WebSocket-enabled Components with QueryAPI

In this article you'll learn how to create a NEAR component that gathers information from a QueryAPI indexer using WebSockets. In this example, the QueryAPI indexer monitors the widget activity on the blockchain, and the NEAR component gets that information using WebSockets.

info QueryAPI is a fully managed solution to build indexer functions, extract on-chain data, store it in a database, and be able to query it using GraphQL endpoints.

QueryAPI indexer

TheWidget Activity indexer keeps track of any widget activity on thesocial.near smart contract. Whenever a Widget transaction is found, the data is stored in a Postgres database.

DB schema

The schema for the indexer's database is pretty simple:

CREATE

TABLE "widget_activity"

("id"

SERIAL

NOT

NULL , "account_id"

VARCHAR

NOT

NULL , "widget_name"

VARCHAR

NOT

NULL , "block_height"

DECIMAL (58 ,

0)

NOT

NULL , "receipt_id"

VARCHAR

NOT

NULL , "block_timestamp"

DECIMAL (20 ,

0)

NOT

NULL , CONSTRAINT

"widgets_pkey"

PRIMARY

KEY

("id")) ;

CREATE

INDEX idx_widget_activity_block_timestamp ON widget_activity (block_timestamp) ;

Indexer logic

In the following code snippet, you can find the simple indexer logic that filters widget transactions from thesocial.near smart contract, and if it finds widget development activity, then it adds a record to thewidget_activity table defined previously.

tip To learn more, check the complete source code of theWidget Activity indexer . // Add your code here const

SOCIAL_DB

=

"social.near" ;

const nearSocialWidgetTxs = block . actions ( ) . filter ((action)

=> action . receiverId

===

SOCIAL_DB) . flatMap ((action)

=> action . operations . map ((operation)

=> operation ["FunctionCall"]) . filter ((operation)

=> operation ?. methodName ===

"set") . map ((functionCallOperation)

=>

({ ... functionCallOperation , args :

base64decode (functionCallOperation . args) , receiptId : action . receiptId ,

// providing receiptId as we need it })) . filter ((functionCall)

=>

{ const accountId =

Object . keys (functionCall . args . data) [0] ; return

Object . keys (functionCall . args . data [accountId]) . includes ("widget") ; })) ;

if

(nearSocialWidgetTxs . length

0)

{ console . log ("Found NEAR Widget Development Activity...") ; const blockHeight = block . blockHeight ; const blockTimestamp = block . header ( ) . timestampNanosec ; console . log (nearSocialWidgetTxs) ; await

Promise . all (nearSocialWidgetTxs . map (async

(widgetEditTx)

=>

{ const accountId =

Object . keys (widgetEditTx . args . data) [0] ; const widgetName =

Object . keys (widgetEditTx . args . data [accountId] ["widget"]) [0] ;

console . log (ACCOUNT_ID: { accountId }) ; console . log (widgetName) ; await

handleWidgetTx (accountId , widgetName , blockHeight , blockTimestamp , widgetEditTx . receiptId) ; console . log (widgetEditTx) ; })) ; } }

This is the JS function that calls the GraphQL mutationInsertWidgetActivity and adds a record to thewidget_activity table:

tip Learn more aboutQueryAPI indexing functions and how to build your own indexers. async

function

handleWidgetTx (accountId , widgetName , blockHeight , blockTimestamp , receiptId)

{ console . log (accountId , blockHeight , blockTimestamp , receiptId) ; try

{ const mutationData =

{ activity :

{ account_id : accountId , widget_name : widgetName , block_height : blockHeight , block_timestamp : blockTimestamp , receipt_id : receiptId , } , } ; await context . graphql (mutation InsertWidgetActivity(activity: roshaan_near_widget_activity_feed_widget_activity_insert_input = {}) { insert_roshaan_near_widget_activity_feed_widget_activity_one(object: activity) { id } } , mutationData) ; }

catch

(e)

{ console . log (Could not add widget activity to DB, { e }) ; } }

Using WebSockets

Once you have a QueryAPI indexer running, you can use WebSockets to get the data in your NEAR Component. You only need to create aWebSocket object pointing to the QueryAPI's GraphQL endpoint.

