Part 3: Building a simple Web Application

In this part of the tutorial, we will learn how to build a simple web application that will interact with the contract from the first part of this tutorial series. You need to have Neutron up and running, and the contract to be instantiated. You can learn how to do all of that by following the first part of this tutorial.

To build the UI, we will be using:

- Next.js
- as a framework
- shadcn-ui
- for components
- CosmosKit
- for wallets

The final result of this tutorial can be found athttps://github.com/neutron-org/onboarding/tree/main/minimal_ui .

Setup Next.js app

1. InstallNode.js
2. v18 or higher.
3. Create Next.js app:
4. npx create-next-app@latest minimal_ui
5. **- Would you like to use TypeScript: Yes**
6. **- Would you like to use ESLint: Yes**
7. **- Would you like to use Tailwind CSS: Yes**
8. **- Would you like to use src/ directory: Yes**
9. **- Would you like to use App Router: Yes**
10. **- Would you like to customize the default import alias (@/*)?: No**
11. cd
12. minimal_ui
13. Install shadcn-ui and add components:
14. npx shadcn@latest init --defaults
15. npx shadcn@latest
16. add
17. button card input label

Add CosmosKit

1. Install packages:
2. npm
3. install
4. --save @cosmos-kit/react @interchain-ui/react @cosmos-kit/core @cosmos-kit/keplr-extension @cosmjs/amino @cosmjs/proto-signing chain-registry @chain-registry/types
5. Create contextsrc/contexts/CosmosKit.tsx
6. with the following content:
7. "use client"
8. ;
9. import
10. {
11. AssetList

```
,
Chain
}
from
"@chain-registry/types"
;
import
{
GasPrice
}
from
"@cosmjs/stargate"
;
// You can add more wallets here
import
{
wallets
as
keplrExtension
}
from
"@cosmos-kit/keplr-extension"
;
import
{
ChainProvider
}
from
"@cosmos-kit/react"
;
import
assert
from
"assert"
;
import
{
assets
,
chains
}
from
"chain-registry"
;
import
React
from
"react"
;
import
"@interchain-ui/react/styles"
;
// This is a Neutron Localnet chain, we need to add it manually because it's not in the chain registry.
// This new chain can be based on Neutron Testnet, we just need to adjust some parameters.
const
localnetChain
:
Chain
=
(
(
)
=>
{
const
chain
=
chains
```

```
.
find
(
(
chain
)
=>
chain
.
chain_name
===
"neutrontestnet"
)
;
assert
(
chain
)
;
return
{
...
chain
,
// Chain ID is a unique identifier for the chain. You can find one in localnet_config.json.
chain_id
:
"ntrntest"
,
// Chain name is another unique identifier for the chain that is used in CosmosKit.
chain_name
:
"neutronlocalnet"
,
// Pretty name is a human readable name for the chain.
pretty_name
:
"Neutron Localnet"
,
apis
:
{
...
chain
.
apis
,
// RPC and REST endpoints are used to communicate with the chain.
// We provide proxy endpoints here, which can be found in the "First launch" section.
rpc
:
[
{
address
:
"http://localhost:3001/proxy"
}
]
,
rest
:
[
{
address
:
"http://localhost:3002/proxy"
}
]
```

```
,
}
,
}
;
}
)
(
)
;
// The same for assets
const
localnetAssets
:
AssetList
=
(
(
)
=>
{
const
asset
=
assets
.
find
(
(
asset
)
=>
asset
.
chain_name
===
"neutrontestnet"
)
;
assert
(
asset
)
;
return
{
...
asset
,
chain_name
:
"neutronlocalnet"
,
}
;
}
)
(
)
;
export
const
CosmosKitProvider
=
(
{
children
,
```

```
}
:
{
children
:
React
.
ReactNode
;
}
)
=>
(
<
ChainProvider
chains
=
{
[
...
chains
,
localnetChain
]
}
assetLists
=
{
[
...
assets
,
localnetAssets
]
}
// We need to specify gas price to be able to sign transactions.
// The provided value works just fine for the localnet.
signerOptions
=
{
{
signingCosmwasm
:
(
)
=>
(
{
gasPrice
:
GasPrice
.
fromString
(
"0.01untrn"
)
,
}
)
,
signingStargate
:
(
)
=>
(
{
gasPrice
```

:
GasPrice
.
fromString
(
"0.01untrn"
)
,
}
)
,
}
}
wallets
=
{
[
...
keplrExtension
]
}

{
children
}
<!--
ChainProvider
-->
)
;
Add context by replacing the content ofsrc/app/layout.tsx
with the following:
import
{
CosmosKitProvider
}
from
"@/contexts/CosmosKit"
;
import
type
{
Metadata
}
from
"next"
;
import
"./globals.css"
;
export
const
metadata
:
Metadata
=
{
title
:
"Minimal UI"
,
}
;
export
default
function
RootLayout
(

```
{
children
,
}
:
Readonly
<
{
children
:
React
.
ReactNode
;
}


)
{
return
(
<
html
lang
=
"
en
"

<
body

<
CosmosKitProvider

{
children
}
<!--
CosmosKitProvider
-->
<!--
body
-->
<!--
html
-->
)
;
}
```

Brief overview of CosmosKit

CosmosKit is a library that allows you to easily communicate with different Cosmos chains and wallets. To use it you setup a context with a list of chains and wallets you want to support.

