Algorand útikalauz stopposoknak

Lipovszki Gábor

2025. jan. 19. 18:54

Tartalomjegyzék

1. Bevezetés **6**
1.1. Néhány szó az Algorandról 6
1.2. Erőforrások az interneten 8
1.2.1. Youtube videók fejlesztőknek 8
1.3. Algorand pénztárcák 9
1.4. Az Algorand blokklánc csomópontjai 10
1.5. Interakció az Algorand blokklánccal 12
1.6. SDK-k 13
1.7. Az Algorand fejlesztő eszközei 13

2. AlgoKit installálás **16**
2.1. AlgoKit installálás `codespaces` alatt 17
2.2. AlgoKit installálás Windows alatt 18
2.2.1. Windows verzió ellenőrzés 18
2.2.2. `VS Code` installálás 18
2.2.3. Git installálás 18
2.2.4. WSL installálás 18
2.2.5. Docker Desktop installálás 19
2.2.6. Node.js installálás 20
2.2.7. Python installálás 20

2.2.8. pipx installálás 21
2.2.9. AlgoKit installálás 21
2.3. Az AlgoKit használata 21

3. Néhány példa az Algorand parancsok használatára 25

4. Néhány példa a Python SDK használatára 30
4.1. Előkészületi lépések 30
4.2. Algorand számla létrehozása 32
4.3. Számla egyenleg kiírása 32
4.4. Fizetési tranzakció kiadása 34
4.5. Számlaegyenlegek kiírása 37
4.6. Pénzeszköz létrehozása 41
4.7. Pénzeszközbe történő „benevezés”, opt-in 44
4.8. Csere atomi tranzakciós csoporttal 47

5. Fejlesztés a Javascript SDK-val 55
5.1. Előkészületi lépések 55
5.2. Algorand számlaszám létrehozása 56
5.3. Számla egyenleg kiírása 57
5.4. Fizetési tranzakció kiadása 59
5.5. Számlaegyenlegek kiírása 63
5.6. Pénzeszköz létrehozása 68
5.7. Pénzeszközbe történő „benevezés”, opt-in 72
5.8. Csere atomi tranzakciós csoporttal 77
5.9. Node.js Web-es alkalmazások 84
5.9.1. Pera WalletConnect mintapélda 84
5.10. Közvetlenül a Web böngészőre támaszkodó alkalmazásfejlesztés 96
5.10.1. A `@perawallet/connect` összepakolása 96
5.10.2. Csocsó eredmények rögzítése az Algorand Testnet blokkláncon 101
5.11. Az első rész összefoglalása 109

6. A TEAL 111

6.1. A TEAL program elemei 111
6.2. Az AVM felépítése (architektúrája) 112
6.3. AVM adattípusok 112
6.3.1. Alulcsordulás és túlcsordulás kezelés 113
6.4. Algorand Smart Signatures 116
6.5. Példa az ASS használatára 116
6.6. Algorand alkalmazások 116
6.6.1. A bytekód maximális mérete 116
6.6.2. A bytekód maximális végrehajtási költsége 116
6.7. Példa az Algorand alkalmazások használatára: “Hello, World” 117
6.7.1. A backend 121
6.7.2. A frontend 128
6.7.3. Az ABI 130
6.7.4. “HelloWorld” mintaprogram ABI használata nélkül . 131
6.7.5. “HelloWorld” mintaprogram ABI használatával . . . 132
6.7.6. A TEAL kód debug-olása 137

7. Esettanulmány: tulajdonrész opciós vételi jog 141
7.1. A feladat leírása 141
7.2. A feladat elvi megoldása 141
7.3. Az okos szerződés magyarázata 143
7.3.1. Az okos szerződés osztályának definiálása 143
7.3.2. A globális állapotváltozók definiálása 144
7.3.3. `createApplication` – az okos szerződés létrehozása után meghívva 145
7.3.4. `bootstrap` – kezdeti paraméterek beállítása 145
7.3.5. A globális állapotváltozók visszaolvasása 147
7.3.6. `buyAsset` – zseton vétele 148
7.3.7. `sendAlgosToCreator` – zseton vételár visszaküldés . 151
7.3.8. `clawback` – zseton visszavétele 152
7.3.9. `clawbackNoIncAmount` – zseton bevonása 153
7.3.10. `deleteAsset` – az ASA törlése 153

7.3.11. `deleteApplication` – az okos szerződés törlése előtt meghívva . . . 154
7.4. Az okos szerződés tesztelése . . . 155
7.4.1. Az egyes Jest tesztek előtti beállítások . . . 156
7.4.2. `bootstrap` teszt . . . 159
7.4.3. `getAppVersion` teszt . . . 160
7.4.4. `getAppCreatorAddress` teszt . . . 160
7.4.5. `getAssetAmountInitial` teszt . . . 161
7.4.6. `getAssetAmount` teszt . . . 161
7.4.7. `getAssetPrice` teszt . . . 161
7.4.8. `getAssetId` teszt . . . 161
7.4.9. `getSellPeriodEnd` teszt . . . 161
7.4.10. `getGlobalState` teszt . . . 162
7.4.11. `opt in to asset` teszt . . . 164
7.4.12. `buyAsset` teszt . . . 164
7.4.13. `buyAsset 2nd time` teszt . . . 167
7.4.14. `sendAlgosToCreator` teszt . . . 168
7.4.15. `clawback` teszt . . . 168
7.4.16. `buyAsset after clawback` teszt . . . 169
7.4.17. `clawback again` teszt . . . 170
7.4.18. `'opt out buyer from asset'` teszt . . . 170
7.4.19. `'deleteAsset'` teszt . . . 171
7.4.20. `'deleteApplication'` teszt . . . 171
7.5. A tesztek futtatása . . . 171
7.6. A frontend . . . 177
7.6.1. A frontend oldali npm scriptek . . . 178
7.6.2. A fronend fejlesztés menete . . . 179
7.6.3. Algorand specifikus részek . . . 180

8. Fejlesztői környezet upgrade, 2024. okt. 29. 186
8.1. git upgrade . . . 186
8.2. Docker Desktop upgrade . . . 187
8.3. node.js upgrade . . . 188

8.4. Python upgrade 189
8.5. pipx upgrade 189
8.6. algokit újrainstallálás 190
8.7. Utómunkák 190

9. Fejlesztői környezet upgrade, 2024. dec. 17. 192
9.1. git upgrade 192
9.2. Docker Desktop upgrade 193
9.3. node.js upgrade 194
9.4. Python upgrade 195
9.5. pipx upgrade 195
9.6. algokit upgrade 196
9.7. Utómunkák 197

10. Fejlesztői környezet upgrade, 2025. jan. 09. 199
10.1. `VS Code` újratelepítés 199
10.2. git upgrade 200
10.3. Docker Desktop upgrade 200
10.4. node.js upgrade 201
10.5. Python upgrade 202
10.6. pipx upgrade 203
10.7. algokit upgrade 203
10.8. Utómunkák 204

1. Bevezetés

Ez az Útikalauz segít eligazodni a „stopposoknak”, azaz a fejlesztőknek az Algorand blokklánc kacskaringós, zegzugos világában. Nem csak elméleti áttekintést nyújt, hanem példákkal is szemlélteti az egyes jelenségeket.

Ne ess pánikba! – ahogy azt az eredeti Útikalauz mondja.

Messze kerüld el! – a feleségem véleménye az Algorandról.

Aki tudja, csinálja, aki nem tudja, tanítja – magyar közmondás

1.1. Néhány szó az Algorandról

Az Algorand-ot Silvio Micali alkotta. Az motiválta, hogy a blokklánc új blokkjai minimális erőforrás-használattal jöjjenek létre, ugyanakkor a biztonság se szenvedjen csorbát. Ennek érdekében egy új kriptográfiai primitívet alkotott, ez a Verifiable Random Function (VRF). Ezzel a konszenzus úgy teremthető meg, hogy az egyes fázisokban a bizottsági tagokat a VRF segítségével, véletlenszerű sorsolással választják ki. Minden lekötött, “staked” Algó egy szavazatot jelent a sorsolás során. Ugyanakkor a VRF-fel utólag ellenőrizhető, hogy jogos volt-e az a fajta véletlen választás, ami bekövetkezett.

Az Algorand elnevezés is Micali professzornak köszönhető: az algorithmic és a random szavak összetételéből született. Összesen 10 milliárd Algó van, ezek a hálózat létrehozásakor egyszerre mind létrejöttek. A kisbefektetők a különféle tőzsdéken tudnak Algót venni. Jelenleg kb. 8,3 milliárd Algó van forgalomban, a maradék Algorand pénzmennyiségből az Algorand Foundation különféle programokat finanszíroz – pl. az önigazgatást (governance), fejlesztéseket, DeFi (Decentralised Finance) támogatást stb.

Az Algorand hálózat 2019-ben kezdte meg a működését. Jelenleg (2024 októberében) kb. 7500 tranzakció/sec az elérhető maximális sebessége, és kb. 2,9 sec alatt válik egy tranzakció véglegessé. A blokklánc elágazás valószínűsége rendkívül kicsi, 10^{-18}, vagyis elágazás évmilliárdok alatt fordul-

hat csupán elő. A tranzakciós díj jelenleg 0,001 Algó, ami a jelenlegi Algó árfolyam mellett kb. 10 fillér.

A kiváló műszaki paraméterek ellenére a CoinCodex szerint az Algorand a tőkésítettségi mutató szempontjából jelenleg (2024 októberében) a hetvenkettedik helyen áll a digitális pénzek rangsorában. A közösség nem díjazta, hogy az Algorand relé hálózatát lényegében „csókosok" üzemeltetik a jelen pillanatban is, és akiknek az Algorand Foundation évente súlyos dollármilliókat fizet.

Az Algorand Foundation főmérnöke az idén, 2024-ben oly módon szeretne ezen változtatni, hogy az új blokkok előállításáért pénz (Algó) járjon, vagyis egyfajta „bányászat" legyen bevezetve a hálózatban. Azt reméli, hogy a relé hálózat fokozatosan leállítható lesz, és az Algorand önfenntartó hálózattá válik. Mivel a bányászat az Algó lekötésével végezhető ("proof of stake", „kockázati bizonyíték" alapján), a blokkok előállításának jutalmazásától azt várja, hogy a jelenlegi 15% körüli tőkelekötés legalább 25%-ra nő az idén, továbbá azt, hogy az Algorand bekerül az első 20 digitális pénz közé.

Mivel a tranzakciós díjak rendkívül alacsonyak, és ezt szeretnék a jövőben is alacsonyan tartani, az első 2–3 évben a tranzakciós díjakból adódó bányászati jutalmat az Algorand Foundation egészítené ki. A főmérnök reményei szerint az Algorand árfolyamnövekedése, a hálózat tranzakciószámának növekedése, valamint a tranzakciós díj észszerű növelése a bányász folyamatot az első 2–3 év után önfenntartóvá fogja tenni.

Az Algorand Foundation idei, 2024-es fejlesztési tervei (roadmap) a következőket tartalmazzák:

- dinamikus blokk idő bevezetése, vagyis a bizottsági tagok leglassabb 5%-ának kihagyása a konszenzus teremtés során, ami további blokk idő csökkentést tesz lehetővé,
- az AlgoKit 2.0-ban az Algorand szerződéseket natív Python nyelven lehet megírni, ami megszűntetheti a fejlesztői szűk keresztmetszetet,

- a nem archív relé csomópontok bevezetésével a tárolt adatmennyiség csökkenthető,
- új blokkok előállításnak jutalmazásával, a „bányászattal” a lekötött Algó mennyisége növelhető,
- a relé típusú hálózati topológiáról a P2P (peer to peer) gossip hálózati topológiára történő fokozatos átállással a Bitcoinhoz és a többi digitális pénzhez hasonló rendszer kialakítása.

1.2. Erőforrások az interneten

- Az Algorand Technologies (korábban Algorand, Inc.) honlapja.
- Az Algorand Foundation honlapja
- Az Algorand Fórum
- Meghívó az Algorand Discord szerverére
- Algorand linkgyűjtemény: Awesome Algo
- Algorand blokklánc vizsgálók: lora, allo’, pera
- Magyar honlap: Algorand.hu

1.2.1. Youtube videók fejlesztőknek

A fejlesztőknek szóló Youtube videók a `https://youtube.com/@algodevs` címen érhetőek el. A „Lejátszási listák” fülre kattintva megjelennek az egyes témákhoz tartozó videók. Néhány ezek közül:

- Algorand Development Environment Setup (fejlesztői környezet létrehozása)
- AVM Explained (az Algorand Virtuális Gép ismertetése)
- PyTeal Tutorial for Beginners (Full Course) (PyTeal tanfolyam)
- Beaker for Beginners (Full Course) (Beaker tanfolyam)

- TealScript for Beginners (Full Course) (TealScript tanfolyam)
- The Differences Between Ethereum & Algorand (Az Ethereum és az Algorand közötti különbségek)
- Bonus Content: Algorand Foundation CTO John Woods (Bónusz: Interjúk az Algorand Foundation főmérnökével, John Woods-szal)
- Beginner Algorand Bootcamp [April 2023] (Algorand képzés kezdőknek, 2023. április)

1.3. Algorand pénztárcák

A felhasználó az Algorand pénztárcán keresztül kerül közvetlen kapcsolatba az Algorand blokklánccal. A felhasználói élményt alapvetően meghatározza a pénztárca minősége, használhatósága.

A pénztárca szoftver vagy hardver pénztárca lehet. A hardver pénztárcák jellemzője, hogy a titkos kulcsot egy kriptográfiailag biztonságos memóriában tárolják, amelyből az adat közvetlenül nem olvasható ki. A hardver pénztárcák a tranzakciók aláírását egy külön célhardverrel végzik, amelyen a felhasználó explicit beavatkozása (gombnyomás) szükséges egy tranzakció aláírásához.

Az egyik legismertebb hardver pénztárca gyártó a Ledger. A Ledger Live alkalmazás a Ledger hardver pénztárcához tartozó kezelőfelület. A Ledger hardver pénztárcával sokféle digitális pénz kezelhető, többek között az Algorand is.

A hivatalos Algorand pénztárca a Pera pénztárca. A Pera kétféle pénztárcát ajánl: web-es felületű pénztárcát és iOS ill. Android alkalmazással megvalósított pénztárcát. A web-es pénztárca biztonsági okok miatt kivezetésre került, jelenleg már csak számlák egyenlege figyelhető meg vele. Az iOS és az Android alkalmazások az App Store-ból vagy a Google Play-ről tölthetők le.

Az Algorand blokkláncnak számos olyan jellemzője van, melyet a hiva-

talos Pera pénztárca még nem támogat, vagy csak részben támogat. Néhány ilyen jellemző:

- a Ledger hardver pénztárca USB-vel nem használható,
- a multisig számlák nincsenek támogatva,
- az Algorand alkalmazások hívása WalletConnect interface használatával lehetséges
- a kijelölt számlák titkos kulcsának együttes mentése egy újabb, 12 szóból álló jelmondat (passphrase) megadásával lehetséges.

Egy másik pénztárca az A-Wallet, amely egy web-es pénztárca. Az A-Wallet támogatja a Ledger HW pénztárcák használatát. Sok érdekes jellemzője van, pl. a multisig (többszörös aláírás) támogatása, a Google Authenticator használata a kétfaktoros aláírás támogatásához, a payment gateway támogatása, szavazás támogatása a blokkláncon stb.

1.4. Az Algorand blokklánc csomópontjai

Ha szeretnénk egy olyan alkalmazást fejleszteni, amely az Algorand blokkláncot használja, akkor ehhez az alkalmazásból el kell tudnunk érni az Algorand valamelyik publikus hálózati csomópontját, vagy saját (privát) hálózati csomópontot kell létrehoznunk. Az Algorand blokkláncban többféle hálózati csomópont lehetséges:

- relé csomópontok, melyek gyors hálózati forgalmat biztosítják
- archív csomópontok, amelyek a teljes Algorand blokklánc történetét tárolják. A blokkláncban való gyors kereshetőséget egy Indexer-nek nevezett prostgreSQL adatbázis kezelő biztosítja.
- nem archív csomópontok, amelyek helytakarékosság miatt mindegyik Algorand számlára vonatkozóan csak az utolsó 1000 tranzakciót tárolják.

Az archív csomópontok elérése szabványos REST API felületen keresz-

tül lehetséges. Például a `nodely.io` ingyenesen használható Algorand végpontjai a playground linken keresztül biztosítják, hogy kipróbálhassuk a REST API parancsainak a működését. Az alacsony szintű curl parancsok használata helyett a fejlesztők SDK-kon (System Development Kit, Rendszerfejlesztő csomag) keresztül tudják kényelmesen elérni az egyes REST API funkciókat.

A fejlesztő dönthet úgy, hogy saját hálózati csomópontot üzemeltet. A saját csomópont archív vagy nem archív csomópont lehet. A csomópontok üzemeltetésével kapcsolatos HW előfeltételek leírása az előbbi hiperhivatkozásnál található meg.

Saját csomópont Linux vagy Docker alatt telepíthető. Linux alatt vannak bináris telepítő készletek, és csomagba szervezett telepítő készletek.

A telepített Algorand csomópont alkotó elemei:

- algod, Algorand démon, amely a blokklánc kezelését végzi
- kmd, Key Management démon, amely a pénztárca kezelését végzi
- indexer, Indexer, a blokklánc adatok gyors elérését biztosító adatbázis. Egyelőre csak docker-ben használható
- conduit, az Algorand blokklánc adatainak eljuttatása külső alkalmazások számára. Egyelőre csak Docker-ben használható
- parancssorban használható programok, pl. `goal`, `kmd`

Az Algorand Foundation a fejlesztők munkájának megkönnyítése érdekében kifejlesztette az AlgoKit-et, amely Docker felület alá telepít egy Algorand csomópontot, és a fejlesztést elősegítő környezetet.

A fejlesztés során használható Algorand blokklánc fajták:

- privát blokklánc, egyetlen csomópontja a fejlesztői gépen van.
- betanet, teszt blokklánc a legelső próbákhoz. Saját bétanet teszt Algót használ, amely a https://bank.betanet.algodev.network/ címről in-

gyenesen letölhető.

- testnet, teszt blokklánc, amely nagyon hasonlít az éles blokkláncra. Saját teszt Algó-t használ, amely a https://bank.testnet.algorand.network/ címről ingyenesen letölthető.
- mainnet, éles blokklánc

1.5. Interakció az Algorand blokklánccal

- Parancsnyelv: Ha saját csomópontot üzemeltetünk, akkor az `algod`, `kmd` és `indexer` processzek kezelése shell parancsokkal lehetséges, pl. a `goal` és a `kmd` parancsok a leginkább használtak.
- REST API: A saját csomópontunk szolgáltatásai REST API felületen keresztül is elérhetőek: a `goal`, `kmd` és `indexer` processzek egy-egy REST API felületet is biztosítanak. Ezen szolgáltatások eléréshez API kulcsokra van szükség. Ha publikáljuk ezeket a kulcsokat, akkor a csomópontunkat mások is használhatják.
- SDK-k: A REST API felületeket az Algorand SDK-k „csomagolják be", vagyis a különböző nyelvi környezetekből (JavaScript, PHP, Java, Rust stb.) kényelmes hozzáférést biztosítanak a processzek REST API felületéhez.

