[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 0 - Getting Started](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Getting Started](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Blockchain 101](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Blockchain History](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Public and Managed Blockchains](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Consensus in Distributed Networks](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Cryptography](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Self-Assessment Quiz](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Go Introduction - First Steps](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Go Basics](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Go Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Control Structures in Go](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Arrays and Slices in Go](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Standard Packages in Go](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Concurrency in Go](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Docker Introduction](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Introduction to the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[The Interchain Ecosystem](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Getting ATOM and Staking It](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[A Blockchain App Architecture](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Accounts](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Transactions](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Messages](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Modules](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Protobuf](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Multistore and Keepers](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[BaseApp](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Queries](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Events](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Context](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Testing](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Relaying with IBC](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Interchain Security](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Bridges](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Migrations](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 1 Quiz](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 2 - First Steps](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[First Steps](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Setup Your Work Environment](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Run a Node, API, and CLI](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Ignite CLI](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Store Object](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create and Save a Game Properly](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Add a Way to Make a Move](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Emit Game Information](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Record the Game Winner](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 2 Exercise](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[What is IBC?](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC Token Transfer](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC Tooling](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[What is CosmJS?](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Your First CosmJS Actions](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Compose Complex Transactions](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Learn to Integrate Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Put Your Games in Order](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Auto-Expiring Games](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Let Players Set a Wager](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Handle wager payments](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Integration tests](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Incentivize Players](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Help Find a Correct Move](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Understand IBC Denoms](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Go Relayer](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Hermes Relayer](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[CosmJS Advanced](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create Custom Objects](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Get an External GUI](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Backend Script for Game Indexing](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC Deep Dive](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[IBC Application Developer Introduction](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Create a Leaderboard Chain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Run in Production](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Prepare the Software to Run](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Prepare a Validator and Keys](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Prepare Where the Node Starts](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Prepare and Do Migrations](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Simulate Production in Docker](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Tally Player Info After Production](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Simulate a Migration in Docker](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Final Exam](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[What's Next?](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

[Continue Your Interchain Journey](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html)

Docs Version Switcher

On this page

[Creating your simple Next.js project](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#creating-your-simple-next-js-project)

[HTML elements](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#html-elements)

[Installing CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#installing-cosmjs)

[Displaying information without user input](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#displaying-information-without-user-input)

[Getting testnet tokens](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#getting-testnet-tokens)

[Detecting Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#detecting-keplr)

[Prepare Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#prepare-keplr)

[Your address and balance](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#your-address-and-balance)

[With a locally running chain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#with-a-locally-running-chain)

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#learn-to-integrate-keplr) **Learn to Integrate Keplr**



Build applications that interact with the Keplr browser extension.   
  
In this section, you will learn more about:

* Detecting Keplr.
* Getting chain information.
* Working with the user interaction flow.

CosmJS allows you to connect with [Keplr (opens new window)↗](https://chrome.google.com/webstore/detail/keplr/dmkamcknogkgcdfhhbddcghachkejeap), the widely used browser extension, to manage your private keys. In a previous section you used the command-line and CosmJS to issue commands to the Cosmos Hub Testnet. In this tutorial, you are working on a browser application that interacts with the Keplr extension.

You will again connect to the Cosmos Hub Testnet. Optionally, connect to your locally running Cosmos blockchain using simapp as explained [before](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html).

To keep the focus on CosmJS and Keplr, you are going to use ready-made pages created by the Next.js framework. Do not worry if you routinely use another framework, the CosmJS-specific code in this tutorial can be applied similarly in Angular, Vue, and other frameworks.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#creating-your-simple-next-js-project) Creating your simple Next.js project

In your project folder create the ready-made Next.js app, which automatically places it in a subfolder for you. This follows [the docs (opens new window)↗](https://nextjs.org/docs):

**Local**

**Docker**



Copy

$ npx create-next-app@latest --typescript

Copy

$ docker run --rm -it \

-v $(pwd):/root -w /root \

node:lts-slim \

npx create-next-app@latest --typescript

Which guides you with:



Copy

...

