



Hedera dApp Days

Session 2: Smart Contracts on Hedera

Gehrig Kunz, Waylon Jepsen, Ed Marquez

DevRel Team



/ed-marquez



@ed__marquez

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



**Session 1:
Introduction**




**Session 2:
Smart Contracts with
Hedera Tokens**



**Session 3:
Conclusions
and Next Steps**

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



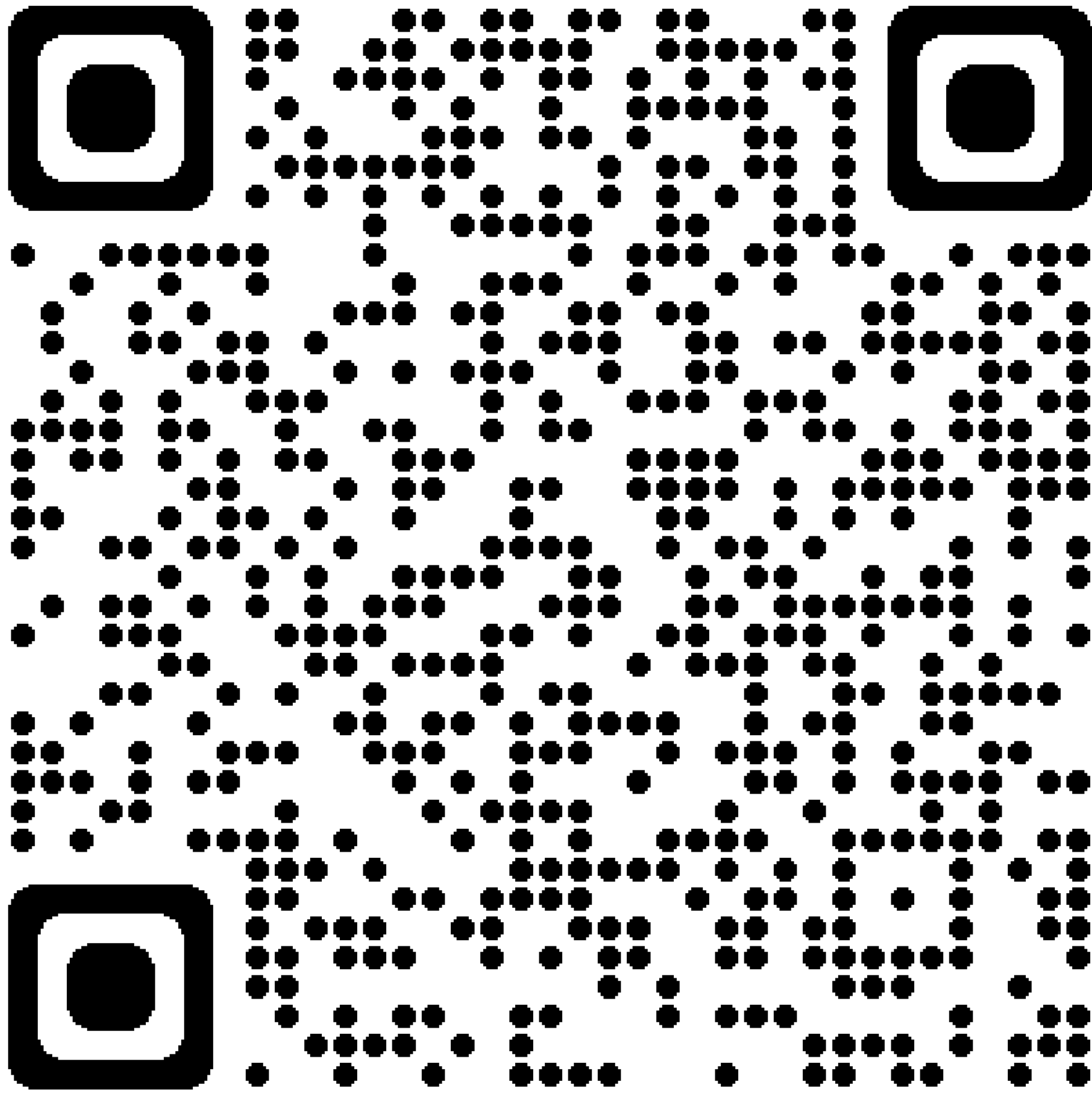
Create a developer Testnet account

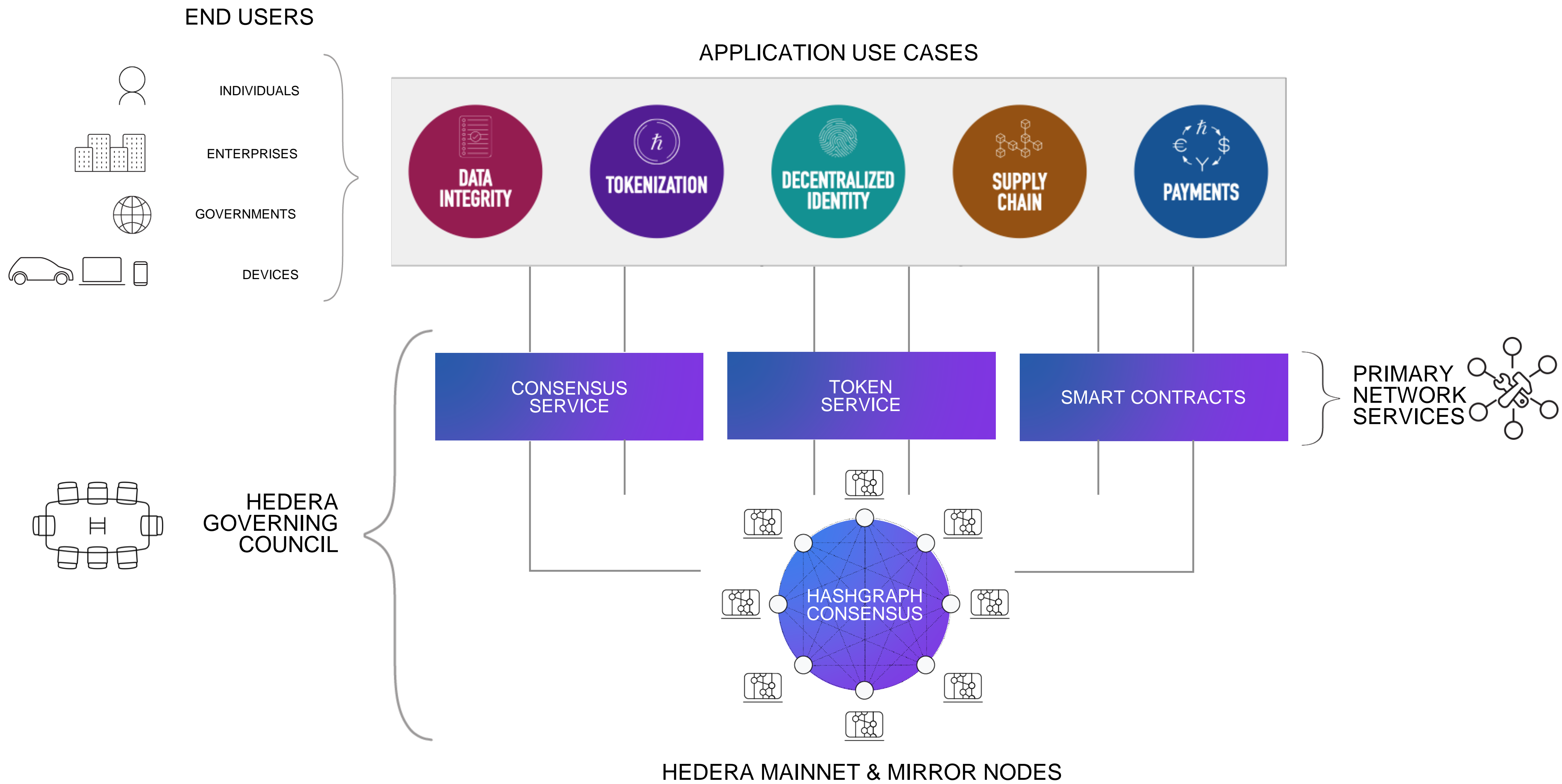
Create a developer portal profile to start building decentralized applications.

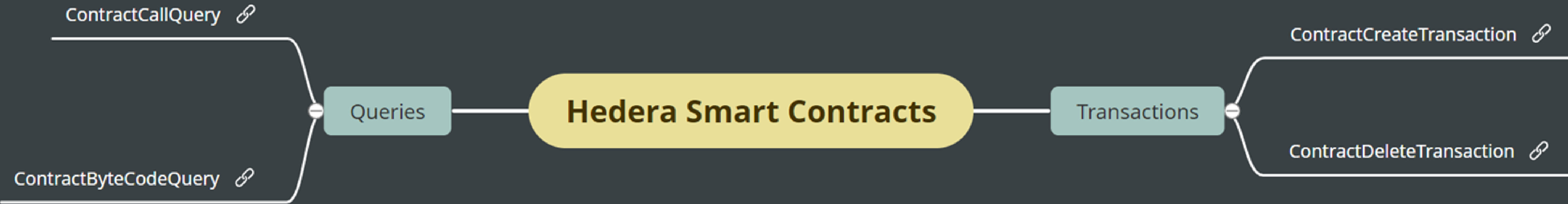
By creating a Hedera profile you agree to the [Terms Of Service](#).

START BUILDING

Already have an account? [Log In](#)







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

Contracts and inheritance similar to OOP libraries

Contracts are compiled and output:

- Bytecode (EVM instructions)
- 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": [
      {

```

Bytecode

```
"data": {
  "bytecode": {
    "functionDebugData": {},
    "generatedSources": [],
    "linkReferences": {},
