**Anexă**

**Configurarea și inițializarea modulului Lisk**

import { Module**,** HttpModule } from '@nestjs/common'**;** import { LiskService } from './lisk.service'**;** import { LiskController } from './lisk.controller'**;** import { CryptoService } from './crypto.service'**;**

import { getNetworkIdentifier } from '@liskhq/lisk-cryptography'**;**

import { Application**,** genesisBlockDevneț configDevnet } from 'lisk-sdk'**;** import { APIClient } from '@liskhq/lisk-api-client'**;** import { Logger } from '../../../common/logger'**;**

import { AccountTransaction } from './transactions/account.transaction'**;** import { DEVNET\_URL } from '../../../common/env\_vars'**;**

import { OrderTransaction } from './transactions/order.transaction'**;**

configDevnet**.**modules**.**chain**.**forging**.**force **=** **true;**

genesisBlockDevnet**.**transactions[0] **=** {

id**:** '7646387794267587684'**,**

type**:** 8**,**

timestamp**:** 0**,**

senderPublicKey**:**

'edf5786bef965f1836b8009e2c566463d62b6edd94e9cced49c1f098c972b92b'**,**

signature**:**

'9f1282585cf91c9da0355f8e75c53363e50c0c1d41e96756b2bda02991ecb351bf67...'**,**

asset**:** { amount**:** '10000000000000000'**,** recipientId**:** '16313739661670634666L' }

}

export **const** lisknet **=** **new** APIClient(

process**.**env**.**MAINNET **?** APIClient**.**constants**.**MAINNET\_NODES **:**

APIClient**.**constants**.**TESTNET\_NODES

)**;**

export **const** devnet **=** **new** APIClient(

[DEVNET\_URL]

)**;**

export **const** networkIdentifier **=** getNetworkIdentifier( "23ce0366ef0a14a91e5fd4b1591fc880ffbef9d988ff8bebf8f3666b0c09597d"**,** "Lisk"**,**

)**;**

@Module({

imports**:** [HttpModule]**,**

providers**:** [{

provide**:** 'LISK\_SETUP'**,**

useFactory**:** () **=>** {

**const** app **= new** Application(genesisBlockDevnețconfigDevnet)**;** app**.**registerTransaction(AccountTransaction)**;** app**.**registerTransaction(OrderTransaction)**;** app

**.**run()

**.**then(() **=>** console**.**log('Lisk Connected'))

**.**catch(err **=>** {

console**.**error(err)**;**

process**.**exit(1)**;**

})**;**

}

}**,**

LiskService**,**

CryptoService

]**,**

controllers**:** [LiskController]

})

export **class** LiskModule {}

**Serviciul principal de logică tranzacțională (Snippets)**

export **function** timestamp() {

**const** millisSinceEpoc **=** Date**.**now() **-** Date**.**parse(EPOCH\_TIME**.**toString())**; const** inSeconds **=** ((millisSinceEpoc) **/** 1000)**.**toFixed(0)**; return** *parseInt*(inSeconds)**-**1**;**

}

**async** updateAccount(user**:** User) {

**const** tx **= new** AccountTransaction({

timestamp**:** timestamp()**,**

networkIdentifier**:** networkIdentifier**,**

asset**:** user**.**asset

})**;**

tx**.**sign(user**.**passphrase)**;**

**try** {

**const** result **= await** devnet**.**transactions**.**broadcast(tx**.**toJSON())log**.**info('Transaction result: '**,** result)**;**

**return** result**;**

} **catch**(err) { log**.**error(err)**;**

**throw new** Error("Fail on update account")**;**

}

}

**async** transact(buyOrder**:** BuyOrder**,** passphrase**:** string**,** asset**:** Asset) { **const** tx **=** trans**.**transfer({