? What is your project named? › cosmjs-keplr

This created a new cosmjs-keplr folder, so open it. There you can find a /pages folder, which contains an index.tsx. That's your first page.

Run it, in the cosmjs-keplr folder:

**Local**

**Docker**



Copy

$ npm run dev

Copy

$ docker run --rm -it \

-v $(pwd):/cosmjs-keplr -w /cosmjs-keplr \

-p 0.0.0.0:3000:3000 \

node:lts-slim \

npm run dev

Which returns:



Copy

ready - started server on 0.0.0.0:3000, url: http://localhost:3000

...

You should see the result, a welcome page with links, in your browser by visiting [http://localhost:3000 (opens new window)↗](http://localhost:3000/). Next.js uses [React (opens new window)↗](https://reactjs.org/) under the hood.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#html-elements) HTML elements

The goal of the exercise is to find token balances, yours and the faucet's, and then send some tokens back to the faucet. Before introducing any CosmJS, you can already create a React component that includes the basic user interface that you need. By convention, create a /components folder and then copy the following code inside a new file called FaucetSender.tsx:

****

**FaucetSender.tsx**

Copy

import { ChangeEvent, Component, MouseEvent } from "react"

import styles from '../styles/Home.module.css'

interface FaucetSenderState {

denom: string

faucetBalance: string

myAddress: string

myBalance: string

toSend: string

}

export interface FaucetSenderProps {

faucetAddress: string

rpcUrl: string

}

export class FaucetSender extends Component<FaucetSenderProps, FaucetSenderState> {

// Set the initial state

constructor(props:FaucetSenderProps) {

super(props)

this.state = {

denom: "Loading...",

faucetBalance: "Loading...",

myAddress: "Click first",

myBalance: "Click first",

toSend: "0",

}

}

// Store changed token amount to state

onToSendChanged = (e: ChangeEvent<HTMLInputElement>) => this.setState({

toSend: e.currentTarget.value

})

// When the user clicks the "send to faucet button"

onSendClicked = async(e: MouseEvent<HTMLButtonElement>) => {

alert("TODO")

}

// The render function that draws the component at init and at state change

render() {

const { denom, faucetBalance, myAddress, myBalance, toSend } = this.state

const { faucetAddress } = this.props

console.log(toSend)

// The web page structure itself

return <div>

<div className={styles.description}>

Send back to the faucet

</div>

<fieldset className={styles.card}>

<legend>Faucet</legend>

<p>Address: {faucetAddress}</p>

<p>Balance: {faucetBalance}</p>

</fieldset>

<fieldset className={styles.card}>

<legend>You</legend>

<p>Address: {myAddress}</p>

<p>Balance: {myBalance}</p>

</fieldset>

<fieldset className={styles.card}>

<legend>Send</legend>

<p>To faucet:</p>

<input value={toSend} type="number" onChange={this.onToSendChanged}/> {denom}

<button onClick={this.onSendClicked}>Send to faucet</button>

</fieldset>

</div>

}

}



The **properties** of FaucetSender.tsx only contain the things it knows at build time. It keeps a **state**, and this state is either updated by the user or after a fetch. It reuses a default style you can find in /styles.

The component is still unused. You do not need the default page that comes with create-next-app, so you can replace the contents of index.tsx with the following code that imports the new component:



Copy

import type { NextPage } from 'next'

import { FaucetSender } from '../components/FaucetSender'

const Home: NextPage = () => {

return <FaucetSender

faucetAddress="cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"

rpcUrl="https://rpc.sentry-01.theta-testnet.polypore.xyz" />

}

export default Home

The faucet address was found in the [previous section](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html), as well as the RPC endpoint that connects to the Cosmos Hub Testnet.

When npm run dev picks up the changes, you should see that your page has changed to what you created. In particular, it alerts you with "TODO" when you click on the button.

