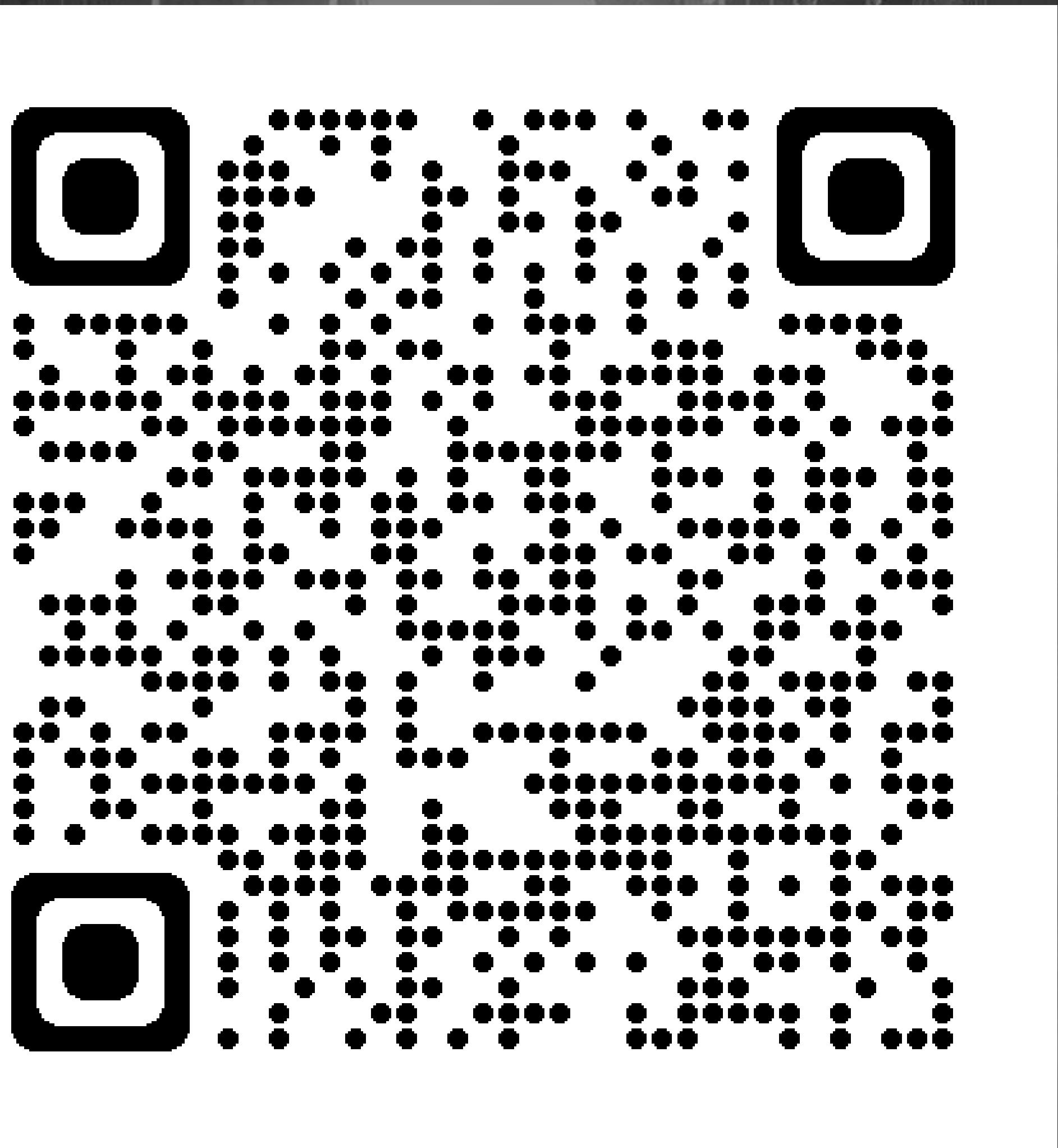


tinyurl.com/45p27hf

[GO TO THE REPO NOW →](#)





Hedera dApp Day

Arizona State University

ED MARQUEZ, DEVELOPER RELATIONS, SWIRLDS LABS

 @ed_marquez

 /in/ed-marquez

hello future

With dApp days, you will learn how to build and deploy applications on Hedera!



Section 1:
Introduction to
Web 3 and Hedera

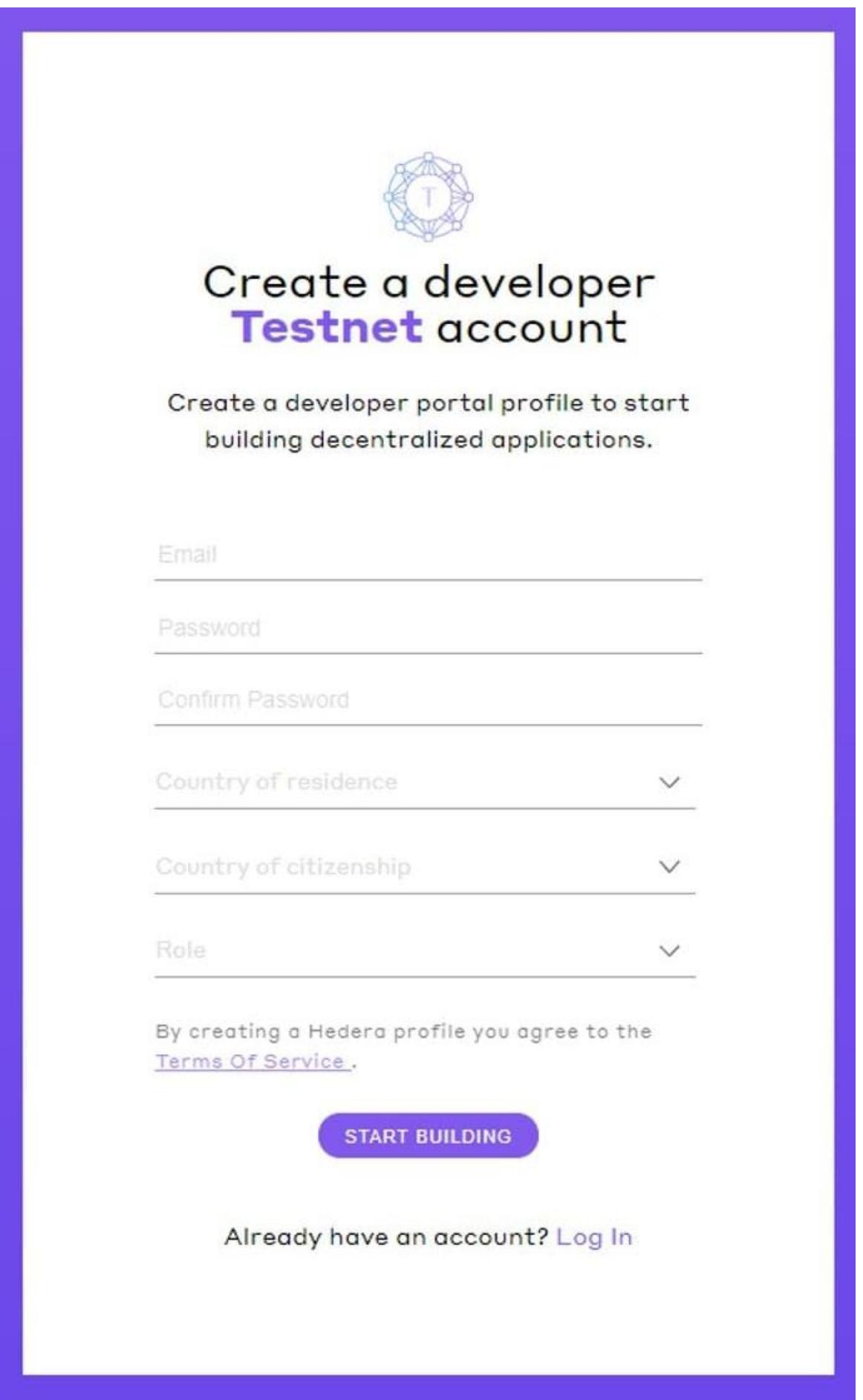


Section 2:
Build on Hedera

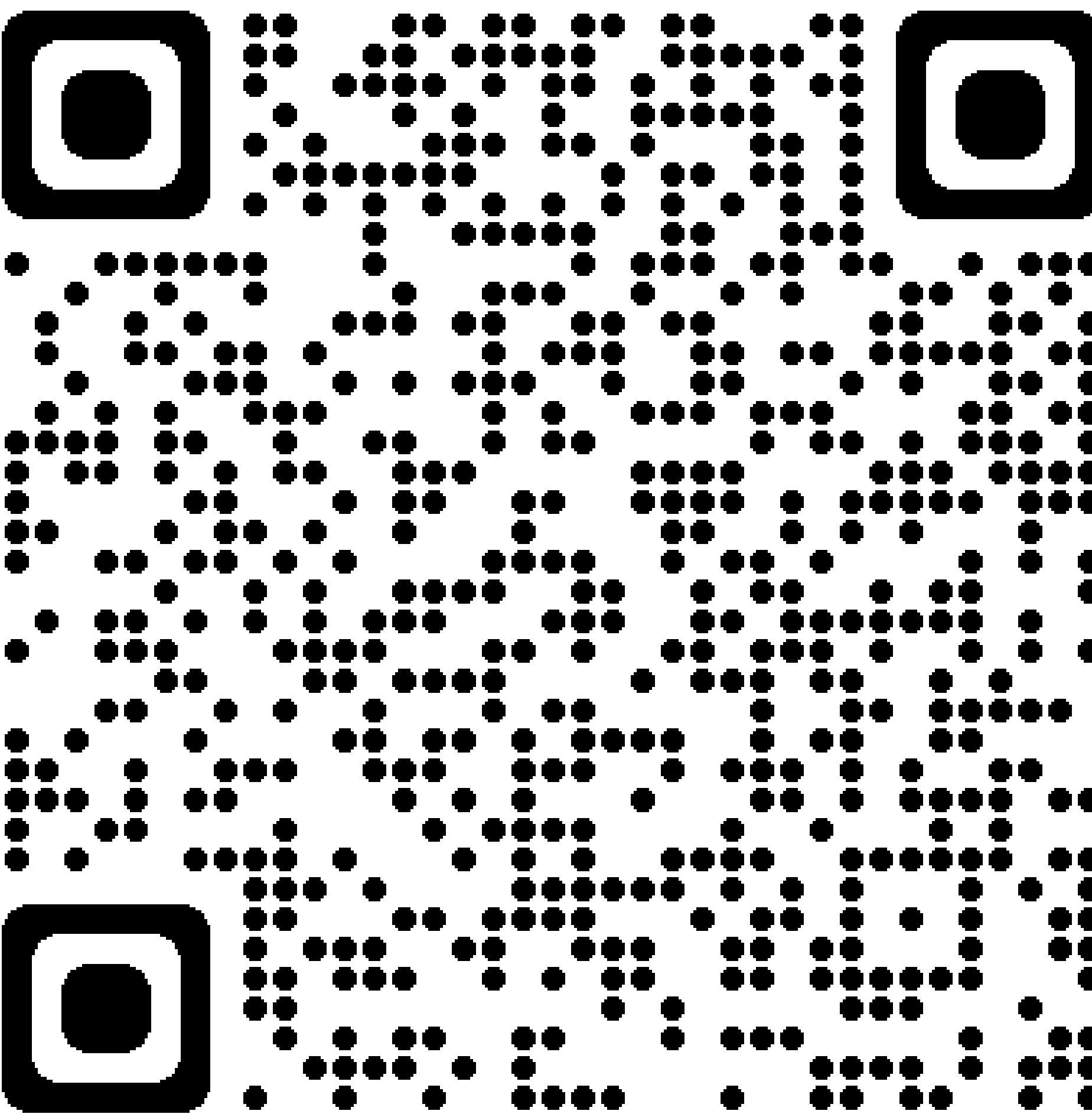


Section 3:
Enter NFT Giveaway
and Next Steps

Get a testnet account! (<https://portal.hedera.com/register>)



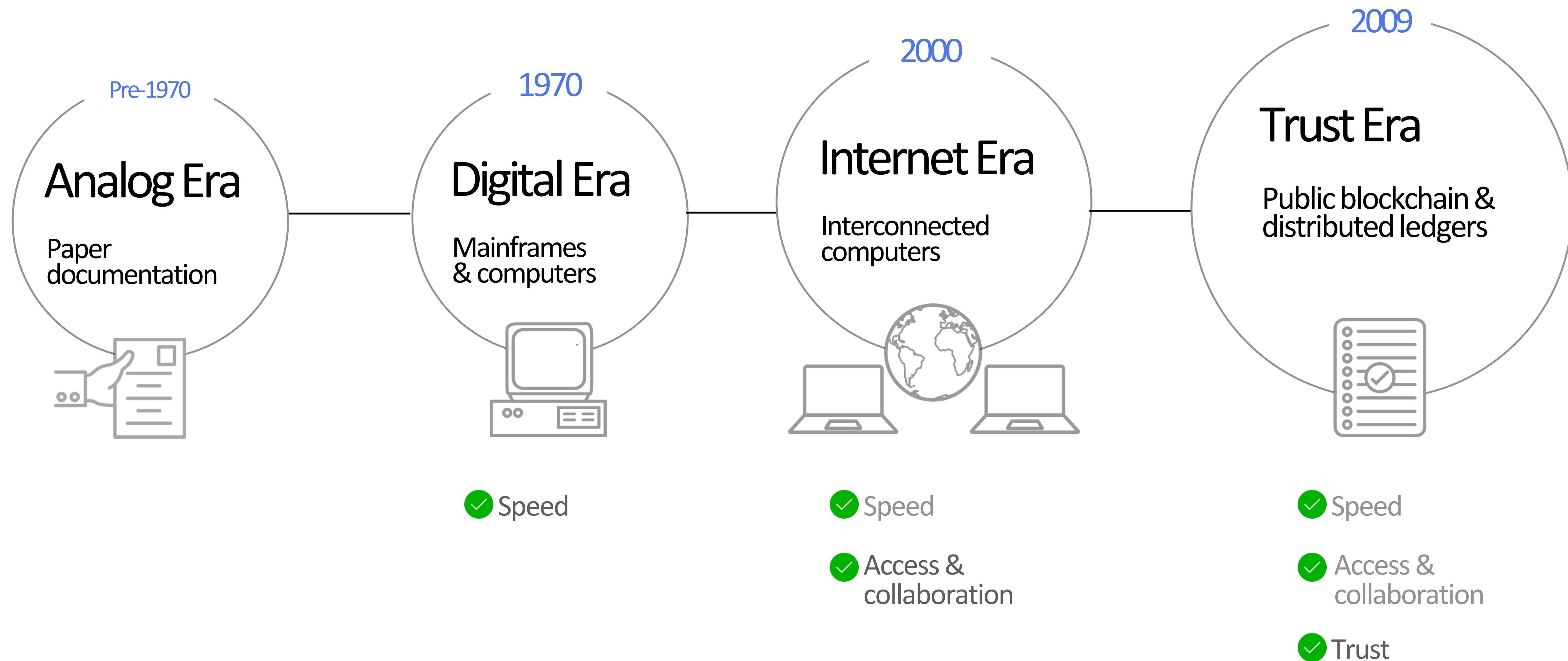
The screenshot shows the "Create a developer Testnet account" form. It features a logo of a stylized "T" inside a hexagonal grid. The main heading is "Create a developer **Testnet** account". Below it, a sub-instruction reads "Create a developer portal profile to start building decentralized applications." The form includes fields for "Email", "Password", "Confirm Password", "Country of residence" (with a dropdown arrow), "Country of citizenship" (with a dropdown arrow), and "Role" (with a dropdown arrow). A note at the bottom states "By creating a Hedera profile you agree to the [Terms Of Service](#)." A purple "START BUILDING" button is centered below the note. At the bottom, a link says "Already have an account? [Log In](#)".



A close-up, profile shot of a person's face and shoulders. They are looking intently at a computer screen in the foreground, which displays a large amount of colorful, syntax-highlighted code. The person has dark hair and is wearing a blue shirt. The background is blurred, showing what appears to be a keyboard and other office equipment.