    "object": "60806040523480156100105760",
    "opcodes": "PUSH1 0x80 PUSH1 0x40 MST",
    "sourceMap": "75:335:0:-:0;;;;;;;;;;;"
```

Solidity documentation



v0.8.11


BASICS



- Introduction to Smart Contracts
- Installing the Solidity Compiler
- Solidity by Example

LANGUAGE DESCRIPTION

- Layout of a Solidity Source File
- Structure of a Contract

—

 RTD ☒ v: v0.8.11 ▼

 » Solidity  [Edit on GitHub](#)

Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.

Solidity is a [curly-bracket language](#). It is influenced by C++, Python and JavaScript, and is designed to target the Ethereum Virtual Machine (EVM). You can find more details about which languages Solidity has been inspired by in the [language influences](#) section.

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

With Solidity you can create contracts for uses such as 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;

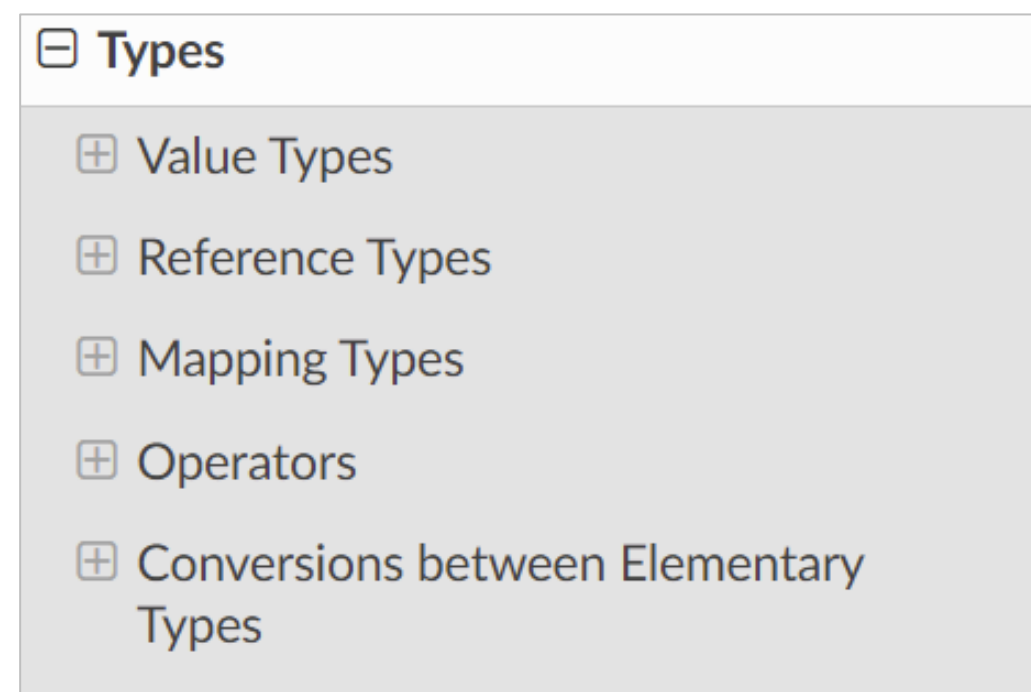
    function addMobileNumber(string memory _name, uint _mobileNumber) public {
        phoneNumbers[_name] = _mobileNumber;
    }

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

}
```