Your page is not very useful yet, make it more so.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#installing-cosmjs) Installing CosmJS

Now that you have a working Next.js project and ready page, it is time to add the necessary CosmJS elements to the project:

**Local**

**Docker**



Copy

$ npm install \

@cosmjs/stargate@v0.28.2 cosmjs-types@v0.4.1 \

--save-exact

Copy

$ docker run --rm -it \

-v $(pwd):/cosmjs-keplr -w /cosmjs-keplr \

node:lts-slim \

npm install \

@cosmjs/stargate@v0.28.2 cosmjs-types@v0.4.1 \

--save-exact

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#displaying-information-without-user-input) Displaying information without user input

When building a user interface, it is good practice to not ask your user's address until it becomes necessary (e.g. if they click a relevant button). You should start by showing information that is knowable without user input. In this case, this is the token denom (denomination) and the faucet's balance. Add the following function that gets the balance from the faucet and place it above the onToSendChanged function inside FaucetSender.tsx:



Copy

+ // Get the faucet's balance

+ updateFaucetBalance = async(client: StargateClient) => {

+ const balances: readonly Coin[] = await client.getAllBalances(

+ this.props.faucetAddress

+ )

+ const first: Coin = balances[0]

+ this.setState({

+ denom: first.denom,

+ faucetBalance: first.amount,

+ })

+ }

onToSendChanged = (e: ChangeEvent<HTMLInputElement>)...

Note that it only cares about the first coin type stored in balances[0]: this is to keep the exercise simple, but there could be multiple coins in that array of balances. It extracts the denom, which is then displayed to the user as the unit to transfer. Add the denom that in the constructor as well so that it runs on load via another specific function:



Copy

constructor(props:FaucetSenderProps) {

...

+ setTimeout(this.init, 500)

}

+ init = async() => this.updateFaucetBalance(

+ await StargateClient.connect(this.props.rpcUrl)

+ )

updateFaucetBalance = async (client: StargateClient)...

The call to setTimeout is so that init is *not* launched on the same pass as the constructor.

After run dev picks the changes, you should see that your page starts showing the relevant information.

Now, add elements that handle your user's information.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#getting-testnet-tokens) Getting testnet tokens

Refer to the previous section on how to [get Cosmos Hub Testnet tokens](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html). This time you should use your Keplr address. If you have not set up one yet, do so now. Your Cosmos Hub Testnet address is the same one that Keplr shows you for the Cosmos Hub mainnet.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#detecting-keplr) Detecting Keplr

Following [Keplr's documentation (opens new window)↗](https://docs.keplr.app/api/#how-to-detect-keplr), it is time to add a function to see if Keplr is installed on the browser. For convenience and type hinting, install the Typescript Keplr types from within the folder of your project:

**Local**

**Docker**



Copy

$ npm install @keplr-wallet/types@0.11.51 \

--save-dev --save-exact

Copy

$ docker run --rm -it \

-v $(pwd):/cosmjs-keplr -w /cosmjs-keplr \

node:lts-slim \

npm install @keplr-wallet/types@0.11.51 \

--save-dev --save-exact

After this package is installed, inform Typescript that window may have a .keplr field with the help of [this helper (opens new window)↗](https://github.com/chainapsis/keplr-wallet/tree/master/docs/api#keplr-specific-features), by adding it below your imports to FaucetSender.tsx:



Copy

import { Window as KeplrWindow } from "@keplr-wallet/types";

declare global {

interface Window extends KeplrWindow {}

}

Detecting Keplr can be done at any time, but to keep the number of functions low for this exercise do it in onSendClicked. You want to avoid detecting Keplr on page load if not absolutely necessary. This is generally considered bad user experience for users who might just want to browse your page and not interact with it. Replace the onSendClicked with the following:



Copy

onSendClicked = async(e: MouseEvent<HTMLButtonElement>) => {

- alert("TODO")

+ const { keplr } = window

+ if (!keplr) {

+ alert("You need to install or unlock Keplr")