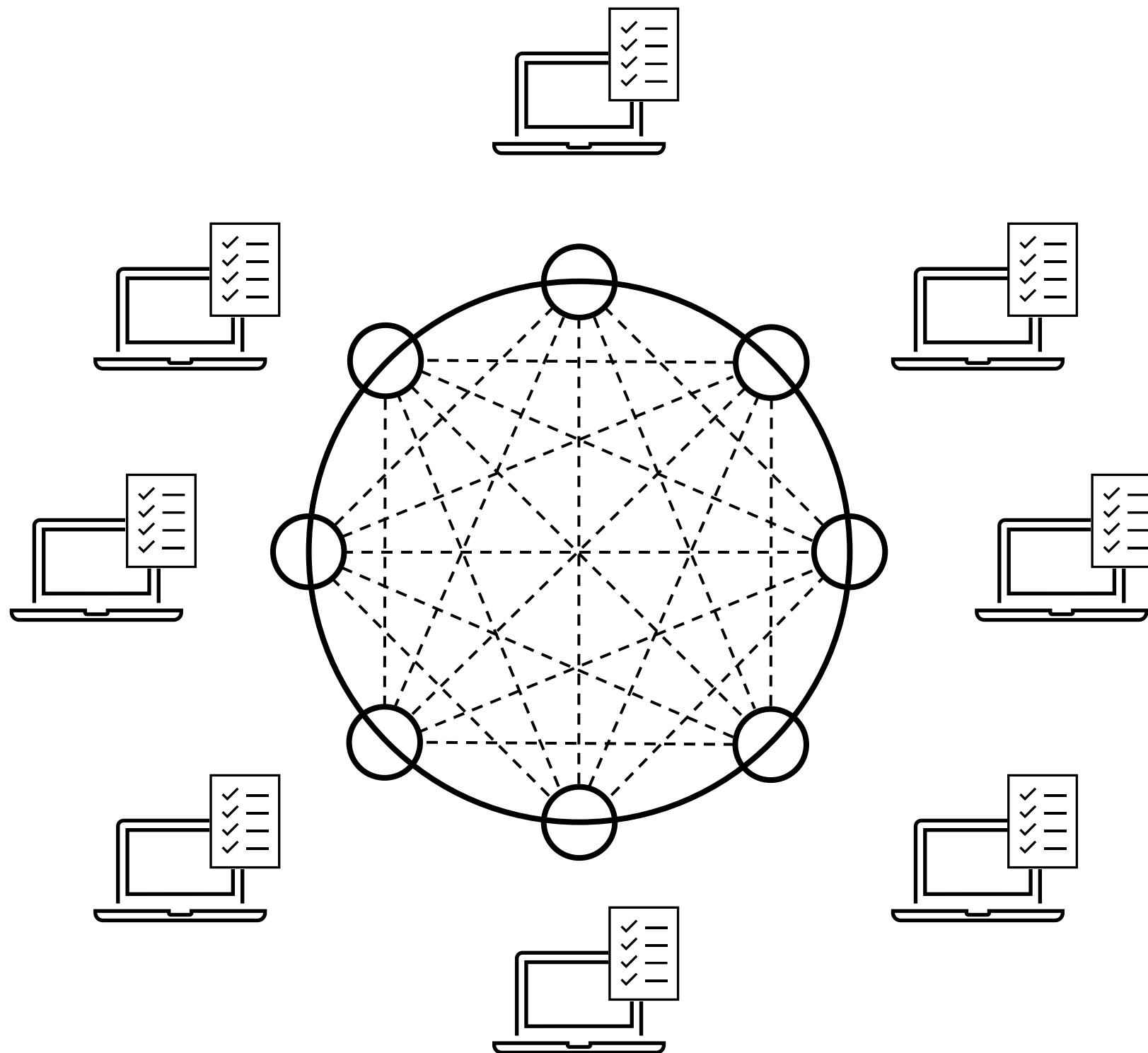
APPLICATION PREVIEW

The web evolves continuously and now it is becoming more decentralized



Distributed ledgers are a key component of Web 3.0 because of their qualities

DISTRIBUTED LEDGER



Entire network records and validates each transaction

CENTRALIZED LEDGER



Single authority verifies, records, and executes transactions

Distributed ledgers are a key component of Web 3.0 because of their qualities

No central point of failure to attack



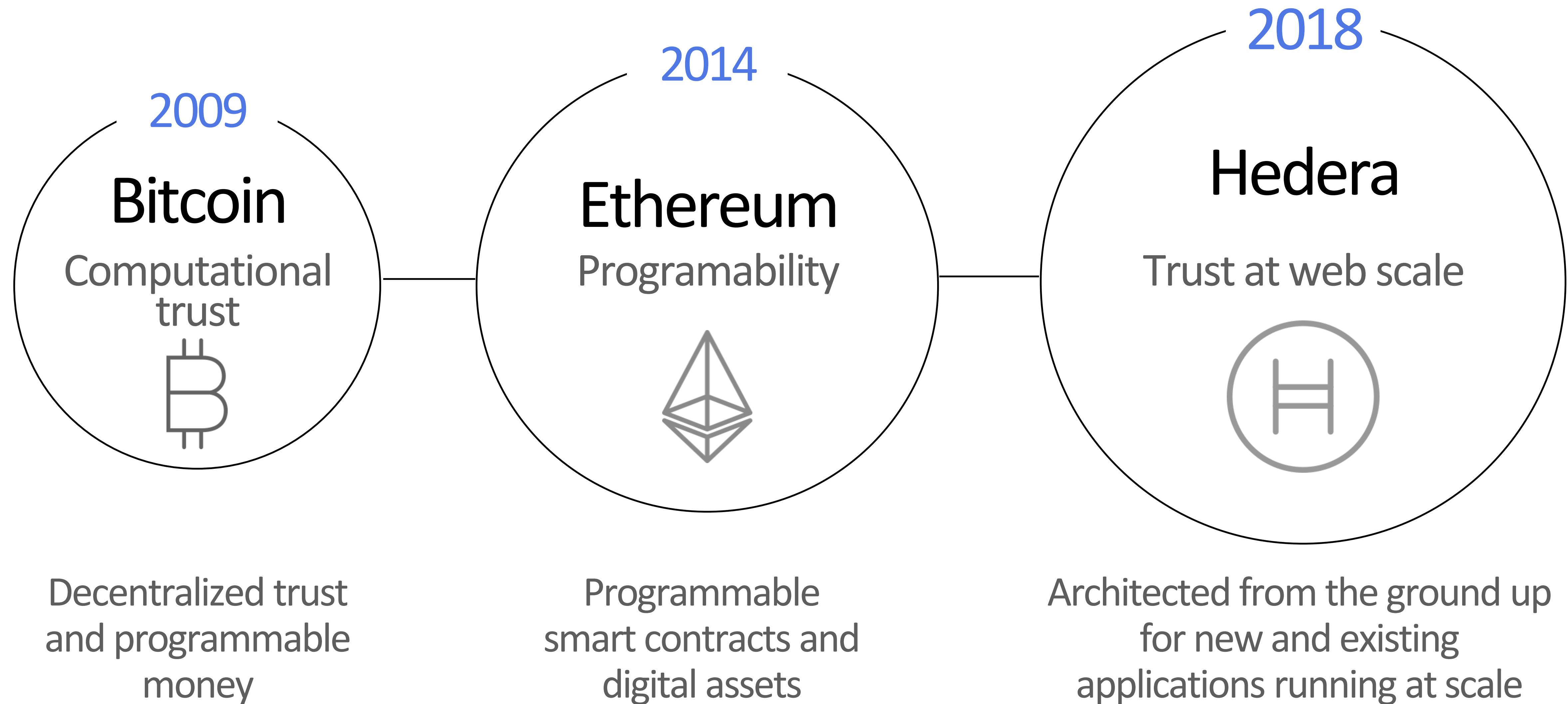
Reliant on strong cryptography to prove data integrity and tamper resistance



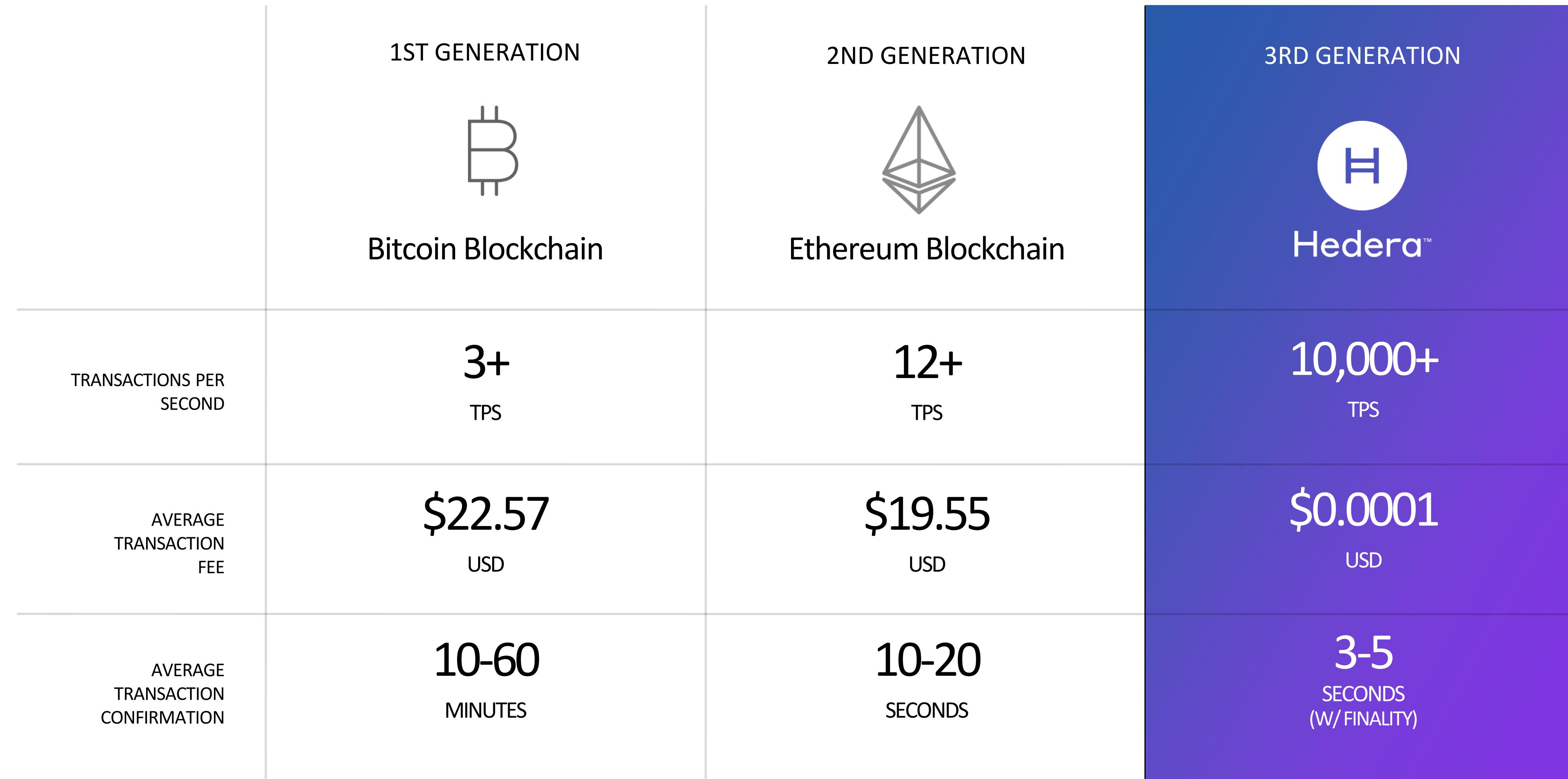
Rules for the ledger are determined by a governing concept of some sort

Require a consensus mechanism to determine the rules for adding new transactions to the state of the ledger

Hedera is a third-generation public distributed ledger



Hedera is a third-generation public distributed ledger



• Cryptocurrency transactions. For Hedera, range shown for transactions not requiring a transaction record, but can receive a transaction receipt.

• Avg. Bitcoin tx fee from 6/26/20 - 9/24/20 from <https://blockchair.com/bitcoin/charts/average-transaction-fee-usd?interval=3m>

• Avg. Ethereum tx fee from 6/26/20 - 9/24/20 from <https://blockchair.com/ethereum/charts/average-transaction-fee-usd?interval=3m>

A comparison of third-generation DLTs

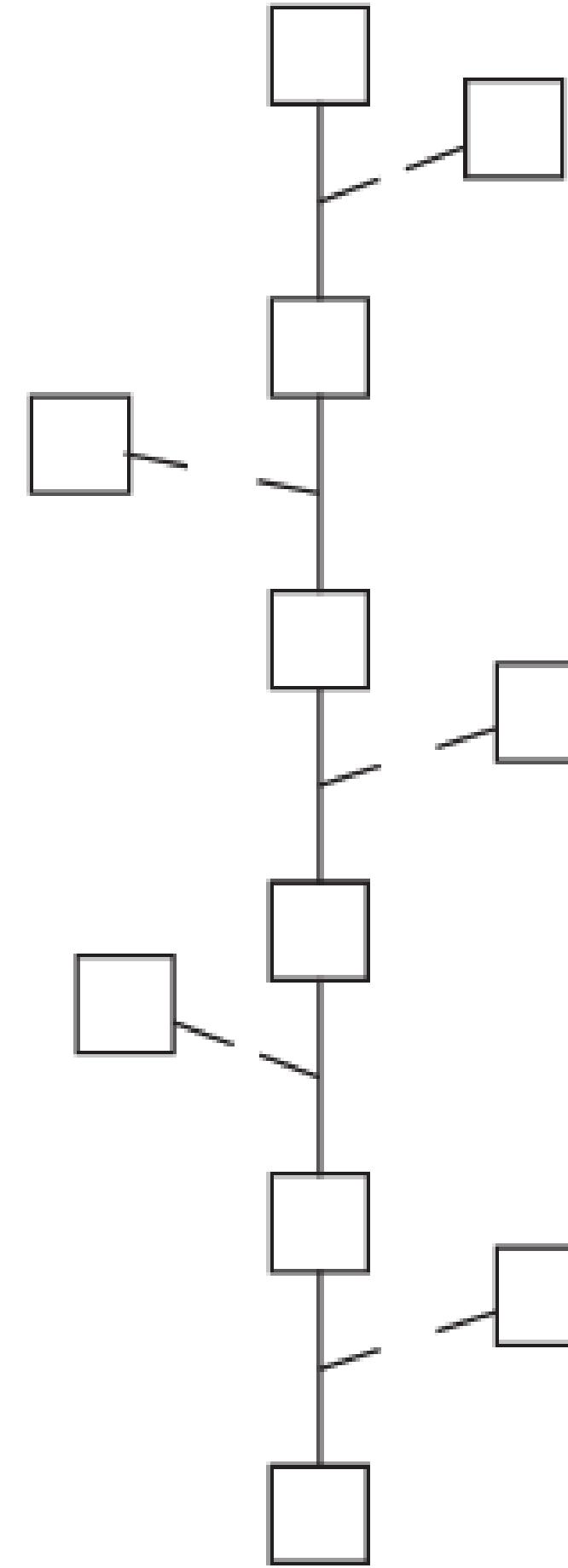
	POLYGON	ALGORAND	(ETHEREUM)	3RD GENERATION
				
TRANSACTIONS PER SECOND	6500* (CLAIMED)	1,000† (APPROX)	12+ TPS	10,000+ TPS
AVERAGE TRANSACTION FEE	\$0.002 USD (VARIABLE)	0.001 ALGO (FIXED)	\$19.55 USD (VARIABLE)	\$0.0001^ USD (FIXED)
AVERAGE TRANSACTION CONFIRMATION	5 - 10 SECONDS (LEADER BLOCK CREATION)	5 SECONDS (LEADER BLOCK CREATION)	10-20 SECONDS (LEADER BLOCK CREATION)	3-5 SECONDS (W/ FINALITY)
ENERGY USE PER TRANSACTION	90+ KWH	0.00534 KWH	102+ KWH	0.00017 KWH

* Actual TPS closer to 53 over a rolling 30 day period - <https://thecoinweekly.com/not-so-fast-are-eth-killers-as-speedy-as-they-claim>

^ Hedera Transaction fees - <https://hedera.com/fees>

† <https://www.algorand.com/resources/algorand-announcements/algorand-2021-performance>