Other types include:

- String
- Booleans
- Byte arrays
- Enums



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](#)

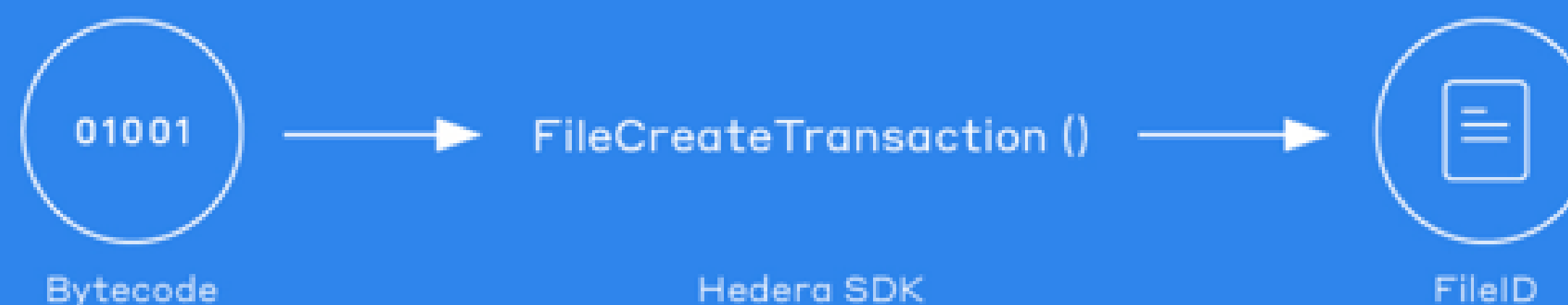


LET'S BUILD A COOL DAPP ON HEDERA!

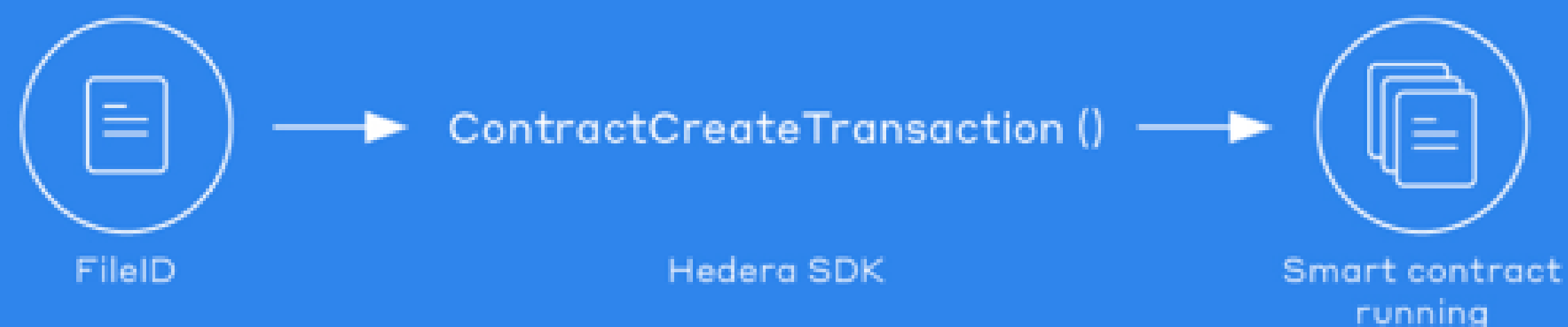
1 Compile to
byte code



2 Add file to
Hedera



3 Create
smart contract

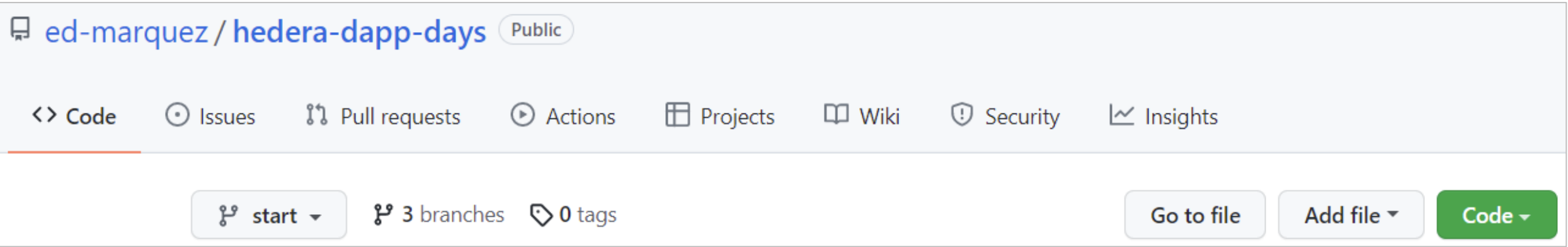


4 Call smart
contract



BUILD A DAPP ON HEDERA (1/3)

