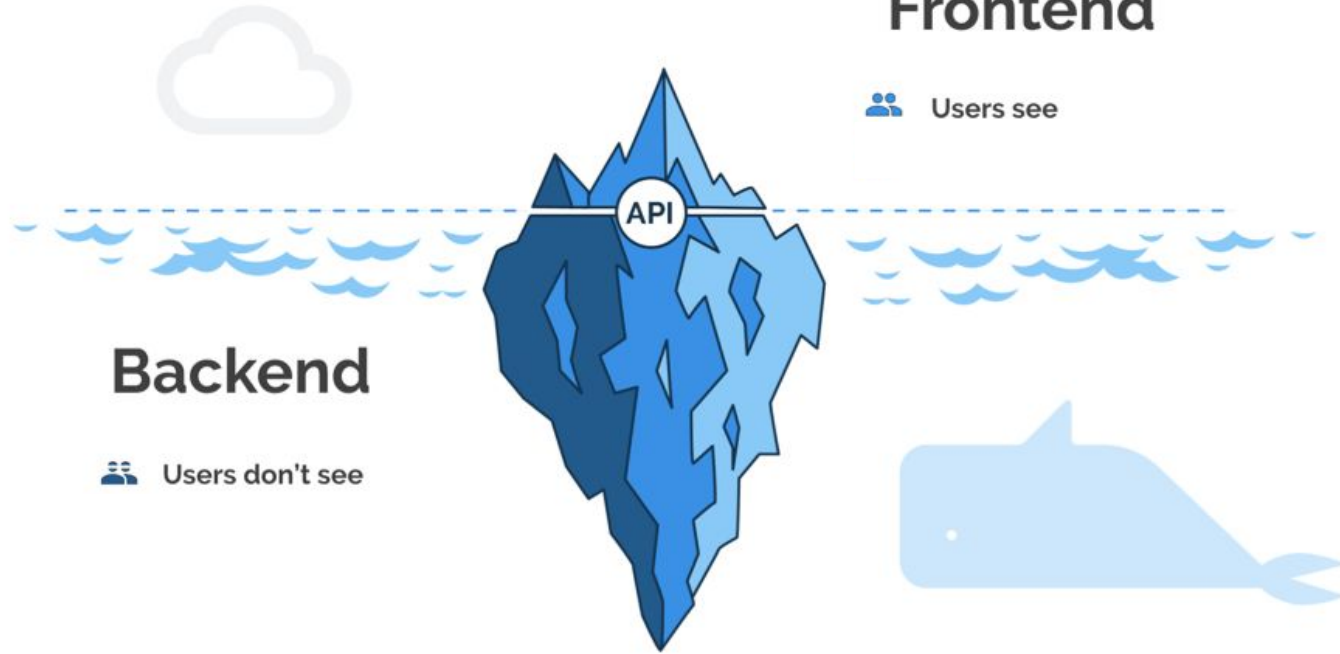
Blockchain Game dApp “Axie Infinity” Onboarding screen