Blockchains



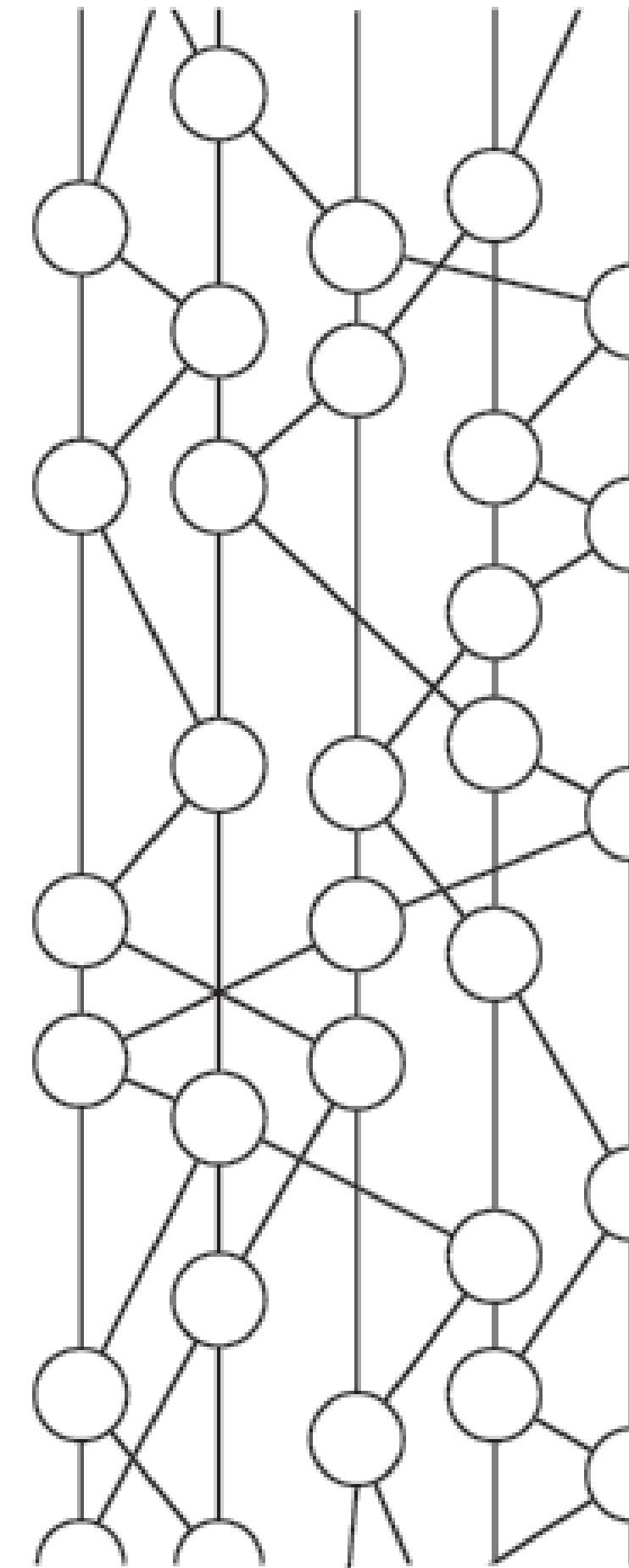
Proof-of-Work (PoW) typically has heavy electrical / computational requirements

No guaranteed consensus, just a
“good enough” probability

Inefficient - “stale” blocks are pruned

Can be difficult to scale beyond
~1,000 tps

Hashgraph



Hashgraph is an **open-source** consensus algorithm and data structure

Uses a directed acyclic graph (DAG) and novel inventions:

- Gossip about Gossip protocol
 - Virtual voting

Hedera currently supports
10,000+ cryptocurrency
transactions per second

Finality within 3-5 seconds

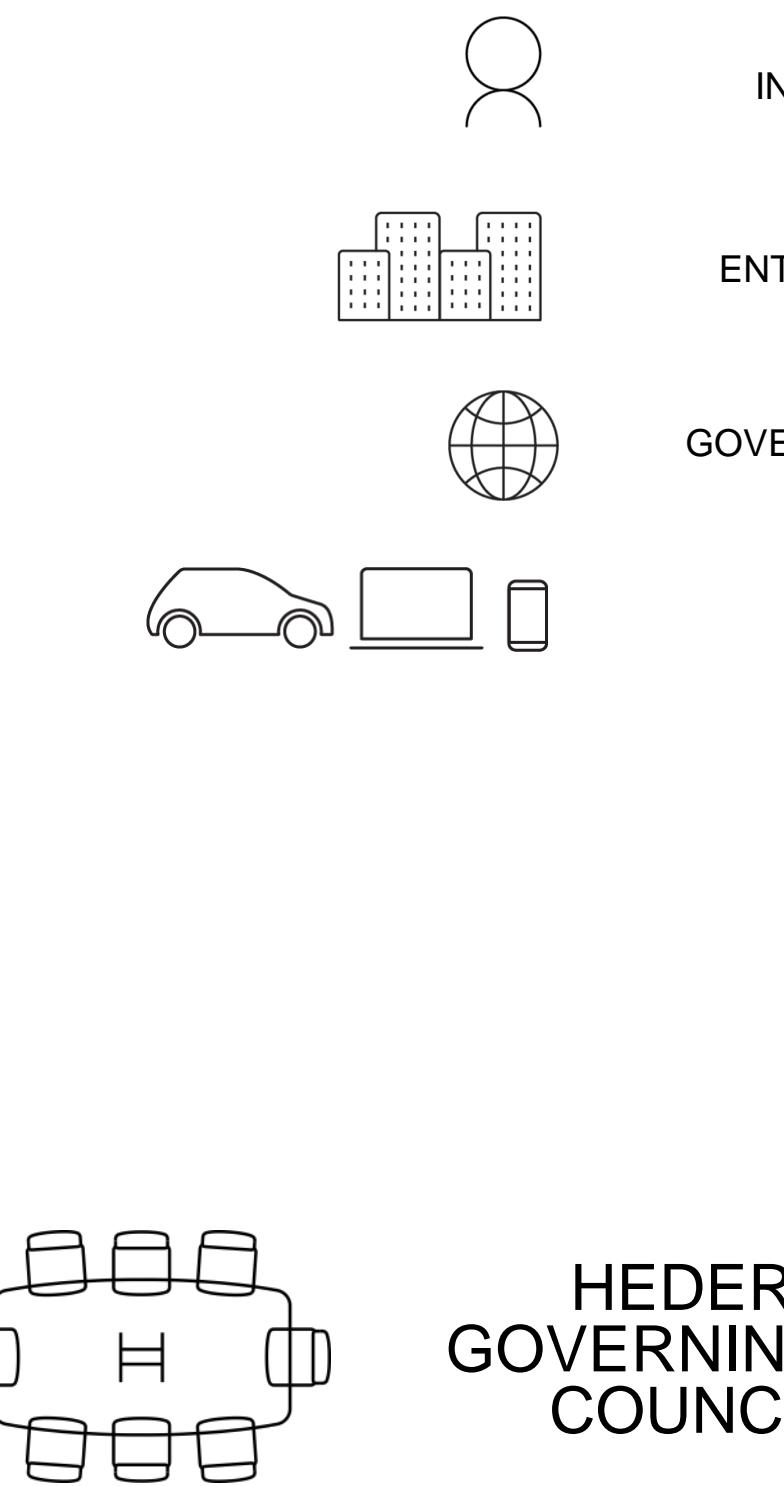
Blockchains



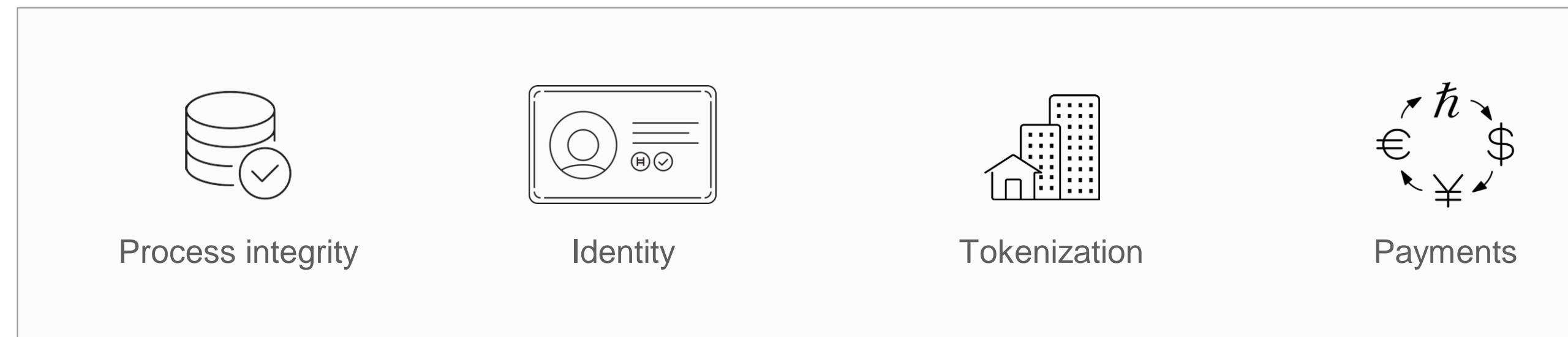
Hashgraph



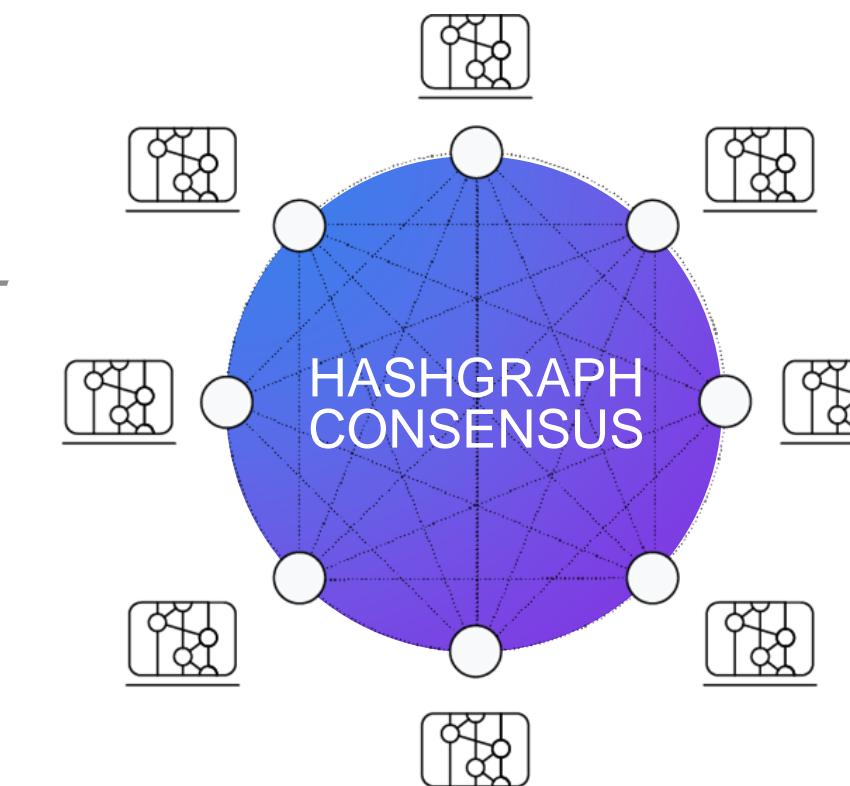
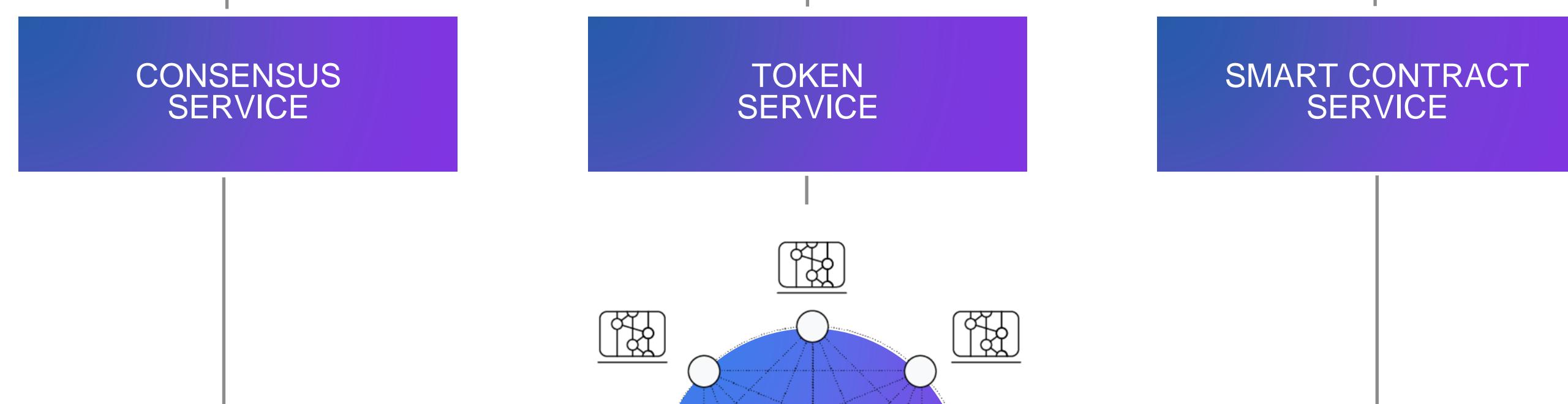
END USERS



APPLICATION USE CASES

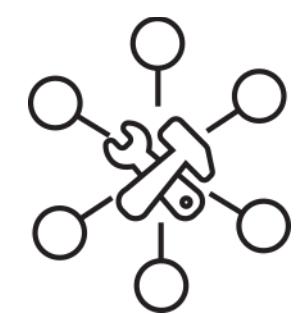


HEDERA GOVERNING COUNCIL



HEDERA MAINNET

HEDERA NETWORK SERVICES



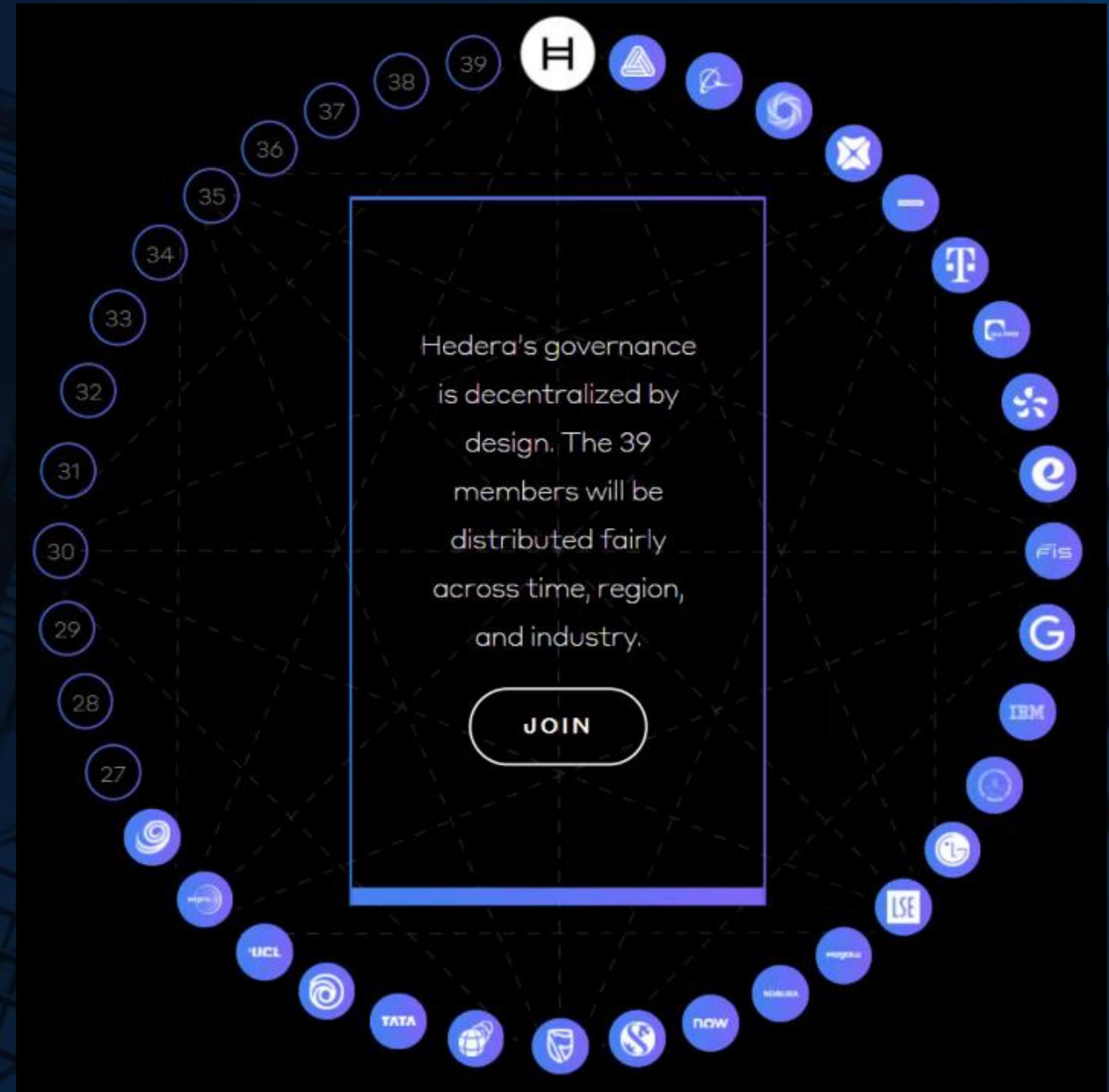
Hedera Governing Council



Up to **39** leading global organizations

20+ unique industries across **5** continents, touching all major markets

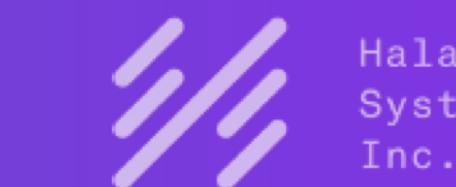
3-year maximum term, with up to 2 consecutive terms



2.6% influence per member (equal vote)

7 committees:
Membership, Marketing, Finance, Audit, Technical Steering/Product Pricing, Legal & Regulatory

Every member required to run a network node



14,000+ DEVELOPERS

Attending global hackathons, meetups, and active in Discord.