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



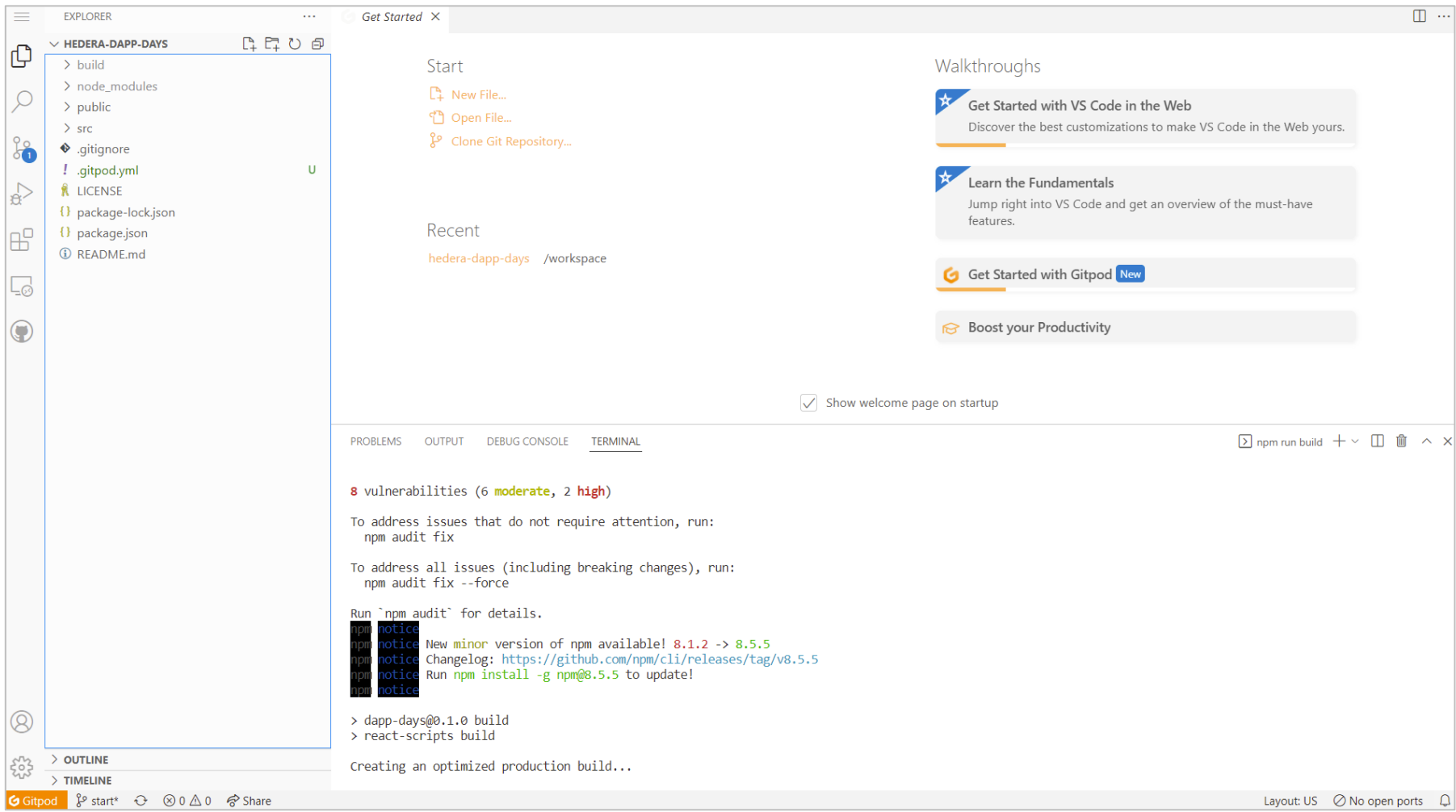
2. **Fork** the repository...



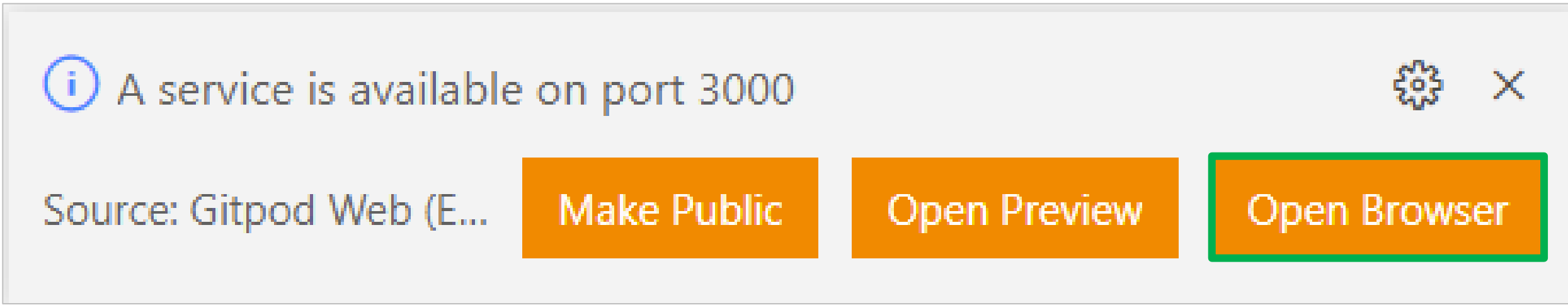
... and **open** your fork in Gitpod using a link that looks as follows:

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

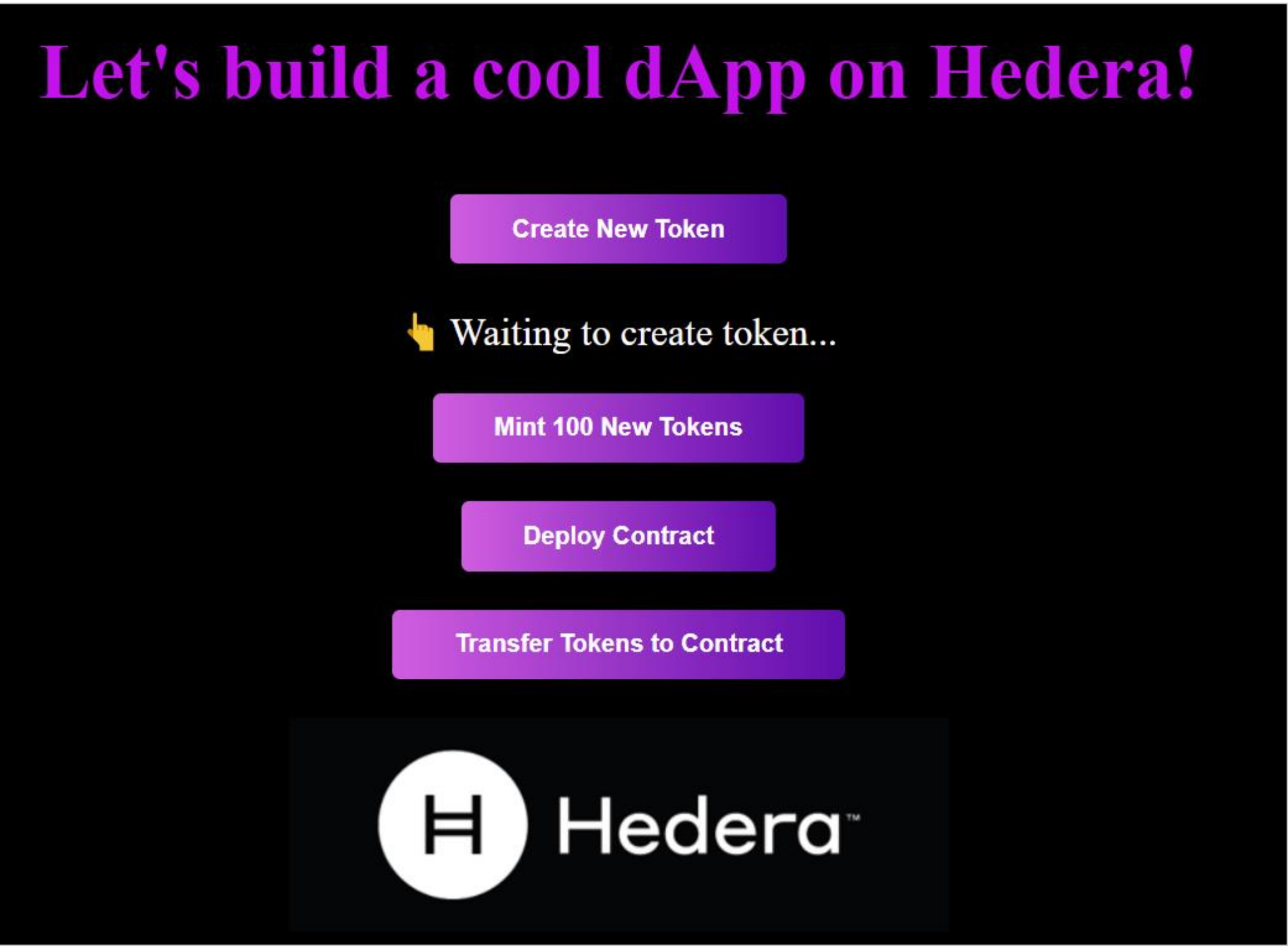
Where **<GITHUB_UN>** is your GitHub username



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



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

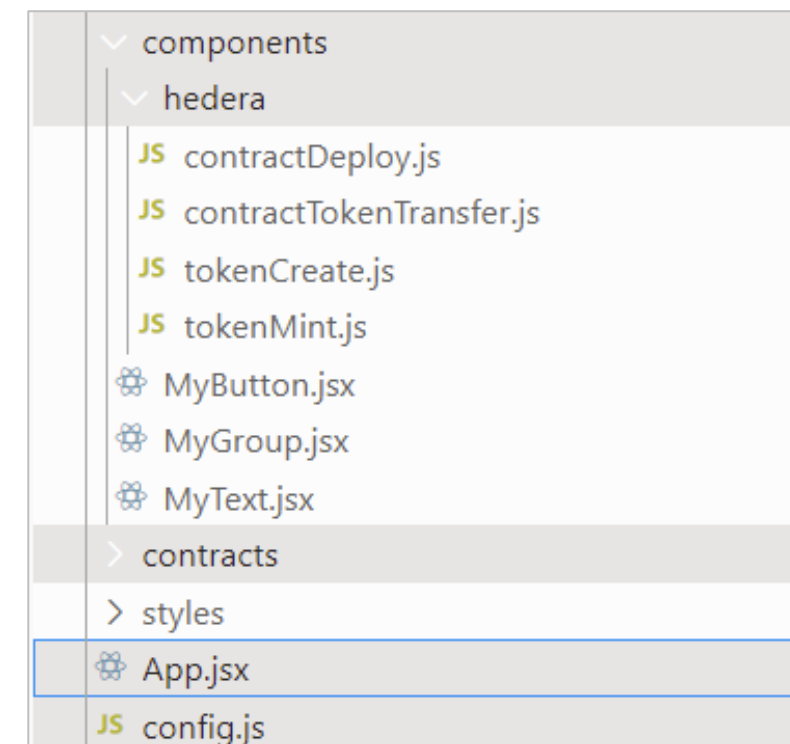


BUILD A DAPP ON HEDERA (2/3)