Setup

Here's a code snippet from the NEAR component that subscribes and processes any activity from theWidget Activity indexer :

tip The code below is only a snippet. If you want the full source code to play around with the component, you can fork theWidget Activity Feed source code and build your own NEAR component. const

GRAPHQL_ENDPOINT

=

"near-queryapi.api.pagoda.co" ;

const

LIMIT

=

10 ; const accountId = props . accountId

||

"roshaan.near"

|| context . accountId ;

State . init ({ widgetActivities :

[ ] , widgetActivityCount :

0 , startWebSocketWidgetActivity :

null , initialFetch :

false , soundEffect : "https://bafybeic7uvzmhuwjficgctpleov5i43rteavwmktyyjrauwi346ntgja4a.ipfs.nftstorage.link/" , }) ;

const widgetActivitySubscription =

subscription IndexerQuery { roshaan_near_widget_activity_feed_widget_activity(order_by: {block_timestamp: desc} limit: { LIMIT }) { account_id block_height block_timestamp id receipt_id widget_name } } ;

```
const subscriptionWidgetActivity =
{ type :
"start" , id :
"widgetActivity" ,
// You can use any unique identifier payload :
{ operationName :
"IndexerQuery" , query : widgetActivitySubscription , variables :
{ } , } , } ; function
processWidgetActivity ( activity )
{ return
{
... activity } ; } function
startWebSocketWidgetActivity ( processWidgetActivities )
{ let ws =
State . get ( ) . ws_widgetActivity ;
if
( ws )
{ ws . close ( ) ; return ; }
```

ws

```
new
WebSocket ( wss:// { GRAPHQL_ENDPOINT } /v1/graphql ,
"graphql-ws" ) ;
ws . onopen
=
( )
=>
{ console . log (Connection to WS has been established ) ; ws . send ( JSON . stringify ( { type :
"connection_init" , payload :
{ headers :
{ "Content-Type" :
"application/json" , "Hasura-Client-Name" :
"hasura-console" , "x-hasura-role" :
"roshaan_near" , } , lazy :
true , } , } ) ) ;
setTimeout ( ( )
=> ws . send ( JSON . stringify ( subscriptionWidgetActivity ) ) ,
50 ) ; } ;
```

ws . onclose

=

( )

=>

{ State . update ({

ws_widgetActivity :

null

}) ; console . log (WS Connection has been closed) ; } ;

ws . onmessage

=

(e)

=>

{ const data =

JSON . parse (e . data) ; console . log ("received data" , data) ; if

(data . type

===

"data"

&& data . id

===

"widgetActivity")

{ processWidgetActivities (data . payload . data) ; } } ;

ws . onerror

=

(err)

=>

{ State . update ({

ws_widgetActivity :

null

}) ; console . log ("WebSocket error" , err) ; } ;

State . update ({

ws_widgetActivity : ws }) ; } info Pay attention to thesubscriptionWidgetActivity JSON payload.

Processing

This is the JS function that process the incoming widget activities generated by the QueryAPI indexer, allowing the NEAR component to create a feed based on the blockchain's widget activity:

tip You can fork theWidget Activity Feed source code and build your own NEAR component. function

processWidgetActivities (incoming_data)

{ let incoming_widgetActivities = incoming_data . roshaan_near_widget_activity_feed_widget_activity . flatMap (processWidgetActivity) ; const newActivities =

[... incoming_widgetActivities . filter ((activity)

=>

{ return

(state . widgetActivities . length

==

0

|| activity . block_timestamp

state . widgetActivities [0] . block_timestamp) ; }) ,] ; const prevActivities = state . prevActivities

||

[ ] ; State . update ({

widgetActivities :

[... newActivities ,

... prevActivities]

}) ; }

if

(state . ws_widgetActivity

===

undefined)

{ State . update ({ startWebSocketWidgetActivity : startWebSocketWidgetActivity , }) ; state . startWebSocketWidgetActivity (processWidgetActivities) ; }

Rendering

Finally, rendering the activity feed on the NEAR component is straight-forward, by iterating through thestate.widgetActivities map:

return

(< div

< Title

Widget

Activity

Feed { " " } < TextLink href = "https://near.org/dataplatform.near/widget/QueryApi.App"

{ " " } Powered

By

QueryAPI { " " } < / TextLink

< / Title

< RowContainer

{ state . widgetActivities . map ((activity , i)

=>

(< Card

< div

< Widget src = "mob.near/widget/TimeAgo" props = { {

blockHeight : activity . block_height

} } /

{ " " } ago < / div

< CardBody

< div key = { i }

< Text bold

Widget

Name :

{ activity . widget_name } < / Text

< Text bold

Account

ID :

{ activity . account_id } < / Text

< / div

< / CardBody

< CardFooter

< TextLink href = {/#/near/widget/ComponentDetailsPage?src= { activity . account_id } /widget/ { activity . widget_name } }

View < / TextLink

< / CardFooter

< / Card

)) } < / RowContainer

< / div

) ; Edit this page Last updatedonApr 10, 2024 bygagdiez Was this page helpful? Yes No Need some help? Chat with us or check ourDev Resources ! Twitter Telegram Discord Zulip

Previous Push Notifications Next NEAR for Ethereum developers