Lesson 12 - Starknet JS

Introduction

Documentation
https://www.starknetjs.com/docs/API/provider

This is modeled on libraries such as Web3.js

The main areas are

- Provider API - connecting to starknet
- Account API - connection with an account
- Signer API - allows signatures
- Contract API - an object representing a contract
- Utils API - Utility methods

Installation

```
npm install starknet@next
```

Provider API

You can create a provider with

```
const provider = new starknet.Provider()
```

or if you want specify the network

```
const provider = new starknet.Provider({
sequencer: {
        network: 'mainnet-alpha' // or 'goerli-alpha'
        }
})
```

To interact with a contract we use the provider we set up

Provider methods

callContract

```
provider.callContract(call [ , blockIdentifier ]) => _Promise
```

The call object has the following structure

- call.contractAddress - Address of the contract
- call.entrypoint - Entrypoint of the call (method name)
- call.calldata - Payload for the invoking method

Response

```
{
result: string[];
}
```

getTransactionReceipt

```
provider.getTransactionReceipt(txHash) => _Promise
```

Response

```
{
transaction_hash: string;
status: 'NOT_RECEIVED' | 'RECEIVED' | 'PENDING' | 'ACCEPTED_ON_L2' |
'ACCEPTED_ON_L1' | 'REJECTED';
actual_fee?: string;
status_data?: string;
messages_sent?: Array<MessageToL1>;
events?: Array<Event>;
l1_origin_message?: MessageToL2;
}
```

Deploy Contract

```
provider.deployContract(payload [ , abi ]) => _Promise
```

Response

```
{
transaction_hash: string;
contract_address?: string;
};
```

Wait For Transaction

`provider.waitForTransaction(txHash [ , retryInterval]) => Promise < void >`

Wait for the transaction to be accepted on L2 or L1.

Other methods

- getBlock
- getClassAt
- getStorageAt
- getTransaction
- declareContract
- waitForTransaction

A useful library is get-starknet which provides connection methods.
If you are connecting with a wallet use the connect method from the get-starknet module

```
const starknet = await connect()
// connect to the wallet
await starknet?.enable({ starknetVersion: "v4" })
const provider = starknet.account
```

Signer API

The Signer API allows you to sign transactions and messages

You can generate a key pair by using the utility functions

`ec.genKeyPair()`
or
`getKeyPair(private_key)`

The signer object is then created with

`new starknet.Signer(keyPair)`

You can then sign messages

```
signer.signMessage(data, accountAddress) => _Promise
```

Code Example

```
const privateKey = stark.randomAddress();
const starkKeyPair = ec.genKeyPair(privateKey);
const starkKeyPub = ec.getStarkKey(starkKeyPair);
```

Account API

The Account object extends the Provider object
To create the account object, an account contract needs to have been deployed, see below for guide to deploy an account contract.

```
const account  = new starknet.Account(Provider, address, starkKeyPair)
```

Account Properties

```
account.address =>string
```

Account Methods

```
account.getNonce() => Promise
account.estimateFee(calls [ , options ]) => _Promise
account.execute(calls [ , abi , transactionsDetail ]) => _Promise
account.signMessage(typedData) => _Promise
account.hashMessage(typedData) => _Promise
account.verifyMessageHash(hash, signature) => _Promise
account.verifyMessage(typedData, signature) => _Promise
```

See guide to creating and deploying an account

Contract

Creating the contract object

```
new starknet.Contract(abi, address, providerOrAccount)

contract.attach(address)` _for changing the address of the connected contract_

contract.connect(providerOrAccount)` _for changing the provider or account_
```

Contract Properties

```
contract.address => string
contract.providerOrAccount => ProviderInterface | AccountInterface
contract.deployTransactionHash => string | null
contract.ab => Abi
```

Contract Interaction

1. View Functions

contract.METHOD_NAME(...args [, overrides]) => Promise < Result >

The type of the result depends on the ABI.
The result object will be returned with each parameter available positionally and if the parameter is named, it will also be available by its name.

The override can identify the block : `overrides.blockIdentifier`

Code Example

```
const bal = await contract.get_balance()
```

2. Write Functions

contract.METHOD*NAME(...args [, overrides]) => _Promise < AddTransactionResponse >*

Overrides can be

- overrides.signature - Signature that will be used for the transaction
- overrides.maxFee - Max Fee for the transaction
- overrides.nonce - Nonce for the transaction

Code Example

```
await contract.increase_balance(13)
```

Utils

Useful Methods

- toBN

  `toBN(number: BigNumberish, base?: number | 'hex'): BN`

  Converts BigNumberish to BN.
  Returns a BN.

- uint256ToBN

  `uint256ToBN(uint256: Uint256): BN`

  Function to convert `Uint256` to `BN` (big number), which uses the `bn.js` library.