100+ APPLICATIONS

In production on Hedera Mainnet, since open access on Sept. 2019

>7,000,000+ TXS/DAY

Far surpassing the daily transaction volume of Ethereum.

>2,450,000,000+ TXS TO DATE

Most used public DLT surpassing total transaction volume of Ethereum.

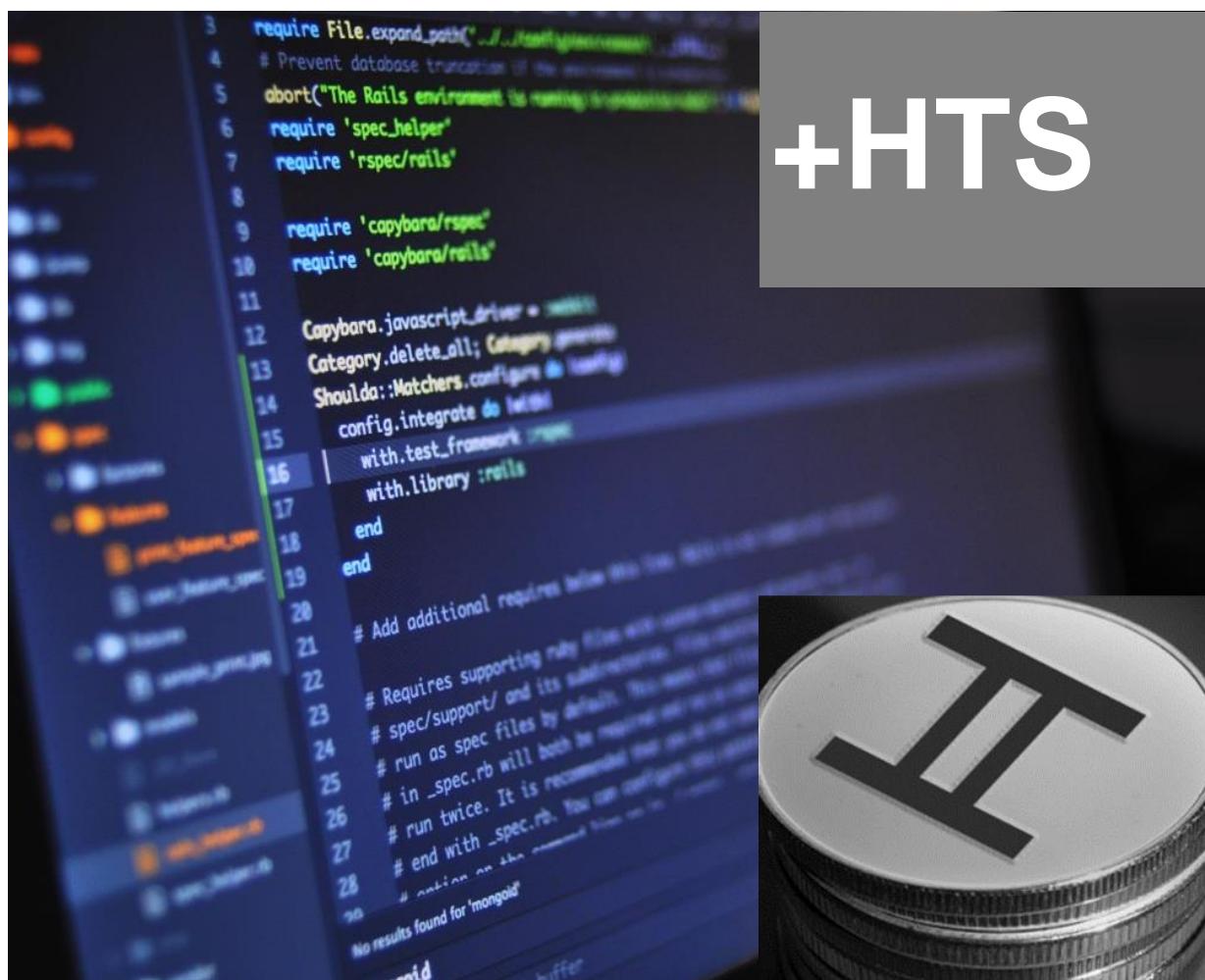
>1,000,000+ TOTAL ACCOUNTS ON MAINNET

66,700 accounts created in April 2022 alone.

With dApp days, you will learn how to build and deploy applications on Hedera!



Section 1:
Introduction to
Web 3 and Hedera



Section 2:
Build on Hedera



Section 3:
Enter NFT Giveaway
and Next Steps

A close-up, profile shot of a person's face and shoulders. They are looking down at a computer screen, which is visible on the left side of the frame. The screen displays a large amount of colorful, syntax-highlighted code, likely JavaScript or similar. The person has dark hair and is wearing a blue t-shirt. The background is blurred, showing what appears to be an office environment.

LET'S BUILD A COOL DAPP ON HEDERA!

BUILD A DAPP ON HEDERA (1/4)

1. Go to the [hedera-dapp-days](#) repository in GitHub

The screenshot shows the GitHub repository page for 'hedera-dapp-days'. At the top, there's a 'Code' tab, followed by 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Below the tabs, it shows 'token2049-start' as the current branch, '3 branches', and '0 tags'. There are buttons for 'Go to file', 'Add file', and 'Code'. The main area displays the repository's code.

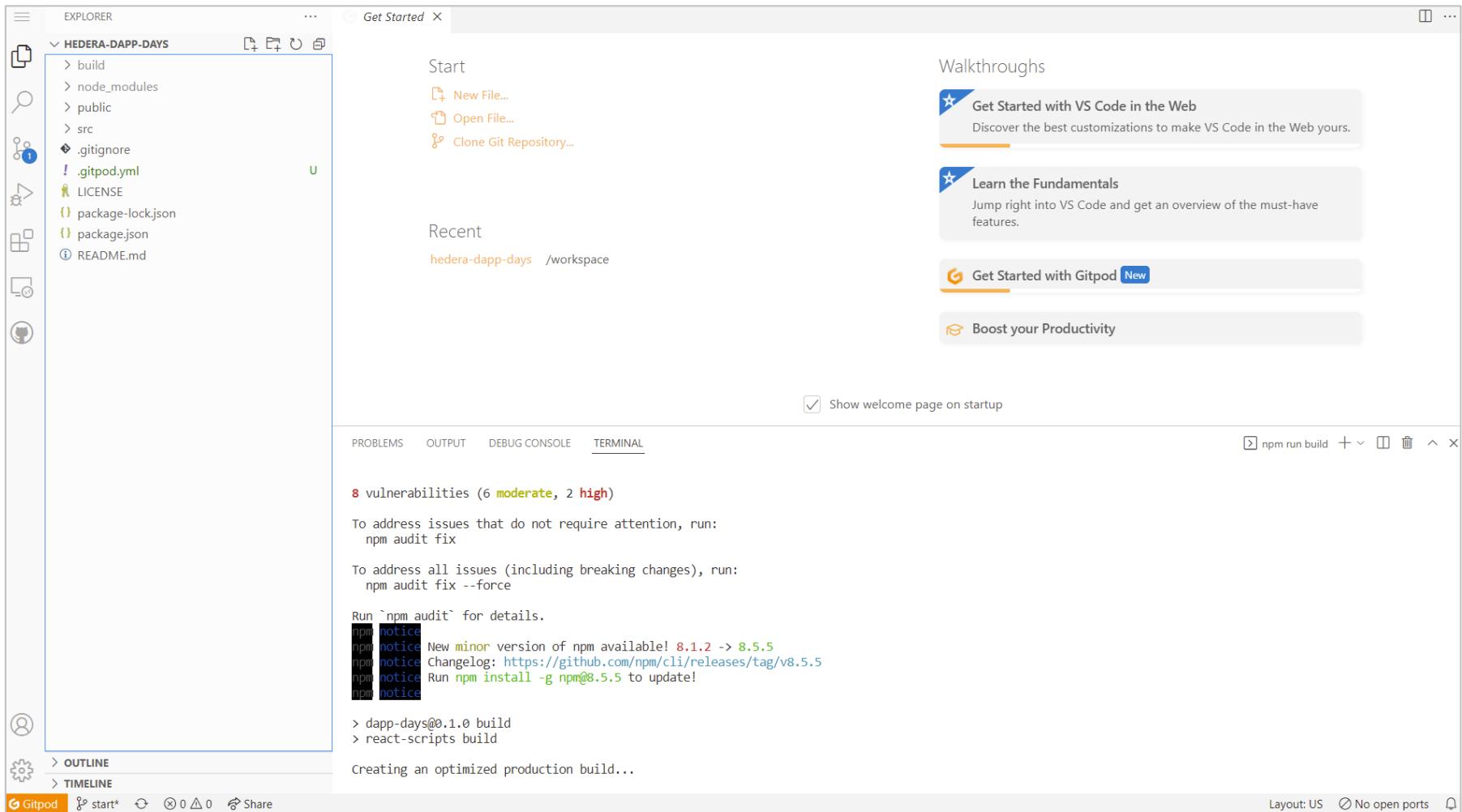
2. Fork the repository...



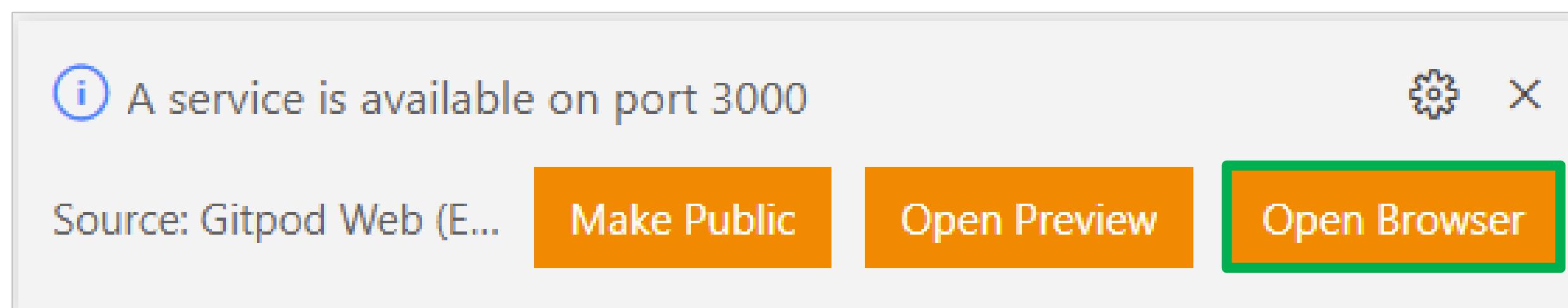
... and open your fork in Gitpod using a link that looks as follows:

`gitpod.io#https://github.com/<GITHUB_UN>/hedera-dapp-days/tree/<EVENT_NAME>-start`

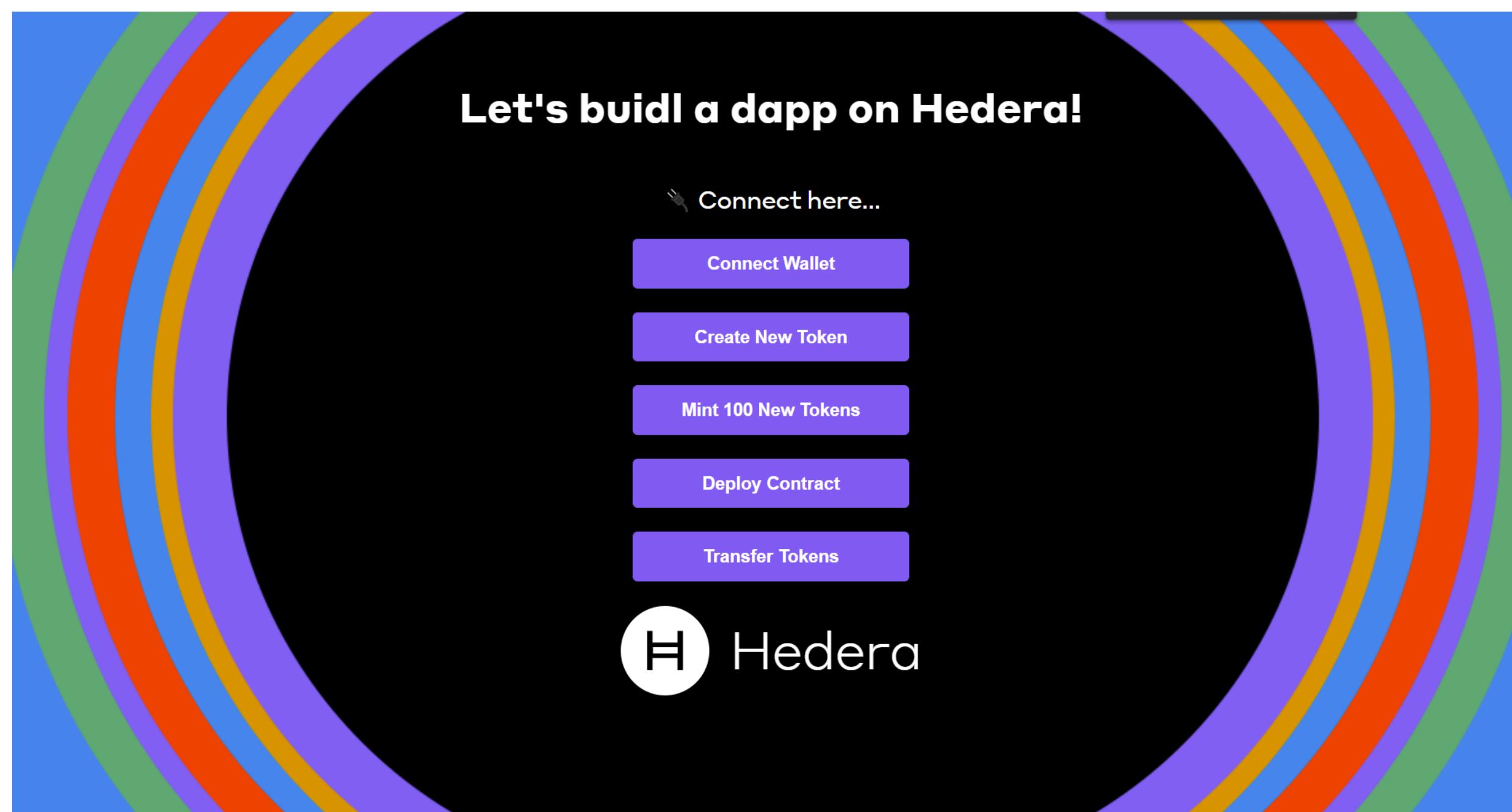
Where `<GITHUB_UN>` is your GitHub username, `<EVENT_NAME>` is the event