Saját csomópont üzemeltetése többféleképpen lehetséges:

- Linux alatt Algorand blokklánc csomópont üzembe helyezésével, lásd Install a node, ezen belül az update script használatával lásd Installing on Linux
- Linux alatt privát Algorand blokklánc csomópont üzembe helyezésével, lásd Create a private network
- Docker (és Docker Compose) alatt: Algorand Sandbox üzembe helyezésével, lásd Algorand Sandbox

1.6. SDK-k

Az Algorand blokklánc a különféle nyelvi környezetekből az SDK-k segítségével érhető el. Az Algorand által biztosított „hivatalos” SDK-k a következők:

- Javascript Algorand SDK
- Python Algorand SDK
- Go Algorand SDK
- Java Algorand SDK

A fejlesztői közösség által készített további SDK-k a következők:

- PHP Algorand SDK
- .NET Algorand SDK
- Rust SDK
- Swift SDK
- Unity SDK

1.7. Az Algorand fejlesztő eszközei

Az Algorand fejlesztőeszközei a hagymahéjakhoz hasonlóan számos rétegben érhetők el. Ennek részben történeti okai vannak. Az egyes szintek a következők:

- bináris szint, az Algorand Virtuális Gép (Algorand Virtual Machine, a továbbiakban AVM) utasításainak a szintje. Az Algorand fejlesztői dokumentáció részletesen leírja az AVM architektúráját és az egyes utasítások működését, lásd v10 opkódok.
- assembly szint: a TEAL assembly szintű programozási nyelv. (A TEAL a Transaction Execution and Approval Language rövidítése.) A TEAL nyelven megírt assembly programot a TEAL compiler fordítja

le az Algorand Virtuális Gép bináris kódjává. A TEAL compiler mind a parancsnyelvi felületen, mind az SDK-kon keresztül elérhető.

- compiler szint: a PyTeal compiler egy Python nyelvben megírt „pszeudo"-Python kódot fordít TEAL-re. A használhatóságát megnehezíti, hogy keveredik benne a normál Python kód és PyTeal fordítónak szóló „pszeudo"-Python kód. A TEAL-hez képest sokkal áttekinthetőbb kód írható benne a vezérlési struktúrák használhatósága miatt.
- compiler szint: a Beaker compiler a PyTeal továbbfejlesztett változata, amely Python objektumként teszti használhatóvá az Algorand blokkláncon használható alkalmazást. Az Algorand blokkláncon az alkalmazás nagyjából az Ethereum szerződésnek felel meg, de számos korlátozással biztosították a blokklánc fejlesztők, hogy az Algorand blokklánc működése gyors maradjon. Mivel az Algorand blokklánc alkalmazások csak 1-2 évvel az „alap" AVM után jelentek meg, így a Beaker is szükségszerűen későbbi „találmány".
- compiler szint: a PuyaPy Python -> TEAL compiler, amely a Python nyelv egy részhalmazáról fordít az Algorand Virtuális Gép assembly nyelvi szintjére (TEAL).
- compiler szint: a TealScript fordítóprogram a TypeScript nyelv egy részhalmazából fordít TEAL szintre.

A fejlesztő eszközök keretrendszerekbe vannak szervezve. Az AlgoKit az egyik legismertebb. Az AlgoKit biztosítja a fejlesztő számára:

- az Algorand blokklánc elérését. A blokklánc lehet privát, vagy a Betanet, Testnet, Mainnet valamelyike. A blokklánc elérése Docker és Docker Compose segítségével, az Algorand Sandbox segítségével történik.
- a PyTeal, Beaker, PuyaPy fejlesztő eszközök elérését. A fejlesztő eszközök számára virtuális környezetet teremt, és a függőségeket is kezeli.
- a TealScript fejlesztő eszköz elérését

- mintapéldákat
- automatikus felhasználói felület generálást az Algorand alkalmazás alapján

Blokklánc vizsgálóként az AlgoKit mellett a LORA ajánlott, mert képes nem csak a Betanet, Testnet, Mainnet tranzakciók megjelenítésére, hanem az Algorand Sandbox által kezelt privát blokklánc tranzakcióinak megjelenítésére is.

Egy másik keretrendszer a TealCraft keretrendszer, amely lehetővé teszi a TealScript böngészőből történő közvetlen használatát.

2. AlgoKit installálás

Az AlgoKit az Algorand Foundation által kifejlesztett keretrendszer, amellyel biztosítható:

- egy lokális (privát) hálózat felállítása, indítása, leállítása, vagy kapcsolódás az Algorand Betanet, Testnet vagy Mainnet hálózatához, lásd `algokit local` parancsok
- fejlesztői keretrendszerek inicializálása, példák betöltése, lásd `algokit init` parancs
- virtuális környezet megteremtése a fejlesztéshez, lásd `algokit bootstrap all` parancs

Az AlgoKit installálásának előfeltételei Windows 10 vagy Windows 11 operációs rendszer alatt:

- `VS Code` (Visual Studio Code) installálása
- Git installálása
- WSL2 (Windows Subsystem for Linux) installálása
- Docker Desktop installálása
- Python installálása
- pipx installálása

Az installálás lépéseiről egy jó összefoglaló található a YouTube-on, lásd Ryan Fox Setup Your Algorand Development Environment on Windows in 10 Minutes (Hozd létre Windows-on az Algorand fejlesztői környezetedet 10 perc alatt) című videóját.

A Github alatt regisztrált felhasználók ingyen havi 120 óra virtualizált környezethez jutnak a codespaces alatt. Ez a környezet web böngészőből használható. Az AlgoKit ebben a környezetben minimális számú lépéssel installálható, mivel a `codespaces` automatikusan biztosítja a következő

komponensek használatát:

- `VS Code` szövegszerkesztő
- `git` verziókezelő
- Docker és Docker Composer
- Node.js

2.1. AlgoKit installálás `codespaces` alatt

A szükséges lépések:

- Lépjünk be a github.com alatti számlánkba
- Készítsünk egy új repository-t
- Menjünk az új repository-ba. Nyomjuk meg a `Code` gombot, válasszuk a `Codespaces` tab-ot.
- Nyomjuk meg a "Create codespace on main" gombot. Létrejön egy új fejlesztői környezet, `VS Code` szerkesztővel.
- Ha nem jelenik meg terminál fül, akkor kattintsunk a ≡ -ra, majd válasszuk a `Terminal` | `New terminal` menüpontokat.
- A terminál fülön adjuk ki: `pipx install algokit`
- A terminál fülön adjuk ki: `algokit localnet start`

Egy-két perc alatt installálódik docker alatt egy Algorand lokális hálózati csomópont.

Készen is vagyunk! Ugorjunk az *Az AlgoKit használata* című következő részhez a 21 oldalra, amely néhány példával az AlgoKit használatát mutatja be!

2.2. AlgoKit installálás Windows alatt

2.2.1. Windows verzió ellenőrzés

Menjünk a `Vezérlőpult` | `Rendszer` lapra. Ellenőrizzük az OS verziószámát. A minimális támogatott verzió: 19045.[1]

2.2.2. `VS Code` installálás

Menjünk a VS Code Download weblapra. Nyomjuk meg a "Download, Windows 10, Windows 11" gombot. Windows esetén `user installer` vagy `system installer` egyaránt letölthető. Indítsuk el a letöltött fájlt. Indítsuk el a `VS Code`-ot.

2.2.3. Git installálás

A `VS Code`-ban a 'View | Terminal' menüelemekre kattintva indítsunk egy `Power Shell` terminál ablakot. A terminál ablakban adjuk ki:

```
git --version
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
winget install --exact --id Git.Git
```

Ellenőrizzük, hogy sikerült-e az installálás:

```
git --version

# => git version 2.44.0.windows.1
```

2.2.4. WSL installálás

A `Windows Subsystem for Linux` installálására azért van szükség, mert a `Docker Desktop` a Linux container-eket ennek a segítségével futtatja.

A `VS Code` terminál ablakában adjuk ki:

[1] A docker.desktop 4.35.0 által megkövetelt minimális Windows verzió 19045 vagy későbbi

```
wsl -l -v
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
wsl --install
```

Amikor megjelenik egy új terminál ablak, adjuk meg az új UNIX felhasználónevünket és jelszavunkat:

```
Enter new UNIX username: felhasználónév
New password: jelszó
Retype password: jelszó újra
```

Írjuk be a start sorba: `Ubuntu`, és indítsuk el az Ubuntu alkalmazást.

A `VS Code` terminál ablakban adjuk ki ismét:

```
wsl -l -v
```

A következő fog megjelenni:

```
  NAME      STATE           VERSION
* Ubuntu    Running         2
```

2.2.5. Docker Desktop installálás

A `VS Code` terminál ablakában adjuk ki:

```
docker version
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
winget install --exact --id Docker.DockerDesktop
```

Az installálás befejeződése után indítsuk újra a Windows-t.

A Docker Desktop ikonra kattintva indítsuk el a Docker-t. Fogadjuk el a feltételeket (“Accept”). Megjegyzés: lehet, hogy az “Accept” az indító ablak mögött van „elbújva”.

A `VS Code` terminál ablakában adjuk ki:

```
docker version
```

Most már nem kapunk hibaüzenetet.

2.2.6. Node.js installálás

A Node.js installálása a TealScript és a React frontend működéséhez szükséges. A `VS Code` terminál ablakában adjuk ki:

```
node --version
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
winget install --exact --id OpenJS.NodeJS
```

Ellenőrzésképpen adjuk ki a következő parancsokat:

```
node --version

# => v21.7.3

npm --version

# => 10.5.0
```

2.2.7. Python installálás

A Python installálása a PuyaPy fordítóprogram működéséhez szükséges, de a korábbi PyTeal és Beaker is Python kódot használ. A `VS Code` terminál ablakában adjuk ki:

```
python --version
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
winget install python.python.3.12
```

Megjegyzés: érdemes a legújabb *stabil* Python verziót installálni, de minimum Python 3.11-et.

2.2.8. pipx installálás

Csukjuk be a VS Code-ot, és indítsuk el újra, hogy a VS Code lássa a path-on a python parancsot. A VS Code terminál ablakban adjuk ki:

```
pipx --version
```

Ha hibaüzenetet kapunk, akkor adjuk ki:

```
python -m pip install pipx
```

A VS Code terminál ablakban adjuk ki ismét:

```
pipx --version

# => 1.5.0
```

Most már nem kapunk hibaüzenetet.

2.2.9. AlgoKit installálás

A VS Code terminál ablakában adjuk ki:

```
pipx install algokit
```

2.3. Az AlgoKit használata

Ellenőrizzük, hogy sikerült-e az installálás. A codespaces vagy a VS Code terminál ablakában adjuk ki:

```
algokit --version

# => algokit, version 2.0.3
```

Indítsuk el az Algorand lokális hálózati környezetet:

```
algokit localnet start
```

Ellenőrizzük, hogy sikeresen elindultak-e a Docker környezet alatt a containerek. A Windows alatt a Docker Desktop ablakban láthatók a konténerek és az elindított image-ek. A codespaces környezetben a

```
docker container ls
```

parancsot kell kiadnunk. Öt image-et kell látnunk:

- `nginx:1.27.0-alpine`
- `algorand/indexer:latest`
- `algorand/conduit:latest`
- `algorand/algod:latest`
- és `postgres:16-alpine`

Ezek a konténerek futtatják az algokit proxy-t, az indexer-t, a conduit-ot (ez egyfajta eseménykezelő), az algod Algorand démont és a postgres adatbáziskezelőt.

A Docker image-ek helyes működése a következő paranccsal is ellenőrizhető:

```
algokit localnet status
```

Példa:

```
@A-Maugli → /workspaces (main) $ pipx install algokit
  installed package algokit 1.13.1, installed using Python 3.10.13
  These apps are now globally available
    - algokit
done! ★ ★
@A-Maugli → /workspaces (main) $ algokit localnet start
Starting AlgoKit LocalNet now...
docker: conduit Pulling
docker: algod Pulling
docker: indexer-db Pulling
docker: indexer Pulling
docker: indexer Pulled
docker: conduit Pulled
docker: indexer-db Pulled
docker: algod Pulled
docker: Network algokit_sandbox_default  Creating
docker: Network algokit_sandbox_default  Created
```

```
docker: Container algokit_sandbox_algod  Creating
...
docker: Container algokit_sandbox_algod  Waiting
docker: Container algokit_sandbox_conduit  Waiting
docker: Container algokit_sandbox_algod  Healthy
docker: Container algokit_sandbox_postgres  Healthy
docker: Container algokit_sandbox_indexer  Healthy
docker: Container algokit_sandbox_conduit  Healthy
Started; execute `algokit explore` to explore LocalNet in a web user interface.
@A-Maugli → /workspaces (main) $ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED       STATUS          PORTS
          NAMES
9690c346b3db   nginx:1.27.0-alpine        "/docker-entrypoint...."   3 days ago   Up 11 seconds   0.0.0.0:4001-4002->4001-4002/tcp, 80/tcp, 0.0.0.0:8980->8980/tcp
          algokit_sandbox_proxy
822b6a5cd201   algorand/indexer:latest    "docker-entrypoint.s..."   3 days ago   Up 12 seconds
          algokit_sandbox_indexer
23287957b7ed   algorand/conduit:latest    "docker-entrypoint.sh"   3 days ago   Up 10 seconds
          algokit_sandbox_conduit
96c14faf89c4   algorand/algod:latest      "/node/run/run.sh"       3 days ago   Up 14 seconds   4160/tcp, 8080/tcp, 9100/tcp, 0.0.0.0:9392->9392/tcp, 0.0.0.0:61091->7833/tcp
          algokit_sandbox_algod
e8e6b7d5e071   postgres:16-alpine         "docker-entrypoint.s..."   3 days ago   Up 14 seconds   0.0.0.0:5443->5432/tcp
          algokit_sandbox_postgres
@A-Maugli → /workspaces (main) $ algokit explore
Opening localnet explorer in your default browser
@A-Maugli → /workspaces (main) $ algokit localnet status
# container engine
Name: docker (change with `algokit config container-engine`)
# algod status
Status: Running
Port: 4001
Last round: 12
Time since last round: 0.0s
```

```
Genesis ID: dockernet-v1
Genesis hash: lJ8dlUKJNEJFsOykfc+fNDxIkxTdc13XLGBHGEFDdrY=
Version: 3.27.0
# conduit status
Status: Running
# indexer-db status
Status: Running
# indexer status
Status: Running
Port: 8980
Last round: 12
Version: 3.7.1
# proxy status
Status: Running
@A-Maugli → /workspaces (main) $
```

A lokális Algorand csomópont működését egy böngészőben a `algokit explore` paranccsal tudjuk ellenőrizni. A `codespaces` környezetben ennek kiadása előtt menjünk a `Ports` fülre, és a `Visibility` oszlopban kattintsunk a jobb egér gombbal a 4001, 4002 és 8980 portoknak megfelelő helyekre, majd válasszuk a `Port visibility` | `Public` menüpontokat.

Ezt követően válasszuk a `Terminal` fület, és adjuk ki a `algokit explore` parancsot. A böngészőben felpattanó ablakban nyomjuk meg az `OK` gombot!

3. Néhány példa az Algorand parancsok használatára

A parancsnyelvi környezet a következő paranccsal érhető el:

```
algokit localnet console
```

`codespaces` használatakor a következő parancsok előzetes kiadására van szükség:

```
pipx install algokit
```

```
algokit localnet start
```

```
algokit localnet stop
```

```
sudo chown -R codespace:codespace  /.config/algokit
```

Példa:

```
@A-Maugli → /workspaces (main) $ algokit localnet console
Opening Bash console on the algod node; execute `exit` to return to
↪  original console
root@ea05aa8270b7:~# goal wallet list
##################################################
Wallet: unencrypted-default-wallet
ID:     b971fb4cc5463c57a8563eed3413c0de
##################################################
root@ea05aa8270b7:~# goal account list
[online]
↪  NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
↪  NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
↪  4000000000000000 microAlgos
[online]
↪  3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
↪  3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
↪  4000000000000000 microAlgos
[online]
↪  4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
↪  4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
↪  2000000000000000 microAlgos
```

```
root@ea05aa8270b7:~# goal wallet new w1
Please choose a password for wallet 'w1':
Please confirm the password:
Creating wallet...
Created wallet 'w1'
Your new wallet has a backup phrase that can be used for recovery.
Keeping this backup phrase safe is extremely important.
Would you like to see it now? (Y/n): y
Your backup phrase is printed below.
Keep this information safe -- never share it with anyone!

ordinary usage hockey nurse shop rebel picnic female element guitar
↪  furnace rain enforce drum metal ostrich arrow safe immune melody dog
↪  mule organ abandon cannon
root@ea05aa8270b7:~# goal account new
Created new account with address
↪  6BW3G56B6PNWB2CZVXQNKYTDW5R5SLVEDMVEM7I2FBIZ2XRTEH7EACLAOY
root@ea05aa8270b7:~# goal account list
[online]
↪  NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
↪  NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
↪  4000000000000000 microAlgos
[online]
↪  3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
↪  3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
↪  4000000000000000 microAlgos
[online]
↪  4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
↪  4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
↪  2000000000000000 microAlgos
[offline]       Unnamed-0
↪  6BW3G56B6PNWB2CZVXQNKYTDW5R5SLVEDMVEM7I2FBIZ2XRTEH7EACLAOY      0
↪  microAlgos    *Default
root@ea05aa8270b7:~# goal account new --wallet w1
Please enter the password for wallet 'w1':
Created new account with address
↪  RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM
root@ea05aa8270b7:~# goal account list
```

```
35 [online]
   ↪ NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
   ↪ NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM
   ↪ 4000000000000000 microAlgos
36 [online]
   ↪ 3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
   ↪ 3B2D7UNBANXVZWPTIFWQZ7AC7MPPLBKCWHAL5MNJPGMAGEZQQA7TDMG57E
   ↪ 4000000000000000 microAlgos
37 [online]
   ↪ 4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
   ↪ 4OFWMGCQX6VS65NNWQAVRVPL3M5O2HN3BPUUXTE2CR3SYZLIFVCH3K5TM4
   ↪ 2000000000000000 microAlgos
38 [offline]       Unnamed-0
   ↪ 6BW3G56B6PNWB2CZVXQNKYTDW5R5SLVEDMVEM7I2FBIZ2XRTEH7EACLAOY      0
   ↪ microAlgos    *Default
39 root@ea05aa8270b7:~# goal account list --wallet w1
40 [offline]       Unnamed-1
   ↪ RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM      0
   ↪ microAlgos
41 root@ea05aa8270b7:~# goal clerk send --from
   ↪ NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM --to
   ↪ RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM --amount
   ↪ 1000000
42 Sent 1000000 MicroAlgos from account
   ↪ NMA5HTMA53EGFBS5523NGEZ3TGSGWLB3VGYMP4YH22VGFEPHL7HUNIOZQM to address
   ↪ RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM,
   ↪ transaction ID: IKASGOOYTBMHIXOL2GTWG5QWUOAXUHOFPTZRRBC43FANR3M3DZTQ.
   ↪ Fee set to 1000
43 Transaction IKASGOOYTBMHIXOL2GTWG5QWUOAXUHOFPTZRRBC43FANR3M3DZTQ
   ↪ committed in round 1
44 root@ea05aa8270b7:~# goal account balance --address
   ↪ RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM
45 1000000 microAlgos
46 root@ea05aa8270b7:~# exit
47 exit
48 @A-Maugli → /workspaces (main) $
```

A példa magyarázata:

- 3. sor: a `goal wallet list` kilistázza a kiindulási pénztárcát (default

wallet)

- 8. sor: a `goal account list` kilistázza a kiindulási pénztárcába felvett számlákat
- 12. sor: új pénztárcát a `goal wallet new` paranccsal tudunk készíteni. A pénztárca neve `w1`
- 24. sor: új számlát a `goal account new` paranccsal tudunk létrehozni. Látszik, hogy a számla a default pénztárcában jött létre.
- 31. sor: új számlát a `w1` pénztárcában a `goal account new --wallet w1` paranccsal tudunk felvenni
- 34. sor: a számlák kilistázása a kiindulási pénztárcából
- 39. sor: a számlák kilistázása a `w1` pénztárcából
- 41. sor: Algót küldeni az egyik számláról a másikra a `goal clerk send` paranccsal tudunk. Meg kell adnunk a forrás számla címét a `--from` után, a cél számla címét a `--to` után, és az elküldeni kívánt összeget, mikro-Algókban: `--amount 1000000`. Itt tehát 1 Algót küldtünk a `NMA5H...IOZQM` számláról a `RTKEC...JGXPM` számlára.
- 44. sor: egy számla egyenlegét a `goal account balance` paranccsal tudjuk megállapítani. Meg kell adnunk a számlához tartozó azonosítót is, a `--address` után.

