Create custom session

Now users can make their own implementation of Session Storage by implementing ISessionStorage interface and pass it to the SessionKeyManager module instance. Let's create a new file called customSession and start writing our own session stoage.

we will import following

```
import

{ ISessionStorage , SessionLeafNode , SessionSearchParam , SessionStatus , }

from

"@biconomy/modules/dist/src/interfaces/ISessionStorage" ; We will need to implement all the interface methods.
```

Here is an example of File storage implementation. It saves the session leafs data and signers in files. For testing purpose developer will need to create two files in the root folder for each user with smartAccountAddress_sessions.json and smartAccountAddress_signers.json names. These files can be created automatically, based on where and how it gets stored. For instance, if the account address is 0x123 then create 0x123_sessions.json and 0x123_signers.json to run this tutorial.

```
import

*

as fs from

"fs" ; import

{ Wallet , Signer }

from

"ethers" ; import

{ ISessionStorage , SessionLeafNode , SessionSearchParam , SessionStatus , }

from

"@biconomy/modules/dist/src/interfaces/ISessionStorage" ;

export

class

SessionFileStorage

implements

ISessionStorage

{ private smartAccountAddress :

string ;

constructor ( smartAccountAddress :

string )

{ this . smartAccountAddress = smartAccountAddress . toLowerCase ( ) ; } // This method reads data from the file and returns it in the JSON format private

async

readDataFromFile ( type :

"sessions"

|

"signers" ) :
```

```
Promise < any
{ return
new
Promise ( ( resolve )
=>
{ fs . readFile ( this . getStorageFilePath ( type ) ,
"utf8" ,
( err , data )
=>
{ if
( err )
{ // Handle errors appropriately resolve ( undefined ) ; }
else
{ if
( ! data )
{ resolve ( null ) ; }
else
{ resolve ( JSON . parse ( data ) ) ; } // resolve(JSON.parse(data)); } } ) ; } ) ; }
private
getStorageFilePath ( type :
"sessions"
|
"signers" ) :
string
{ return
./ { this . smartAccountAddress } _ { type } .json  ; }
private
async
writeDataToFile ( data :
any , type :
"sessions"
|
"signers" , ) :
Promise < void
{ return
new
Promise ( ( resolve , reject )
=>

{ const filePath =

this . getStorageFilePath ( type ) ; fs . writeFile ( filePath ,

JSON . stringify ( data ) ,

"utf8" ,

( err )

=>

{ if

( err )

{ // Handle errors appropriately reject ( err ) ; }

else

{ resolve ( ) ; } } ) ; } ) ; }

private

validateSearchParam ( param : SessionSearchParam ) :

void

{ if

( param . sessionID )

{ return ; }

else

if

( ! param . sessionID && param . sessionPublicKey && param . sessionValidationModule )

{ return ; }

else

{ throw

new

Error ( "Either pass sessionId or a combination of sessionPublicKey and sessionValidationModule address." , ) ; } } //
Session store is in the form of mekrleRoot and leafnodes, each object will have a root and an array of leafNodes. private

async

getSessionStore ( ) :

Promise < any

{ try

{ const data =

await

this . readDataFromFile ( "sessions" ) ; return data ||

{ merkleRoot :

"" , leafNodes :

[ ]

} ; }
```

```
catch

( error )

{ // Handle errors appropriately throw error ; } }

private

async

getSignerStore ( ) :

Promise < any

{ try

{ const data =

await

this . readDataFromFile ( "signers" ) ; return data ||

{ } ; }

catch

( error )

{ // Handle errors appropriately throw error ; } }

private

getStorageKey ( type :

"sessions"

|

"signers" ) :

string

{ return

{ this . smartAccountAddress } _ { type }  ; }

private

toLowercaseAddress ( address :

string ) :

string

{ return address . toLowerCase ( ) ; }