+ return

+ }

}

Hopefully, when you click on the button it does not show an alert. It does not do anything else either. As an optional confirmation, if you disable Keplr from Chrome's extension manager, when you click the button the page tells you to install it.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#prepare-keplr) Prepare Keplr

Keplr is now detected. By default, Keplr lets its users only connect to the blockchains it knows about. Unfortunately, the Cosmos Hub Testnet is not one of them, but there is a feature where you can instruct it to handle any Cosmos blockchain, provided you give its parameters. Here is [an example (opens new window)↗](https://github.com/chainapsis/keplr-example/blob/master/src/main.js). In the case of Cosmos Hub Testnet, these parameters are available, as mentioned on the [testnet page (opens new window)↗](https://github.com/cosmos/testnets/tree/master/public#add-to-keplr). Add a new function for them as shown in the expandable box:

****

**getTestnetChainInfo**

Copy

getTestnetChainInfo = (): ChainInfo => ({

chainId: "theta-testnet-001",

chainName: "theta-testnet-001",

rpc: "https://rpc.sentry-01.theta-testnet.polypore.xyz/",

rest: "https://rest.sentry-01.theta-testnet.polypore.xyz/",

bip44: {

coinType: 118,

},

bech32Config: {

bech32PrefixAccAddr: "cosmos",

bech32PrefixAccPub: "cosmos" + "pub",

bech32PrefixValAddr: "cosmos" + "valoper",

bech32PrefixValPub: "cosmos" + "valoperpub",

bech32PrefixConsAddr: "cosmos" + "valcons",

bech32PrefixConsPub: "cosmos" + "valconspub",

},

currencies: [

{

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

},

{

coinDenom: "THETA",

coinMinimalDenom: "theta",

coinDecimals: 0,

},

{

coinDenom: "LAMBDA",

coinMinimalDenom: "lambda",

coinDecimals: 0,

},

{

coinDenom: "RHO",

coinMinimalDenom: "rho",

coinDecimals: 0,

},

{

coinDenom: "EPSILON",

coinMinimalDenom: "epsilon",

coinDecimals: 0,

},

],

feeCurrencies: [

{

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

gasPriceStep: {

low: 1,

average: 1,

high: 1,

},

},

],

stakeCurrency: {

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

},

coinType: 118,

features: ["stargate", "ibc-transfer", "no-legacy-stdTx"],

})

You need to add another import from the @keplr-wallet package so that your script understands what ChainInfo is:



Copy

- import { Window as KeplrWindow } from "@keplr-wallet/types"

+ import { ChainInfo, Window as KeplrWindow } from "@keplr-wallet/types"

Note that it mentions the chainId: "theta-testnet-001". In effect, this adds the Cosmos Hub Testnet to Keplr's registry of blockchains, under the label theta-testnet-001. Whenever you want to prompt the user to add the Cosmos Hub Testnet to Keplr, add the line:



Copy

await window.keplr!.experimentalSuggestChain(this.getTestnetChainInfo())

This needs to be done once, which in this case is in the onSendClicked function after having detected Keplr, but repeating the line elsewhere is generally not a problem.

Keplr is now detected and prepared. Now try to do something useful with the user's information.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#your-address-and-balance) Your address and balance

In onSendClicked, similar to the previous section, you can:

1. Prepare Keplr, with keplr.experimentalSuggestChain.
2. Get the signer for your user's accounts, with KeplrWindow's window.getOfflineSigner.
3. Create your signing client.
4. Get the address and balance of your user's first account.
5. Send the requested coins to the faucet.
6. Inform and update.