Egy érdekesség a Bitcoin, Ethereum világából érkezőknek: a `w1` nevű új pénztárca létrehozásakor kiírattuk a "backup phrase"-t, „visszaállító kifejezést". Figyeljük meg, hogy ez 25 szóból áll. A 25 szó egy 256 bites entrópiát kódol, és ellenőrző összeggel védi a visszaállító kifejezést. Ez a 256 bites entrópia megfelel a BIP39-es szabványban szereplő 256 bites entrópiának, de **nem úgy van kódolva**, ahogyan azt a BIP39 szabvány előírja, hanem az Algorand 11 bites tömbformátumának megfelelően. A részleteket lásd a What's the rationale behind the bespoke 25-word mnemonic standard? (Milyen meggondolás van az állítólagos 25-szavas Algorand mnemonik szabvány mögött?) című fórum cikkben.

Természetesen semmi akadálya, hogy az Algorand 25 szavas mnemonikját (jelmondatát) átalakítsuk a BIP39-ben szereplő 24 szavas mnemonikra (jelmondatra), és fordítva. A hierarchikus címképzés ebből az entrópiából történik, lásd pl. a BIP39 weblapot. Itt tehát a BIP39 szabványnak megfelelő 24 szó vagy az Algorandban használt 25 szó nem privát címet kódol, hanem 256 bites entrópiát.

Az esetleges zavart tovább fokozza, hogy lehetőség van az egyes számlákhoz tartozó titkos kulcs kiexportálására is. Ezek is 25 szavas mnemonik formájában íródnak ki. A pénztárcák esetén a 25 szó az entrópiát kódolja, és a segítségével tetszőleges számú számla állítható elő (publikus kulcs, titkos kulcs).

Igazán ínyenceknek azt is elárulhatjuk, hogy a Ledger HW pénztárca esetében viszont úgy valósították meg a HW pénztárcán futó Algorand App-ot, hogy az entrópia kódolására csak 24 szót használnak, és egy egészen más titkosító algoritmust használnak. A algorand.oortnet.com címen található egy helyes kis segédprogram, amely a Ledger HW pénztárca mnemonikja, azaz visszaállító kifejezése alapján képes előállítani az egyes, hierarchikus Algorand számlákhoz tartozó címeket, kulcsokat, és mnemonikokat. Ez azt jelenti, hogy ha nem férünk hozzá a Ledger HW pénztárcánkhoz (mert elromlott, ellopták stb.), akkor is hozzá tudunk férni az általa kódolt Algorand számlákhoz.

Megjegyzés: A BIP39 honlapon az Algorand pénznem nem választható ki, de a `https://github.com/Coinomi/bip39-coinomi/releases` címről letölthető egy olyan standalone (önállóan futtatható) változat, amely az Algorand esetén is működőképes.

4. Néhány példa a Python SDK használatára

A Python SDK használatát Ryan Fox "Algorand Bootcamp for Beginners" előadásain használt példákkal mutatom be. A Python mintapéldák megtalálhatók a `https://github.com/A-Maugli/akt02` repository-ban, a hellow/playground/python_api könyvtárban.

4.1. Előkészületi lépések

Indítsuk el a `codespaces` munkaterületünket. Mikor a munkaterület betöltődött, indítsuk el a privát blokkláncot:

```
@A-Maugli → /workspaces/akt02 (main) $ algokit --version
algokit, version 1.13.0
@A-Maugli → /workspaces/akt02 (main) $ algokit localnet start
algod has a new version available, run `algokit localnet reset --update`
↪  to get the latest version
Starting AlgoKit LocalNet now...
docker: Container algokit_sandbox_algod  Creating
docker: Container algokit_sandbox_postgres  Creating
docker: Container algokit_sandbox_algod  Created
docker: Container algokit_sandbox_postgres  Created
...
docker: Container algokit_sandbox_indexer  Healthy
Started; execute `algokit explore` to explore LocalNet in a web user
↪  interface.
```

A projekt eredetileg az `algokit init` paranccsal lett inicializálva. A projekthez tartozó függőségeket (Python könyvtárak, node.js modulok) az `algokit bootstrap all` paranccsal tudjuk betölteni:

```
@A-Maugli → /workspaces/akt02 (main) $ algokit bootstrap all
Poetry not found; attempting to install it...
? We couldn't find `poetry`; can we install it for you via pipx so we can
↪  install Python dependencies? Yes
Installing Python dependencies and setting up Python virtual environment
↪  via Poetry
poetry: Creating virtualenv playground in /workspaces/akt02/hellow/.venv
```

```
poetry: Installing dependencies from lock file
poetry:
poetry: Package operations: 25 installs, 0 updates, 0 removals
poetry:
poetry: - Installing exceptiongroup (1.2.0)
poetry: - Installing idna (3.6)
...
poetry: - Installing py-algorand-sdk (2.5.0)
poetry: - Installing semantic-version (2.10.0)
poetry: - Installing tabulate (0.9.0)
poetry: - Installing algokit-utils (2.2.1)
poetry: - Installing pyteal (0.24.1)
poetry: - Installing beaker-pyteal (1.1.1)
poetry:
poetry: Installing the current project: playground (0.1.0)
Finished bootstrapping /workspaces/akt02
```

A Ports fülön tegyük globálissá a portokat. Ehhez a lakat szimbólumra kell kattintanunk az egérrel. A lakat ennek hatására kinyílik.

A Python fejlesztés egy virtuális környezetben történik. A virtuális környezetet a Poetry biztosítja. Aktiváljuk a virtuális környezetet:

```
source .venv/bin/activate
```

Készítsünk a playground könyvtáron belül egy python_api könyvtárat a grafikus felület segítségével, majd menjünk ebbe a könyvtárba.

```
@A-Maugli → /workspaces/akt02 (main) $ cd hellow
@A-Maugli → /workspaces/akt02/hellow (main) $ source .venv/bin/activate
(playground-py3.10) @A-Maugli → /workspaces/akt02/hellow (main) $ cd
↪  playground
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground (main) $
↪  mkdir python_api
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground (main) $ cd
↪  python_api
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
↪  (main) $
```

4.2. Algorand számla létrehozása

A fájl explorer-rel vagy a terminálban a `nano 01-account_generation.py` segítségével hozzuk létre a következő fájlt:

```
from algosdk import account, mnemonic

def generate_algorand_keypair():
    private_key, address = account.generate_account()
    print("My address: {}".format(address))
    print("My private key: {}".format(private_key))
    print("My passphrase:
    ↪  {}".format(mnemonic.from_private_key(private_key)))

generate_algorand_keypair()
```

Futtassuk a `01-make_account.py` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
↪  (main) $ python 01-account_generation.py
My address: 3API2XIBSH7IKMSZ3SNRGOINISSNXVAYT2OV6BLL4GBZRAVWTMYAWCXMPI
My private key: y1mWx1K1ZMGbHojwGOSWKtMTs0XuCIKOATJ9PFc2AyrYHo1dAZH+hTJZ3
↪  JsTOQ1EpNvUGJ6dXwVr4YOYgrabMA==
My passphrase: defense floor glow festival siren utility visit marine
↪  lawn away enroll crawl chicken holiday impulse angry space alert
↪  october sick purpose snow exotic ability rather
```

Gratulálok! Sikerült egy Python programból egy számlaszámot és a hozzá tartozó privát kulcsot létrehozni, valamint a privát (titkos) kulcsot 25 szavas mnemonikus formában kiírni!

4.3. Számla egyenleg kiírása

A fájl explorer-rel vagy a terminálban a `nano 02-account_balance.py` segítségével hozzuk létre a következő fájlt:

```
from algosdk import kmd
from algosdk.wallet import Wallet
from algosdk.v2client import algod
```

```
import json

# define sandbox values for kmd client
kmd_address = "http://localhost:4002"
kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

# define sandbox values for algod client
algod_address = "http://localhost:4001"
algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

def main() :
    # create KMDClient
    kmd_client = kmd.KMDClient(kmd_token, kmd_address)

    # connect to default wallet
    wallet = Wallet("unencrypted-default-wallet", "", kmd_client)

    # gather the three default accounts
    wallet_addresses = wallet.list_keys()
    addr1 = wallet_addresses[0]
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # check account balance
    account_info = algod_client.account_info(addr1)
    print("{} balance: {} microAlgos".format(account_info.get('address'),
    ↪  account_info.get('amount')) +
    ↪  "\n")
    account_info = algod_client.account_info(addr2)
    print("{} balance: {} microAlgos".format(account_info.get('address'),
    ↪  account_info.get('amount')) +
    ↪  "\n")
    account_info = algod_client.account_info(addr3)
    print("{} balance: {} microAlgos".format(account_info.get('address'),
    ↪  account_info.get('amount')) +
    ↪  "\n")
```

```
37
38 main()
```

Futtassuk le a `02-account_balance.py` fájlt:

```
1  (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
   ↪  (main) $ python 02-account_balance.py
2  DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE balance:
   ↪  4000000000000000 microAlgos
3
4  DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ balance:
   ↪  2000000000000000 microAlgos
5
6  ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74 balance:
   ↪  4000000000000000 microAlgos
```

Látható, hogy a pénztárca kezdetben három számlaszámot tartalmaz, és ezekben összesen 10^{16} mikro-Algó van, azaz 10 milliárd Algó. (10 milliárd Algó = 10 000 millió Algo = 10^{10} Algo)

4.4. Fizetési tranzakció kiadása

A fájl explorer-rel vagy a terminálban a `nano 03-payment_transaction.py` segítségével hozzuk létre a következő fájlt:

```
1  from algosdk import transaction
2  import json
3  import base64
4
5  from algosdk import kmd
6  from algosdk.wallet import Wallet
7  from algosdk.v2client import algod
8
9  # define sandbox values for kmd client
10 kmd_address = "http://localhost:4002"
11 kmd_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
12
13 # define sandbox values for algod client
```

```
algod_address = "http://localhost:4001"
algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

def main() :
    # create KMDClient
    kmd_client = kmd.KMDClient(kmd_token, kmd_address)

    # connect to default wallet
    wallet = Wallet("unencrypted-default-wallet", "", kmd_client)

    # gather the three default accounts
    wallet_addresses = wallet.list_keys()
    addr1 = wallet_addresses[0]
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # build unsigned transaction
    params = algod_client.suggested_params()
    receiver = addr2
    note = "Hello World".encode()
    amount = 1000000
    unsigned_txn = transaction.PaymentTxn(addr1, params, receiver,
    ↪  amount, None, note)

    # sign transaction
    signed_txn = unsigned_txn.sign(wallet.export_key(addr1))

    #submit transaction
    txid = algod_client.send_transaction(signed_txn)
    print("Successfully sent transaction with txID: {}".format(txid))

    # wait for confirmation
    try:
        confirmed_txn = transaction.wait_for_confirmation(algod_client,
        ↪  txid, 4)
    except Exception as err:
        print(err)
```

```
        return

    print("Transaction information: {}".format(
        json.dumps(confirmed_txn, indent=4)))
    print("Decoded note: {}".format(base64.b64decode(
        confirmed_txn["txn"]["txn"]["note"]).decode()))

main()
```

Futtassuk le a `03-payment_transaction.py` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
↪  (main) $ python 03-payment_transaction.py
Successfully sent transaction with txID:
↪  2ICYN4NT6O52XTN3YNKM5RCDBHSMEIA5PK7LA66TJN3M3KZF4HSQ
Transaction information: {
    "confirmed-round": 1,
    "pool-error": "",
    "txn": {
        "sig": "ZDb6f6E3OkybPW6KBF7gaosEQBpAZCXIMLgeYSG23Bbg/YQnUJz8ZsFz8
        ↪  /R7nTsYf1im1H5O9umuKjUxocOkAA==",
        "txn": {
            "amt": 1000000,
            "fee": 1000,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "lv": 1000,
            "note": "SGVsbG8gV29ybGQ=",
            "rcv": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF6
            ↪  2WRUQ",
            "snd": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJR
            ↪  ZYYOE",
            "type": "pay"
        }
    }
}
Decoded note: Hello World
```

Látható, hogy sikerült a tranzakció, a tranzakció azonosító `2ICYN...F4HSQ`. Kiírattuk magát a tranzakciós rekordot is. (txn: tranzakció, amt: amount,

fee, fv: first block value, gen: genezis, gh: genezis hash, lv: last block value, note, rcv: receiver, snd: sender, pay: payment txn)

A `note` mezőt dekódolt formában is kiírattuk.

4.5. Számlaegyenlegek kiírása

A fájl explorer-rel vagy a terminálban a `nano 04-account_info.py` segítségével hozzuk létre a következő fájlt:

```
from algosdk import kmd
from algosdk.wallet import Wallet
from algosdk.v2client import algod
import json

# define sandbox values for kmd client
kmd_address = "http://localhost:4002"
kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

# define sandbox values for algod client
algod_address = "http://localhost:4001"
algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

def main() :
    # create KMDClient
    kmd_client = kmd.KMDClient(kmd_token, kmd_address)

    # connect to default wallet
    wallet = Wallet("unencrypted-default-wallet", "", kmd_client)

    # gather the three default accounts
    wallet_addresses = wallet.list_keys()
    addr1 = wallet_addresses[0]
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)
```

```
29
30      # check account details
31      account_info = algod_client.account_info(addr3)
32      print("Account information: {}".format(
33          json.dumps(account_info, indent=4)))
34      account_info = algod_client.account_info(addr2)
35      print("Account information: {}".format(
36          json.dumps(account_info, indent=4)))
37      account_info = algod_client.account_info(addr1)
38      print("Account information: {}".format(
39          json.dumps(account_info, indent=4)))
40  main()
```

Futtassuk le a `04-account_info.py` fájlt:

```
1   (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
    ↪  (main) $ python 04-account_info.py
2   Account information: {
3       "address":
        ↪  "ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74",
4       "amount": 4000000000000000,
5       "amount-without-pending-rewards": 4000000000000000,
6       "apps-local-state": [],
7       "apps-total-schema": {
8           "num-byte-slice": 0,
9           "num-uint": 0
10      },
11      "assets": [],
12      "created-apps": [],
13      "created-assets": [],
14      "min-balance": 100000,
15      "participation": {
16          "selection-participation-key":
            ↪  "2m3w5viSs8ZXPhW0+Mns+z8oX3dkJEH3M+DbJQWtN/U=",
17          "state-proof-key": "e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFA
            ↪  IcaO7snXb68TdI5b+Fz2hGNl8hE8qbAQSsNhopp/Q==",
18          "vote-first-valid": 0,
19          "vote-key-dilution": 10000,
20          "vote-last-valid": 30000,
21          "vote-participation-key":
            ↪  "ERZwnIHHihb37ERB93xwo6ejSbA3TyAtggzegnrylS8="
```

```
    },
    "pending-rewards": 0,
    "reward-base": 0,
    "rewards": 0,
    "round": 1,
    "status": "Online",
    "total-apps-opted-in": 0,
    "total-assets-opted-in": 0,
    "total-created-apps": 0,
    "total-created-assets": 0
}
Account information: {
    "address":
    ↪  "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
    "amount": 2000000001000000,
    "amount-without-pending-rewards": 2000000001000000,
    "apps-local-state": [],
    "apps-total-schema": {
        "num-byte-slice": 0,
        "num-uint": 0
    },
    "assets": [],
    "created-apps": [],
    "created-assets": [],
    "min-balance": 100000,
    "participation": {
        "selection-participation-key":
        ↪  "JB7ZLgYDgt54LEZ4wpWEpKZOswglTFkLImXprW0yi3A=",
        "state-proof-key": "/Kp6qQ9X2DBrBTlQOBhCzmuDqmdEap/HDPAd9dxI8YUR5
        ↪  +HYgyQvn8456ImNm7vMMT28XgBSY4em3Hl4bOIi4A==",
        "vote-first-valid": 0,
        "vote-key-dilution": 10000,
        "vote-last-valid": 30000,
        "vote-participation-key":
        ↪  "k1S3OrIKTr8D9CoB154NuK0hbikQSYspUPNdGEIWcMY="
    },
    "pending-rewards": 0,
    "reward-base": 0,
    "rewards": 0,
    "round": 1,
    "status": "Online",
```

```
    "total-apps-opted-in": 0,
    "total-assets-opted-in": 0,
    "total-created-apps": 0,
    "total-created-assets": 0
}
Account information: {
    "address":
    ↪  "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
    "amount": 3999999998999000,
    "amount-without-pending-rewards": 3999999998999000,
    "apps-local-state": [],
    "apps-total-schema": {
        "num-byte-slice": 0,
        "num-uint": 0
    },
    "assets": [],
    "created-apps": [],
    "created-assets": [],
    "min-balance": 100000,
    "participation": {
        "selection-participation-key":
        ↪  "tSvVZrUkGrQbS4YTJ7//K/ew56tJGd9rODmXFSKO1To=",
        "state-proof-key": "uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev
        ↪  6O1wo5U9cPx6t+tBK7iwmNgL8OIaMHP3eUkisdIpQ==",
        "vote-first-valid": 0,
        "vote-key-dilution": 10000,
        "vote-last-valid": 30000,
        "vote-participation-key":
        ↪  "EPGO+Y9ojVIgfstV72u8U4sCPLYLgqhys9XV0IzudrM="
    },
    "pending-rewards": 0,
    "reward-base": 0,
    "rewards": 0,
    "round": 1,
    "status": "Online",
    "total-apps-opted-in": 0,
    "total-assets-opted-in": 0,
    "total-created-apps": 0,
    "total-created-assets": 0
}
```

A három számlára vonatkozóan bőséges információt kaptunk. Az "assets", azaz "pénzeszközök" mező még üres. Az ASA, azaz az Algorand Standard Assets a blokkláncon létrehozott, nem Algorand token-eket jelenti. Az ASA segítségével tetszőleges pénzeszközt létre lehet hozni, akár még „Fabatkát" is.