Then you can use theuseChain hook to connect to communicate with a specific chain like this:

tip The code below is an example of how to use CosmosKit. Don't copy it to the project! const

{ address , connect , disconnect , getCosmWasmClient , getSigningCosmWasmClient , }

=

useChain ("neutronlocalnet" ,

true) ;

// address is the address of the connected wallet. // It is undefined if no wallet is connected. console . log ("Your address:" , address ??

"Not connected") ;

// connect is a function that opens a modal to connect to a wallet. await

connect ( ) ;

// disconnect is a function to disconnect from a wallet. await

disconnect ( ) ;

// getCosmWasmClient is a function that returns a CosmWasmClient for the connected chain. // This client is used to make queries. const client =

await

getCosmWasmClient ( ) ; const balance =

await client . getBalance (address ,

"untrn") ; console . log ("Your balance:" , balance) ;

// getSigningCosmWasmClient is a function that returns a SigningCosmWasmClient for the connected chain. // This client is used to sign transactions. const client =

await

getSigningCosmWasmClient ( ) ; const

{ transactionHash }

=

await client . sendTokens (address , receiver , [{ amount :

"1000000" , denom :

"untrn" , } ,] , "auto") ; console . log ("TX hash:" , transactionHash) ;

Create a wallet button component

This component will be used to open CosmosKit wallets modal and perform connection to a wallet.

1. Create a component with wallet button insrc/components/WalletButton.tsx
2. :
3. import
4. {
5. Button
6. }
7. from
8. "@/components/ui/button"
9. ;
10. import
11. {
12. useChain
13. }
14. from
15. "@cosmos-kit/react"
16. ;
17. const
18. formatAddress
19. =
20. (
21. address
22. :
23. string
24.)

```
=>
{
return
address
.
slice
(
0
,
11
)
+
"..."
+
address
.
slice
(
-
3
)
;
}
;
export
const
WalletButton
=
(
)
=>
{
const
{
address
,
connect
}
=
useChain
(
"neutronlocalnet"
,
true
)
;
return
(
<
Button
variant
=
{
address
?
"outline"
:
"default"
}
type
=
"
button
"
onClick
=
{
connect
```

```
}

{
address
?
formatAddress
(
address
)
:
"Connect Wallet"
}
<!--
Button
-->
)
;
}
;
Replace thesrc/app/page.tsx
with the following content:
"use client"
;
import
{
WalletButton
}
from
"@/components/WalletButton"
;
export
default
function
Home
(
)
{
return
(
<
main
className
=
"
flex flex-col gap-8 items-center justify-center h-screen
"


<
WalletButton
/>
<!--
main
-->
)
;
}
```

First launch

Now we have a simple UI that can connect to a wallet. Let's try it out.

Prepare your browser

1. InstallKeplr Wallet Extension
2. .
3. When prompted to create a wallet, clickImport an existing wallet

4. and then
5. clickUse recovery phrase or private key
6. .
7. Enter mnemonic fromPart 1
8. :
9. kiwi valid tiger wish shop time exile client metal view spatial ahead

Run CORS Proxy

Before we can do anything with the local chain, we need to launch a Local CORS Proxy for the localnet. Run the following commands in a separate terminals and keep them in the background:

npx local-cors-proxy@latest --proxyUrl http://0.0.0.0:26657 --port 3001 npx local-cors-proxy@latest --proxyUrl http://0.0.0.0:1317 --port 3002

Launch the app

npm run dev And open it in the browser: http://localhost:3000

Now you should see a wallet button and should be able to connect to a wallet.

Interact with the contract

In this UI we provide an example of interacting with the contract created in Part 1 .

Contract address The address of your contract might be different from what you see in this tutorial. Make sure that you are replacing the addresses from the commands below with the address ofyour contract! To interact with the contract we'll be usingCosmWasmClient andSigningCosmWasmClient fromCosmJS .

Querying value from the contract:

const client =

await

getCosmWasmClient ( ) ;

const

{ current_value }

=

(await client . queryContractSmart ("neutron1nyuryl5u5z04dx4zsqgvsuw7fe8gl2f77yufynauuhklnnmnjncqcls0tj" , { current_value :

{ } }))

as

Promise < { current_value :

string

}

;

console . log (current_value) ; Executing a message:

const client =

await

getSigningCosmWasmClient ( ) ;

const

{ transactionHash }

=

await client . execute (address , "neutron1nyuryl5u5z04dx4zsqgvsuw7fe8gl2f77yufynauuhklnnmnjncqcls0tj" , { increase_count :

{ amount :

"1" , } , } , "auto") ;

console . log (transactionHash) ; Let's create a component that will do this for us.