In practice, the code for onSendClicked looks like this:



Copy

onSendClicked = async(e: MouseEvent<HTMLButtonElement>) => {

// Detect Keplr

const { keplr } = window

if (!keplr) {

alert("You need to install Keplr")

return

}

// Get the current state and amount of tokens that we want to transfer

const { denom, toSend } = this.state

const { faucetAddress, rpcUrl } = this.props

// Suggest the testnet chain to Keplr

await keplr.experimentalSuggestChain(this.getTestnetChainInfo())

// Create the signing client

const offlineSigner =

window.getOfflineSigner!("theta-testnet-001")

const signingClient = await SigningStargateClient.connectWithSigner(

rpcUrl,

offlineSigner,

)

// Get the address and balance of your user

const account: AccountData = (await offlineSigner.getAccounts())[0]

this.setState({

myAddress: account.address,

myBalance: (await signingClient.getBalance(account.address, denom))

.amount,

})

// Submit the transaction to send tokens to the faucet

const sendResult = await signingClient.sendTokens(

account.address,

faucetAddress,

[

{

denom: denom,

amount: toSend,

},

],

{

amount: [{ denom: "uatom", amount: "500" }],

gas: "200000",

},

)

// Print the result to the console

console.log(sendResult)

// Update the balance in the user interface

this.setState({

myBalance: (await signingClient.getBalance(account.address, denom))

.amount,

faucetBalance: (

await signingClient.getBalance(faucetAddress, denom)

).amount,

})