“Decentraland” Metaverse dApp Onboarding screen

 Users see

Users see



Backend

 Users don't see

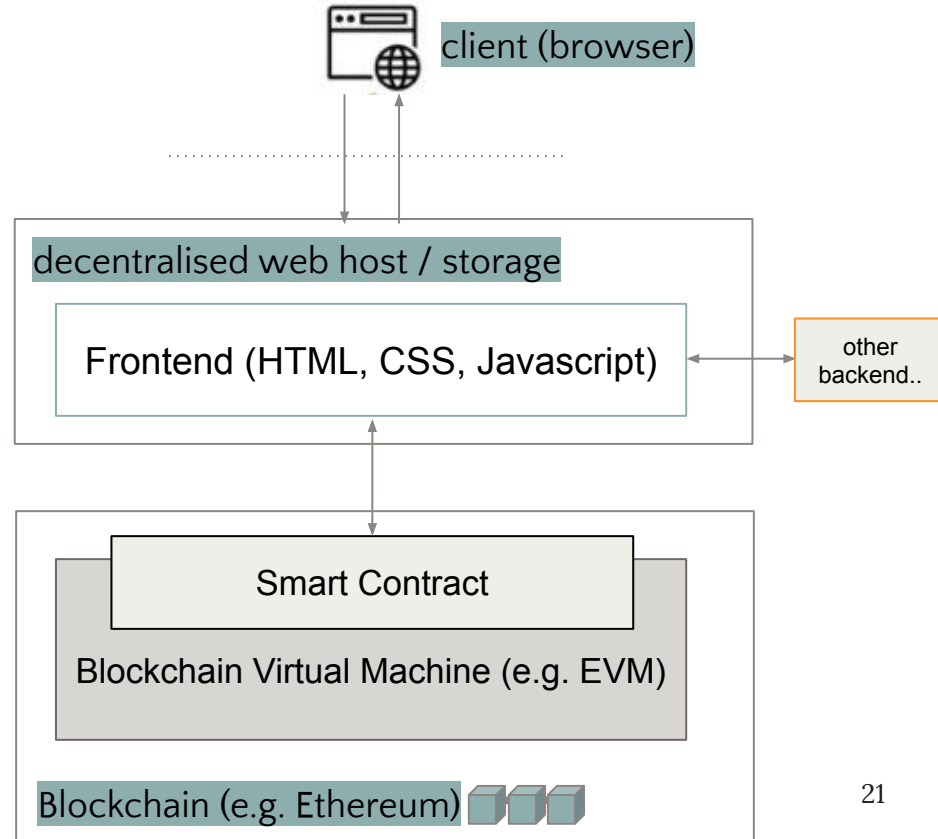
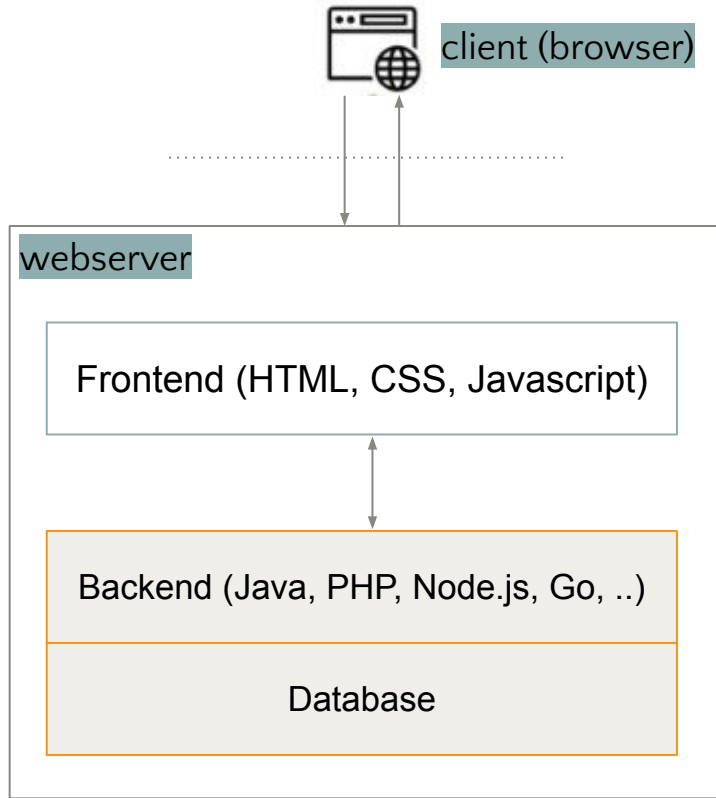
<https://blog.back4app.com/wp-content/uploads/2020/00/Firebase-vs-Back4app-2-2.png>

The **frontend** of applications, websites, dApps, etc., constitutes everything a user can see and interact with. We often refer to it as the “client-side”. This includes colors, styles, images, animations, graphs, tables, buttons, text, menus, and much more.