1. Create a component insrc/components/Counter.tsx
2. :
3. import
4. {
5. Button
6. }
7. from
8. "@/components/ui/button"
9. ;
10. import
11. {
12. Card
13. ,
14. CardContent
15. ,
16. CardFooter
17. ,
18. CardHeader
19. ,
20. CardTitle
21. ,
22. }
23. from
24. "@/components/ui/card"
25. ;
26. import
27. {
28. Input
29. }
30. from
31. "@/components/ui/input"
32. ;
33. import
34. {
35. Label
36. }
37. from
38. "@/components/ui/label"
39. ;
40. import
41. {
42. useChain
43. }
44. from
45. "@cosmos-kit/react"
46. ;
47. import
48. assert
49. from
50. "assert"
51. ;
52. import
53. {
54. useCallback
55. ,
56. useEffect
57. ,
58. useState
59. }
60. from

```
"react"
;
// Your contract address here
const
CONTRACT_ADDRESS
=
"neutron1nyuryl5u5z04dx4zsqgvsuw7fe8gl2f77yufynauuhklnnmnjncqcls0tj"
;
const
useCounter
=
(
)
=>
{
const
{
address
,
getCosmWasmClient
,
getSigningCosmWasmClient
}
=
useChain
(
"neutronlocalnet"
,
true
)
;
const
[
value
,
setValue
]
=
useState
<
string
|
undefined


(
)
;
const
fetchValue
=
useCallback
(
async
(
)
=>
{
const
client
=
await
getCosmWasmClient
(
)
;
const
{
current_value
```

```
}
=
(
await
client
.
queryContractSmart
(
CONTRACT_ADDRESS
,
{
current_value
:
{
}
}
)
)
as
{
current_value
:
string
}
;
setValue
(
current_value
)
;
}
,
[
getCosmWasmClient
]
)
;
const
increaseValue
=
useCallback
(
async
(
amount
:
string
)
=>
{
assert
(
address
,
"Address is required"
)
;
const
client
=
await
getSigningCosmWasmClient
(
)
;
const
{
transactionHash
```

```
}
=
await
client
.
execute
(
address
,
CONTRACT_ADDRESS
,
{
increase_count
:
{
amount
,
}
,
}
,
"auto"
)
;
void
fetchValue
(
)
;
return
transactionHash
;
}
,
[
address
,
getSigningCosmWasmClient
,
fetchValue
]
)
;
useEffect
(
(
)
=>
{
void
fetchValue
(
)
;
}
,
[
fetchValue
]
)
;
return
{
value
,
increaseValue
}
;
```

```
}
;
export
const
Counter
=
(
)
=>
{
const
{
address
}
=
useChain
(
"neutronlocalnet"
,
true
)
;
const
{
value
,
increaseValue
}
=
useCounter
(
)
;
const
[
amount
,
setAmount
]
=
useState
(
""
)
;
const
handleAmountChange
=
useCallback
(
(
event
:
React
.
ChangeEvent
<
HTMLInputElement


)
=>
{
setAmount
(
event
.
currentTarget
.
```

```
value
)
;
}
,
[
]
)
;
const
handleIncreaseClick
=
useCallback
(
async
(
)
=>
{
if
(
!
address
||
!
amount
)
return
;
const
transactionHash
=
await
increaseValue
(
amount
)
;
console
.
log
(
transactionHash
)
;
setAmount
(
""
)
;
}
,
[
address
,
amount
,
increaseValue
]
)
;
return
(
<
Card
className
=
"
```

w-[350px]
"

<
CardHeader

<
CardTitle

Counter contract
<!--
CardTitle
-->
<!--
CardHeader
-->
<
CardContent

<
div
className
=
"
grid w-full items-center gap-4
"

<
div
className
=
"
flex flex-col space-y-1.5
"

<
Label
htmlFor
=
"
value
"

Current value
<!--
Label
-->
<
Input
id
=
"
value
"
value
=
{
value
??
""
}
disabled
/>

```
<!--
div
-->
<
div
className
=
"
flex flex-col space-y-1.5
"


<
Label
htmlFor
=
"
amount
"


Amount to increase
<!--
Label
-->
<
Input
id
=
"
amount
"
type
=
"
number
"
min
=
"
1
"
value
=
{
amount
}
onChange
=
{
handleAmountChange
}
/>
<!--
div
-->
<!--
div
-->
<!--
CardContent
-->
<
CardFooter
className
=
"
flex justify-between
"
```

```
<
Button
disabled
=
{
!
address
||
!
amount
}
type
=
"
button
"
onClick
=
{
handleIncreaseClick
}

Increase
<!--
Button
-->
<!--
CardFooter
-->
<!--
Card
-->
)
;
}
;
AddCounter
component tosrc/app/page.tsx
:
"use client"
;
import
{
Counter
}
from
"@/components/Counter"
;
import
{
WalletButton
}
from
"@/components/WalletButton"
;
export
default
function
Home
(
)
{
return
(
<
main
```

```
598.  className
599.  =
600.  "
601.  flex flex-col gap-8 items-center justify-center h-screen
602.  "
603.
604.  <
605.  WalletButton
606.  />
607.  <
608.  Counter
609.  />
610.  <!--
611.  main
612.  -->
613.  )
614.  ;
615.  }
```

Now you should be able to see the counter value in UI and be able to increase itPrevious Part 2: Calling Modules and Contracts Next Integration tests for chain