async

getSessionData ( param : SessionSearchParam ) :

Promise < SessionLeafNode

{ const sessions =

( await

this . getSessionStore ( ) ) . leafNodes ; console . log ( "Got sessions" , sessions ) ; const session = sessions [ 0 ] ;

if

( ! session )

{ throw
```

```
new
Error ( "Session not found." ) ; } return session ; }
async
addSessionData ( leaf : SessionLeafNode ) :
Promise < void
{ console . log ( "Add session Data" , leaf ) ; const data =
await
this . getSessionStore ( ) ; leaf . sessionValidationModule =
this . toLowercaseAddress ( leaf . sessionValidationModule , ) ; leaf . sessionPublicKey =
this . toLowercaseAddress ( leaf . sessionPublicKey ) ; data . leafNodes . push ( leaf ) ; await
this . writeDataToFile ( data ,
"sessions" ) ;
// Use 'sessions' as the type }
async
updateSessionStatus ( param : SessionSearchParam , status : SessionStatus , ) :
Promise < void
{ this . validateSearchParam ( param ) ;
const data =
await
this . getSessionStore ( ) ; const session = data . leafNodes . find ( ( s : SessionLeafNode )
=>
{ if
( param . sessionID )
{ return s . sessionID === param . sessionID ; }
else
if
( param . sessionPublicKey && param . sessionValidationModule )
{ return
( s . sessionPublicKey === this . toLowercaseAddress ( param . sessionPublicKey )
&& s . sessionValidationModule === this . toLowercaseAddress ( param . sessionValidationModule ) ) ; }
else
{ return
undefined ; } } ) ;
if
( ! session )
{ throw
new
```

```
Error ( "Session not found." ) ; }
session . status = status ; await
this . writeDataToFile ( data ,
"sessions" ) ;
// Use 'sessions' as the type }
async
clearPendingSessions ( ) :
Promise < void
{ const data =
await
this . getSessionStore ( ) ; data . leafNodes = data . leafNodes . filter ( ( s : SessionLeafNode )
=> s . status !==
"PENDING" , ) ; await
this . writeDataToFile ( data ,
"sessions" ) ;
// Use 'sessions' as the type }
async
addSigner ( signer ? : Wallet ) :
Promise < Wallet
{ const signers =
await
this . getSignerStore ( ) ; if
( ! signer )
{ signer = Wallet . createRandom ( ) ; } signers [ this . toLowercaseAddress ( signer . address ) ]
=
{ privateKey : signer . privateKey , publicKey : signer . publicKey , } ; await
this . writeDataToFile ( signers ,
"signers" ) ;
// Use 'signers' as the type return signer ; }
async
getSignerByKey ( sessionPublicKey :
string ) :
Promise < Signer
{ const signers =
await
this . getSignerStore ( ) ; console . log ( "Got signers" , signers ) ; const signerData = signers [ this . toLowercaseAddress (
sessionPublicKey ) ] ; if
( ! signerData )
```

```
{ throw
new
Error ( "Signer not found." ) ; } const signer =
new
Wallet ( signerData . privateKey ) ; return signer ; }
async
getSignerBySession ( param : SessionSearchParam ) :
Promise < Signer
{ const session =
await
this . getSessionData ( param ) ; console . log ( "got session" , session ) ; return
this . getSignerByKey ( session . sessionPublicKey ) ; }
async
getAllSessionData ( param ? : SessionSearchParam , ) :
Promise < SessionLeafNode [ ]
{ const sessions =
( await
this . getSessionStore ( ) ) . leafNodes ; if
( ! param ||
! param . status )
{ return sessions ; } return sessions . filter ( ( s : SessionLeafNode )
=> s . status === param . status ) ; }
async
getMerkleRoot ( ) :
Promise < string
{ return
( await
this . getSessionStore ( ) ) . merkleRoot ; }
async
setMerkleRoot ( merkleRoot :
string ) :
Promise < void
{ const data =
await
this . getSessionStore ( ) ; data . merkleRoot = merkleRoot ; await
this . writeDataToFile ( data ,
"sessions" ) ;
```

// Use 'sessions' as the type } }Previous Initialize Smart Account Next Session Module