- getStarkKey

  `getStarkKey(keyPair: KeyPair): string`

  Public key defined over a Stark-friendly elliptic curve that is different from the standard Ethereum elliptic curve

- getKeyPairFromPublicKey

  `getKeyPairFromPublicKey(publicKey: BigNumberish): KeyPair`

  Takes a public key and casts it into `elliptic` KeyPair format.
  Returns keyPair with public key only, which can be used to verify signatures, but can't sign anything.

- sign

  `sign(keyPair: KeyPair, msgHash: string): Signature`

  Signs a message using the provided key.
  keyPair should be an KeyPair with a valid private key.
  Returns an Signature.

- verify

  `verify(keyPair: KeyPair | KeyPair[], msgHash: string, sig: Signature): boolean`

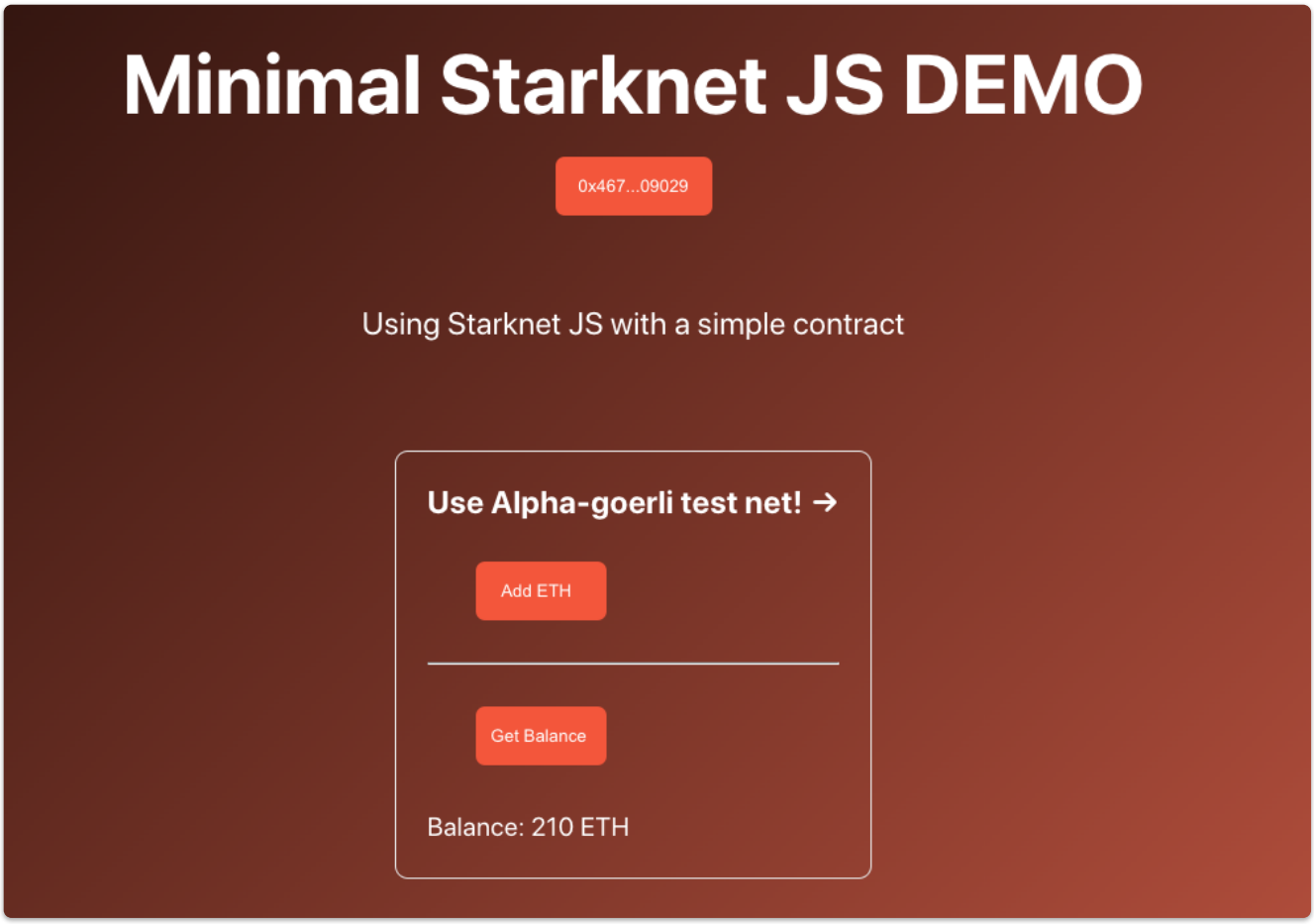
  Verifies a message using the provided key.
  keyPair should be an KeyPair with a valid public key.

sig should be an Signature.
Returns true if the verification succeeds.

Example in repo

Code

Based on tutorial from @darlingtonnnam



Links

Starknet.js workshop : https://github.com/0xs34n/starknet.js-workshop
Tutorial on medium :https://medium.com/@darlingtonnnam/an-in-depth-guide-to-getting-started-with-starknet-js-a55c04d0ccb7