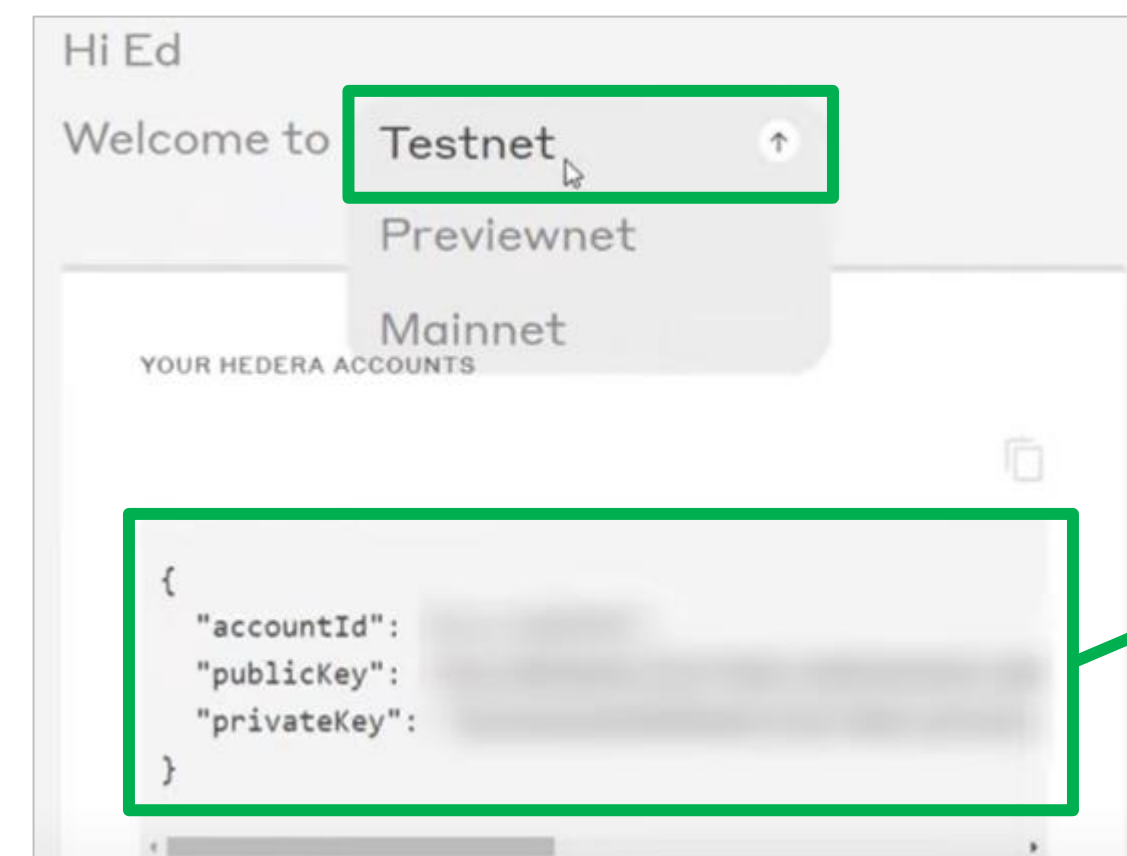
3. If you're not too familiar with React, explore the project in Gitpod. Most of the relevant files for dAppDays are in the **src** directory

Files and directories of interest include:

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



4. Log into the [Hedera Portal](#), select **Testnet**, and copy/paste your testnet account credentials (values only) to the **config.js** file in the Gitpod project



```
src > JS config.js > ...
1  // TESTNET CREDENTIALS
2  const operator = {
3    id: "0.0.2...",
4    pb_key: "302...",
5    pvkey: "302...",
6  };
7
8  export default operator;
9
```

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. tokenCreate.js

```
async function tokenCreateFcn() {
  const operatorId = AccountId.fromString(operator.id);
  const operatorKey = PrivateKey.fromString(operator.pvkey);
  const client = Client.forTestnet().setOperator(operatorId, operatorKey);

  console.log("- Creating token");

  const tokenCreateTx = new TokenCreateTransaction()
    .setTokenName("dAppDayToken")
    .setTokenSymbol("DDT")
    .setTreasuryAccountId(operatorId)
    .setInitialSupply(100)
    .setDecimals(0)
    .setSupplyKey(operatorKey)
    .freezeWith(client);
  const tokenCreateSign = await tokenCreateTx.sign(operatorKey);
  const tokenCreateSubmit = await tokenCreateSign.execute(client);
  const tokenCreateRec = await tokenCreateSubmit.getRecord(client);
  const tId = tokenCreateRec.receipt.tokenId;
  const supply = tokenCreateTx._initialSupply.low;

  return [tId, supply];
}
```

5b. tokenMint.js

```
async function tokenMintFcn(tId) {
  const operatorId = AccountId.fromString(operator.id);
  const operatorKey = PrivateKey.fromString(operator.pvkey);
  const client = Client.forTestnet().setOperator(operatorId, operatorKey);

  console.log("- Minting new tokens!");
  const tokenMintTx = new TokenMintTransaction().setTokenId(tId).setAmount(100).freezeWith(client);
  const tokenMintSign = await tokenMintTx.sign(operatorKey);
  const tokenMintSubmit = await tokenMintSign.execute(client);
  const tokenMintRec = await tokenMintSubmit.getRecord(client);
  const supply = tokenMintRec.receipt.totalSupply;

  return supply;
}
```