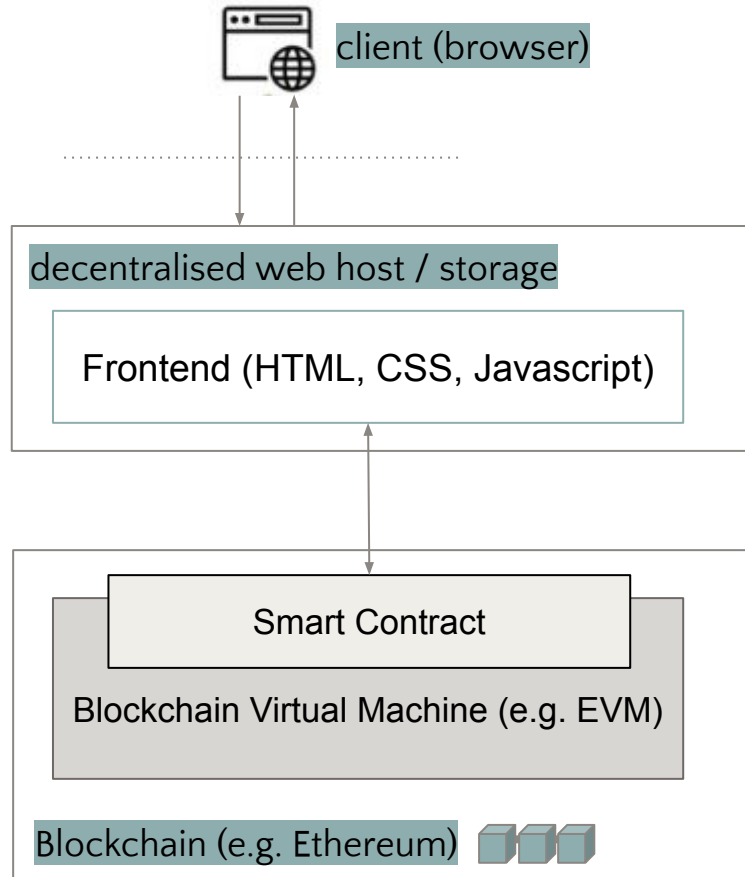


“

Client-Server-Model traditional web app vs. dApp



The dApp tech stack

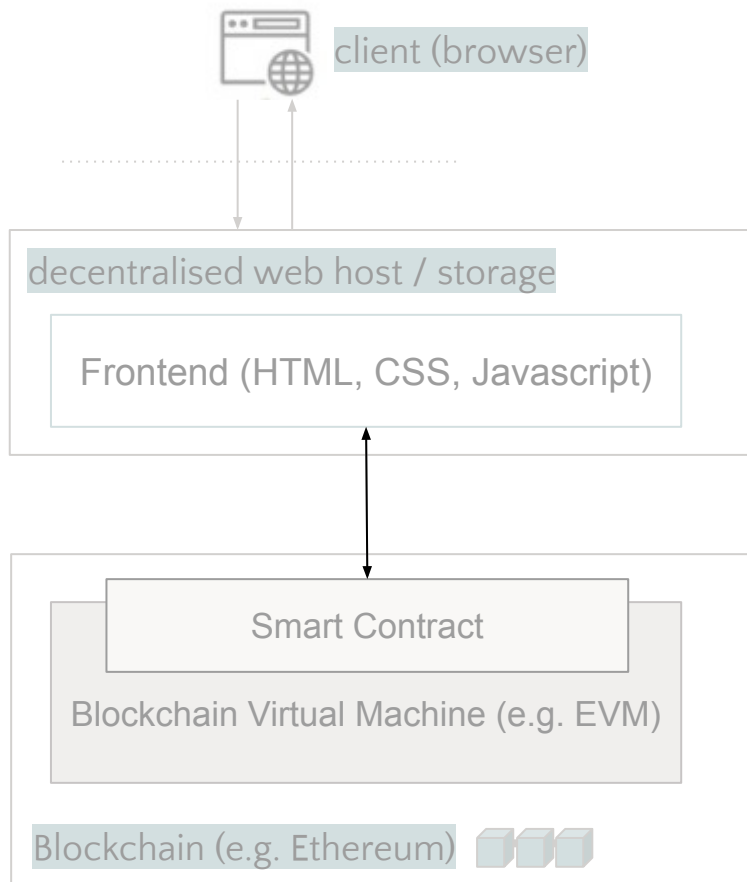


HTML, CSS & Javascript (*with a little bit of extra super powers..*)



depending on your chosen Blockchain platform; Solidity, Rust, AssemblyScript, Cadence, ..

The dApp tech stack



for communication with smart contracts you need to connect to **one of the blockchain nodes** by