amount**:** convertLSKToBeddows(buyOrder**.**offeredPrice**.**toString())**,**

networkIdentifier**:** networkIdentifier**,**

recipientId**:** buyOrder**.**sellerId**,**

passphrase**:** passphrase

})**;**

**try** {

**await** devnet**.**transactions**.**broadcast(tx)**;**

} **catch**(err) {

**throw new** InsufficientFunds("That wasn't very cash money of you")**;**

}

**try** {

**await this.**addAssetToUserPortfolio(assețpassphrase)**;** } **catch**(err) {

**throw new** Error("Failed to transfer asset to portfolio")

}

**try** {

**await this.**removeSellOrder(buyOrder**.**sellOrderId)**;** } **catch**(err) {

**throw new** Error("Failed to remove sell order")**;**

}

**return;**

}

**async** addSellOrder(sellOrder**:** SellOrder) {

**let** orderAccount**:** OrderAccount **= await this.**getOrderAccount()**;**

**let** asset**:** Orders **=** orderAccount**.**asset**;** asset**.**sellOrders**.**push(sellOrder)**;**

**const** tx **= new** OrderTransaction({

timestamp**:** timestamp()**,**

networkIdentifier**:** networkIdentifier**,**

targetAccountAddress**:** getAddressFromPassphrase(sellPass)**,**

asset**:** asset

})**;**

tx**.**sign(sellPass)**;**

**try** {

**const** result **= await** devnet**.**transactions**.**broadcast(tx**.**toJSON())**;** log**.**info('Transaction result: '**,** result)**;**

**return** result**;**

} **catch**(err) { log**.**error(err)**;**

**return** {status**:** 400**,** message**:** "Please try again"}**;**

}

}

**async** addBuyOrder(buyOrder**:** BuyOrder**,** passphrase**:** string) {

**let** orderAccount**:** OrderAccount **= await this.**getOrderAccount()**;**

**let** asset**:** Orders **=** orderAccount**.**asset**;**

asset**.**buyOrders**.**push(buyOrder)**;**

**const** relevantSellOrder **=** asset**.**sellOrders**.**filter((sellOrder) **=>** sellOrder**.**id **==** bu

**try** {

**await this.**transact(buyOrder**,** passphrase**,** relevantSellOrder**.**assetData)**;**

} **catch**(err) { log**.**error(err)**;**

**throw new** HttpException(err**.**message**,** 400)**;**

}

**const** tx **= new** OrderTransaction({

timestamp**:** timestamp()**,**

networkIdentifier**:** networkIdentifier**,**

targetAccountAddress**:** getAddressFromPassphrase(sellPass)**,**

asset**:** asset

})**;**

tx**.**sign(sellPass)**;**

**try** {

**const** result **= await** devnet**.**transactions**.**broadcast(tx**.**toJSON())**;** log**.**info('Transaction result: '**,** result)**;**

**return** result**;**

**...**

}

**Exemplu tranzacție personalizată**

export **class** AccountTransaction **extends** BaseTransaction {

**private** accountAsset**:** any**;**

**protected** verifyAgainstTransactions(transactions**:** readonly import("@liskhq/lisk-tra **throw new** Error("Method not implemented.")**;**

}

**protected** assetFromSync(raw**:** any)**:** object { **throw new** Error("Method not implemented.")**;**

}

constructor(transObj**:** TransactionInfo) {

**super**(transObj)**;**

}

**static** get TYPE() {

**return** 13**;**

}

**static** get FEE() {

**return** `0`**;**

}**;**

**async** prepare(store) {

**await** store**.**account**.**cache([

{

address**:** **this.**senderId

}**,**

])**;**

}

validateAsset() {

**const** errors **=** []**;**

**const** asset**:** any **= this.**asset**;**

**return** errors**;**

}

applyAsset(store) {

**const** errors **=** []**;**

**const** asset**:** Partial**<**User**> = this.**asset**;**

**let** sender**:** any **=** store**.**account**.**get(**this.**senderId)**;**

**const** newObj **=** { **...**sender**,** asset**:** asset }**;** store**.**account**.**set(sender**.**addresș newObj)**;**

**return** errors**;**

}

undoAsset(store) {

**const** sender **=** store**.**account**.**get(**this.**senderId)**; const** nullAssetObj **=** { **...**sender**,** asset**: null** }**;** store**.**account**.**set(sender**.**addresș nullAssetObj)**;**

**return** []**;**

}

assetToBytes() {

**return** Buffer**.**from(JSON**.**stringify(**this.**asset))**;**

}

assetToJSON() {

**return this.**asset**;**

}

}