}

Expand

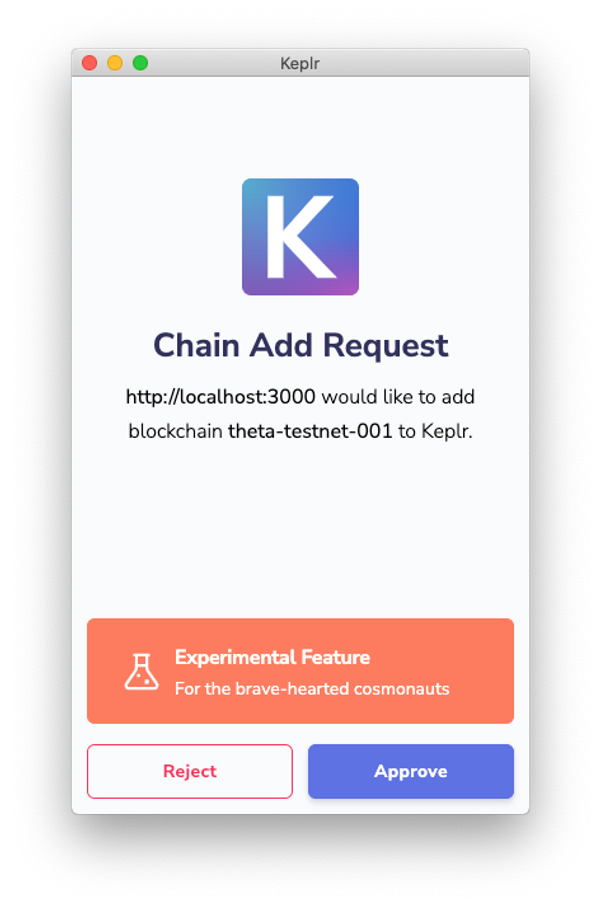




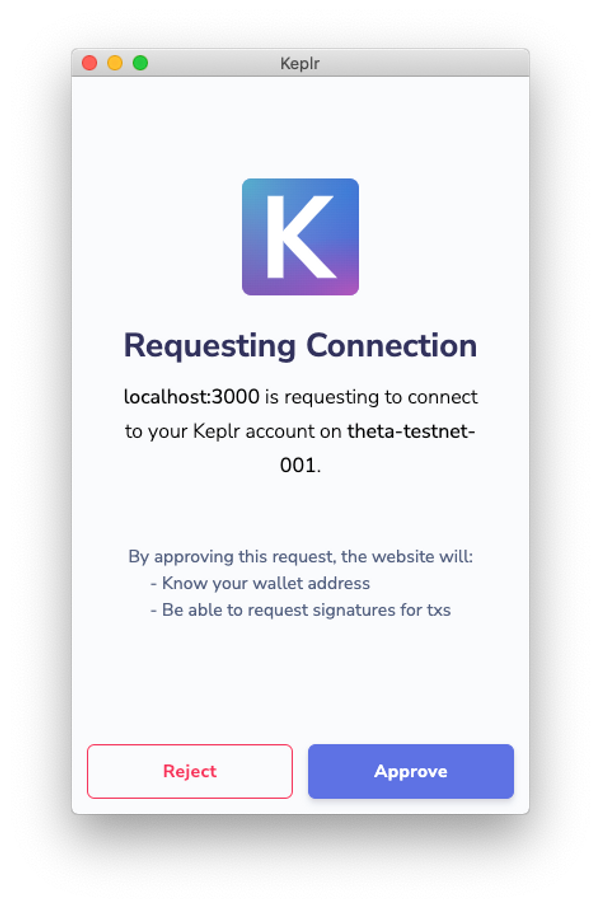
Keplr is only tasked with signing transactions. The transactions are broadcast with the RPC endpoint of your choice.

Now run the full script. In the refreshed page, enter an amount of uatom (for example 1000000) and click Send to faucet. A number of events happen:

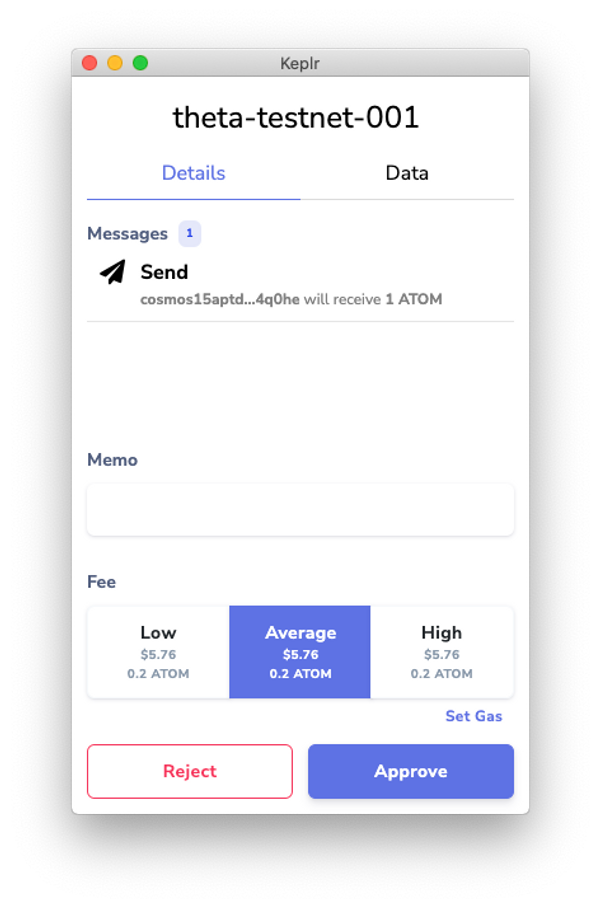
1. Keplr asks for confirmation that you agree to add the testnet network. It does not install any network without your approval, as that would be a security risk. It asks this only the first time you add a given network, which is why doing it in onSendClicked is harmless.



1. Keplr asks whether you agree to share your account information, because this involves a potential security risk. Again, it asks this only once per web page + network combination.



1. Your address and balance fields are updated and visible.
2. Keplr asks whether you agree to sign the transaction, a very important action that requires approval **every time**.



After this is done, your balance updates again, and in the browser console you see the transaction result.