BUILD A DAPP ON HEDERA (3/3)

5c. 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 contractDeployFcn(tokenId) {
    const operatorId = AccountId.fromString(operator.id);
    const operatorKey = PrivateKey.fromString(operator.pvkey);
    const client = Client.forTestnet().setOperator(operatorId, operatorKey);

    // STEP 1 =====
    console.log(`STEP 1 =====`);
    // const bytecode = "../contracts/AssoTransHTS_sol_AssoTransHTS.bin";
    const bytecode = "6080...";
    console.log(`- Done`);

    // STEP 2 =====
    console.log(`STEP 2 =====`);
    //Create a file on Hedera and store the hex-encoded bytecode
    const fileCreateTx = new
FileCreateTransaction().setKeys([operatorKey]).setContents(bytecode).freezeWith(client);
    const fileCreateSign = await fileCreateTx.sign(operatorKey);
    const fileSubmit = await fileCreateSign.execute(client);
    const fileCreateRx = await fileSubmit.getReceipt(client);
    const bytecodeFileId = fileCreateRx.fileId;
    console.log(`- The smart contract bytecode file ID is: ${bytecodeFileId}`);

    // STEP 3 =====
    console.log(`STEP 3 =====`);
    // Create the smart contract
    const contractInstantiateTx = new ContractCreateTransaction()
        .setBytecodeFileId(bytecodeFileId)
        .setGas(3000000)
        .setConstructorParameters(new ContractFunctionParameters().addAddress(tokenId.toSolidityAddress()));
    const contractInstantiateSubmit = await contractInstantiateTx.execute(client);
    const contractInstantiateRx = await contractInstantiateSubmit.getReceipt(client);
    const cId = contractInstantiateRx.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;
}
```

5d. contractTokenTransfer.js

```
async function contractTokenTransferFcn(tokenId, contractId) {
    const operatorId = AccountId.fromString(operator.id);
    const operatorKey = PrivateKey.fromString(operator.pvkey);
    const client = Client.forTestnet().setOperator(operatorId, operatorKey);

    // // STEP 4 =====
    console.log(`STEP 4 =====`);
    //Execute a contract function (transfer)
    const contractExecTx2 = new ContractExecuteTransaction()
        .setContractId(contractId)
        .setGas(3000000)
        .setFunction("tokenAssoTrans", new ContractFunctionParameters().addInt64(50))
        .freezeWith(client);
    const contractExecSign2 = await contractExecTx2.sign(operatorKey);
    const contractExecSubmit2 = await contractExecSign2.execute(client);
    const contractExecRx2 = await contractExecSubmit2.getReceipt(client);

    console.log(`- Token transfer from Operator to contract: ${contractExecRx2.status.toString()}`);