1. setting up your own node
2. use third-party node provider

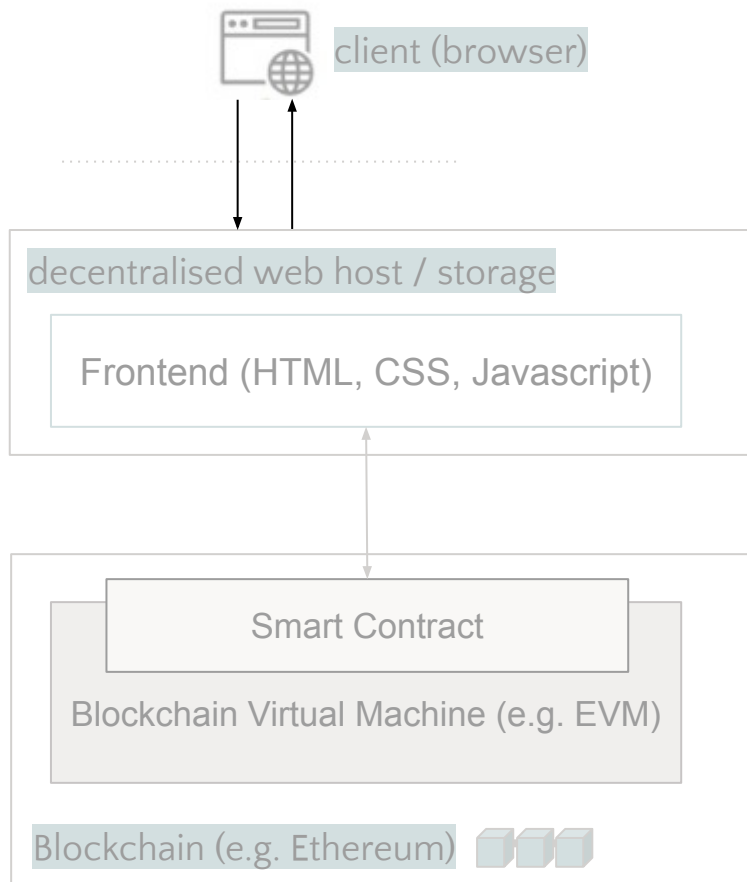
 alchemy

 Ankr



INFURA

The dApp tech stack



when transactions are involved, a “**signer**” makes sure to identify the one executing the transaction → usually realized by a **browser wallet**, like Metamask





Web 2.0 vs. Web3 frontends

What stays the same:

- all frontends are just websites that are (in the end) written in HTML, CSS and Javascript and consumed by a webbrowser

What's different:

- use cases
- authentication handling
- interaction with backend



Web3 frontend Best Practices

- Web3 frontend functions could be implemented with plain Javascript or any Javascript frontend framework (React, Vue.js, Angular, ..) with the help of blockchain-specific libraries, like *web3.js* and *ethers.js*
- *but* **React** and its superset **Next.js** have gained a lot of popularity due to a wide ecosystem around Web3 specific libraries



Web3 Javascript libraries

EVM-based Blockchains

- web3.js
- Ethers.js
- web3modal
- .. and a whole lot more

Solana

- @solana/web3.js

NEAR

- near-api-js

Flow

- FCL.js



Web3 React libraries

EVM-based Blockchains

- rainbowkit
- Wagmi
- Web3-react
- web3-ui
- etc.