As Gitpod loads, you will see this prompt. Click **Open Browser**



After that, you will see the template application in a new tab

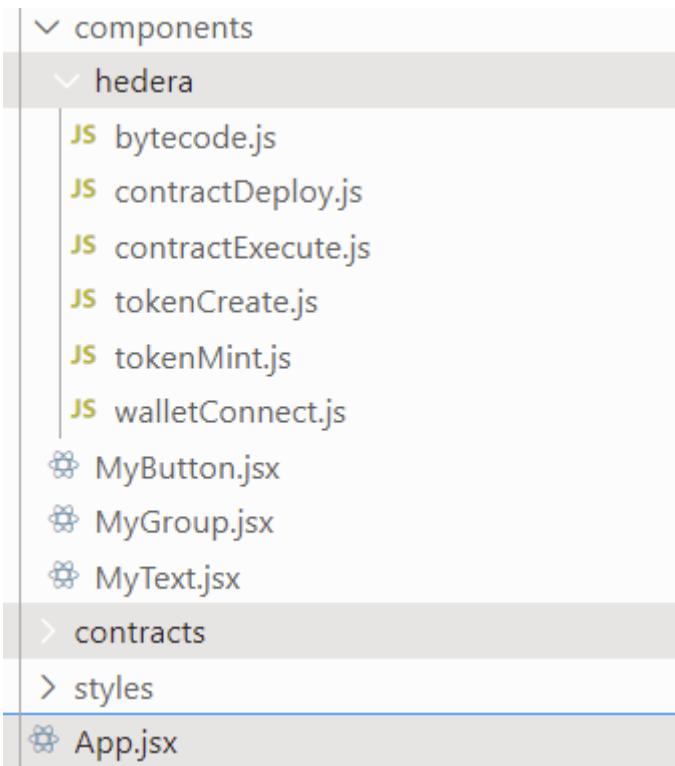


BUILD A DAPP ON HEDERA (2/4)

3. If you're not too familiar with React, explore the project in Gitpod. Most of the relevant files for dAppDays are in the **src** directory

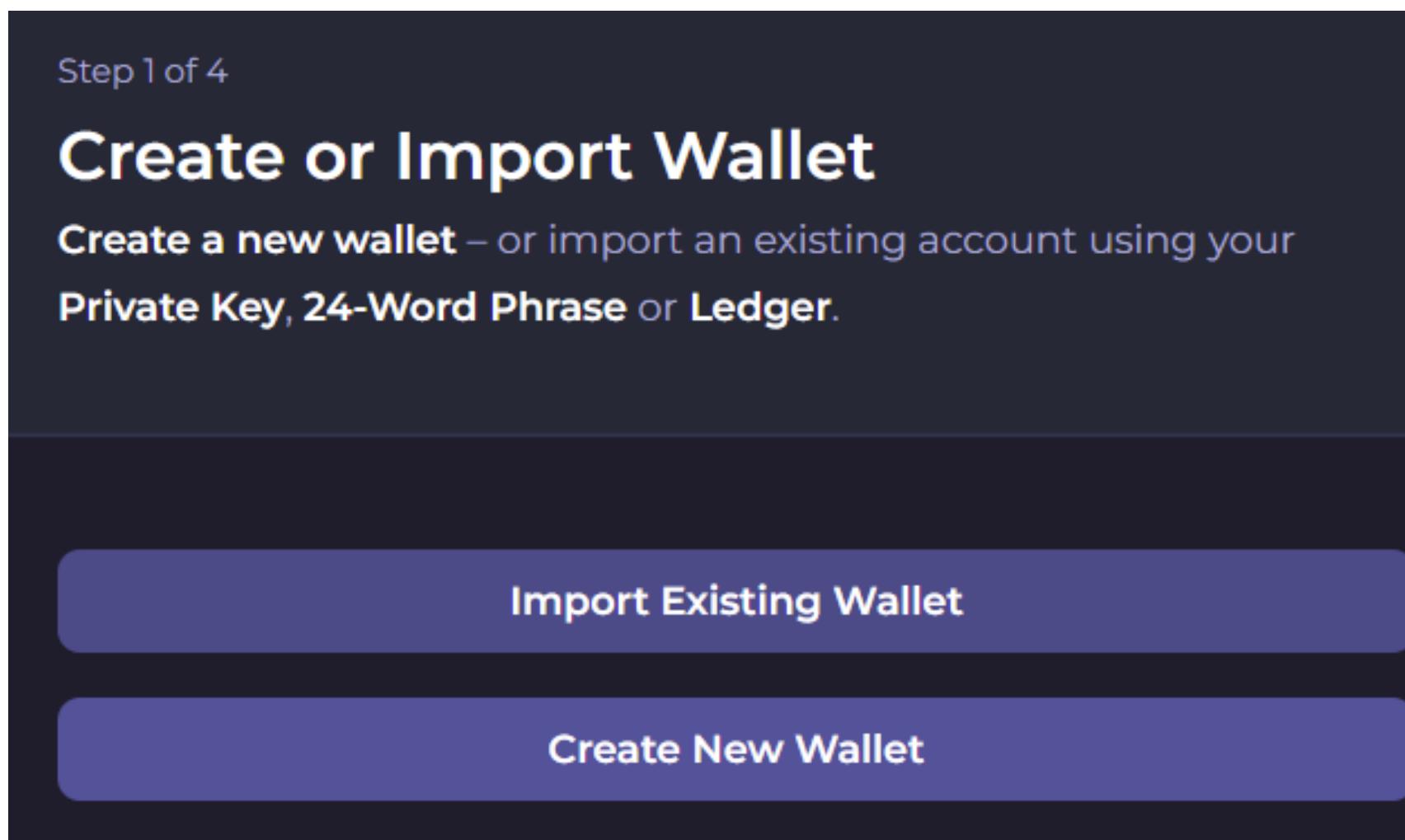
Files and directories of interest include:

- **App.jsx**
- **src/components**
- **src/components/hedera**
- **src/contracts**



4. Open the HashPack wallet extension to...

- Import your existing Hedera testnet account credentials, OR to..
- Create a new testnet account from HashPack



5. Code the Hedera components and the Solidity contract. All files you will code have boilerplate code as a starting point. Add the following:

5a. *walletConnect.js*

```
async function walletConnectFn() {
    console.log(`\n=====`);
    console.log("- Connecting wallet...");

    let saveData = {
        topic: "",
        pairingString: "",
        privateKey: "",
        pairedWalletData: null,
        pairedAccounts: []
    };
    let appMetadata = {
        name: "Hedera dApp Days",
        description: "Let's build a dapp on Hedera",
        icon: "https://raw.githubusercontent.com/swirls-matt/hedera-dapp-days/token-2049-start/src/assets/hederaLogo.png",
    };
    let hashconnect = new HashConnect();

    // First init and store the pairing private key for later (this is NOT your account private key)
    const initData = await hashconnect.init(appMetadata);
    saveData.privateKey = initData.privKey;
    console.log(`- Private key for pairing: ${saveData.privateKey}`);

    // Then connect, storing the new topic for later
    const state = await hashconnect.connect();
    saveData.topic = state.topic;
    console.log(`- Pairing topic is: ${saveData.topic}`);

    // Generate a pairing string, which you can display and generate a QR code from
    saveData.pairingString = hashconnect.generatePairingString(state, "testnet", false);

    // Find any supported local wallets
    hashconnect.findLocalWallets();
    hashconnect.connectToLocalWallet(saveData.pairingString);

    return [hashconnect, saveData];
}
```

BUILD A DAPP ON HEDERA (3/4)

5b. *tokenCreate.js*

```
async function tokenCreateFn(walletData, accountId) {
    console.log(`\n=====
    - Creating HTS token...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    const url = `https://testnet.mirrornode.hedera.com/api/v1/accounts?account.id=${accountId}`;
    const mirrorQuery = await axios(url);
    const supplyKey = PublicKey.fromString(mirrorQuery.data.accounts[0].key.key);

    const tokenCreateTx = await new TokenCreateTransaction()
        .setTokenName("dAppDayToken")
        .setTokenSymbol("DDT")
        .setTreasuryAccountId(accountId)
        .setAutoRenewAccountId(accountId)
        .setAutoRenewPeriod(7776000)
        .setInitialSupply(400)
        .setDecimals(0)
        .setSupplyKey(supplyKey)
        .freezeWithSigner(signer);
    const tokenCreateSubmit = await tokenCreateTx.executeWithSigner(signer);
    const tokenCreateRx = await provider.getTransactionReceipt(tokenCreateSubmit.transactionId);
    const tId = tokenCreateRx tokenId;
    const supply = tokenCreateTx._initialSupply.low;
    console.log(`- Created HTS token with ID: ${tId}`);
    return [tId, supply, tokenCreateSubmit.transactionId];
}
```

5c. *tokenMint.js*

```
async function tokenMintFn(walletData, accountId, tId) {
    console.log(`\n=====
    - Minting ${amount} tokens...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    const tokenMintTx = await new TokenMintTransaction()
        .setTokenId(tId)
        .setAmount(amount)
        .freezeWithSigner(signer);
    const tokenMintSubmit = await tokenMintTx.executeWithSigner(signer);
    const tokenCreateRx = await provider.getTransactionReceipt(tokenMintSubmit.transactionId);
    const supply = tokenCreateRx.totalSupply;
    console.log(`- Tokens minted. New supply is ${supply}`);

    return [supply, tokenMintSubmit.transactionId];
}
```

5d. *contracts → AssoTransHTS.sol*

```
contract AssoTransHTS is HederaTokenService {
    address tokenAddress;

    constructor(address _tokenAddress) public {
        tokenAddress = _tokenAddress;
    }

    function tokenAssoTrans(int64 _amount) external {
        int response1 = HederaTokenService.associateToken(address(this), tokenAddress);
        int response2 = HederaTokenService.transferToken(tokenAddress, msg.sender, address(this), _amount);
    }
}
```

...and *contractDeploy.js*

```
async function contractDeployFn(walletData, accountId, tokenId) {
    console.log(`\n=====`);
    console.log(`- Deploying smart contract on Hedera...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    //Create a file on Hedera and store the hex-encoded bytecode
    const fileCreateTx = await new FileCreateTransaction()
        .setContents(bytecode)
        .freezeWithSigner(signer);
    const fileSubmit = await fileCreateTx.executeWithSigner(signer);
    const fileCreateRx = await provider.getTransactionReceipt(fileSubmit.transactionId);
    const bytecode fileId = fileCreateRx fileId;
    console.log(`- The smart contract bytecode file ID is: ${bytecode fileId}`);

    // Create the smart contract
    const contractCreateTx = await new ContractCreateTransaction()
        .setBytecodeFileId(bytecode fileId)
        .setGas(3000000)
        .setConstructorParameters(
            new ContractFunctionParameters().addAddress(tokenId.toSolidityAddress())
        )
        .freezeWithSigner(signer);
    const contractCreateSubmit = await contractCreateTx.executeWithSigner(signer);
    const contractCreateRx = await provider.getTransactionReceipt(contractCreateSubmit.transactionId);
    const cId = contractCreateRx.contractId;
    const contractAddress = cId.toSolidityAddress();
    console.log(`- The smart contract ID is: ${cId}`);
    console.log(`- The smart contract ID in Solidity format is: ${contractAddress} \n`);

    return [cId, contractCreateSubmit.transactionId];
}
```

BUILD A DAPP ON HEDERA (4/4)

5e. *contractExecute.js*

```
async function contractExecuteFn(walletData, accountId, tokenId, contractId) {
    console.log(`\n=====`);
    console.log(`- Executing the smart contract...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    //Execute a contract function (transfer)
    const contractExecTx = await new ContractExecuteTransaction()
        .setContractId(contractId)
        .setGas(3000000)
        .setFunction("tokenAssoTrans", new ContractFunctionParameters().addInt64(50))
        .freezeWithSigner(signer);
    const contractExecSign = await contractExecTx.signWithSigner(signer);
    const contractExecSubmit = await contractExecSign.executeWithSigner(signer);
    const contractExecRx = await provider.getTransactionReceipt(contractExecSubmit.transactionId);
    console.log(`- Token transfer from Operator to contract: ${contractExecRx.status.toString()}`);

    const bCheck = await signer.getAccountBalance();
    console.log(
        `- Operator balance: ${bCheck.tokens._map.get(tokenId.toString())} units of token ${tokenId}`
    );
}

return contractExecSubmit.transactionId;
}
```

6. Next steps for you...

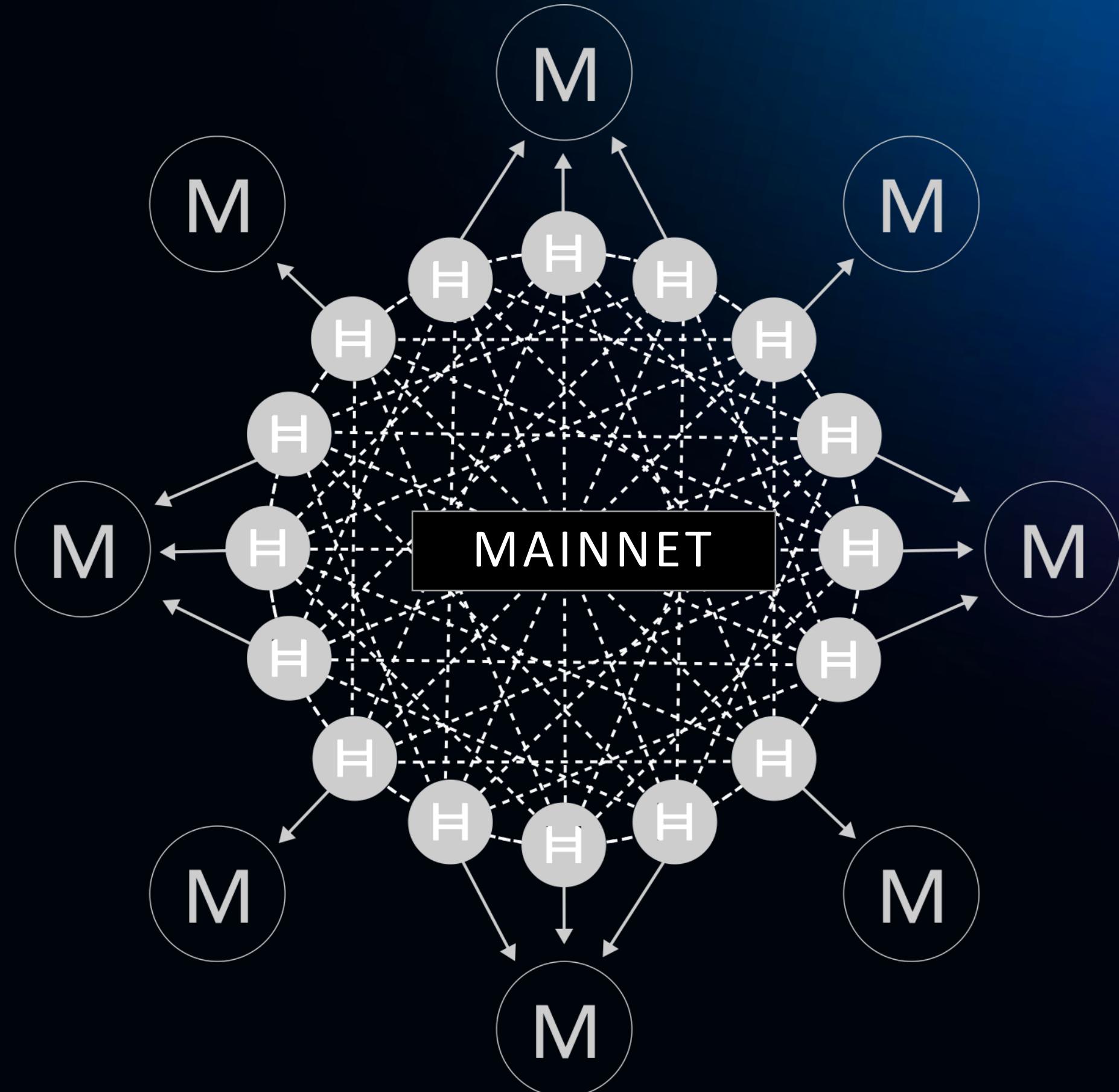
- Add more functionality to your application. Here are a few ideas:
 - Send *hbar* to and from the contract
 - Create tokens directly via the contract (*hint: HTS precompiles*)
 - Exchange *hbar* and HTS tokens via the contract
 - Add error handling to the React application
- Try libraries and tools to speed up your development:
 - [Run a local Hedera local network for testing](#)
 - [Use hether.js / ether.js / web3.js with Hedera](#)
- Check out the documentation and try:
 - [Examples](#)
 - [Tutorials](#)
 - [Demo applications](#)
 - [SDK implementations](#)
- Join another Hedera dApp Day

HEDERA MAINNET & MIRRORNET



MAINNET

- Can submit HAPI (Hedera API) transactions to the Hedera network
- Contributes to consensus on transactions
- Creates events on the Hedera network
- Requires HBAR cryptocurrency payment for transactions & queries