4.6. Pénzeszköz létrehozása

A fájl explorer-rel vagy a terminálban a `nano 05-asset_create.py` segítségével hozzuk létre a következő fájlt:

```
1  from algosdk import kmd, transaction
2  from algosdk.wallet import Wallet
3  from algosdk.v2client import algod
4
5  import json
6  import base64
7
8  # define sandbox values for kmd client
9  kmd_address = "http://localhost:4002"
10 kmd_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
11
12 # define sandbox values for algod client
13 algod_address = "http://localhost:4001"
14 algod_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
15
16 def main() :
17     # create KMDClient
18     kmd_client = kmd.KMDClient(kmd_token, kmd_address)
19
20     # connect to default wallet
21     wallet = Wallet("unencrypted-default-wallet", "", kmd_client)
22
23     # gather the three default accounts and corresponding mnemonic
       ↪  passphrase
24     wallet_addresses = wallet.list_keys()
25     addr1 = wallet_addresses[0]
```

```
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # build unsigned transaction
    params = algod_client.suggested_params()
    unsigned_txn = transaction.AssetConfigTxn(sender=addr1,
            sp=params,
            total=10000,  # Fungible tokens have total issuance greater than 1
            decimals=2,    # Fungible tokens typically have decimals greater than 0
            default_frozen=False,
            unit_name="FUNTOK",
            asset_name="Fun Token",
            manager=addr1,
            strict_empty_address_check=False,
            reserve="",
            freeze="",
            clawback="",
            url="https://path/to/my/fungible/asset/metadata.json",
            metadata_hash="", # Typically include hash of metadata.json (bytes)
    )

    # sign transaction
    signed_txn = unsigned_txn.sign(wallet.export_key(addr1))

    #submit transaction
    txid = algod_client.send_transaction(signed_txn)
    print("Successfully sent transaction with txID: {}".format(txid))

    # wait for confirmation
    try:
        confirmed_txn = transaction.wait_for_confirmation(algod_client, txid, 4)
    except Exception as err:
        print(err)
        return
```

```
63
64      print("Transaction information: {}".format(
65          json.dumps(confirmed_txn, indent=4)))
66
67      # write the asset index to an environment file
68      f = open('asset.index', 'w+')
69      f.write(f'{confirmed_txn["asset-index"]}')
70      f.close()
71
72  main()
```

Az új pénzeszköz neve a "Fun Token", („Vicces zseton"), az elemi egység neve FUNTOK, összesen 10000 db elemi egységet hozunk kezdetben létre, és 100 elemi egység felel meg egy egységnek, a decimals = 2 beállítás miatt, vagyis ténylegesen 100,00 FUNTOK zsetont hozunk létre.

Futtassuk le a `05-asset_create.py` fájlt:

```
1   Transaction information: {
2       "asset-index": 1002,
3       "confirmed-round": 2,
4       "pool-error": "",
5       "txn": {
6           "sig": "9KHcv6feb9b1fkX6Wg6oRvlqj1/ttyaDjX95XOQWmv2ITe/c6zt3KHFPM⌋
            ↪  BgbNZ9BMaoNkFIYOyrOa6XF5ne+Aw==",
7           "txn": {
8               "apar": {
9                   "an": "Fun Token",
10                  "au": "https://path/to/my/fungible/asset/metadata.json",
11                  "dc": 2,
12                  "m": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFF⌋
                    ↪  JRZYYOE",
13                  "t": 10000,
14                  "un": "FUNTOK"
15              },
16              "fee": 1000,
17              "fv": 1,
18              "gen": "dockernet-v1",
19              "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
20              "lv": 1001,
```

```
            "snd": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJR
            ↪  ZYYOE",
            "type": "acfg"
        }
    }
}
```

Létrejött az új “Fun Token” asset. Látható, hogy a létrehozott pénzeszköz indexe, az “asset-index” értéke 1002. Az “apar” a paramétereket tartalmazza: an: asset name, au: asset url, dc: decimals, m: manager address, t: total units, un: unit name. A tranzakció típusa a 22. sorban látható: `"type": "acfg"`, ahol az `acfg` jelentése `asset configuration`.

4.7. Pénzeszközbe történő „benevezés”, opt-in

Az Algorandban mindaddig nem lehet egy számlára ASA-t küldeni, amíg az adott számla ki nem fejezi a hajlandóságát a pénzeszköz fogadására. Ezt a hajlandóságot angolul opt-in -nek hívjuk, ami azt jelenti, hogy „benevezünk” valamire, itt nevezetesen az adott ASA-ra.

Az opt-in úgy történik, hogy az a számla, amelyik szeretne „benevezni” az adott ASA-ra, 0 egységet küld önmagának az az adott ASÁ-ból. Ezt a mechanizmust az AssetOptInTxn függvénynév tökéletesen elfedi.

Megjegyzendő, hogy minden egyes opt-in 0.1 Algóval növeli a számlán lévő megkövetelt minimális Algó egyenleget. Opt-in-ek nélkül a minimális egyenleg 0.1 Algó, és ezt minden egyes ASA vagy NFT benevezés megemeli 0.1 Algóval.

A fájl explorer-rel vagy a terminálban a `nano 06-asset_opt_in.py` segítségével hozzuk létre a következő fájlt:

```
from algosdk import kmd, transaction
from algosdk.wallet import Wallet
from algosdk.v2client import algod

import json
```

```
import base64

# define sandbox values for kmd client
kmd_address = "http://localhost:4002"
kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

# define sandbox values for algod client
algod_address = "http://localhost:4001"
algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

def get_asset_index(default_index = 1010):
    # try to read the asset index from our environment file
    try:
        index = int(open('asset.index', 'r').readline())
    # otherwise return the default index
    except:
        index = default_index
    return index

def main() :
    # create KMDClient
    kmd_client = kmd.KMDClient(kmd_token, kmd_address)

    # connect to default wallet
    wallet = Wallet("unencrypted-default-wallet", "", kmd_client)

    # gather the three default accounts and corresponding mnemonic
    ↪  passphrase
    wallet_addresses = wallet.list_keys()
    addr1 = wallet_addresses[0]
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # build unsigned transaction
    params = algod_client.suggested_params()
    sender = addr2
```

```
    index = get_asset_index(default_index = 2) # ensure this matches the
    ↪  asset-index returned by asset_create.py
    unsigned_txn = transaction.AssetOptInTxn(sender, params, index)

    # sign transaction
    signed_txn = unsigned_txn.sign(wallet.export_key(addr2))

    # submit transaction
    txid = algod_client.send_transaction(signed_txn)
    print("Successfully sent transaction with txID: {}".format(txid))

    # wait for confirmation
    try:
        confirmed_txn = transaction.wait_for_confirmation(algod_client,
        ↪  txid, 4)
    except Exception as err:
        print(err)
        return

    print("Transaction information: {}".format(
        json.dumps(confirmed_txn, indent=4)))

main()
```

Futtassuk le a `06-asset_opt_in.py` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
↪  (main) $ python 06-asset_opt_in.py
Successfully sent transaction with txID:
↪  7DCSMMAVZNVHZO2AZMON5GKIEE4LJ4PDGWGAQFQWX6Y76POGWTCA
Transaction information: {
    "confirmed-round": 3,
    "pool-error": "",
    "txn": {
        "sig": "DtNu3ZMV1jd+3EQHUeSbVa3oqP9FCJavz5kk5TsBomICJZ4OZ/KYubPT4
        ↪  0B186gasqHXnRoK3mkeex4s+L7PAw==",
        "txn": {
            "arcv": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF
            ↪  62WRUQ",
            "fee": 1000,
```

```
            "fv": 2,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "lv": 1002,
            "snd": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF6
            ↪  2WRUQ",
            "type": "axfer",
            "xaid": 1002
        }
    }
}
```

A tranzakció típus: “axfer”, “asset transfer”, és a “xaid” a “transfer asset id”, és az “arcv”, az “address receiver” az a cím, amelyik az ASA-ra benevezett.

4.8. Csere atomi tranzakciós csoporttal

Az Algorandban az ASA sok mindent képviselhet. Tegyük fel, hogy példánkban egy másik pénzeszközt jelent. Ahhoz, hogy az ASA-t el tudjuk adni Algorand-ért, biztosítani kell, hogy két tranzakció menjen egyszerre végbe, vagy ne menjen végbe egy sem:

- a vevő Algorand-dal fizet az eladónak az ASA-ért
- az eladó elküldi az ASA-t a vevőnek

Az Algorandban a „tranzakció csoport” fogalma biztosítja a fenti cserét.

A fájl explorer-rel vagy a terminálban a `nano 07-atomic_transaction.py` segítségével hozzuk létre a következő fájlt:

```
from algosdk import transaction
import json
import base64

from algosdk import kmd
from algosdk.wallet import Wallet
from algosdk.v2client import algod

# define sandbox values for kmd client
```

```
kmd_address = "http://localhost:4002"
kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

# define sandbox values for algod client
algod_address = "http://localhost:4001"
algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

def get_asset_index(default_index = 1010):
    # try to read the asset index from our environment file
    try:
        index = int(open('asset.index', 'r').readline())
    # otherwise return the default index
    except:
        index = default_index
    return index

def main() :
    # create KMDClient
    kmd_client = kmd.KMDClient(kmd_token, kmd_address)

    # connect to default wallet
    wallet = Wallet("unencrypted-default-wallet", "", kmd_client)

    # gather the three default accounts and corresponding mnemonic
    ↪  passphrase
    wallet_addresses = wallet.list_keys()
    addr1 = wallet_addresses[0]
    addr2 = wallet_addresses[1]
    addr3 = wallet_addresses[2]

    # create algod client
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # build unsigned payment transaction
    params = algod_client.suggested_params()
    sender = addr2
    receiver = addr1
    amount = 1000000
    txn_1 = transaction.PaymentTxn(sender, params, receiver, amount)
```

```
48
49      # build unsigned asset transfer transaction
50      sender = addr1
51      receiver = addr2
52      amount = 100 # remember this ASA has 2 decimal places, so this is
        ↪  1.00 FUNTOK
53      index = get_asset_index(default_index = 1010) # ensure this matches
        ↪  the asset-index returned by asset_create.py
54      txn_2 = transaction.AssetTransferTxn(sender, params, receiver,
        ↪  amount, index)
55
56      # group transactions
57      gid = transaction.calculate_group_id([txn_1, txn_2])
58      txn_1.group = gid
59      txn_2.group = gid
60
61      # sign transaction
62      stxn_1 = txn_1.sign(wallet.export_key(addr2))
63      stxn_2 = txn_2.sign(wallet.export_key(addr1))
64
65      # assemble transaction group
66      signed_group = [stxn_1, stxn_2]
67
68      #submit atomic transaction group
69      txid = algod_client.send_transactions(signed_group)
70      print("Successfully sent transaction with txID: {}".format(txid))
71
72      # wait for confirmation
73      try:
74          confirmed_txn = transaction.wait_for_confirmation(algod_client,
            ↪  txid, 4)
75      except Exception as err:
76          print(err)
77          return
78
79      print("Transaction information: {}".format(
80          json.dumps(confirmed_txn, indent=4)))
81
82  main()
```

Futtassuk le a `07-atomic_transaction.py` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api (main) $ python 07-atomic_transaction.py
Successfully sent transaction with txID: 6AW7M7GDZKS2YA575LEOSRD3NKR7LF7RWTGY7IEWSL56B67MPY4A
Transaction information: {
    "confirmed-round": 4,
    "pool-error": "",
    "txn": {
        "sig": "WJz54qOqdzBtHHN417rmeKTxfqlBPkAllg+XytRtzrV42VAUswRcemomUEbV3vKm4IaZIDEVJr4sIQifM3XBCQ==",
        "txn": {
            "amt": 1000000,
            "fee": 1000,
            "fv": 3,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "grp": "SBs5jPxhxYFk7y/WhsILxTXn7QlBByaETrD5xTHECPI=",
            "lv": 1003,
            "rcv": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
            "snd": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
            "type": "pay"
        }
    }
}
```

Az tranzakció egyik felét írattuk ki, a “pay” “payment” txn-t, azaz a fizetési tranzakciót. De van egy másik tranzakció is a tranzakciós csoportban: az “axtr”, azaz az “asset transfer” tranzakció.

Az `04-account_info.py` jön a segítségünkre:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api (main) $ python 04-account_info.py
Account information: {
    "address": "ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74",
```

```
    "amount": 4000000000000000,
    "amount-without-pending-rewards": 4000000000000000,
    "apps-local-state": [],
    "apps-total-schema": {
        "num-byte-slice": 0,
        "num-uint": 0
    },
    "assets": [],
    "created-apps": [],
    "created-assets": [],
    "min-balance": 100000,
    "participation": {
        "selection-participation-key":
        ↪  "2m3w5viSs8ZXPhW0+Mns+z8oX3dkJEH3M+DbJQWtN/U=",
        "state-proof-key": "e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFA
        ↪  IcaO7snXb68TdI5b+Fz2hGNl8hE8qbAQSsNhopp/Q==",
        "vote-first-valid": 0,
        "vote-key-dilution": 10000,
        "vote-last-valid": 30000,
        "vote-participation-key":
        ↪  "ERZwnIHHihb37ERB93xwo6ejSbA3TyAtggzegnrylS8="
    },
    "pending-rewards": 0,
    "reward-base": 0,
    "rewards": 0,
    "round": 4,
    "status": "Online",
    "total-apps-opted-in": 0,
    "total-assets-opted-in": 0,
    "total-created-apps": 0,
    "total-created-assets": 0
}
Account information: {
    "address":
    ↪  "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
    "amount": 1999999999998000,
    "amount-without-pending-rewards": 1999999999998000,
    "apps-local-state": [],
    "apps-total-schema": {
        "num-byte-slice": 0,
        "num-uint": 0
```

```
    },
    "assets": [
        {
            "amount": 100,
            "asset-id": 1002,
            "is-frozen": false
        }
    ],
    "created-apps": [],
    "created-assets": [],
    "min-balance": 200000,
    "participation": {
        "selection-participation-key":
        ↪  "JB7ZLgYDgt54LEZ4wpWEpKZOswglTFkLImXprW0yi3A=",
        "state-proof-key": "/Kp6qQ9X2DBrBTlQOBhCzmuDqmdEap/HDPAd9dxI8YUR5
        ↪  +HYgyQvn8456ImNm7vMMT28XgBSY4em3Hl4bOIi4A==",
        "vote-first-valid": 0,
        "vote-key-dilution": 10000,
        "vote-last-valid": 30000,
        "vote-participation-key":
        ↪  "k1S3OrIKTr8D9CoB154NuKOhbikQSYspUPNdGEIWcMY="
    },
    "pending-rewards": 0,
    "reward-base": 0,
    "rewards": 0,
    "round": 4,
    "status": "Online",
    "total-apps-opted-in": 0,
    "total-assets-opted-in": 1,
    "total-created-apps": 0,
    "total-created-assets": 0
}
Account information: {
    "address":
    ↪  "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
    "amount": 3999999999997000,
    "amount-without-pending-rewards": 3999999999997000,
    "apps-local-state": [],
    "apps-total-schema": {
        "num-byte-slice": 0,
        "num-uint": 0
```

```
    },
    "assets": [
        {
            "amount": 9900,
            "asset-id": 1002,
            "is-frozen": false
        }
    ],
    "created-apps": [],
    "created-assets": [
        {
            "index": 1002,
            "params": {
                "creator": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAO
                ↪  KQBWFFJRZYYOE",
                "decimals": 2,
                "default-frozen": false,
                "manager": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAO
                ↪  KQBWFFJRZYYOE",
                "name": "Fun Token",
                "name-b64": "RnVuIFRva2Vu",
                "total": 10000,
                "unit-name": "FUNTOK",
                "unit-name-b64": "RlVOVE9L",
                "url": "https://path/to/my/fungible/asset/metadata.json",
                "url-b64": "aHR0cHM6Ly9wYXRoL3RvL215L2Z1bmdpYmxlL2Fzc2V0L
                ↪  21ldGFkYXRhLmpzb24="
            }
        }
    ],
    "min-balance": 200000,
    "participation": {
        "selection-participation-key":
        ↪  "tSvVZrUkGrQbS4YTJ7//K/ew56tJGd9rODmXFSKO1To=",
        "state-proof-key": "uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev
        ↪  6O1wo5U9cPx6t+tBK7iwmNgL8OIaMHP3eUkisdIpQ==",
        "vote-first-valid": 0,
        "vote-key-dilution": 10000,
        "vote-last-valid": 30000,
        "vote-participation-key":
        ↪  "EPGO+Y9ojVIgfstV72u8U4sCPLYLgqhys9XV0IzudrM="
```

```
113     },
114     "pending-rewards": 0,
115     "reward-base": 0,
116     "rewards": 0,
117     "round": 4,
118     "status": "Online",
119     "total-apps-opted-in": 0,
120     "total-assets-opted-in": 1,
121     "total-created-apps": 0,
122     "total-created-assets": 1
123 }
124 (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api
    ↪  (main) $
```

A példa 42..48. sorában látszik, hogy a `DUWWU...2WRUQ` címre az 1002-es "asset-id"-ból 1,00 FUNTOK megérkezett, ami a 2 tizedesjegy miatt 100-nak látszik. Ugyanakkor a 79..85. sorban az is látszik, hogy az ASÁ-t kibocsájtó `DPO7A...ZYYOE"` számla egyenlege 100-zal csökkent, és már csak 9900 egység van a "FUNTOK"-ból, ami a 2 tizedes miatt 99,00-nek felel meg.

5. Fejlesztés a Javascript SDK-val

Az Algorand Javascript SDK segítségével többféle környezetben is lehetséges a fejlesztés:

- Node.js környezetben, a konzol használatával
- Node.js környezetben, Web böngésző használatával
- közvetlenül a Web böngészőben, „összepakolt" JS könyvtár(ak) használatával

A következő részben az előzőleg már látott példákat mutatjuk be újra, ezúttal az Algorand JavaScript SDK használatával, Node.js környezetben, a konzol használatával. A JavaScript mintapéldák megtalálhatók a `https://github.com/A-Maugli/akt02` repository-ban, a hellow/playground/js_api könyvtárban.

5.1. Előkészületi lépések

Indítsuk el az utoljára használt Codespaces munkaterületünket, vagy Windows alatt a `VS Code`-ot. Mikor a munkaterület betöltődött, indítsuk el a privát blokkláncot:

```
@A-Maugli → /workspaces/akt02 (main) $ algokit --version
algokit, version 1.13.0
@A-Maugli → /workspaces/akt02 (main) $ algokit localnet stop
Stopping AlgoKit LocalNet now...
docker: Container algokit_sandbox_indexer  Stopping
docker: Container algokit_sandbox_indexer  Stopped
docker: Container algokit_sandbox_conduit  Stopping
docker: Container algokit_sandbox_conduit  Stopped
LocalNet Stopped; execute `algokit localnet start` to start it again.
@A-Maugli → /workspaces/akt02 (main) $ algokit localnet start
```

A Ports fülön tegyük globálissá a portokat. Ehhez a lakat szimbólumra kell kattintanunk az egérrel. A lakat ennek hatására kinyílik.

Hozzunk létre egy könyvtárat a `/workspace/akt02/hellow/playground` alatt, `js_api` névvel. Lépjünk be ebbe a könyvtárba:

```
cd /workspace/akt02/hellow/playground/js_api
```

Hozzunk létre egy `package.json` csomagot:

```
npm init
```

```
npm install algosdk
```

Ennek hatására a csomag függőségként a `packages.json` fájlba is bekerül.

5.2. Algorand számlaszám létrehozása

A fájl explorerrel vagy egy terminálban a nano szövegszerkesztővel hozzuk létre a következő fájlt: `nano 01-account_generation.js`

```
const algosdk = require('algosdk');
const DEBUG = 0;

const createAccount = function() {
    try {
        const myAccount = algosdk.generateAccount();
        console.log("Account address:", myAccount.addr);
        if (DEBUG) console.log("Private key:", myAccount.sk);
        let b64_sk = Buffer.from(myAccount.sk).toString('base64');
        console.log("Private key:", b64_sk);
        const accountMnemonic = algosdk.secretKeyToMnemonic(myAccount.sk);
        console.log("Account mnemonic:", accountMnemonic);
        return myAccount;
    }
    catch (err) {
        console.log("err:", err);
    }
}

createAccount();
```

Installáljuk a függőségeket az `algokit bootstrap all` paranccsal:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ algokit bootstrap all
Installing npm dependencies
npm:
npm: added 13 packages, and audited 14 packages in 1s
npm:
npm: 3 packages are looking for funding
npm: run `npm fund` for details
npm:
npm: found 0 vulnerabilities
```