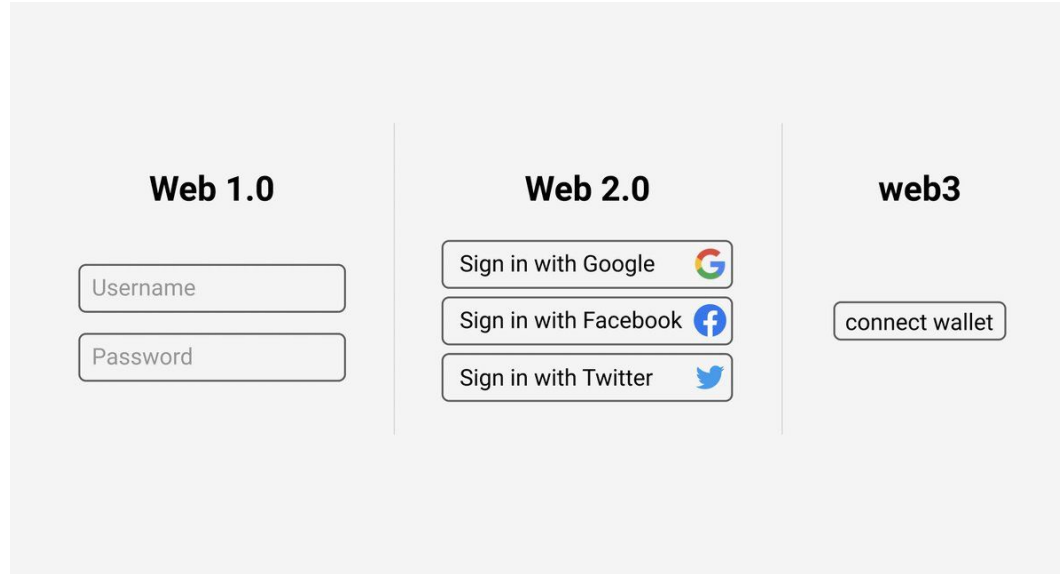
Solana

- wallet-adapter

Authentication handling



“



<https://images.app.goo.gl/CBcEbqESk2M3LFg27>

Authentication handling **Web / Browser Wallets**

- act as entrance point for users and their browsers to dApps
- commonly used wallet software: Metamask (EVM), WalletConnect, Phantom (Solana), NEAR wallet, etc.



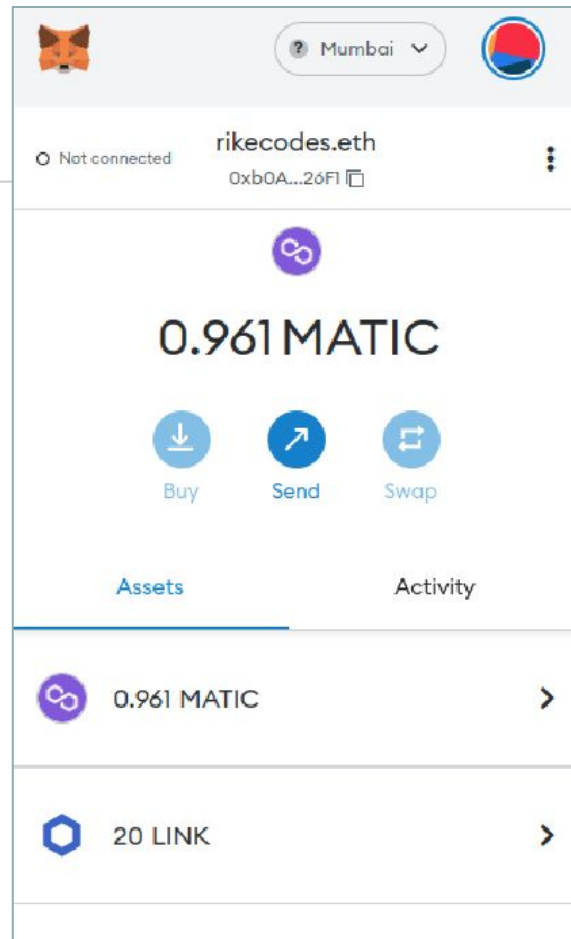
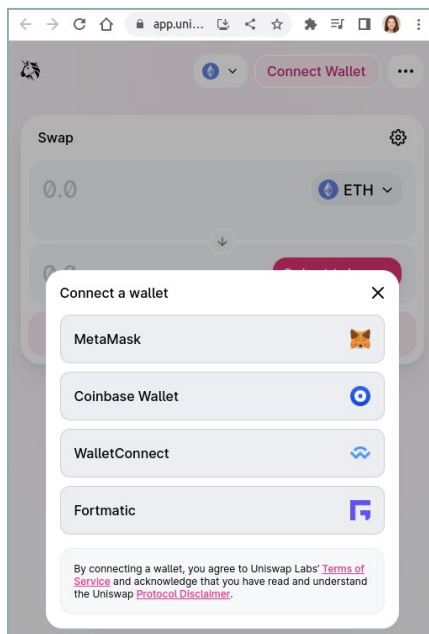
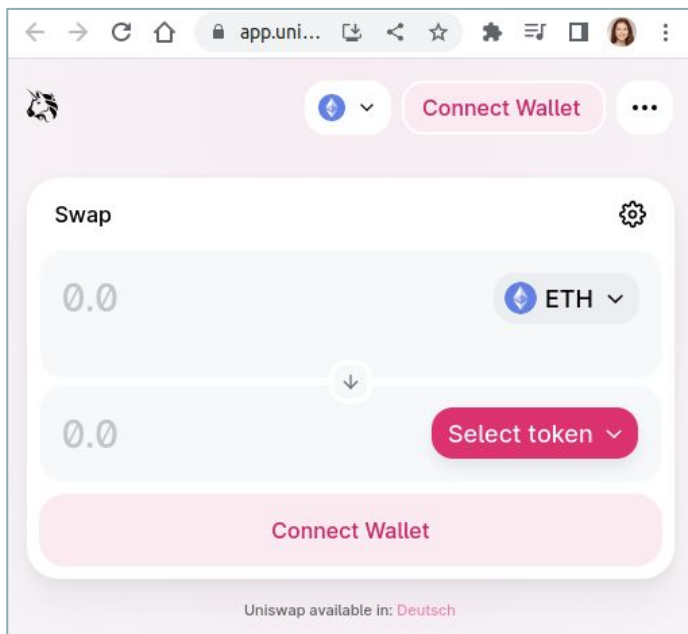
What it looks like in action..