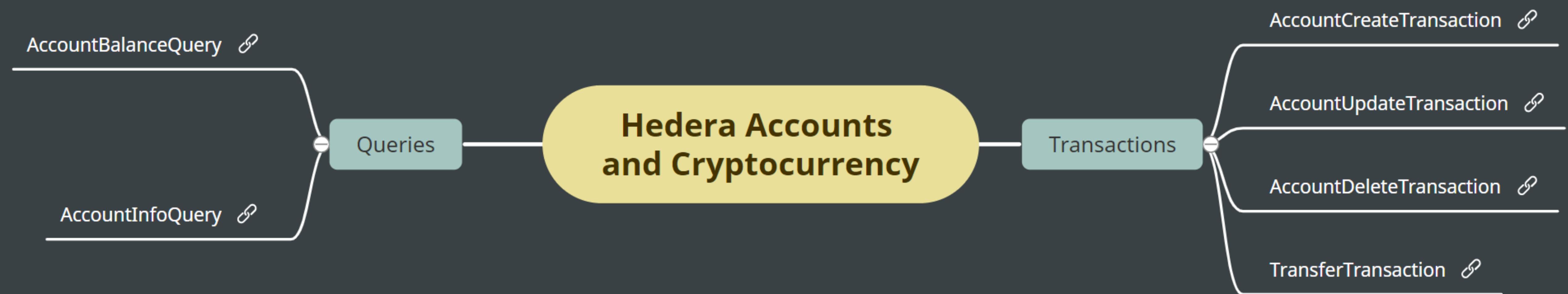
MIRRORNET

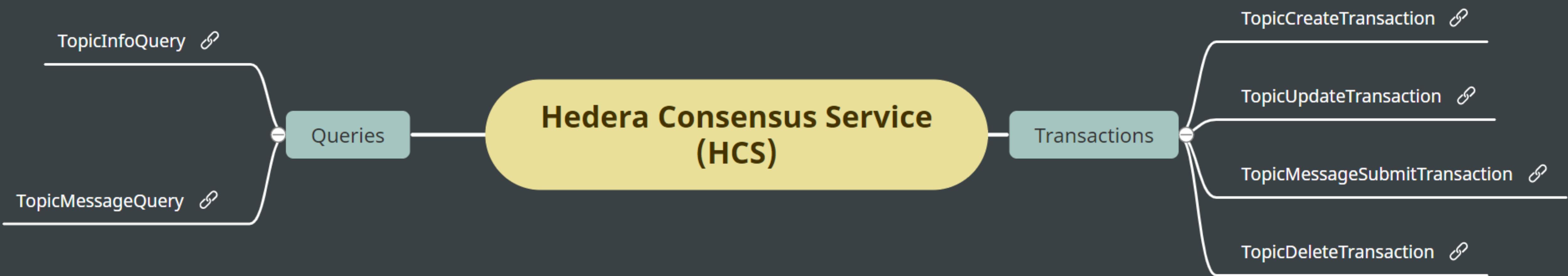
- Maintains a history of some or all the Hedera network state and ledger of transactions
- Value-added services (managed read-only node, etc.)
- Enables analytical insight into an application's state / transactions
- Publish and subscribe capabilities

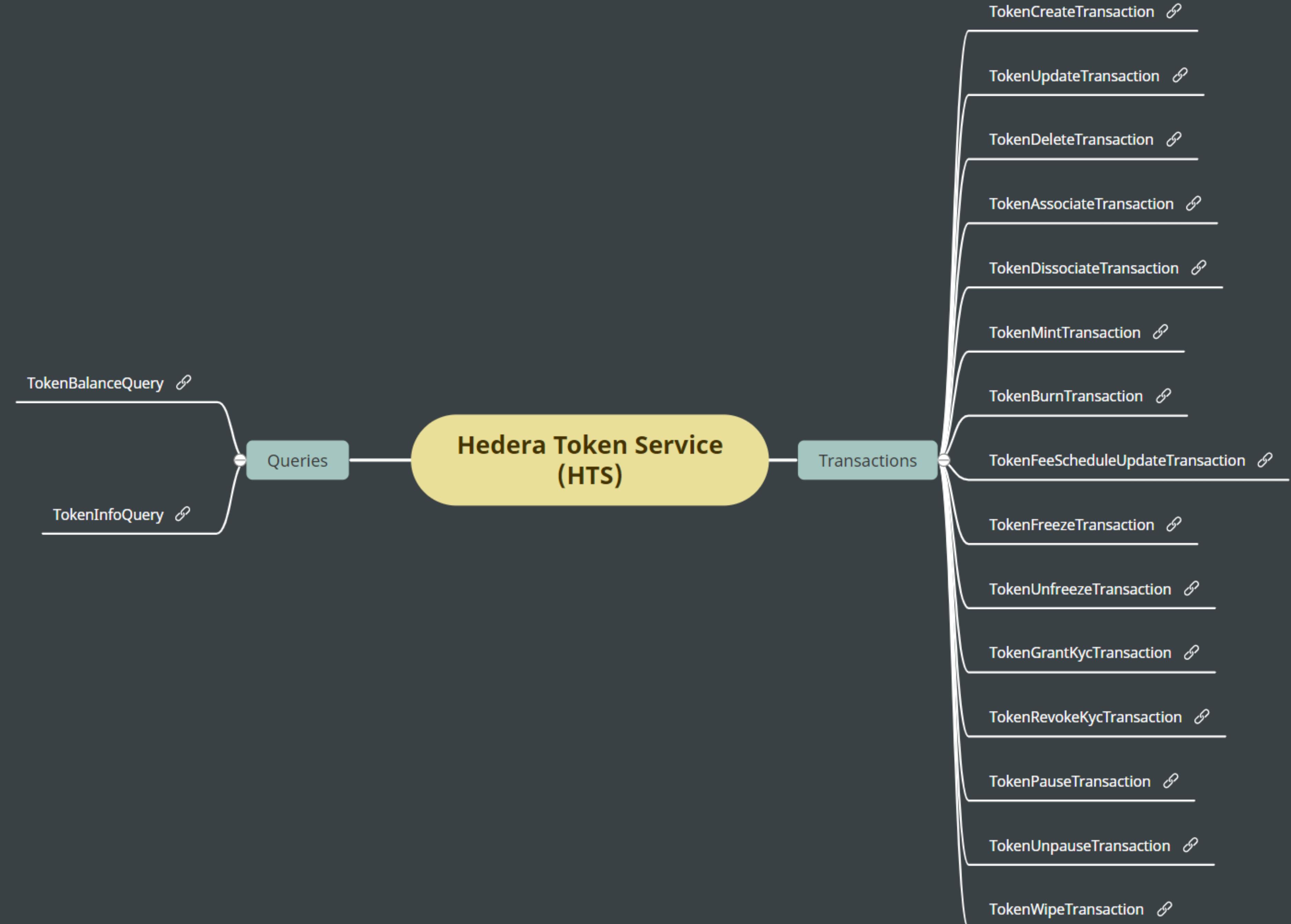


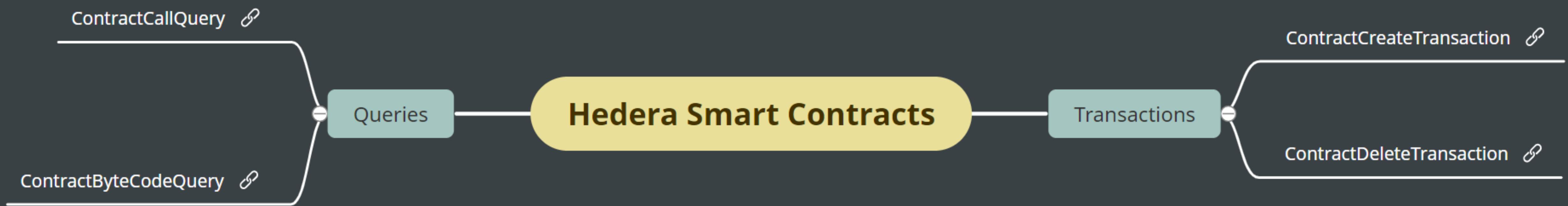
Hedera™ Hashgraph

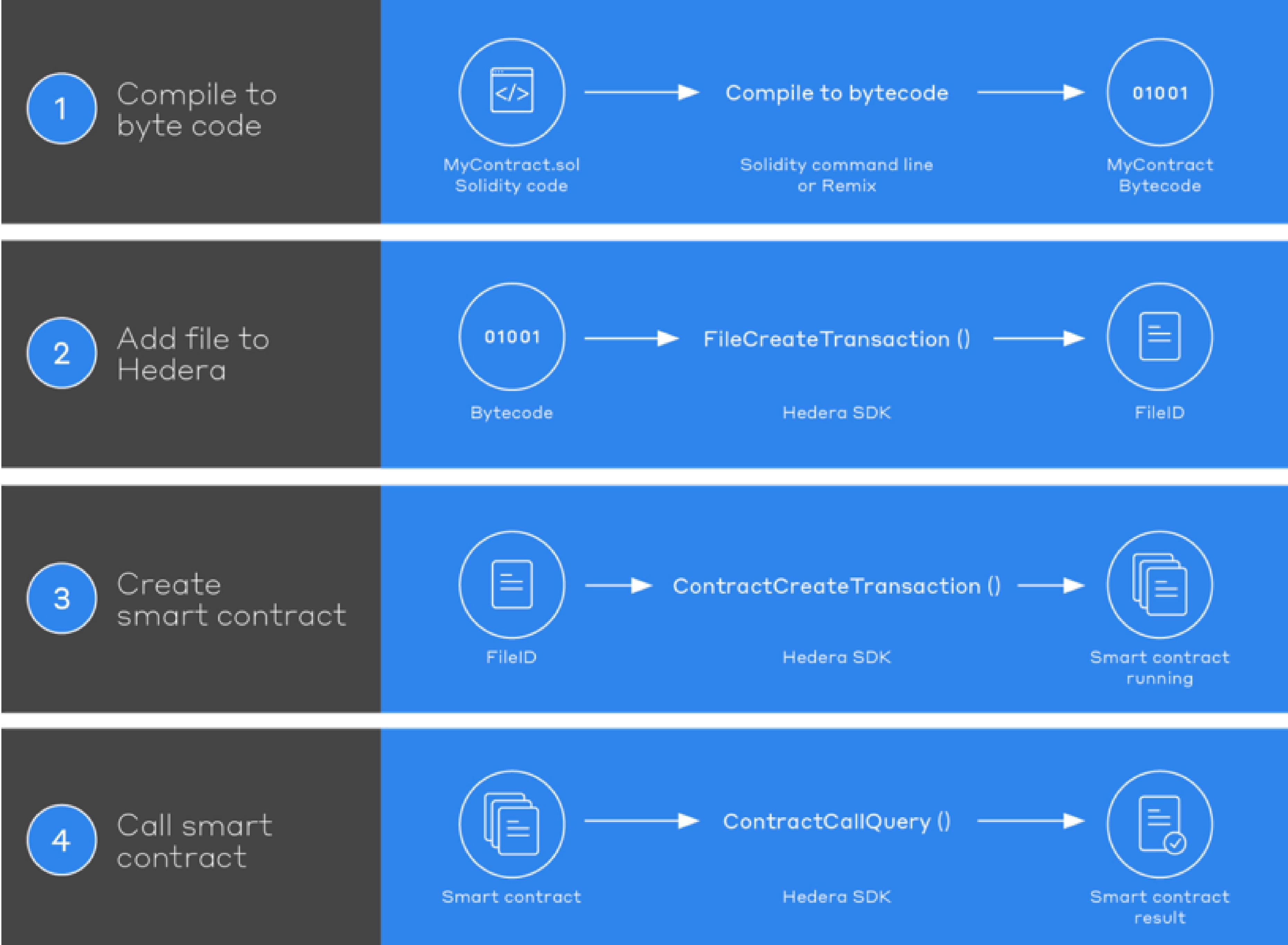
hedera.com











Solidity is a contract-oriented, high-level programming language

Contracts and inheritance similar to OOP
libraries

Contracts are compiled and output:

- Bytecode (EVM instructions)
- Application Binary Interface (ABI) definition (contract's interface e.g., inputs, return values, etc.)

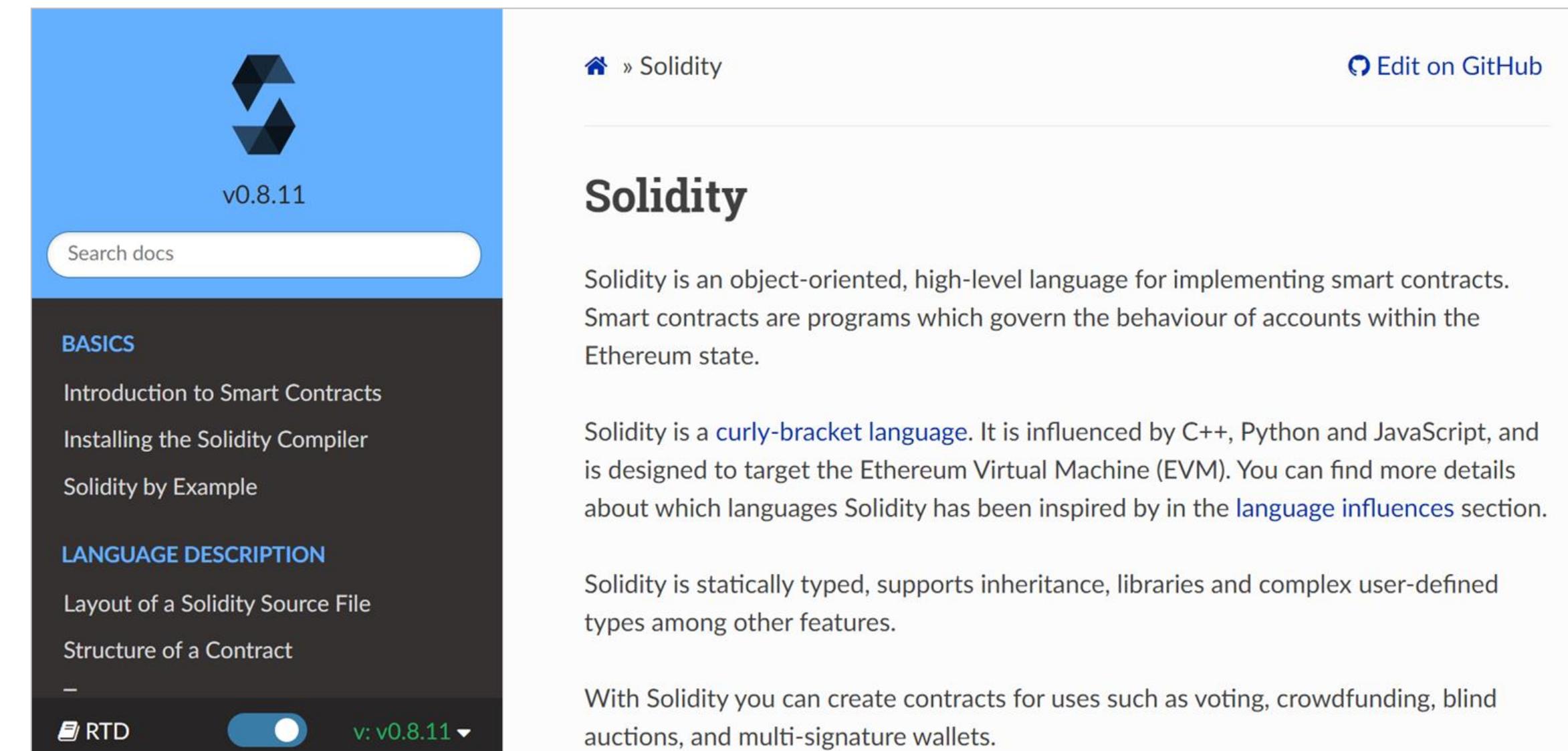
ABI Definition

```
[  
 {  
   "inputs": [  
     {  
       "internalType": "string",  
       "name": "_name",  
       "type": "string"  
     },  
     {  
       "internalType": "uint256",  
       "name": "_mobileNumber",  
       "type": "uint256"  
     }  
   ],  
   "name": "addMobileNumber",  
   "outputs": [],  
   "stateMutability": "nonpayable",  
   "type": "function"  
 },  
 {  
   "inputs": [  
     {  
       "internalType": "string",  
       "name": "_name",  
       "type": "string"  
     }  
   ],  
   "name": "getMobileNumber",  
   "outputs": [  
     {  
       "internalType": "uint256",  
       "name": "mobileNumber",  
       "type": "uint256"  
     }  
   ]  
 }]
```

Bytecode

```
"data": {  
   "bytecode": {  
     "functionDebugData": {},  
     "generatedSources": [],  
     "linkReferences": {},  
     "object": "608060405234801561001057600  
     "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTO  
     "sourceMap": "75:335:0:-:0;;;;;;;"  
   }  
 }
```

Solidity documentation



The screenshot shows the official Solidity documentation website. The header features the Solidity logo and the version v0.8.11. Below the header is a search bar labeled "Search docs". The main content area is divided into two columns. The left column contains sections titled "BASICS" (with links to "Introduction to Smart Contracts", "Installing the Solidity Compiler", and "Solidity by Example") and "LANGUAGE DESCRIPTION" (with links to "Layout of a Solidity Source File" and "Structure of a Contract"). The right column contains descriptive text about Solidity, mentioning it is an object-oriented, high-level language for implementing smart contracts, influenced by C++, Python, and JavaScript, and designed to target the Ethereum Virtual Machine (EVM). It also notes that Solidity is statically typed, supports inheritance, libraries, and complex user-defined types. At the bottom of the page, there is a note about creating contracts for various applications like voting, crowdfunding, blind auctions, and multi-signature wallets.