Futtassuk a `01-account_generation.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 01-account_generation.js
Account address:
↪  MLLDAU2SRDZBTT5KCMNPK6B4O6N6ZYSUEX32AEPQH7Q25EWCK7MJCVK3BY
Private key: WGOJVOOy9vZR33nzIzAfpNhJNEdSxUoae/nzgr8T8A9i1jBTUojyGc+qExr1
↪  eDx3m+ziVCX3oBHwP+GuksJX2A==
Account mnemonic: prison certain present comfort uniform laundry whisper
↪  keep lazy scene auto medal neck snack mutual shed enable unaware
↪  witness connect lecture agent legend abandon crumble
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

5.3. Számla egyenleg kiírása

A fájl explorer-rel vagy a terminálban a `nano 02-account_balance.js` segítségével hozzuk létre a következő fájlt:

```
algosdk = require('algosdk');

const DEBUG=0;

// define sandbox values for kmd client
const kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const kmd_server = "http://localhost";
```

```
const kmd_server_port = 4002;

// define sandbox values for algod client
const algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const algod_server = "http://localhost";
const algod_server_port = 4001;

async function main() {
    // create kmd client
    const kmd_client = new algosdk.Kmd(kmd_token, kmd_server,
    ↪  kmd_server_port);

    // list wallets
    wallets = await kmd_client.listWallets();
    if(DEBUG) console.log('wallets:', wallets);

    // get wallet index for default wallet
    wallets.wallets.forEach(item => {
        if (item.name ==='unencrypted-default-wallet') {
            wallet_id = item.id;
        }
    })
    // get wallet_handle for default wallet
    wallet_handle = await kmd_client.initWalletHandle(wallet_id, '');
    if(DEBUG) console.log('wallet_handle:', wallet_handle);

    // get accounts (addresses) from default wallet
    wallet_addresses = await
    ↪  kmd_client.listKeys(wallet_handle.wallet_handle_token);
    if(DEBUG) console.log('wallet_addresses:', wallet_addresses);

    // create algod client
    const algod_client = new algosdk.Algodv2(algod_token, algod_server,
    ↪  algod_server_port);

    // check account balance for each account
    wallet_addresses.addresses.forEach(async (addr) => {
        if(DEBUG) console.log('addr', addr);
        account_info = await algod_client.accountInformation(addr).do();
        if(DEBUG) console.log('account_info', account_info);
```

```
        console.log('%s balance: %s microAlgos', account_info.address,
        ↪  account_info.amount);

    });

    // release wallet handle
    let hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_h
    ↪  andle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Futtassuk le a `02-account_balance.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 02-account_balance.js
DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE balance:
↪  3999999999997000 microAlgos
DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ balance:
↪  1999999999998000 microAlgos
ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74 balance:
↪  4000000000000000 microAlgos
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

Látható, hogy a pénztárca kezdetben három számlaszámot tartalmaz, és ezekben összesen 10^{16} mikro-Algó van, azaz 10 milliárd Algó. (10 milliárd Algó = 10 000 millió Algo = 10^{10} Algo)

5.4. Fizetési tranzakció kiadása

A fájl explorer-rel vagy a terminálban a `nano 03-payment_transaction.js` segítségével hozzuk létre a következő fájlt:

```
const algosdk = require('algosdk');

const DEBUG=0;

```

```
// define sandbox values for kmd client
const kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const kmd_server = "http://localhost";
const kmd_server_port = 4002;

// define sandbox values for algod client
const algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const algod_server = "http://localhost";
const algod_server_port = 4001;

async function getWalletId(
    kmdClient,
    walletName) {

    // list wallets
    wallets = await kmdClient.listWallets();
    if(DEBUG) console.log('wallets:', wallets);

    // get wallet index for default wallet
    let walletId='';
    wallets.wallets.forEach(item => {
        if (item.name === walletName) {
            walletId = item.id;
        }
    })
    return walletId;
}

async function main() {
    // create kmd client
    kmd_client = new algosdk.Kmd(kmd_token, kmd_server, kmd_server_port);

    // get wallet id for default wallet
    wallet_id = await getWalletId(kmd_client,
    ↪  'unencrypted-default-wallet');
    if(DEBUG) console.log('wallet_id:', wallet_id);

    // get wallet_handle for default wallet
    wallet_handle = await kmd_client.initWalletHandle(wallet_id, '');
```

```
43      if(DEBUG) console.log('wallet_handle:', wallet_handle);
44
45      // get accounts (addresses) from default wallet
46      wallet_addresses = await
        ↪  kmd_client.listKeys(wallet_handle.wallet_handle_token);
47      if(DEBUG) console.log('wallet_addresses:', wallet_addresses);
48      addr1 = wallet_addresses.addresses[0];
49      addr2 = wallet_addresses.addresses[1];
50      addr3 = wallet_addresses.addresses[2];
51
52      // create algod client
53      algod_client = new algosdk.Algodv2(algod_token, algod_server,
        ↪  algod_server_port);
54
55      // build unsigned transaction
56      params = await algod_client.getTransactionParams().do();
57      params['fee'] = 0;
58      if (DEBUG) console.log('params:', params);
59      unsigned_txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
60          from: addr1,
61          to: addr2,
62          amount: algosdk.algosToMicroalgos(0.15),
63          note: algosdk.encodeObj("Hello World"),
64          suggestedParams: params
65      });
66      if (DEBUG) console.log('unsigned_txn:', unsigned_txn);
67
68      // sign transaction
69      addr1_sk = await
        ↪  kmd_client.exportKey(wallet_handle.wallet_handle_token, "",
        ↪  addr1);
70      if (DEBUG) console.log('addr1_sk:', addr1_sk);
71      signed_txn = unsigned_txn.signTxn(addr1_sk.private_key);
72      if (DEBUG) console.log('signed_txn:', signed_txn);
73
74      // submit transaction
75      tx_id = await algod_client.sendRawTransaction(signed_txn).do();
76      console.log("Successfully sent transaction with tx_id: ", tx_id);
77      console.log('tx_id["txId"]:', tx_id['txId']);
78
79      // wait for confirmation
```

```
    console.log('Awaiting confirmation (this may take several
    ↪  seconds)...');
    const roundTimeout = 7;
    const confirmed_txn = await algosdk.waitForConfirmation(
      algod_client,
      tx_id['txId'],
      roundTimeout
    );
    console.log('Transaction is successfully completed');

    // log confirmed transaction
    console.log('confirmed_txn:', confirmed_txn);

    // log decoded note
    let decoded_note =
    ↪  algosdk.decodeObj(confirmed_txn['txn']['txn']['note']);
    console.log('decoded_note:', decoded_note);

    // release wallet handle
    let hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_h
    ↪  andle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Futtassuk le a `03-payment_transaction.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 03-payment_transaction.js
Successfully sent transaction with tx_id: { txId:
↪  'MIPIZFQWBSQJGMSZ5DR4BP5WK5OWWT7KN7YTQAHDM4FM6TCTNUPQ' }
tx_id["txId"]: MIPIZFQWBSQJGMSZ5DR4BP5WK5OWWT7KN7YTQAHDM4FM6TCTNUPQ
Awaiting confirmation (this may take several seconds)...
Transaction is successfully completed
confirmed_txn: {
  'confirmed-round': 5,
  'pool-error': '',
  txn: {
    sig: Uint8Array(64) [
```

```
      135,  55,  63, 198, 207, 182, 235,  66, 129, 201, 133,
       65, 195, 236, 195, 242, 242,  95, 182, 253, 181, 189,
      151, 179, 127, 251,   8,  21,  20,  37,   8,  91, 177,
       93,  49,  14, 226,  71,  67, 126, 128, 238,  73,  29,
       82, 105,  21,  67, 195, 245,  68,  25, 127,  26,  84,
      110,  62, 218, 168,  41,  84, 190,  50,   4
    ],
    txn: {
      amt: 150000,
      fee: 1000,
      fv: 4,
      gen: 'dockernet-v1',
      gh: [Uint8Array],
      lv: 1004,
      note: [Uint8Array],
      rcv: [Uint8Array],
      snd: [Uint8Array],
      type: 'pay'
    }
  }
}
decoded_note: Hello World
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

Látható, hogy sikerült a tranzakció, a tranzakció azonosító `MIPIZ...TNUPQ`. Kiírattuk magát a tranzakciós rekordot is. (txn: tranzakció, amt: amount, fee, fv: first block value, gen: genezis, gh: genezis hash, lv: last block value, note, rcv: receiver, snd: sender, pay: payment txn)

A `note` mezőt dekódolt formában is kiírattuk.

5.5. Számlaegyenlegek kiírása

A fájl explorer-rel vagy a terminálban a `nano 04-account_info.js` segítségével hozzuk létre a következő fájlt:

```
var algosdk = require('algosdk');

```

```
const DEBUG=0;

// define sandbox values for kmd client
const kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const kmd_server = "http://localhost";
const kmd_server_port = 4002;

// define sandbox values for algod client
const algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const algod_server = "http://localhost";
const algod_server_port = 4001;

async function main() {
    // create kmd client
    const kmdClient = new algosdk.Kmd(kmd_token, kmd_server,
    ↪  kmd_server_port);

    // list wallets
    wallets = await kmdClient.listWallets();
    if(DEBUG) console.log('wallets:', wallets);

    // get wallet index for default wallet
    let wallet_id=''
    wallets.wallets.forEach(item => {
        if (item.name === 'unencrypted-default-wallet') {
            wallet_id = item.id;
        }
    })
    // get wallet_handle for default wallet
    wallet_handle = await kmdClient.initWalletHandle(wallet_id, '');
    if(DEBUG) console.log('wallet_handle:', wallet_handle);

    // get accounts (addresses) from default wallet
    wallet_addresses = await
    ↪  kmdClient.listKeys(wallet_handle.wallet_handle_token);
    if(DEBUG) console.log('wallet_addresses:', wallet_addresses);

    // create algod client
```

```
    const algodClient = new algosdk.Algodv2(algod_token, algod_server,
    ↪  algod_server_port);

    // check account balance for each account
    wallet_addresses.addresses.forEach(async (addr) => {
        if(DEBUG) console.log('addr', addr);
        account_info = await algodClient.accountInformation(addr).do();
        console.log('account_info', account_info);
    });

    // release wallet handle
    let hr = await kmdClient.releaseWalletHandle(wallet_handle['wallet_ha
    ↪  ndle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Futtassuk le a `04-account_info.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 04-account_info.js
account_info {
  address: 'DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE',
  amount: 3999999999846000,
  'amount-without-pending-rewards': 3999999999846000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [ { amount: 9900, 'asset-id': 1002, 'is-frozen': false } ],
  'created-apps': [],
  'created-assets': [ { index: 1002, params: [Object] } ],
  'min-balance': 200000,
  participation: {
    'selection-participation-key':
    ↪  'tSvVZrUkGrQbS4YTJ7//K/ew56tJGd9rODmXFSK01To=',
    'state-proof-key': 'uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1w
    ↪  o5U9cPx6t+tBK7iwmNgL8OIaMHP3eUkisdIpQ==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
```

```
    'vote-participation-key':
    ↪  'EPGO+Y9ojVIgfstV72u8U4sCPLYLgqhys9XV0IzudrM='
  },
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 5,
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 1,
  'total-created-apps': 0,
  'total-created-assets': 1
}
account_info {
  address: 'DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ',
  amount: 2000000000148000,
  'amount-without-pending-rewards': 2000000000148000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [ { amount: 100, 'asset-id': 1002, 'is-frozen': false } ],
  'created-apps': [],
  'created-assets': [],
  'min-balance': 200000,
  participation: {
    'selection-participation-key':
    ↪  'JB7ZLgYDgt54LEZ4wpWEpKZOswglTFkLImXprW0yi3A=',
    'state-proof-key': '/Kp6qQ9X2DBrBT1QOBhCzmuDqmdEap/HDPAd9dxI8YUR5+HYg
    ↪  yQvn8456ImNm7vMMT28XgBSY4em3Hl4bOIi4A==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
    'vote-participation-key':
    ↪  'k1S3OrIKTr8D9CoB154NuK0hbikQSYspUPNdGEIWcMY='
  },
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 5,
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 1,
```

```
  'total-created-apps': 0,
  'total-created-assets': 0
}
account_info {
  address: 'ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74',
  amount: 4000000000000000,
  'amount-without-pending-rewards': 4000000000000000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [],
  'created-apps': [],
  'created-assets': [],
  'min-balance': 100000,
  participation: {
    'selection-participation-key':
    ↪  '2m3w5viSs8ZXPhWO+Mns+z8oX3dkJEH3M+DbJQWtN/U=',
    'state-proof-key': 'e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFAIcaO
    ↪  7snXb68TdI5b+Fz2hGNl8hE8qbAQSsNhopp/Q==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
    'vote-participation-key':
    ↪  'ERZwnIHHihb37ERB93xwo6ejSbA3TyAtggzegnrylS8='
  },
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 5,
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 0,
  'total-created-apps': 0,
  'total-created-assets': 0
}
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

A három számlára vonatkozóan bőséges információt kaptunk. A 8. és 36. sorban láthatóak az előzőleg, a `python_api` segítségével létrehozott "assets", azaz "pénzeszközök". Az ASA, azaz az Algorand Standard Assets a blokk-

láncon létrehozott, nem Algorand token-eket jelenti. Az ASA segítségével tetszőleges pénzeszköz létrehozható, akár még a „Fabatka” is.

5.6. Pénzeszköz létrehozása

A fájl explorer-rel vagy a terminálban a `nano 05-asset_create.js` segítségével hozzuk létre a következő fájlt:

```
1  const algosdk = require('algosdk');
2  const fs = require('fs');
3  const DEBUG=0;
4
5  // define sandbox values for kmd client
6  const kmd_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
7  const kmd_server = "http://localhost";
8  const kmd_server_port = 4002;
9
10 // define sandbox values for algod client
11 const algod_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
12 const algod_server = "http://localhost";
13 const algod_server_port = 4001;
14
15 async function getWalletId(
16     kmdClient,
17     walletName) {
18
19     // list wallets
20     wallets = await kmdClient.listWallets();
21     if(DEBUG) console.log('wallets:', wallets);
22
23     // get wallet index for default wallet
24     let walletId='';
25     wallets.wallets.forEach(item => {
26         if (item.name === walletName) {
27             walletId = item.id;
28         }
29     })
30     return walletId;
```

```
}

async function main() {
    // create kmd client
    kmd_client = new algosdk.Kmd(kmd_token, kmd_server, kmd_server_port);

    // connect to default wallet
    wallet_id = await getWalletId(kmd_client,
    ↪  'unencrypted-default-wallet');
    wallet_handle = await kmd_client.initWalletHandle(wallet_id, '');

    // gather the first three accounts from the wallet
    wallet_addresses = await
    ↪  kmd_client.listKeys(wallet_handle.wallet_handle_token);
    addr1 = wallet_addresses.addresses[0];
    addr2 = wallet_addresses.addresses[1];
    addr3 = wallet_addresses.addresses[2];

    // create algod client
    algod_client = new algosdk.Algodv2(algod_token, algod_server,
    ↪  algod_server_port);

    // get params
    params = await algod_client.getTransactionParams().do();
    if (DEBUG) console.log('params:', params);

    // build asset create txn
    unsigned_txn =
    ↪  algosdk.makeAssetCreateTxnWithSuggestedParamsFromObject({
        from: addr1,
        suggestedParams: params,
        total: 10000,   // Fungible token, number of total coins: 10000 /
        ↪  100, because decimals is 2
        decimals: 2,
        defaultFrozen: false,
        unitName: "FUNTOK",
        assetName: "Fun Token",
        manager: addr1,
        reserve: undefined,
        freeze: undefined,
        clawback: undefined,
```

```
        assetURL: "https://path/to/my/fungible/asset/metadata.json",
        assetMetadataHash: undefined
    })
    if (DEBUG) console.log('unsigned_txn:', unsigned_txn);

    // sign transaction
    addr1_sk = await
    ↪  kmd_client.exportKey(wallet_handle.wallet_handle_token, '',
    ↪  addr1);
    if (DEBUG) console.log('addr1_sk:', addr1_sk);
    signed_txn = unsigned_txn.signTxn(addr1_sk.private_key);
    if (DEBUG) console.log('signed_txn:', signed_txn);

    // submit transaction
    tx_id = await algod_client.sendRawTransaction(signed_txn).do();
    console.log("Successfully sent transaction with tx_id: ", tx_id);
    console.log('tx_id["txId"]:', tx_id['txId']);

    // wait for confirmation
    console.log('Awaiting confirmation (this may take several
    ↪  seconds)...');
    const roundTimeout = 7;
    const confirmed_txn = await algosdk.waitForConfirmation(
      algod_client,
      tx_id['txId'],
      roundTimeout
    );
    console.log('Transaction is successfully completed');

    // log confirmed transaction
    console.log('confirmed_txn:', confirmed_txn);

    // write asset id to an environment file
    asset_index = (confirmed_txn['asset-index']).toString();
    fs.writeFile('5_asset_index.txt', asset_index, err => {
        if (err) {
            console.log(err);
        }
        else {
            console.log('File 5_asset_index.txt written successfully')
        }
```

```
    });

    // release wallet handle
    let hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_h
    ↪  andle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Az új pénzeszköz neve a “Fun Token”, („Vicces zseton”), az elemi egység neve FUNTOK, összesen 10000 db elemi egységet hozunk kezdetben létre, és 100 elemi egység felel meg egy egységnek, a decimals = 2 beállítás miatt, vagyis ténylegesen 100,00 FUNTOK zsetont hozunk létre.

Futtassuk le a `05-asset_create.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 05-asset_create.js
Successfully sent transaction with tx_id: { txId:
↪  '6MRNWTGOORZF7KESPQIS2TTWVIGXJSMMMIQJWKZ5UKGDTWCAJCXA' }
tx_id["txId"]: 6MRNWTGOORZF7KESPQIS2TTWVIGXJSMMMIQJWKZ5UKGDTWCAJCXA
Awaiting confirmation (this may take several seconds)...
Transaction is successfully completed
confirmed_txn: {
  'asset-index': 1007,
  'confirmed-round': 6,
  'pool-error': '',
  txn: {
    sig: Uint8Array(64) [
       89, 214, 148, 184, 195, 166,  94,  96,  97, 177,  83,
        5, 197, 148, 243,  68, 131, 166,  73, 135, 164, 255,
      221,  67, 198, 251,   3, 104, 127,  90,   2, 136,  81,
       56,  33, 250, 248, 179, 151, 226, 162, 146, 192,  46,
       32, 105, 237,  69, 133,  79, 139, 118,  75, 207, 170,
       47,   7, 121,  75,  48,  55,  22, 128,   2
    ],
    txn: {
      apar: [Object],
      fee: 1000,
```

```
      fv: 5,
      gen: 'dockernet-v1',
      gh: [Uint8Array],
      lv: 1005,
      snd: [Uint8Array],
      type: 'acfg'
    }
  }
}
File 5_asset_index.txt written successfully
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

Létrejött az új "Fun Token" asset. Látható, hogy a létrehozott pénzeszköz indexe, az "asset-index" értéke 1007. Az "apar" a paramétereket tartalmazza, de a paraméterek ezek a listán nem látszanak, an: asset name, au: asset url, dc: decimals, m: manager address, t: total units, un: unit name. A tranzakció típusa a 27. sorban látható: `type: 'acfg'`, vagyis asset configuration.