“

Authentication handling

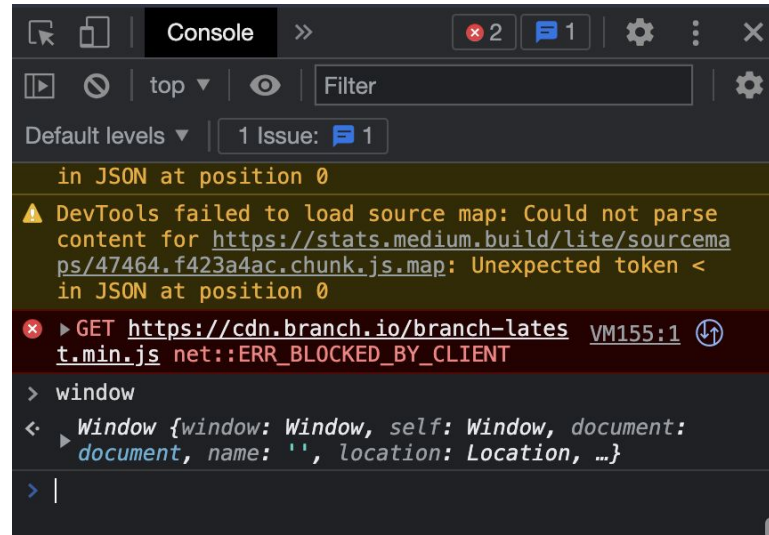
What the user sees





Authentication handling **window.ethereum** Object

- every browser exposes the **window** object by default
- browser add-ons/plugins like web wallets do automatically “inject” themselves into your browser window object, e.g. Metamask creates ethereum object





Authentication handling

How to implement (1)



```
import { useWallet } from '@web3-ui/core';

function App() {
  const { connectWallet, connection, connected } = useWallet();

  if (!connected) {
    return (
      <button onClick={connectWallet}>Connect wallet</button>
    )
  }

  return (
    <p>{connection.userAddress}</p>
  )
}
```




Authentication handling

How to implement (2)



```
import { ConnectButton } from "@rainbow-me/rainbowkit";

function App() {
  return (
    <ConnectButton />
  )
}
```


Interaction with the backend



“



Interaction with the backend

Blockchain nodes & JSON-RPC

- after connecting to a Blockchain **node / provider**, dApps may read data and send transactions to the Blockchain network
- every node implements a **JSON-RPC specification** (a light-weight remote procedure call (RPC) protocol) based on JSON data format, that a client must follow



// example JSON-RPC request

```
{"jsonrpc": "2.0", "method": "getBlockHeight", "id": 1}
```

// example JSON-RPC response

```
{"jsonrpc": "2.0", "result": 143129, "id": 1}
```




Interaction with the backend

Establish provider connection

- libraries like web3.js and ethers.js make it easy to connect to any **node / provide**
- depending on your setup using a **third-party RPC “node-as-a-service” provider** from e.g. Alchemy, Infura, Etherscan, etc. can be beneficial



```
var ether = require("ethers")

let provider;
// init custom provider by node provider service Alchemy
provider = new ethers.providers.AlchemyProvider(null, "API_KEY");
// or: use provider given by browser extension, e.g. Metamask
provider = new ethers.providers.Web3Provider(window.ethereum, "any");
// or: use default provider fallback
provider = new ethers.providers.getDefaultProvider();
```


Interaction with the backend

Reading data from blockchain



“



Reading data from blockchain

Call a provider method



```
var ether = require("ethers");  
  
const provider; // see provider initialization  
const rikeBalance = await provider.getBalance("0x03D28Df4b4c3a4bb1eA5D0a518E4D045172a6559");
```




Reading data from blockchain

Handling Big numbers

- libraries like web3.js and ethers.js make it easy to handle BigNumbers, a number format popular around cryptocurrencies



```
var ether = require("ethers")  
ethers.BigNumber(bigNumberComingFromSmartContract).toNumber();
```




Reading data from blockchain **TheGraph**

- TheGraph protocol makes querying data, of any kind, smart contracts super easy, almost in the way we know it from querying languages such as SQL and GraphQL



Interaction with the backend

**Editing data & making
transactions**



“

What it looks like in action..