If you want to double check if you got everything right, you can find the full component's code in the expandable box below:

****

**Final FaucetSender.tsx file**

Copy

import { Coin, SigningStargateClient, StargateClient } from "@cosmjs/stargate"

import { AccountData, OfflineSigner } from "@cosmjs/proto-signing"

import { ChainInfo, Window as KeplrWindow } from "@keplr-wallet/types"

import { ChangeEvent, Component, MouseEvent } from "react"

import styles from "../styles/Home.module.css"

declare global {

interface Window extends KeplrWindow {}

}

interface FaucetSenderState {

denom: string

faucetBalance: string

myAddress: string

myBalance: string

toSend: string

}

export interface FaucetSenderProps {

faucetAddress: string

rpcUrl: string

}

export class FaucetSender extends Component<

FaucetSenderProps,

FaucetSenderState

> {

// Set the initial state

constructor(props: FaucetSenderProps) {

super(props)

this.state = {

denom: "Loading...",

faucetBalance: "Loading...",

myAddress: "Click first",

myBalance: "Click first",

toSend: "0",

}

setTimeout(this.init, 500)

}

// Connecting to the endpoint to fetch the faucet balance

init = async () =>

this.updateFaucetBalance(

await StargateClient.connect(this.props.rpcUrl),

)

// Get the faucet's balance

updateFaucetBalance = async (client: StargateClient) => {

const balances: readonly Coin[] = await client.getAllBalances(

this.props.faucetAddress,

)

const first: Coin = balances[0]

this.setState({

denom: first.denom,

faucetBalance: first.amount,

})

}

// Store changed token amount to state

onToSendChanged = (e: ChangeEvent<HTMLInputElement>) =>

this.setState({

toSend: e.currentTarget.value,

})

// When the user clicks the "send to faucet button"

onSendClicked = async (e: MouseEvent<HTMLButtonElement>) => {

// Detect Keplr

const { keplr } = window

if (!keplr) {

alert("You need to install Keplr")

return

}

// Get the current state and amount of tokens that we want to transfer

const { denom, toSend } = this.state

const { faucetAddress, rpcUrl } = this.props

// Suggest the testnet chain to Keplr

await keplr.experimentalSuggestChain(this.getTestnetChainInfo())

// Create the signing client

const offlineSigner: OfflineSigner =

window.getOfflineSigner!("theta-testnet-001")

const signingClient = await SigningStargateClient.connectWithSigner(

rpcUrl,

offlineSigner,

)

// Get the address and balance of your user

const account: AccountData = (await offlineSigner.getAccounts())[0]

this.setState({

myAddress: account.address,

myBalance: (await signingClient.getBalance(account.address, denom))

.amount,

})

// Submit the transaction to send tokens to the faucet

const sendResult = await signingClient.sendTokens(

account.address,

faucetAddress,

[

{

denom: denom,

amount: toSend,

},

],

{

amount: [{ denom: "uatom", amount: "500" }],

gas: "200000",

},

)

// Print the result to the console

console.log(sendResult)

// Update the balance in the user interface

this.setState({

myBalance: (await signingClient.getBalance(account.address, denom))

.amount,

faucetBalance: (

await signingClient.getBalance(faucetAddress, denom)

).amount,

})

}

// The Cosmos Hub Testnet chain parameters

getTestnetChainInfo = (): ChainInfo => ({

chainId: "theta-testnet-001",

chainName: "theta-testnet-001",

rpc: "https://rpc.sentry-01.theta-testnet.polypore.xyz/",

rest: "https://rest.sentry-01.theta-testnet.polypore.xyz/",

bip44: {

coinType: 118,

},

bech32Config: {

bech32PrefixAccAddr: "cosmos",

bech32PrefixAccPub: "cosmos" + "pub",

bech32PrefixValAddr: "cosmos" + "valoper",

bech32PrefixValPub: "cosmos" + "valoperpub",

bech32PrefixConsAddr: "cosmos" + "valcons",

bech32PrefixConsPub: "cosmos" + "valconspub",

},

currencies: [

{

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

},

{

coinDenom: "THETA",

coinMinimalDenom: "theta",

coinDecimals: 0,

},

{

coinDenom: "LAMBDA",

coinMinimalDenom: "lambda",

coinDecimals: 0,

},

{

coinDenom: "RHO",

coinMinimalDenom: "rho",

coinDecimals: 0,

},

{

coinDenom: "EPSILON",

coinMinimalDenom: "epsilon",

coinDecimals: 0,

},

],

feeCurrencies: [

{

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

gasPriceStep: {

low: 1,

average: 1,

high: 1,

},

},

],

stakeCurrency: {

coinDenom: "ATOM",

coinMinimalDenom: "uatom",

coinDecimals: 6,

coinGeckoId: "cosmos",

},

coinType: 118,

features: ["stargate", "ibc-transfer", "no-legacy-stdTx"],

})