5.7. Pénzeszközbe történő „benevezés", opt-in

Az Algorandban mindaddig nem lehet egy számlára ASA-t küldeni, amíg az adott számla ki nem fejezi a hajlandóságát a pénzeszköz fogadására. Ezt a hajlandóságot angolul opt-in -nek hívjuk, ami azt jelenti, hogy „benevezünk" valamire, itt nevezetesen az adott ASA-ra.

Az opt-in úgy történik, hogy az a számlaszám, amely szeretne benevezni az adott ASÁ-ra, 0 egységet küld magának az adott ASA-ból, lásd a 72. sorban lévő függvényt:
`makeAssetTransferTxnWithSuggestedParamsFromObject`.

Az opt-in megemeli a számlán tartandó minimális Algorand egyenleget. Kezdetben ez a minimális egyenleg 0.1 Algó, amelyet minden egyes ASA (Algorand Standard Asset) vagy NFT (non-fungible token) további 0.1 Algóval megemel.

A fájl explorer-rel vagy a terminálban a `nano 06-asset_opt_in.js` segítségével hozzuk létre a következő fájlt:

```
const algosdk = require('algosdk');
const fs = require('fs');
const DEBUG=0;

// define sandbox values for kmd client
const kmd_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const kmd_server = "http://localhost";
const kmd_server_port = 4002;

// define sandbox values for algod client
const algod_token =
↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
const algod_server = "http://localhost";
const algod_server_port = 4001;

async function getWalletId(
    kmdClient,
    walletName) {

    // list wallets
    wallets = await kmdClient.listWallets();
    if(DEBUG) console.log('wallets:', wallets);

    // get wallet index for default wallet
    let walletId='';
    wallets.wallets.forEach(item => {
        if (item.name === walletName) {
            walletId = item.id;
        }
    })
    return walletId;
}

function getAssetIndex(fname) {
    try {
        var data = fs.readFileSync(fname, 'utf8');
        var asset_index = Number(data);
```

```
        if (DEBUG) console.log(asset_index);
    } catch (err) {
        console.error(err);
    }
    return asset_index;
}

async function main() {
    // create kmd client
    kmd_client = new algosdk.Kmd(kmd_token, kmd_server, kmd_server_port);

    // connect to default wallet
    const wallet_name = 'unencrypted-default-wallet';
    const wallet_pw = '';
    wallet_id = await getWalletId(kmd_client, wallet_name);
    wallet_handle = await kmd_client.initWalletHandle(wallet_id,
    ↪  wallet_pw);

    // gather the first three accounts from the wallet
    wallet_addresses = await
    ↪  kmd_client.listKeys(wallet_handle.wallet_handle_token);
    addr1 = wallet_addresses.addresses[0];
    addr2 = wallet_addresses.addresses[1];
    addr3 = wallet_addresses.addresses[2];

    // create algod client
    algod_client = new algosdk.Algodv2(algod_token, algod_server,
    ↪  algod_server_port);

    // get asset index from file
    asset_index = getAssetIndex('5_asset_index.txt');
    if (DEBUG) console.log('asset_index:', asset_index);

    // get params
    params = await algod_client.getTransactionParams().do();
    if (DEBUG) console.log('params:', params);

    // build asset optin transaction
    unsigned_txn =
    ↪  algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
        suggestedParams: params,
```

```
74          from: addr2,
75          to: addr2,
76          assetIndex: asset_index,
77          amount: 0
78      });
79      if (DEBUG) console.log('unsigned_txn:', unsigned_txn);
80
81      // sign transaction
82      addr2_sk = await
        ↪  kmd_client.exportKey(wallet_handle.wallet_handle_token, '',
        ↪  addr2);
83      if (DEBUG) console.log('addr1_sk:', addr2_sk);
84      signed_txn = unsigned_txn.signTxn(addr2_sk.private_key);
85      if (DEBUG) console.log('signed_txn:', signed_txn);
86
87      // submit transaction
88      tx_id = await algod_client.sendRawTransaction(signed_txn).do();
89      console.log("Successfully sent transaction with tx_id: ", tx_id);
90      console.log('tx_id["txId"]:', tx_id['txId']);
91
92      // wait for confirmation
93      console.log('Awaiting confirmation (this may take several
        ↪  seconds)...');
94      const roundTimeout = 7;
95      const confirmed_txn = await algosdk.waitForConfirmation(
96        algod_client,
97        tx_id['txId'],
98        roundTimeout
99      );
100     console.log('Transaction is successfully completed');
101
102     // log confirmed transaction
103     console.log('confirmed_txn:', confirmed_txn);
104
105     // release wallet handle
106     let hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_h
        ↪  andle_token']);
107     if (DEBUG) console.log('wallet handle released, hr:', hr)
108 }
109
```

```
main();
```

Futtassuk le a `06-asset_opt_in.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 06-asset_opt_in.js
Successfully sent transaction with tx_id: { txId:
↪  'MC4NDEDTIW7TDRIZ2HETCADG4JF7WR4RJCGUVUSSHDVHNYLFWHLA' }
tx_id["txId"]: MC4NDEDTIW7TDRIZ2HETCADG4JF7WR4RJCGUVUSSHDVHNYLFWHLA
Awaiting confirmation (this may take several seconds)...
Transaction is successfully completed
confirmed_txn: {
  'confirmed-round': 7,
  'pool-error': '',
  txn: {
    sig: Uint8Array(64) [
       21,  72,  32,  49, 110, 132, 206,  97, 160,  12, 242,
       37, 177,  24,   0, 138,   8, 106, 193,   4,  98,  70,
      202, 194, 177, 178, 169,   3,  42,  32, 187, 228, 235,
       79, 210, 248, 246,  23, 112, 177, 111, 164, 166,  59,
      177,  71, 147, 120,  56, 122, 228, 224,  40, 186,  53,
       70, 152, 218,  47, 242, 249,  78,  62,   9
    ],
    txn: {
      arcv: [Uint8Array],
      fee: 1000,
      fv: 6,
      gen: 'dockernet-v1',
      gh: [Uint8Array],
      lv: 1006,
      snd: [Uint8Array],
      type: 'axfer',
      xaid: 1007
    }
  }
}
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

A tranzakció típus: “axfer”, “asset transfer”, és a “xaid” a “transfer asset id”,

és az “arcv”, az “address receiver” az a cím, amelyik az ASA-ra benevezett.

5.8. Csere atomi tranzakciós csoporttal

Az Algorandban az ASA sok mindent képviselhet. Tegyük fel, hogy példánkban egy másik pénzeszközt jelent. Ahhoz, hogy az ASA-t el tudjuk adni Algorand-ért, biztosítani kell, hogy két tranzakció menjen egyszerre végbe, vagy ne menjen végbe egy sem:

- a vevő Algorand-dal fizet az eladónak az ASA-ért
- az eladó elküldi az ASA-t a vevőnek

Az Algorandban a „tranzakció csoport” fogalma biztosítja a fenti cserét.

A fájl explorer-rel vagy a terminálban a `nano 07-atomic_transaction.js` segítségével hozzuk létre a következő fájlt:

```
1  const algosdk = require('algosdk');
2  const fs = require('fs');
3  const DEBUG=0;
4
5  // define sandbox values for kmd client
6  const kmd_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
7  const kmd_server = "http://localhost";
8  const kmd_server_port = 4002;
9
10 // define sandbox values for algod client
11 const algod_token =
   ↪  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
12 const algod_server = "http://localhost";
13 const algod_server_port = 4001;
14
15 async function getWalletId(
16     kmdClient,
17     walletName) {
18
19     // list wallets
20     wallets = await kmdClient.listWallets();
```

```
    if(DEBUG) console.log('wallets:', wallets);

    // get wallet index for default wallet
    let walletId='';
    wallets.wallets.forEach(item => {
        if (item.name === walletName) {
            walletId = item.id;
        }
    })
    return walletId;
}

function getAssetIndex(fname) {
    var asset_index = undefined;
    try {
        var data = fs.readFileSync(fname, 'utf8');
        asset_index = Number(data);
        if (DEBUG) console.log(asset_index);
    } catch (err) {
        console.error(err);
    }
    return asset_index;
}

async function main() {
    // create kmd client
    kmd_client = new algosdk.Kmd(kmd_token, kmd_server, kmd_server_port);

    // connect to wallet
    const wallet_name = 'unencrypted-default-wallet';
    const wallet_pw = '';
    wallet_id = await getWalletId(kmd_client, wallet_name);
    wallet_handle = await kmd_client.initWalletHandle(wallet_id,
    ↪  wallet_pw);

    // gather the first three accounts from the wallet
    wallet_addresses = await
    ↪  kmd_client.listKeys(wallet_handle.wallet_handle_token);
    addr1 = wallet_addresses.addresses[0];
    addr2 = wallet_addresses.addresses[1];
    addr3 = wallet_addresses.addresses[2];
```

```
60
61      // create algod client
62      algod_client = new algosdk.Algodv2(algod_token, algod_server,
        ↪  algod_server_port);
63
64      // get asset index from file
65      asset_index = getAssetIndex('5_asset_index.txt');
66      if (DEBUG) console.log('asset_index:', asset_index);
67
68      // get params
69      params = await algod_client.getTransactionParams().do();
70      if (DEBUG) console.log('params:', params);
71
72      // build unsigned payment transaction
73      txn_1 = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
74          suggestedParams: params,
75          from: addr2,
76          to: addr1,
77          amount: algosdk.algosToMicroalgos(1.0)
78      });
79
80      // build unsigned asset transfer transaction
81      txn_2 = algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
82          suggestedParams: params,
83          from: addr1,
84          to: addr2,
85          assetIndex: asset_index,
86          amount: 100     // this ASA has 2 decimal places, so this is 1.00
            ↪  FUNTOK
87      });
88
89      // compute group id for transactions
90      gid = algosdk.computeGroupID([txn_1, txn_2]);
91      txn_1.group = gid;
92      txn_2.group = gid;
93
94      // sign transactions
95      addr2_sk = await
        ↪  kmd_client.exportKey(wallet_handle.wallet_handle_token,
        ↪  wallet_pw, addr2);
96      if (DEBUG) console.log('addr2_sk:', addr2_sk);
```

```
    stxn_1 = await txn_1.signTxn(addr2_sk.private_key);

    addr1_sk = await
    ↪  kmd_client.exportKey(wallet_handle.wallet_handle_token,
    ↪  wallet_pw, addr1);
    if (DEBUG) console.log('addr1_sk:', addr1_sk);
    stxn_2 = await txn_2.signTxn(addr1_sk.private_key);

    // assemble transaction group
    signed_group = [stxn_1, stxn_2];

    // submit atomic transaction group
    tx_id = await algod_client.sendRawTransaction(signed_group).do();
    console.log("Successfully sent transaction gropup with tx_id: ",
    ↪  tx_id);
    console.log('tx_id["txId"]:', tx_id['txId']);

    // wait for confirmation
    console.log('Awaiting confirmation (this may take several
    ↪  seconds)...');
    const roundTimeout = 7;
    const confirmed_txn = await algosdk.waitForConfirmation(
      algod_client,
      tx_id['txId'],
      roundTimeout
    );
    console.log('Transaction group is successfully completed');

    // log confirmed transaction
    if (DEBUG) console.log('confirmed_txn:', confirmed_txn);

    // release wallet handle
    hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_handl
    ↪  e_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Futtassuk le a `07-atomic_transaction.js` fájlt:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 07-atomic_transaction.js
Successfully sent transaction gropup with tx_id: { txId:
↪  'EEFGBYXM3AICQCPRUAXOJN3CPXIM6GJQ7G5NHIQUTPWKFUS7OJ7A' }
tx_id["txId"]: EEFGBYXM3AICQCPRUAXOJN3CPXIM6GJQ7G5NHIQUTPWKFUS7OJ7A
Awaiting confirmation (this may take several seconds)...
Transaction group is successfully completed
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

Az `04-account_info.js` segítségével kiírathatjuk, hogy sikerült-e a csere:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $ node 04-account_info.js
account_info {
  address: 'DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ',
  amount: 1999999999146000,
  'amount-without-pending-rewards': 1999999999146000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [
    { amount: 100, 'asset-id': 1002, 'is-frozen': false },
    { amount: 100, 'asset-id': 1007, 'is-frozen': false }
  ],
  'created-apps': [],
  'created-assets': [],
  'min-balance': 300000,
  participation: {
    'selection-participation-key':
    ↪  'JB7ZLgYDgt54LEZ4wpWEpKZOswglTFkLImXprW0yi3A=',
    'state-proof-key': '/Kp6qQ9X2DBrBT1QOBhCzmuDqmdEap/HDPAd9dxI8YUR5+HYg
    ↪  yQvn8456ImNm7vMMT28XgBSY4em3Hl4bOIi4A==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
    'vote-participation-key':
    ↪  'k1S3OrIKTr8D9CoB154NuKOhbikQSYspUPNdGEIWcMY='
  },
```

```
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 8,
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 2,
  'total-created-apps': 0,
  'total-created-assets': 0
}
account_info {
  address: 'DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE',
  amount: 4000000000844000,
  'amount-without-pending-rewards': 4000000000844000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [
    { amount: 9900, 'asset-id': 1002, 'is-frozen': false },
    { amount: 9900, 'asset-id': 1007, 'is-frozen': false }
  ],
  'created-apps': [],
  'created-assets': [
    { index: 1002, params: [Object] },
    { index: 1007, params: [Object] }
  ],
  'min-balance': 300000,
  participation: {
    'selection-participation-key':
    ↪  'tSvVZrUkGrQbS4YTJ7//K/ew56tJGd9rODmXFSK01To=',
    'state-proof-key': 'uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1w
    ↪  o5U9cPx6t+tBK7iwmNgL8OIaMHP3eUkisdIpQ==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
    'vote-participation-key':
    ↪  'EPGO+Y9ojVIgfstV72u8U4sCPLYLgqhys9XV0IzudrM='
  },
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 8,
```

```
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 2,
  'total-created-apps': 0,
  'total-created-assets': 2
}
account_info {
  address: 'ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74',
  amount: 4000000000000000,
  'amount-without-pending-rewards': 4000000000000000,
  'apps-local-state': [],
  'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
  assets: [],
  'created-apps': [],
  'created-assets': [],
  'min-balance': 100000,
  participation: {
    'selection-participation-key':
    ↪  '2m3w5viSs8ZXPhW0+Mns+z8oX3dkJEH3M+DbJQWtN/U=',
    'state-proof-key': 'e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFAIcaO
    ↪  7snXb68TdI5b+Fz2hGNl8hE8qbAQSsNhopp/Q==',
    'vote-first-valid': 0,
    'vote-key-dilution': 10000,
    'vote-last-valid': 30000,
    'vote-participation-key':
    ↪  'ERZwnIHHihb37ERB93xwo6ejSbA3TyAtggzegnrylS8='
  },
  'pending-rewards': 0,
  'reward-base': 0,
  rewards: 0,
  round: 8,
  status: 'Online',
  'total-apps-opted-in': 0,
  'total-assets-opted-in': 0,
  'total-created-apps': 0,
  'total-created-assets': 0
}
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api
↪  (main) $
```

A 41. sorokban látszik, hogy a `DPO7A...YYOE` címre az 1007-es "asset-id"-ből 1,00 FUNTOK megérkezett, ami a 2 tizedesjegy miatt 100-nak látszik.

5.9. Node.js Web-es alkalmazások

A Web-es fejlesztés történhet egyszerű HTML használatával, vagy különféle keretrendszerek (pl. React, Angular, Vue stb.) használatával.

Akár egyszerű HTML-ben történik a fejlesztés, akár keretrendszerrel, a közös ezekben a webfejlesztésekben az, hogy a sok-sok Node.js modult valamilyen „web packer" program segítségével néhány nagyobb fájlba pakolják össze, és ezeket az összepakolt fájlokat egy webszerver program segítségével teszik elérhetővé a böngésző számára.

Fejlesztői üzemmódban a fájlok összepakolása és Web szerveren keresztül történő elérése „azonnal", a forrás minden egyes módosítása után megtörténik. A végleges változat elkészítéséhez a „web packer" alaposabb elemzést végez, és optimalizálja az összepakolt könyvtárak méretét.

5.9.1. Pera WalletConnect mintapélda

A WalletConnect Node.js modul feladata, hogy egy alkalmazás kapcsolatba tudjon lépni a felhasználó pénztárcájával, és a felhasználó engedélyével különféle műveleteket végezhessen.

Az Algorand Pera pénztárcához elkészített Pera Connect SDK segítségével különféle műveleteket lehet kiadni: pl. connect, disconnect, reconnect, payment transaction stb. A Pera Connect dokumentációja a `https://docs.perawallet.app/references/pera-connect` címen található. A Pera mintapéldákat is adott a Pera Connect-hez. A plain vanilla Javascript mintapélda továbbfejlesztésével született az alábbi kis minta, amely a `https://github.com/A-Maugli/pera-connect-vanillajs-demo_mod5` repositoryban érhető el.

A fejlesztés Node.js alatt történt. A `package.json` fájl tartalma:

```
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5$ cat
↪  package.json
{
  "name": "perawallet-connect-vanillajs-demo",
  "version": "1.0.0",
  "description": "Pera WalletConnect demo on Algorand Testnet",
  "scripts": {
    "start": "parcel serve ./src/index.html --open --dist-dir
    ↪  ./dist/bundled_noopt",
    "build": "parcel build ./src/index.html --no-scope-hoist --dist-dir
    ↪  ./dist/bundled && serve ./dist/bundled",
    "clean": "rm -rf ./.parcel-cache ./dist ./node_modules
    ↪  ./package-lock.json"
  },
  "dependencies": {
    "@perawallet/connect": "^1.3.4",
    "algosdk": "^2.7.0",
    "jquery": "^3.7.1",
    "parcel": "^2.11.0"
  },
  "keywords": [
    "@perawallet/connect",
    "Algorand Testnet",
    "parcel v2"
  ],
  "devDependencies": {
    "process": "^0.11.10"
  }
}
```

A használt NPM modulok: `@perawallet/connect`, `algosdk` és `jquery`. A használt Web packer a `parcel` csomag 2.11.0 verziója.