“




Editing data & make transactions

Initialize a signer in your frontend

- a signer is an abstraction of a Blockchain account (managed usually by a browser wallet like Metamask), which can be used to sign messages and execute transactions



```
var ethers = require('ethers')  
  
// use provider given by browser extension, e.g. Metamask  
const provider = new ethers.providers.Web3Provider(window.ethereum, "any");  
const signer = provider.getSigner();
```

Editing data & making transactions

Send transaction (e.g. ETH to any other account)

```
var ethers = require('ethers')

const signer; // see signer initialisation
const tx = {
  from: "0x611E72c39419168FfF07F068E76a077588225798", // sender's account
  to: "0x03D28Df4b4c3a4bb1eA5D0a518E4D045172a6559", // receiver's account
  value: ethers.utils.parseEther("0.05"),
  gasLimit: ethers.utils.hexlify(10000),
  gasPrice: ethers.utils.hexlify(parseInt(await provider.getGasPrice())),
};
signer.sendTransaction(tx).then((transaction) => {
  console.info("Send finished!");
});
```




Editing data & making transactions

Connect to smart contract and call a method specified in ABI

```
var ether = require("ethers")

const abi = [
  // abi data
];
const signer; // see signer initialization

const connectedContract = new ethers.Contract("0xdAC...", abi, signer);

const nftTxn = await connectedContract.makeAnEpicNFT();
await nftTxn.wait();
```




What is an ABI?

An ABI (written in JSON format) includes information about a **smart contract** on which functions exist and how you can call them and get data back.

When calling non-built-in methods of smart contracts, you definitively need to provide an ABI for your frontend library to work with.

Decentralized Deployment



“



Decentralized Deployment

Decentralized Storages

- Blockchains are not designed for storing large amounts of data, so **decentralized storages (dStorage)** came into existing, e.g. Arweave, Filecoin, Storj, Fleek, etc.
- possible data types cover images, audio, video files, but also HTML documents, CSS & Javascript files → **a whole static website frontend can be stored in a decentralised way!**
 - checkout Fleek, Spheron, and others

Thoughts on Web3 UX



“




Onboarding / Missing web wallet

- If a user is new to Web3 and has no wallet (yet), a dApp should point it out in a friendly way → good Web3 apps find a way to onboard / educate users without sounding intimidating and overbearing.



Don't freak the user out

- also experienced users don't like to be asked for a wallet connection (or worse: permission allowance) immediately after they accessed a dApp.



Integrate user feedback, especially during waiting times..

- Some transactions take a while, e.g. for minting an NFT. This can leave people to think your app is broken, so use loaders and other well-outlined user feedback.



Reduce technical jargon

HODL, bullish, custodial, wallets, contracts, seed phrase??

- The majority of dApps are aimed at the general masses, so keep the language accessible, especially on the homepage



Transparency of transactions

- all relevant information regarding transactions should be available for users



Follow Web3 values & visions

- don't ask users for passwords, email addresses or other personal information
- don't force cookies, ads and tracking
- publish your dApp under Open Source license
- tech agnostic approach



Check out Web3 Design Principles

- <https://medium.com/@lyricalpolymath/web3-design-principles-f21db2f240c1>

A note on security



“



dApp frontend security

- enhance your frontend's security by e.g.
 - enforcing HTTPS
 - using X-XSS-Protection Header
 - ..
- Read more: <https://graph.org/Article-08-08>



Resources & Tutorials

- [Building a web3 frontend with React](#)
- [The Complete Guide to Full Stack Web3 Development](#)

Resources collection

- riike.dev/web3



More Resources

- [pointer.gg](#)
- [rabbithole.gg](#)
- [buildspace.so](#)
- [LearnWeb3DAO](#)
- [Cryptozombies](#)
- [Web3 University](#)



Thank you!

Any questions?

Twitter: @rikecodes

Web: rike.dev

Discord: rikecodes#3756

LinkedIn: <https://www.linkedin.com/in/rike-exner/>