// The render function that draws the component at init and at state change

render() {

const { denom, faucetBalance, myAddress, myBalance, toSend } =

this.state

const { faucetAddress } = this.props

// The web page structure itself

return (

<div>

<div className={styles.description}>

Send back to the faucet

</div>

<fieldset className={styles.card}>

<legend>Faucet</legend>

<p>Address: {faucetAddress}</p>

<p>Balance: {faucetBalance}</p>

</fieldset>

<fieldset className={styles.card}>

<legend>You</legend>

<p>Address: {myAddress}</p>

<p>Balance: {myBalance}</p>

</fieldset>

<fieldset className={styles.card}>

<legend>Send</legend>

<p>To faucet:</p>

<input

value={toSend}

type="number"

onChange={this.onToSendChanged}

/>{" "}

{denom}

<button onClick={this.onSendClicked}>Send to faucet</button>

</fieldset>

</div>

)

}

}

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#with-a-locally-running-chain) With a locally running chain

What if you wanted to experiment with your own chain while in development?

Keplr does not know about locally running chains by default. As you did with Cosmos Hub Testnet, you must inform Keplr about your chain: change ChainInfo to match the information about your chain, and change rpcUrl so that it points to your local port.



If you would like to get started on integrating Keplr into your own checkers game, you can go straight to the related exercise in [CosmJS for Your Chain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/) to start from scratch.

More specifically, you can jump to:

* [Integrate CosmJS and Keplr](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/4-cosmjs-gui.html), for a more elaborate integration between Keplr, CosmJS, custom messages, and a pre-existing Checkers GUI.

synopsis

To summarize, this section has explored:

* How to use CosmJS to connect with Keplr, a browser extension widely used to manage private keys, to find your token balance and that of the faucet and then send some tokens back to the faucet.
* How to create a simple app in the Next.js framework for the purposes of performing the exercise, though the CosmJS-specific code is also applicable to Angular, Vue, and other frameworks.
* Best practices regarding when and when not to ask for your user's address, such as limiting your user interface to only showing information that is knowable without user input until making a request is absolutely necessary.
* How to add a function to detect whether or not Keplr is installed on the browser, also minimizing the occasions when information requests are made in line with best practices.
* How to prepare Keplr to handle any Cosmos blockchain (or for use with locally running chains, such as during development) by providing it with the necessary parameters for a specific chain, before experimenting with accessing useful information from the chain.

previous

[](https://ida.interchain.io/tutorials/7-cosmjs/3-multi-msg.html)

**[Compose Complex Transactions](https://ida.interchain.io/tutorials/7-cosmjs/3-multi-msg.html)**

up next

**[Create Custom CosmJS Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/5-create-custom.html)**

[[](https://ida.interchain.io/tutorials/7-cosmjs/5-create-custom.html)](https://ida.interchain.io/tutorials/7-cosmjs/5-create-custom.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Creating your simple Next.js project](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#creating-your-simple-next-js-project)

[HTML elements](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#html-elements)

[Installing CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#installing-cosmjs)

[Displaying information without user input](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#displaying-information-without-user-input)

[Getting testnet tokens](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#getting-testnet-tokens)

[Detecting Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#detecting-keplr)

[Prepare Keplr](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#prepare-keplr)

[Your address and balance](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#your-address-and-balance)

[With a locally running chain](https://ida.interchain.io/tutorials/7-cosmjs/4-with-keplr.html#with-a-locally-running-chain)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)