A megvalósított `npm` script-ek:

1. `npm run start` – interaktív web packer + web szerver indítás
2. `npm run build` – web packer optimalizált build indítás, külön web szerver indítás

3. `npm run clean` – a nem feltétlenül szükséges fájlok törlése

Egy lehetséges munkamenet:

```
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5$ npm run clean

> perawallet-connect-vanillajs-demo@1.0.0 clean
> rm -rf ./.parcel-cache ./dist ./node_modules ./package-lock.json

npm notice
npm notice New minor version of npm available! 10.2.4 -> 10.4.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.4.0
npm notice Run npm install -g npm@10.4.0 to update!
npm notice
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5$ npm install
npm WARN deprecated @walletconnect/types@1.8.0: WalletConnect's v1 SDKs are now deprecated. Please upgrade to a v2 SDK. For details see: https://docs.walletconnect.com/
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compatibility
npm WARN deprecated @walletconnect/client@1.8.0: WalletConnect's v1 SDKs are now deprecated. Please upgrade to a v2 SDK. For details see: https://docs.walletconnect.com/

added 256 packages, and audited 257 packages in 40s

91 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5$ du -sh
391M        .
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5$ npm run build

> perawallet-connect-vanillajs-demo@1.0.0 build
```

```
> parcel build ./src/index.html --no-scope-hoist --dist-dir
↪  ./dist/bundled && serve ./dist/bundled

★ Built in 5.73s

dist/bundled/index.html                      1.66 KB    1.23s
dist/bundled/index.0ed18bef.js             667.56 KB    1.41s
dist/bundled/App-94e9365e.cefc48d1.js      320.25 KB    1.66s
dist/bundled/index.runtime.7745c7dc.js       2.21 KB    592ms


   ┌──────────────────────────────────────────┐
   │                                          │
   │   Serving!                               │
   │                                          │
   │   - Local:    http://localhost:3000      │
   │   - Network:  http://10.0.2.15:3000      │
   │                                          │
   │   Copied local address to clipboard!     │
   │                                          │
   └──────────────────────────────────────────┘


 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 47 ms
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /index.runtime.7745c7dc.js
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 3 ms
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /index.0ed18bef.js
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 8 ms
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /App-94e9365e.cefc48d1.js
 HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 4 ms
^C
 INFO  Gracefully shutting down. Please wait...
```

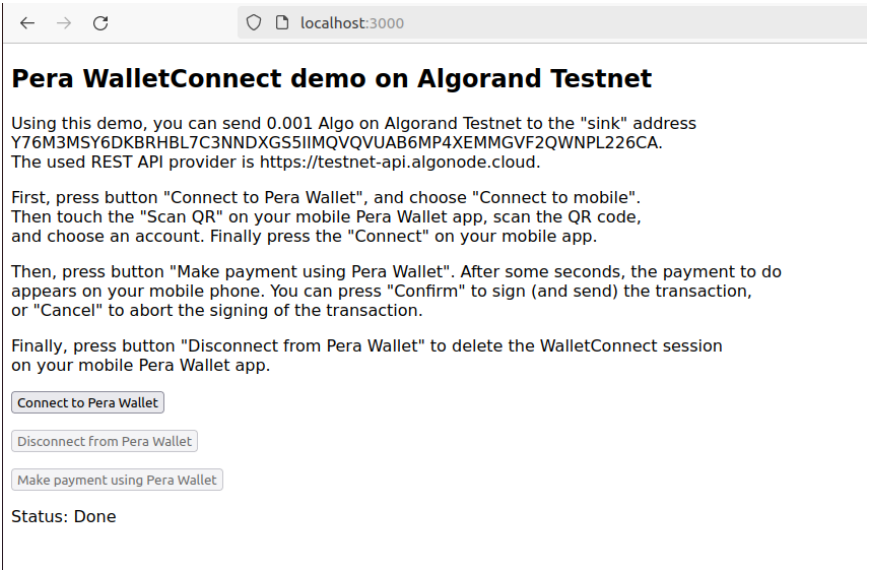
A munkamenet magyarázata:

- 1. sor: törlünk mindet, amit csak lehet
- 11. sor: installáljuk a `node_modules` könyvtárba a Node.js modulokat
- 22. sor: kiíratjuk, hogy mennyi diszk területet használ a projektünk. Apait-anyait magához rántott, mert 391 Mbyte lett az összesített mé-

rete.

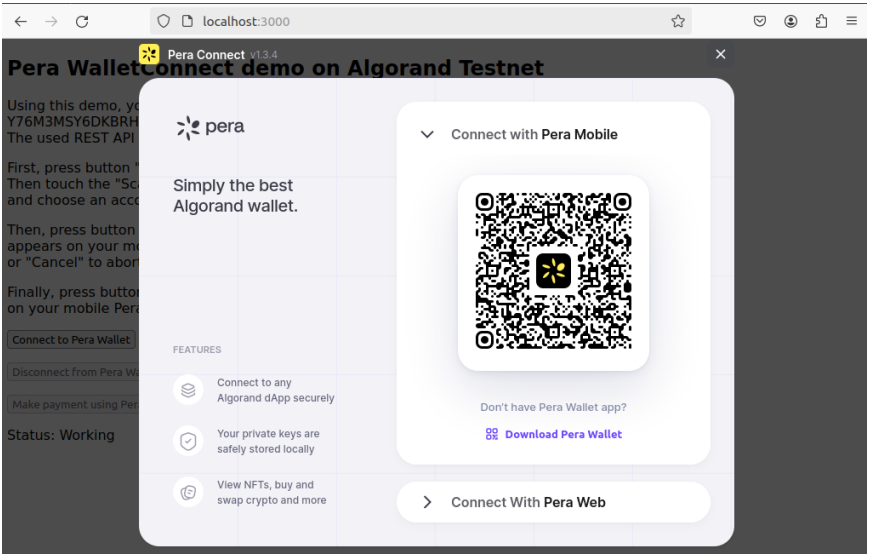
- 24. sor: felépítjük az optimalizált, összepakolt alkalmazást, és elindítunk egy web szervert

A böngészőben elindított demó alkalmazás az 1. ábrán látható. Funkciói:



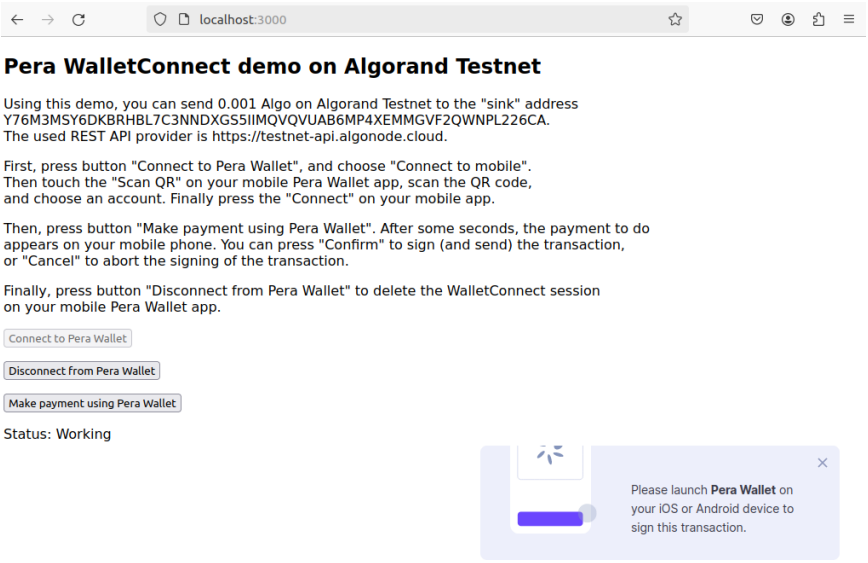
1. ábra. A Pera WalletConnect demó alkalmazás induláskor

- "Connect to Pera Wallet" – a demó alkalmazás összekapcsolása a Pera pénztárcával. A gomb megnyomása után választani tudunk, hogy a demó alkalmazás egy Web-es pénztárcával vagy egy iOS/Android pénztárcával kapcsolódjon össze. Válasszuk a "Connect to mobile" („Kapcsolódás mobil pénztárcához") lehetőséget. Ekkor a képernyőn megjelenik egy QR kód, lásd 2. ábra. A mobilunkon nyissuk meg a Pera pénztárcát, és pásztázzuk be a QR kódot. Válasszunk ki a mobilon egy számlát, amit szeretnénk összekapcsolni a Pera WalletConnect demó alkalmazással.



2. ábra. Kapcsolódás a mobil Pera pénztárcához

- "Make payment using PeraWallet" – fizetési művelet végzése, 0,001 Algo küldése az Algorand „mindent elnyelő" címére, lásd 3. ábra. A gomb megnyomása után pár másodperccel megjelenik egy fizetési tranzakciós képernyő a mobilunkon. A tranzakció a "Confirm" („Megerősítés") gomb megnyomásával írható alá és küldhető el, a "Cancel" („Elhalasztás") gomb megnyomásával pedig törölhető.
- "Disconnect from Pera Wallet" – a WalletConnect munkamenet törlését végzi. A munkamenet a mobilon is törölhető, a Settings | WalletConnect Sessions részben.



3. ábra. Fizetési tranzakció kezdeményezése

Az demó alkalmazás forrása a `src` könyvtárban van. Az `src/index.html` tartalma:

```
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5/src$ cat index.html
<!DOCTYPE html>
<html>

<head>
  <title>Pera WalletConnect demo on Algorand Testnet</title>
  <meta charset="UTF-8">
  <style>
    div.c1 {
      margin-top: 1em;
      margin-bottom: 1em;
    }

    body {
```

```
      font-family: sans-serif;
    }
  </style>
  <script type="module" src="./pera_walletconnect_demo.js"></script>
</head>

<body>
  <h2>Pera WalletConnect demo on Algorand Testnet</h2>
...
  <div class="c1">
    <button id="connect_pera_wallet">Connect to Pera Wallet</button>
  </div>
  <div class="c1">
    <button id="disconnect_pera_wallet">Disconnect from Pera
    ↪  Wallet</button>
  </div>
  <div class="c1">
    <button id="make_payment_pera_wallet">Make payment using Pera
    ↪  Wallet</button>
  </div>
  <p aria-label="Status of last command"
  ↪  id="wallet_connect_status">Status: </p>
  <p aria-label="Optional error message" id="wallet_connect_error">Error:
  ↪  </p>
</body>

</html>
```

Az `index.html` a weblapot írja le. Az ismertető szövege `<p>` tagok között van, ezt követi a funkciókhoz tartozó három nyomógomb létrehozása a `<button>` segítségével. A böngésző által megjelenített weblapot az 1. ábra mutatja.

A nyomógombokat kezelő Javascript a `src/pera_walletconnect_demo.js` fájlban található.

```
lipi@lipi-VirtualBox:~/Downloads/pera-connect-vanillajs-demo_mod5/src$
↪  cat pera_walletconnect_demo.js
"use strict";

```

```
4  import algosdk from "algosdk";
5  //import { PeraWalletConnect } from "@perawallet/connect";
6  import * as pwcsdk from "@perawallet/connect";
7  import $ from "jquery";
8
9  const working = "Status: Working";
10 const done = "Status: Done";
11
12 const peraWallet = new pwcsdk.PeraWalletConnect();
13
14 const token = "";
15 const server = "https://testnet-api.algonode.cloud";
16 const port = 443;
17 const client = new algosdk.Algodv2(token, server, port);
18
19 let accountAddress = "";
20 const sinkAddress =
   ↪  "Y76M3MSY6DKBRHBL7C3NNDXGS5IIMQVQVUAB6MP4XEMMGVF2QWNPL226CA";
21
22
23 $(window.document).ready(function () {
24   console.log("document ready");
25   addEventListeners();
26   reconnectSession();
27 });
28
29 function addEventListeners() {
30 $("#connect_pera_wallet").on("click", function () {
31   handleConnectWallet();
32 });
33
34 $("#disconnect_pera_wallet").on("click", function () {
35   handleDisconnectWallet();
36 });
37
38 $("#make_payment_pera_wallet").on("click", function () {
39   handlePayment();
40 })
41 }
42
43 function reconnectSession() {
```

```
  setStatus(working);
  clearError();
  // Reconnect to the session when the component is mounted
  peraWallet
    .reconnectSession()
    .then((accounts) => {
      if (peraWallet.connector !== null) {
        peraWallet.connector.on("disconnect", handleDisconnectWallet);
      }
      if (accounts.length) {
        accountAddress = accounts[0];
        setButtonState(true);
      }
      else {
        setButtonState(false);
      }
      setStatus(done);
    })
    .catch((error) => {
      console.log('Error in reconnectSession: ' + error.name + ' ' +
      ↪  error.message);
      setStatus(done);
      setError(error);
    });
}

function setButtonState(connected) {
  if (connected) {
    $("#connect_pera_wallet").prop('disabled', true);
    $("#disconnect_pera_wallet").prop('disabled', false);
    $("#make_payment_pera_wallet").prop('disabled', false);
  }
  else {
    $("#connect_pera_wallet").prop('disabled', false);
    $("#disconnect_pera_wallet").prop('disabled', true);
    $("#make_payment_pera_wallet").prop('disabled', true);
  }
}

function setStatus(msg) {
  $("#wallet_connect_status").text(msg);
```

```
}

function setError(error) {
  const error_name = error.name;
  const error_message = error.message.replace(/\n/g, '<br/>');
  if (error_message != "") {
    $("#wallet_connect_error").html('Error: ' + error_name + ' ' +
    ↪  error_message);
    $("#wallet_connect_error").show(0);
  }
  else {
    $("#wallet_connect_error").text('No errors');
    $("#wallet_connect_error").hide();
  }
}

function clearError() {
  setError({ name: '', message: '' })
}

function handleConnectWallet() {
  setStatus(working);
  clearError();
  peraWallet
    .connect()
    .then((newAccounts) => {
      peraWallet.connector.on("disconnect", handleDisconnectWallet);
      accountAddress = newAccounts[0];
      setButtonState(true);
      setStatus(done);
    })
    .catch((error) => {
      if (error?.data?.type !== "CONNECT_MODAL_CLOSED") {
        console.log('Error in handleConnectWallet: ' + error.name + ' ' +
        ↪  error.message);
        setStatus(done);
        setError(error);
      }
    });
}

```

```
function handleDisconnectWallet() {
  setStatus(working);
  clearError();
  peraWallet
    .disconnect()
    .then(() => {
      setButtonState(false);
      setStatus(done);
    })
    .catch((error) => {
      console.log('Error in handleDisconnectWallet: ' + error.name + ' '
      ↪  + error.message);
      setStatus(done);
      setError(error);
    });
  accountAddress = "";
}

async function handlePayment() {
  setStatus(working);
  clearError();
  const params = await client.getTransactionParams().do();
  const txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
    from: accountAddress,
    to: sinkAddress,
    amount: algosdk.algosToMicroalgos(0.001),
    suggestedParams: params
  });
  let txGroup = [{ txn, signers: [accountAddress] }];
  try {
    const signedTxnGroup = await peraWallet.signTransaction([txGroup]);
    const { txId } = await client.sendRawTransaction(signedTxnGroup).do();
    console.log('Info in handlePayment: ', 'Payment txn sent to Algorand
    ↪  network');
    console.log('Info in handlePayment: ', 'txId: ', txId);
    setStatus(done);
  } catch (error) {
    console.log('Error in handlePayment: ' + error.name + ' ' +
    ↪  error.message);
    setStatus(done);
    setError(error);
```

```
161   }
162 }
```

Mikor DOM felépül, akkor megtörténik a nyomógombok eseménykezelőinek beállítása (25. sor, 29..41. sor) és a legutolsó WalletConnect munkamenet visszaállítása (26. sor, 43..67. sor). A gombok engedélyezése/tiltása a 69..80. sorokban történik, aszerint, hogy volt-e már sikeres kapcsolódás a pénztárcával. A státusz üzenet kiírása a 82..84. sorokban történik, az esetleges hibaüzenet kiírása a 86..97. sorokban. A pénztárcával való összekapcsolódás kezelése a 103..121. sorokban van, a szétkapcsolás kezelése a 123..138. sorokban. A fizetési tranzakció kezelése a 140..162. sorokban látható.

5.10. Közvetlenül a Web böngészőre támaszkodó alkalmazásfejlesztés

Ez a fajta alkalmazásfejlesztés kihagyja a Node.js szintet. A HTML, CSS és Javascript fájlokat a fejlesztő közvetlenül a Web böngészőben futtathatja. A használt Javascript könyvtárakat az alkalmazás összepakolt formájukban hivatkozza meg. A belövéskor a böngésző fejlesztői környezete használható (F12).

Az algosdk JavaScript könyvtár összepakolt formában pl. CDN-ekről is elérhető. A teljes könyvtár kb. 300 Kbyte méretű. Azt, hogy hogyan lehet a `@perawallet/connect` Node.js modulból könyvtárat készíteni, a következő rész mutatja be.

5.10.1. A `@perawallet/connect` összepakolása

A `@perawallet/connect` csak a Node.js alatt érhető el, és nincs meg „összepakolt”, a web böngészők által is fogyasztható formában. A Web böngészőre támaszkodó fejlesztéshez ezt is össze kell pakolni valamelyik ismert „web packer”, pl. a webpack használatával.

A `https://github.com/A-Maugli/pera-conect-vanillajs-demo-webpack` repository-ban található egyfajta megoldás erre a feladatra.

A `@perawallet/connect` Node.js modult a `src/pwc.js` fájl hivatkozza meg:

```
//import {PeraWalletConnect} from "@perawallet/connect";
const pwcsdk = require("@perawallet/connect");
exports.pwcsdk = pwcsdk;

const algosdk = require("algosdk");
exports.algosdk = algosdk;
```

A fájl kiexportálja mind a Pera wallet connect SDK-t, lásd `pwcsdk`, mind az Algorand JS SDK-t, lásd `algosdk`.

A webpack konfigurációs állománya, a `webpack.config.js` a következő:

```
const path = require('path');

module.exports = {
  mode: 'production',
  entry: './src/pwc.js',
  output: {
    filename: 'perawalletconnect.min.js',
    path: path.resolve(__dirname, 'dist/browser'),
    library: {
      type: 'umd',
      name: 'pwc',
    },
  },
  devtool: 'source-map',
  resolve: {
    // Add '.ts' as resolvable extensions
    extensions: ['.ts', '.js'],
  },
  module: {
    rules: [
      // All files with a '.ts' extension will be handled by 'ts-loader'.
      {
        test: /\.ts\$/,
        loader: 'ts-loader',
        options: {
```

```
26            configFile: path.resolve(__dirname, 'tsconfig-browser.json'),
27          },
28        },
29
30        // All output '.js' files will have any sourcemaps re-processed by
          ↪  'source-map-loader'.
31        ////{ test: /\.js\$/, loader: 'source-map-loader' },
32      ],
33      // Don't parse tweetnacl module -
          ↪  https://github.com/dchest/tweetnacl-js/wiki/Using-with-Webpack
34      noParse: [/[\\/]tweetnacl[\\/]/, /[\\/]tweetnacl-auth[\\/]/],
35    },
36  };
```

Az 'entry' belépési pontot a könyvtárba a `./src/pwc.js` fájl definiálja (5. sor). A könyvtár típusa 'umd' lesz (univerzálisan elérhető modul), a neve pedig 'pwc', ami a Pera wallet connect rövídítéséből adódott (10. és 11. sor). Az eredmény a `dist/browser` könyvtárban áll elő (8. sor).

Az összepakolt könyvtár a `public/test.html` segítségével tesztelhető:

```
1   <!DOCTYPE html>
2   <html>
3
4   <head>
5     <title>Test for @perawallet/connect packed JavaScript library</title>
6     <meta charset="UTF-8">
7     <!--
8     <script type="module" src="./browser/algosdk.min.js"></script>
9     -->
10    <!-- Note: perawalletconnect.min.js contains both pwcsdk and algosdk -->
11    <!--       but algosdk version is v2.1 -->
12    <script type="module" src="./jslib/perawalletconnect.min.js"></script>
13  </head>
14
15  <body>
16  <h2>Test for @perawallet/connect packed JavaScript library</h2>
17  <p>Use F12 to see Console</p>
18  <p>Expected:</p>
19  <p>success: Pera wallet connection created</p>
```

```
20  <p>success: algod client created</p>
21  <script>
22  window.onload = function(e) {
23    const pwcsdk = pwc.pwcsdk;
24    const algosdk = pwc.algosdk;
25
26    // Create pera wallet connection
27    try {
28      const peraWallet = new pwcsdk.PeraWalletConnect();
29      console.log('success: Pera wallet connection created');
30    }
31    catch (e) {
32      console.log('error: ',  e);
33    }
34
35    // Create algod client
36    const token = "";
37    const server = "https://testnet-api.algonode.cloud";
38    const port = 443;
39    try {
40      const client = new algosdk.Algodv2(token, server, port);
41      console.log('success: algod client created');
42    }
43    catch (e) {
44      console.log('error: ', e);
45    }
46  }
47  </script>
48  </body>
49  </html>
```

Az összepakolt könyvtárra történő hivatkozás a böngészőkben megszokott módon, a `<script>...</script>` paranccsal történik (12. sor). A könyvtár tesztelését a 21..47. sorok közötti Javascript végzi.