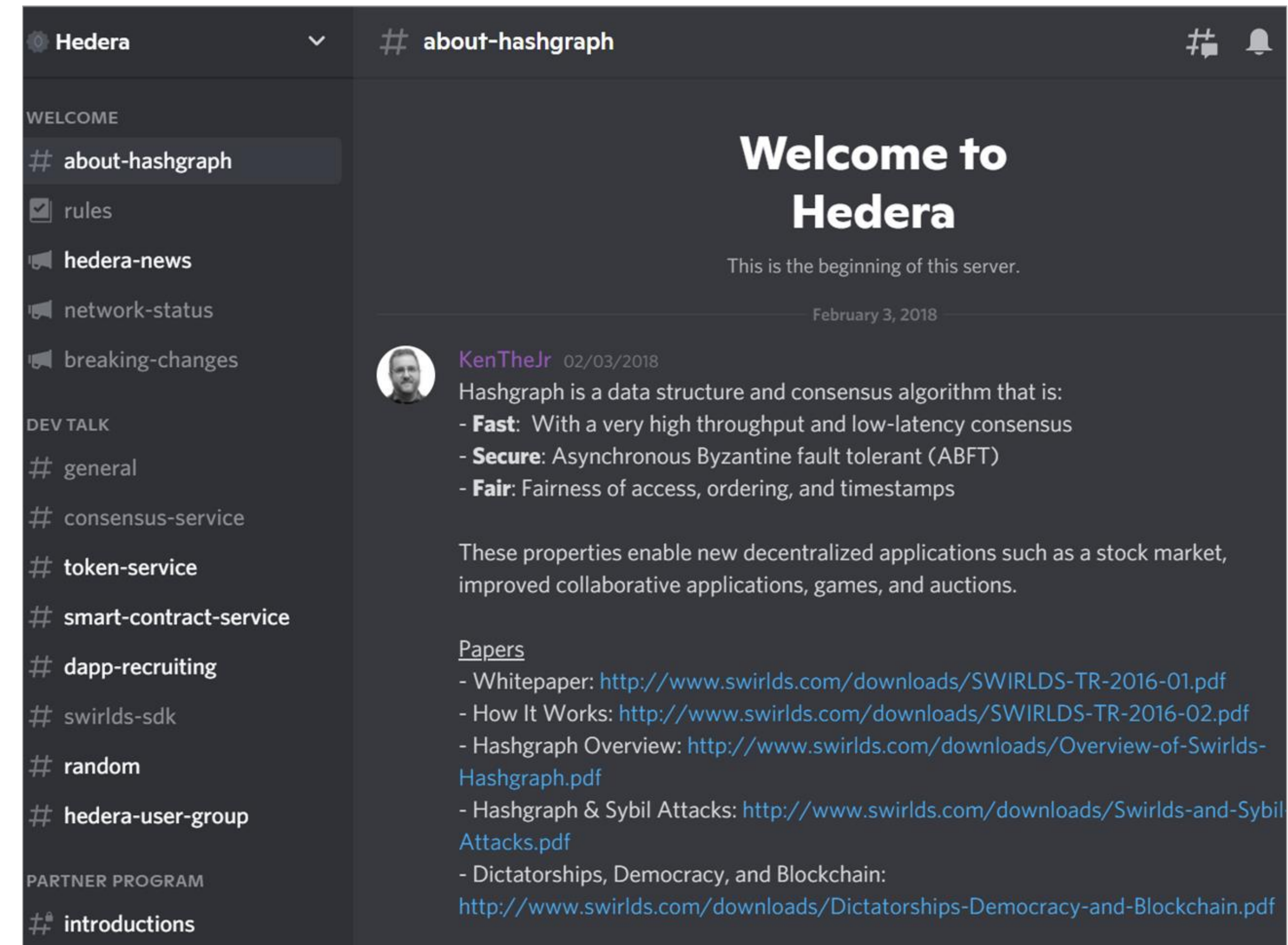
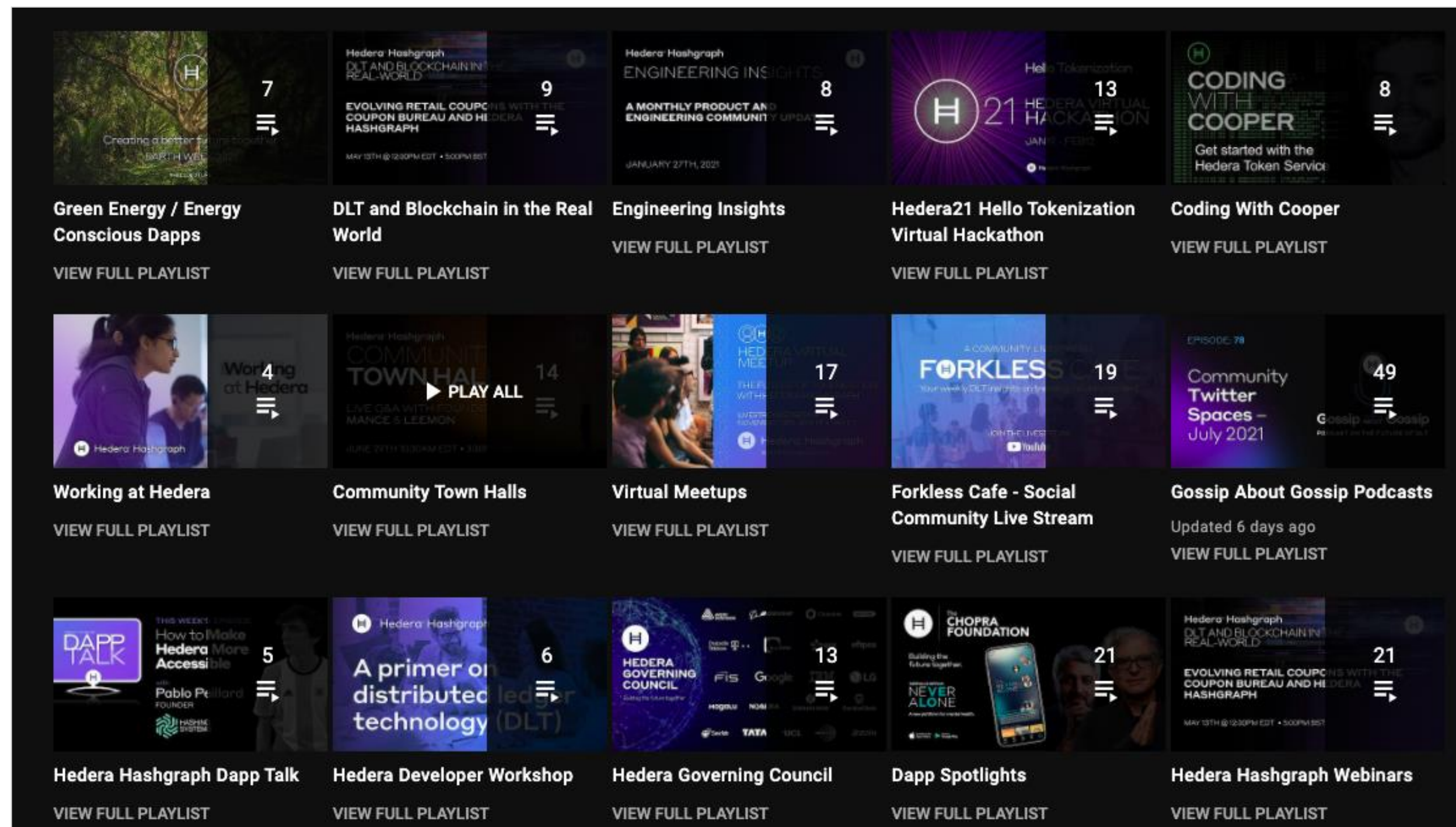
    const bCheck = await new AccountBalanceQuery().setAccountId(operatorId).execute(client);
    console.log(`- Operator balance: ${bCheck.tokens._map.get(tokenId.toString())} units of token ${tokenId}`);

    const cCheck = await new ContractInfoQuery().setContractId(contractId).execute(client);
    console.log(
        `- Contract balance: ${
            cCheck.tokenRelationships._map.get(tokenId.toString()).balance.low
        } units of token ${tokenId}`
    );
}
```

6. Next steps for you...

- Explore your transactions using a [network explorer](#) and/or the [mirror node REST API](#)
- Add more functionality to your application. Here are a few ideas:
 - Send *hbar* to the contract
 - Exchange *hbar* and HTS tokens via the contract

Continue learning with tutorials and with others on Discord



Hedera YouTube Channel

Join the Developer Discord



Hedera™ Hashgraph

THE TRUST LAYER OF THE INTERNET



/ed-marquez



@ed__marquez

hedera.com

©2020 Hedera Hashgraph, LLC. All rights reserved.