Ethereum virtual machine (EVM) executes compiled smart contract (bytecode)

Executing smart contract functions costs **gas**, which is the unit of account for execution costs

Gas reflects the cost necessary to pay for computing resources consumed by execution of smart contract

Each instruction in EVM and persistent storage write cost gas

Gas is measured and paid during the execution of each contract

Hedera processes up to:

- 15 million gas/second network-wide
- 4 million gas/second per contract call



Total Gas (non-Hedera Service transaction) = Intrinsic Gas + EVM Operation Gas

Total Gas (Hedera Service transaction) = Intrinsic Gas + EVM Operation Gas + Hedera Service Gas

Contracts in Solidity are like classes in OOP and they contain data and functions

Persistent data is stored in state variables

Functions can modify these variables

Contract execution is paid with gas

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

pragma solidity >=0.7.0 <0.9.0;

contract Storage {
    uint256 number;

    function store(uint256 num) public {
        number = num;
    }

    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Functions can have different types of visibility:

- Public
- Private
- External
- Internal

Function with a uint256 parameter and no return value:

```
function store(uint256 num) public {
    number = num;
}
```

Function with no parameter and a return value:

```
function retrieve() public view returns (uint256){
    return number;
}
```

<https://solidity-by-example.org/>

Solidity is a statically typed language, so the **type** of each variable (state and local) must be specified

Address: Holds a 20-byte value

Integers: Signed and unsigned integers of various sizes

- *uint* and *int* are aliases for *uint256* and *int256*, respectively

Mappings: Declared using the syntax

```
mapping(KeyType => ValueType) VariableName
```

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

pragma solidity >=0.7.0 <0.9.0;

contract MyFirstContract {

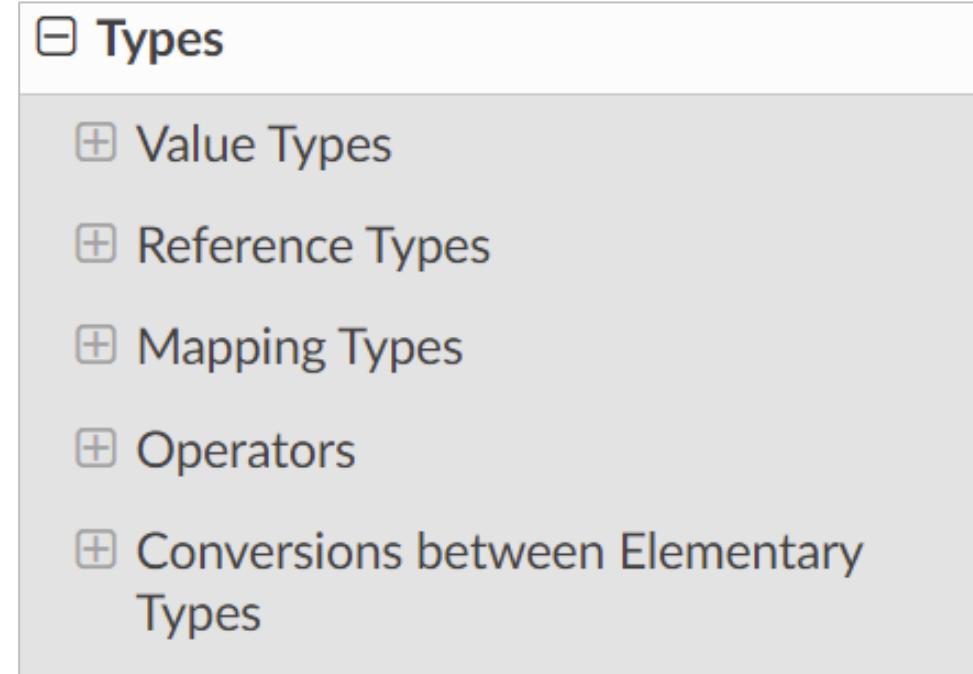
    mapping (string => uint) public phoneNumbers;

    function addMobileNumber(string memory _name, uint _mobileNumber) public {
        phoneNumbers[_name] = _mobileNumber;
    }

    function getMobileNumber(string memory _name) public view returns (uint) {
        return phoneNumbers[_name];
    }
}
```

Other types include:

- String
- Booleans
- Byte arrays
- Enums & Structs



Types

Solidity is a statically typed language, which means that the type of each variable (state and local) needs to be specified. Solidity provides several elementary types which can be combined to form complex types.

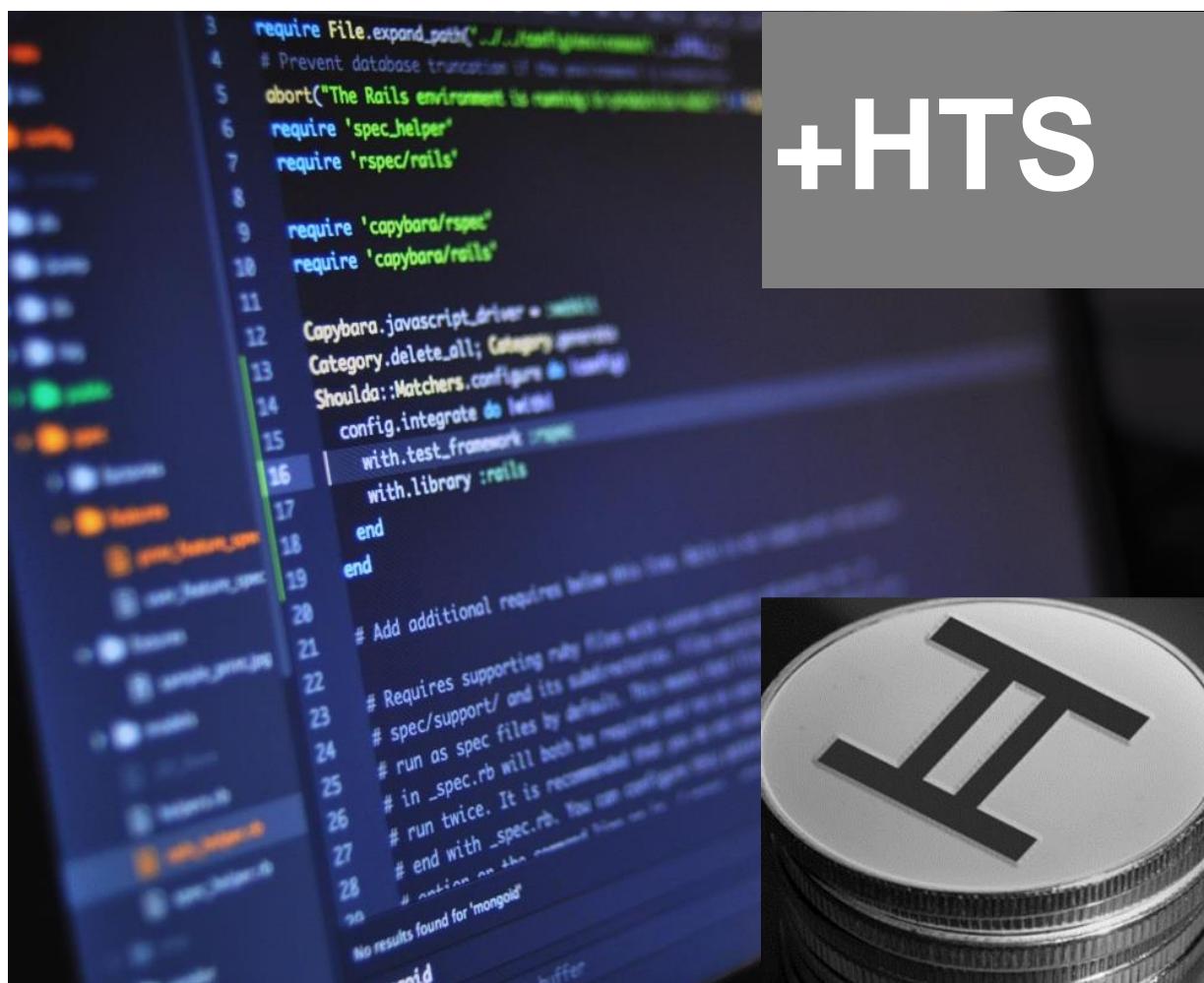
In addition, types can interact with each other in expressions containing operators. For a quick reference of the various operators, see [Order of Precedence of Operators](#).

[Read more about types in Solidity](#)

With dApp days, you will learn how to build and deploy applications on Hedera!



Section 1:
Introduction to
Web 3 and Hedera

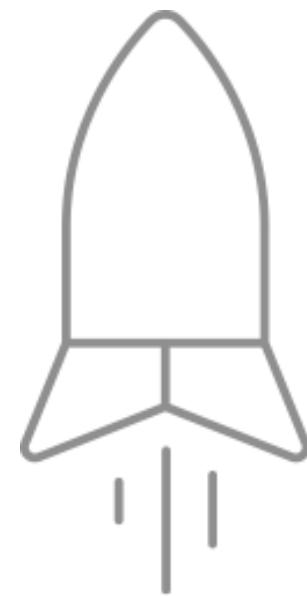


Section 2:
Build on Hedera



Section 3:
Enter NFT Giveaway
and Next Steps

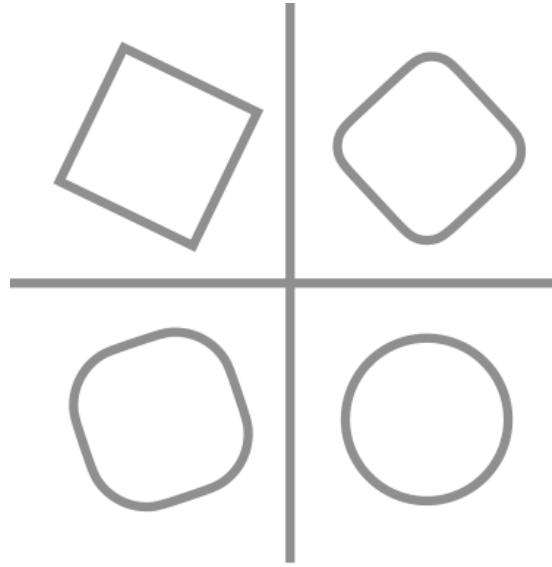
Hedera helps you meet strict requirements in the following areas...



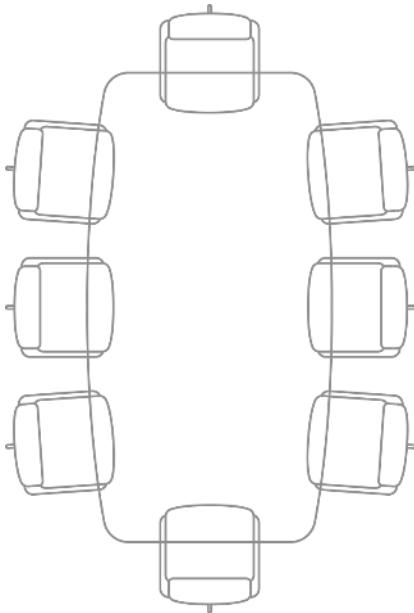
Performance



Security



Stability

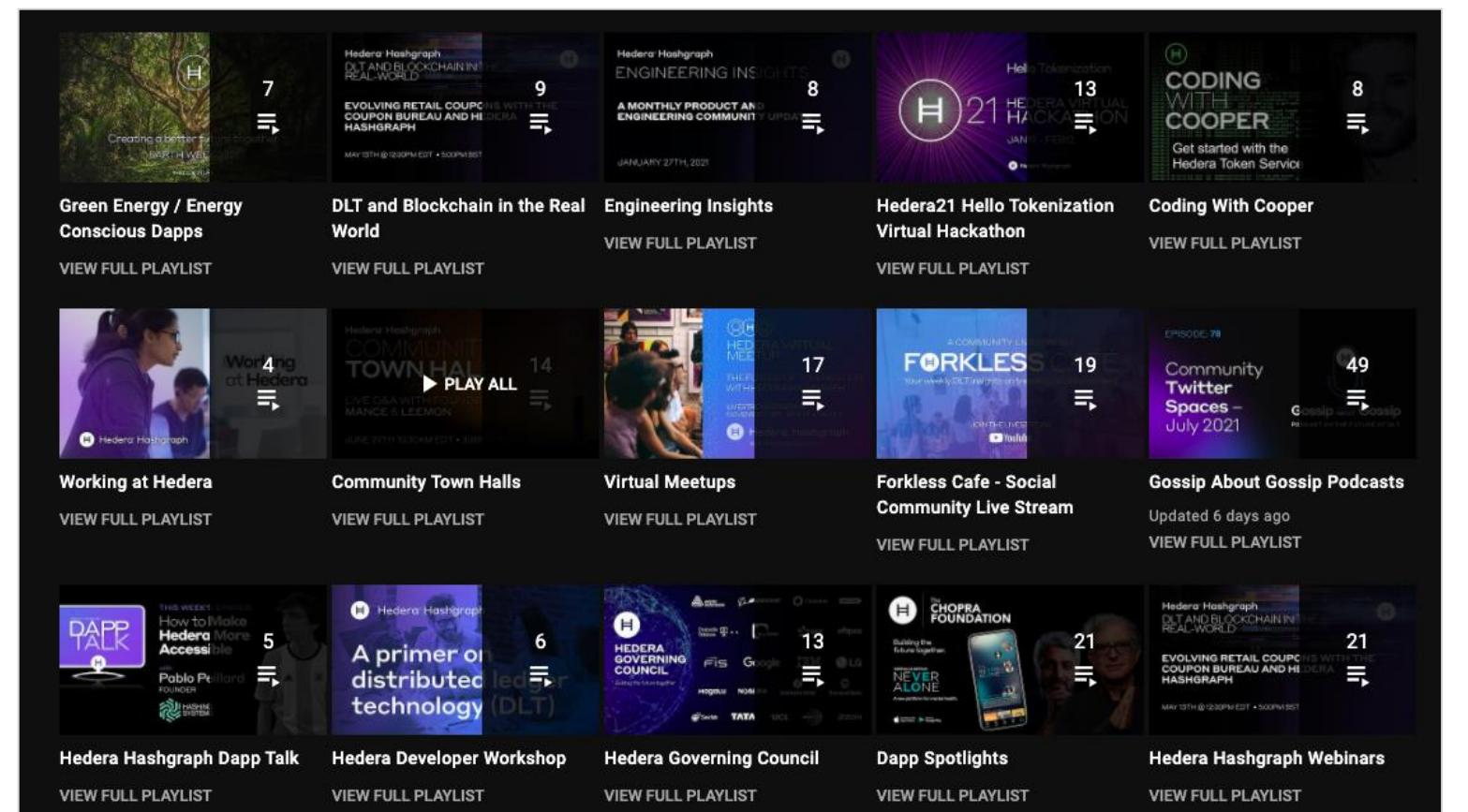


Governance

Continue learning with tutorials, videos, and conversations!

The screenshot shows the 'get-started' page of the Hedera Hashgraph website. It features a navigation bar with links to Network, Devs, Use cases, HBAR, Governance, and About. Below the navigation is a 'GET STARTED' button and a search bar. The main content area is titled 'For developers' and includes three sections: 'Quickstart', 'Tutorials', and 'Starter projects'. Each section has a brief description and a 'GET STARTED' button.

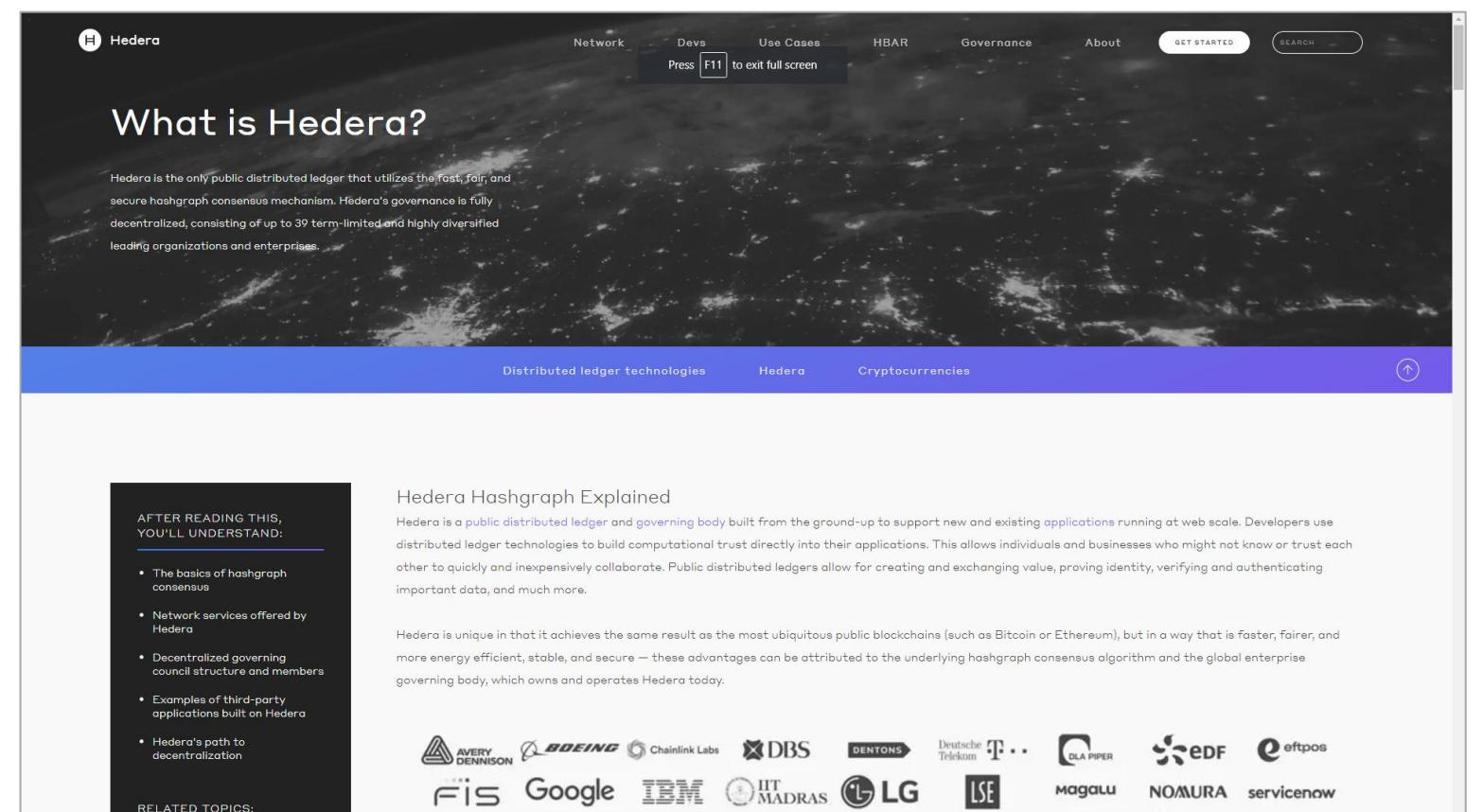
hedera.com/get-started



[Hedera YouTube Channel](#)

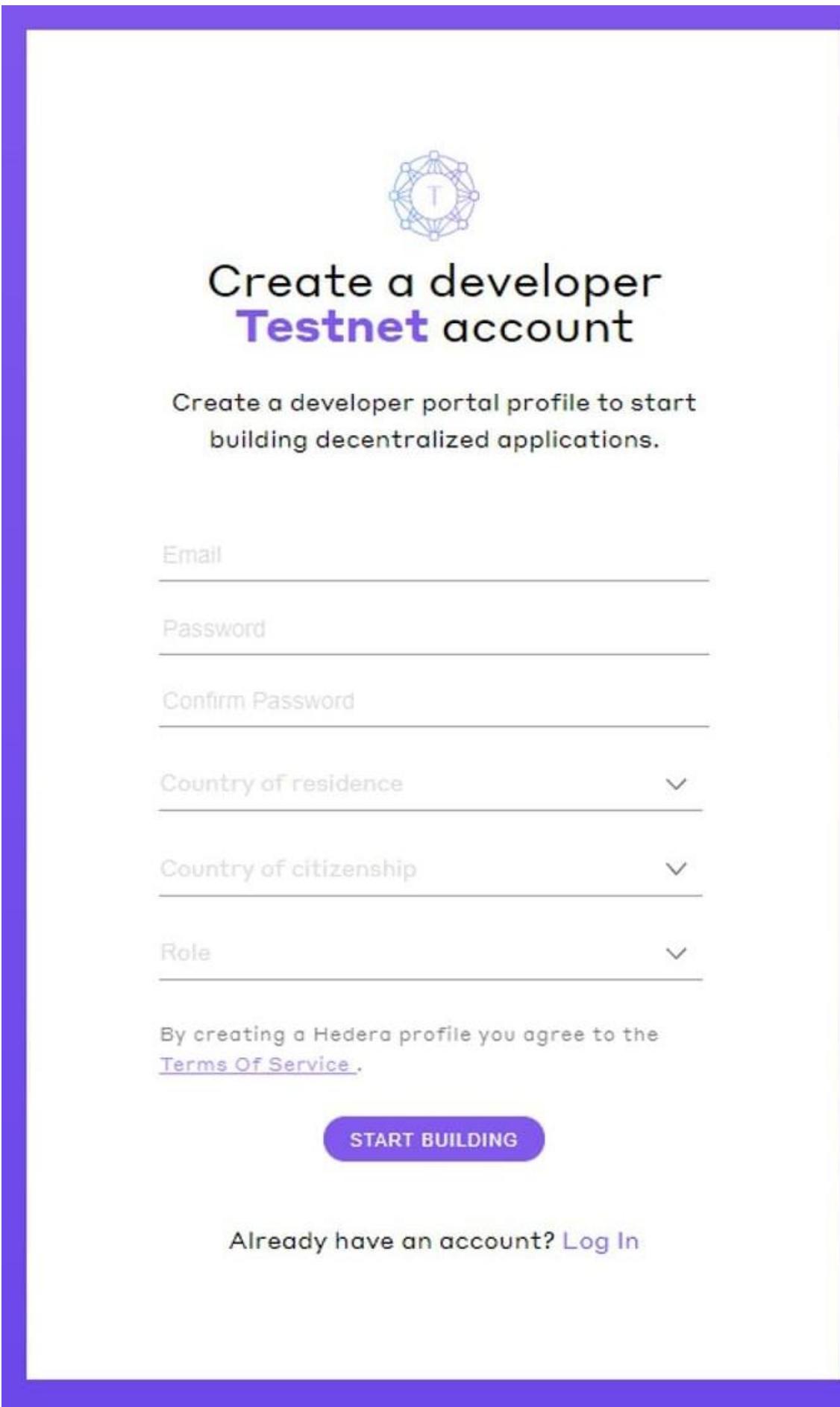
The screenshot shows the Hedera Developer Discord server interface. On the left, there is a sidebar with categories like WELCOME, DEV TALK, and PARTNER PROGRAM, each containing several channels. A message from a user named 'KenTheJr' is displayed, dated February 3, 2018. The message discusses the properties of Hashgraph, mentioning its fast, secure, and fair characteristics. It also links to various documents and白皮书 (whitepapers) such as the Whitepaper, How It Works, Hashgraph Overview, and Hashgraph & Sybil Attacks.

[Join Developer Discord](#)

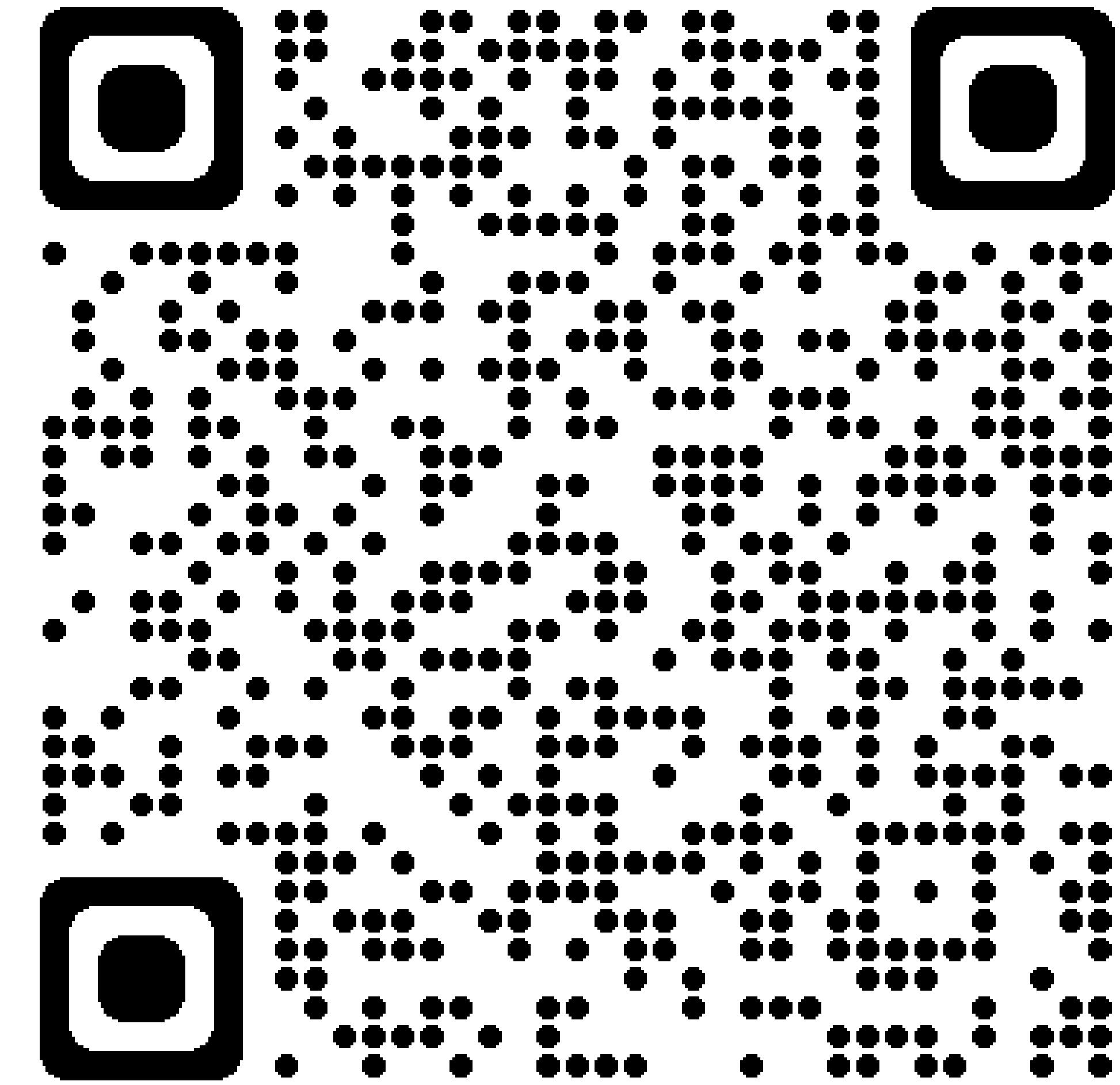


[Hedera Learning Center](#)

Learn more and start building web 3, but first get a testnet account!



The screenshot shows the 'Create a developer Testnet account' page. It features a logo with a stylized 'T' inside a hexagonal grid. The main heading is 'Create a developer **Testnet** account'. Below it, a sub-instruction reads 'Create a developer portal profile to start building decentralized applications.' There are four input fields: 'Email', 'Password', 'Confirm Password', and 'Role'. Each field has a dropdown arrow icon to its right. Below these fields is a note: 'By creating a Hedera profile you agree to the [Terms Of Service](#)'. At the bottom left is a purple 'START BUILDING' button, and at the bottom right is a link 'Already have an account? [Log In](#)'.



<https://portal.hedera.com/register>

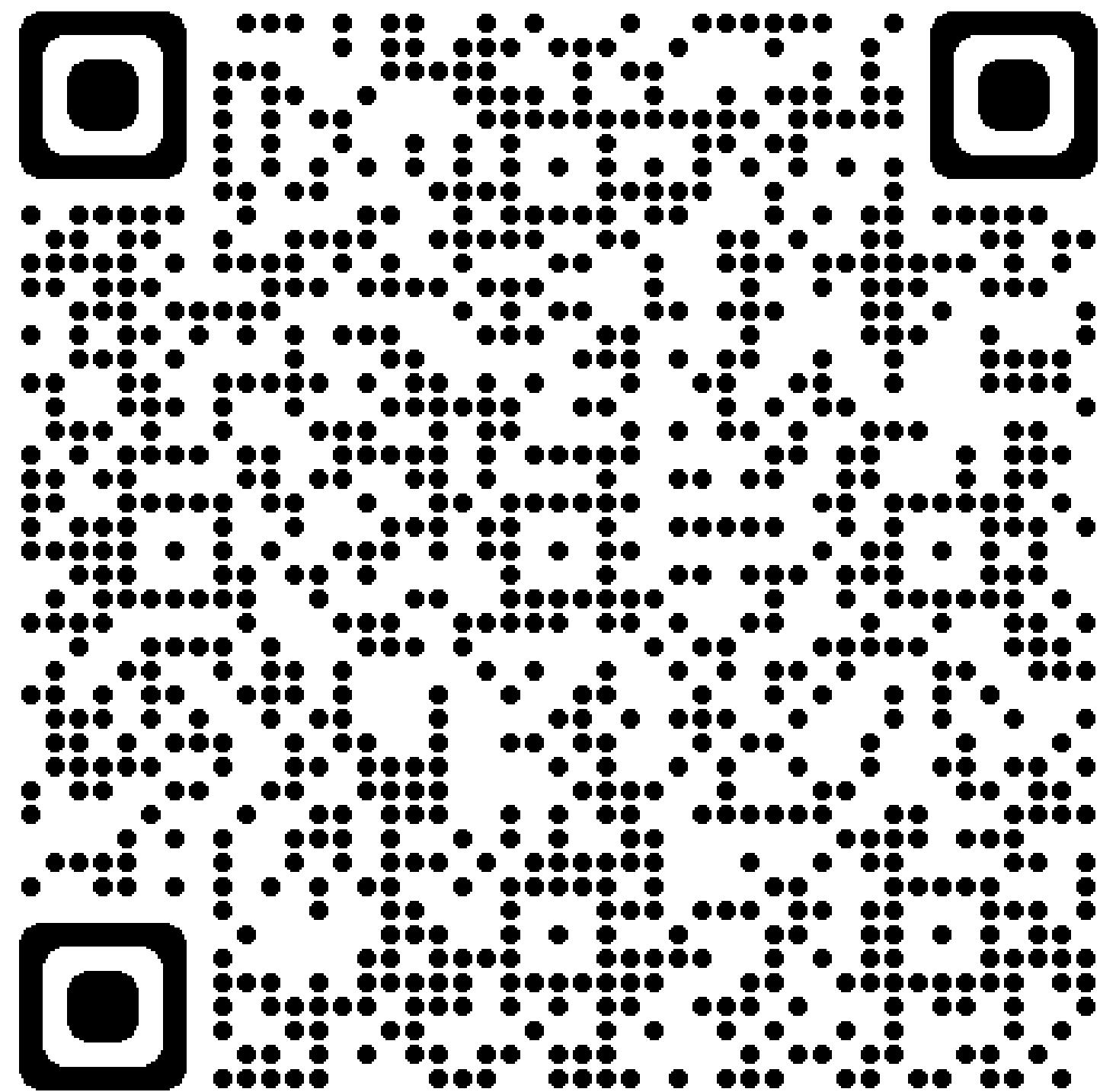
NFTT

GIVEAWAY!!!

A collage background featuring large, stylized question marks cut out of gold and white paper. In the upper right corner, a person's hands are shown holding a pen and a small notebook, appearing to write or draw. The overall theme is inquiry and feedback.

FEEDBACK AND OPEN Q&A

tinyurl.com/dappdays-survey



FEEDBACK AND OPEN Q&A



Hedera dApp Day

Arizona State University

ED MARQUEZ, DEVELOPER RELATIONS, SWIRLDS LABS

 @ed_marquez

 /in/ed-marquez

hello future