Megvárjuk, amíg a DOM előáll, vagyis a `window.onload` eseménykezelőbe tesszük a további teszt sorokat. Itt előállítjuk a `pwc` objektumból a `pwcsdk` és `algosdk` objektumot (23. és 24. sor), majd kiadunk egy `PeraWalletConnect()` hívást (28. sor), és naplózzuk a siker vagy hiba té-

nyét a konzolon.

A `pwc`-be a `@perawallet/connect` függőségei miatt az Algorand JS SDK 2.1 változata is belekerült. A teszt azt is bemutatja, hogy az összepakolt könyvtárral valóban létre tudunk hozni egy Algorand klienst is (40. sor).

Egy másik példa az előző pera wallet connect demót mutatja be újra. A hozzá tartozó két fájl:

- `public/index.html`
- `public/pera_walletconnect_demo.js`

A `public/index.html` az összepakolt könyvtárat a következőképpen hivatkozza:

```
1   <script type="module" src="./jslib/perawalletconnect.min.js"
2     integrity="sha384-1/BfpY6oNlkbLhIQ2HqXVz2NzZb2zw5D6iDz6Qkdi1E6dCxZyJm⌋
      ↪  1+NN9SMLamlXm"
3     crossorigin="anonymous"></script>
```

Itt már megadtuk a könyvtár épségének az ellenőrzésére a könyvtár sha384-es ellenőrző összegét is.

A `public/pera_walletconnect_demo.js` fájlnak csak az eleje más a Node.js változathoz képest:

```
1   "use strict";
2
3   //import algosdk from "algosdk";
4   ////import { PeraWalletConnect } from "@perawallet/connect";
5   //import * as pwcsdk from "@perawallet/connect";
6   //import $ from "jquery";
7
8   const working = "Status: Working";
9   const done = "Status: Done";
10
11  const pwcsdk = pwc.pwcsdk;
12  const peraWallet = new pwcsdk.PeraWalletConnect();
```

```
13 ...
```

5.10.2. Csocsó eredmények rögzítése az Algorand Testnet blokkláncon

Az Algorand tranzakciók `note` mezőjében max. 1 Kbyte információ adható meg. Egy Algorand tranzakció költsége 0,001 Algó, ami a jelenlegi Algó árfolyam mellett kb. 10 filléres díjnak felel meg.[2] A tranzakciók költsége csak akkor nagyobb, ha hálózati torlódás van. Az Algorand hálózat jelenleg 7500 tps sebességű, ahol a tps a tranzakció per sec rövidítése. Mivel a hálózati forgalom jelenleg 50..100 tps, nagyon valószínűtlen a 0,001 Algónál nagyobb díj. Egyébként is, az alkalmazás a Testnet blokkláncon fut, ahol ingyen lehet teszt Algókhoz jutni.

A demó alkalmazás a `https://github.com/A-Maugli/csocso-vanillajs-pwc` repository-ban van, a `src` könyvtárban. Az `index.html` meghivatkozza a használt Javascript könyvtárakat, ezek az algosdk.min.js, a jquery-3.7.1.min.js, és a perawalletconnect.min.js könyvtárak. A HTML fájl két része: a `Store Game Result` (23..35. sor) és a `Read Game Results` (41..47. sor).

A HTML felületen megadható a játékosok neve és az eredmény. A `Store game result` gomb megnyomásakor az alkalmazás a `JW6L2Z...FNKILM` számláról a `UUOB7Z...C7Q62E` „univerzális nyelő” számlára elküld 0,001 Algót, és a tranzakció `note` mezőjébe JSON formátumban beírja a mérkőzés eredményét. A fizetési tranzakció a Pera wallet connect segítségével történik, biztonságos módon, így a demó alkalmazás sehol sem tartalmazza a `JW6L2Z...FNKILM` forrás számla privát kulcsát.

A `Read game results` gomb megnyomásakor egy keresési műveletet küldünk az indexernek, és visszaolvassuk azokat a tranzakciókat, amelyekben a forrás a `JW6L2Z...FNKILM` számla volt. Szűrést végzünk, és csak azokat a tranzakciókat írjuk ki, amelyekben a `note` mezőben lévő JSON

[2] A tranzakciós díj a jövőben változhat. Érdemes az SDK konstansokat használni, pl. a JS SDK-ban ezt: `const minFee = algosdk.ALGORAND_MIN_TX_FEE`

4. ábra. A Csocsó demó alkalmazás felhasználói felülete

struktúra a megfelelő formátumú.

A demót megvalósító `csocso.js` tartalma:

```
"use strict";

//const algod = require('algosdk');
//const pwcsdk = require('@perawallet/connect');
//const $ = require('jquery');

const ver = "0.2.12"; // Use PeraWalletConnect();

/* Algonode.io API */
const baseServer = "https://testnet-api.algonode.cloud";
const baseServerIdx = "https://testnet-idx.algonode.cloud";
const port = 443;
const token = "";

```

```
const working = "Working";
const ready = "Ready";
var addr_src, addr_dst;
var account_addr = '';  // address selected on PeraWallet

const testing = false;
if (testing) {
  debugger;
  addr_src =
  ↪  'CDRLCYZICK7XEBZ7M4HKYWNB7ZZLO4BFOQ5RGZNMHQ5ZH7EZWUDFQ6Z32U';  //
  ↪  Test account 1
  addr_dst = 'UUOB7ZC2IEE4A7JO4WY4TXKXWDFNATM43TL73IZRAFIFFOE6ORPKC7Q62E';
}
else {
  addr_src =
  ↪  'JW6L2ZCQT3UIQH5AFM3CVW3C7M3QFYHXA3EU4WHREOATRWMXP6MBFNKILM';  //
  ↪  Csocsó account
  addr_dst = 'UUOB7ZC2IEE4A7JO4WY4TXKXWDFNATM43TL73IZRAFIFFOE6ORPKC7Q62E';
}

const algod_client = new algosdk.Algodv2(token, baseServer, port);
console.log('algod_client created');
const indexer_client = new algosdk.Indexer(token, baseServerIdx, port);
console.log('indexer_client created');

const pwcsdk = pwc.pwcsdk;
const peraWallet = new pwcsdk.PeraWalletConnect();
console.log('peraWallet created');

async function reconnectSessionA() {
  let accounts = await peraWallet.reconnectSession();
  if (peraWallet.connector !== null) {
    peraWallet.connector.on("disconnect", handleDisconnectWalletA);
  }
  return accounts;
}

async function handleConnectWalletA() {
  let newAccounts = await peraWallet.connect();
  peraWallet.connector.on("disconnect", handleDisconnectWalletA);
  return newAccounts;
```

```
52  }
53
54  async function handleDisconnectWalletA() {
55    await peraWallet.disconnect();
56    console.log('Info in handleDisconnectWallet: ', 'disconnected');
57    let accountAddress = "";
58    return accountAddress;
59  }
60
61  function algo_send_tx(algod_client, note) {
62    (async () => {
63      // Reconnect/connect wallet
64      let accounts = await reconnectSessionA();
65      if (accounts.length == 0) {
66        accounts = await handleConnectWalletA();
67      }
68      if (accounts.length == 0) {
69        throw ('Wallet connect error');
70      }
71      if (accounts[0] !== addr_src) {
72        await handleDisconnectWalletA();
73        throw ('Please connect to ' + addr_src);
74      }
75      else {
76        account_addr = accounts[0];
77      }
78
79      //  get params from algod
80      let params = await algod_client.getTransactionParams().do();
81
82      let obj = {
83        "from": account_addr,
84        "to": addr_dst,
85        "amount": 1,
86        "note": algosdk.encodeObj(note),
87        "suggestedParams": params
88      };
89      let txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject(obj);
90      // sign transaction
91      let txGroup = [{ txn, signers: [account_addr] }];
92      let signedTxnGroup = await peraWallet.signTransaction([txGroup]);
```

```
    // submit transaction
    let tx = await algod_client.sendRawTransaction(signedTxnGroup).do();
    //console.log("Transaction id: " + tx.txId);
    $('#tx_id').val(tx.txId);
    const waitRounds = 5;
    await algosdk.waitForConfirmation(algod_client, tx.txId, waitRounds);
    $('#send_tx_status').val(ready);
  })().catch(e => {
    console.log(e);
    $('#send_tx_status').val(e);
  });
}

function algo_send_tx_outer() {
  if ($('#send_tx_status').val() !== working) {
    $('#send_tx_status').val(working);
    let game_name = $('#game').val();
    let user1 = $('#user1').val();
    let user2 = $('#user2').val();
    let goal1 = $('#goal1').val();
    let goal2 = $('#goal2').val();
    let ms_since_1970 = (new Date()).valueOf();
    let note = {
      game: game_name,
      user1: user1,
      user2: user2,
      goal1: goal1,
      goal2: goal2,
      date: ms_since_1970
    };
    //console.log("Note: " + JSON.stringify(note));
    algo_send_tx(algod_client, note);
  }
}

function make_list_from_tx(tx) {
  let list = "";
  let num_tx = tx.transactions.length;
  for (let i = 0; i < num_tx; i++) {
    let note1 = { game: "", user1: "", user2: "", goal1: "", goal2: "",
    ↪  date: 0 };
```

```
133     try {
134       if (typeof (tx.transactions[i].note) !== 'undefined') {
135         //console.log('i:'+' note:'+ tx.transactions[i].note);
136         const buff = Buffer.from(tx.transactions[i].note, 'base64');   //
            ↪  Node.js Buffer.from
137         //const buff = base64js.toByteArray(tx.transactions[i].note);
138         note1 = algosdk.decodeObj(buff);
139       }
140     }
141     catch (e) {/*alert(e);*/ };
142
143     if ((tx.transactions[i]['tx-type'] === "pay") && (note1.game ===
        ↪  "Csocsó") &&
144       (note1.goal1 !== undefined) && (note1.goal2 !== undefined)) {
145       list += "<li>" + note1.user1 + ' - ' + note1.user2 + ': ' +
146         note1.goal1 + ' : ' + note1.goal2;
147       list += "   ";
148       list += "\t Round: " + tx.transactions[i]['confirmed-round'];
149       list += "   ";
150       list += "\t Fee: " + tx.transactions[i].fee / 1000000.0;
151       list += "   ";
152       if (typeof note1.date !== "undefined") {
153         list += "   ";
154         let date = new Date(note1.date);
155         let date1 = date.toLocaleDateString("hu-HU");
156         let time1 = date.toLocaleTimeString("hu-HU");
157         list += date1 + " " + time1;
158       }
159       list += "</li>";
160     }
161   }
162   return list
163 }
164
165
166 function algo_get_tx() {
167   if ($('#get_tx_status').val() !== working) {
168     (async () => {
169       $('#get_tx_status').val(working);
170       $('#game_list').empty();
171
```

```
      let next_token = "";
      let tx_limit = 50;
      let num_tx = 1;
      let list = "";

      while (num_tx > 0) {
        let tx = await indexer_client
          .lookupAccountTransactions(addr_src)
          .limit(tx_limit)
          .nextToken(next_token).do();
        num_tx = tx.transactions.length;
        list += make_list_from_tx(tx);
        next_token = tx['next-token'];
      }
      $('#get_tx_status').val(ready);
      $("#game_list").append(list);
      $("#game_list").show();
    })().catch(e => {
      console.log(e);
    });
  }
}

$(window.document).ready(function () {
  console.log("document ready");
  const game_name = "Csocsó";
  const user1 = "CsG";
  const user2 = "LG";
  $('#game').val(game_name);
  $('#user1').val(user1);
  $('#user2').val(user2);
  $('#goal1').val('0');
  $('#goal2').val('0');
  $('#tx_id').val('');
  $('#send_tx_status').val('');

  $('#game_list').empty();
  $('#get_tx_status').val('');

  $('#store_game_result').on("click", function () {
    console.log("Handler for `Store game result` is called.");
```

```
213       algo_send_tx_outer();
214     });
215
216     $('#read_game_results').on("click", function () {
217       console.log("Handler for `Read game results` is called.");
218       algo_get_tx();
219     });
220
221   });
```

Az Algod és Indexer szolgáltatás REST végpontjának megadása a 10. és 11. sorban történt. Az `algod_client` és `indexer_client` létrehozása a 31. és 33. sorban látható. A `peraWallet` létrehozása (37. sor) után a végrehajtás a `$(document).ready` sorban (195. sor) folytatódik. A HTML mezők kezdeti feltöltése után (197..209. sor) a gombokhoz tartozó eseménykezelők beállítása történik: az `id="store_game_result"` azonosítójú gombé a 211..214. sorban, az `id="read_game_results"` azonosítójú gombé a 216..219. sorban. A gombnyomáskor hívott függvények a `algo_send_tx_outer()` és a `algo_get_tx()`.

Lássuk ezek után a függvényeket. A Pera pénztárca kapcsolatot kezelő függvények (Pera wallet connect) a következők: a `reconnectSessionA` aszinkron függvény (40..46. sor) egy pénztárca kapcsolatot próbál újraépíteni. Ha sikeres a futása, akkor az `accounts` tömbben adja vissza a pénztárca műveletekben használható számlaszámokat. A `handleConnectWalletA` aszinkron függvény (48..52. sor) egy pénztárca kapcsolatot hoz létre. Ha sikeres a futása, akkor a `newAccounts` tömbben adja vissza a pénztárca műveletekben használható számlaszámokat. A `handleDisconnectWalletA` aszinkron függvény egy pénztárca kapcsolatot szüntet meg.

Az `algo_send_tx` függvény a `note` tartalmát küldi el a blokkláncra. A függvény megpróbál egy létező pénztárca kapcsolatot használni (64. sor). Ha ez a kapcsolat nem ajánlja fel azt a címet, amin a tranzakciót küldeni szeretnénk, akkor hibát dob (71..73. sor), egyébként pedig az `account_addr` változóba teszi a számlaszámot (76. sor). Ezután beolvassa az Algorand

hálózatról a paramétereket (pl. kezdő blokk, genezis blokk, stb., lásd 80. sor), és létrehoz egy fizetési tranzakciót (82..89. sor). A tranzakciót a Pera pénztárcával aláíratja (91..92. sor), majd az aláírt tranzakciót elküldi az Algorand hálózatnak (94. sor). Kiírja a tranzakció azonosítót a HTML formra (96. sor), és várakozni kezd mindaddig, de max. 5 körön keresztül, amíg be nem kerül a blokkláncba a tranzakció (98. sor). Ha a tranzakció beíródott a blokkláncba, akkor kiírja a formra, hogy a tx elküldése rendben megtörtént (99. sor), egyébként pedig a kivétel kezelésével kiírja a hibát a formra (102. sor).

A `algo_send_tx_outer()` függvény beolvassa a formról a paramétereket (109..114. sor), összeállítja a `note` rekordot (115..122. sor), és meghívja az előzőleg ismertetett `algo_send_tx` függvényt.

A `algo_get_tx` függvény – az indexer segítségével – ciklusban visszaolvassa azokat a tranzakciókat a blokkláncról, melyekben a forrás cím az általunk megadott cím, lásd 177..185. sor. A kiolvasás részletekben történik, egyszerre `tx_limit` (50) tranzakciót olvas vissza. A visszaolvasott tranzakciókat a `make_list_from_tx` függvénnyel alakítja át megjeleníthető formátumúra (hívása a 183. sorban, a függvény a 128..163. sorban látható.)

Szeretném arra a „furcsaságra” felhívni a figyelmet, hogy a hálózattal való kommunikációt a `.do()` hatására végzi el a kliens JS API függvény, lásd pl. a 80., 94. és 181. sort.

5.11. Az első rész összefoglalása

Az első részben bemutattuk:

- az AlgoKit installálását
- az Algorand `goal` parancs használatát
- a Python API használatát
- a JS API használatát
- a Pera pénztárca kezelését a walletconnect interfész segítségével

- a Node.js alatti modulok összepakolását Web böngésző számára
- a `note` mező használatát nyilvántartási feladatokra

A következő részben az Algorand blokklánc programozhatóságáról fogunk szólni. Bemutatjuk majd:

- az AVM néhány utasítását
- a TEAL használatát parancsnyelvi környezetben
- a PyTeal használatát smart signature-ok vagy Algorand szerződések írására
- a LORA blokklánc vizsgáló használatát az Algorand szerződések tesztelésére
- a Beaker használatát Algorand szerződések írására, telepítésére, tesztelésére
- a TealScript használatát Algorand szerződések írására, telepítésére, tesztelésére
- a TEAL kód debug-olását
- a PuyaPy natív Python fordítóprogramot, amellyel Python-ban írt programokat lehet közvetlenül TEAL szerződésekké lefordítani

6. A TEAL

A TEAL az Algorand Virtuális Munkagép (AVM) „assembly nyelve”. A lefordított program a bytekód, melyet az AVM közvetlenül értelmezni képes.

6.1. A TEAL program elemei

A TEAL program elemei:

- `#pragma version`, amely a használni kívánt AVM verzóját adja meg. Jelenleg 1 és 11 közötti verzió adható meg.
- cimkék, amelyekre az ugró utasításokban lehet hivatkozni, pl. `l10:`, `subr_hello:`
- szimbolikus műveleti kódok, amelyek a nevükkel a végrehajtott műveletre emlékeztetnek, pl. `dup`, `callsub`, `return`
- a szimbolikus műveleti kódhoz tartozó paraméterek, pl. az ugró utasítások esetén egy cimke, vagy paraméter betöltéskor a betöltendő paraméter neve, pl.
  `asset_param_get AssetUnitName`
- megjegyzések, pl.
  `// ez itt egy megjegyzés`

A TEAL az AVM egyes műveleteihez egy-egy szimbolikus kódot rendel. A műveletek paraméterei lehetnek magában a bytekódban tárolva, vagy az AVM architektúrájának megfelelő egyéb tárterületeken, pl. a stack-en, az átmeneti tároló területen stb.

Pl. a `dup` utasítás AVM kódja `0x49`. Ez az utasítás a stack tetején lévő értéket újra feldobja a stack-re.

6.2. Az AVM felépítése (architektúrája)

Az AVM következő végrehajtandó utasítását a `PC`, a program számláló mutatja. Az egyes utasításokkal a következő adatok érhetők el:

- a stack, amely 1000 elemű, és egy eleme képes egy 64 bites előjel nélküli egész szám vagy egy max. 4096 hosszú `byte[]` string tárolására.
- egy átmeneti tároló terület, amely 256 rekeszből áll, és mindegyik rekesz képes egy 64 bites előjel nélküli egész szám vagy egy max. 4096 hosszú `byte[]` string tárolására.
- a tranzakcióban szereplő számlák vagy ASA-k mezői, pl. a küldő fél Algorand címe, a fogadó fél Algorand címe, a küldött összeg stb.
- globális paraméterek, pl. `MinTxFee` minimális tranzakciós díj, vagy a hálózat azonosítására szolgáló `GenesisHash` stb.
- az okos szerződést létrehozó számlához tartozó globális kulcs-érték párok, max. 64 db
- az okos szerződésbe benevező számlá(k)hoz tartozó lokális kulcs-érték párok, max. 16 db
- mailbox-ok, melyek max. 32K hosszúak lehetnek, de korlátlan számú lehet belőlük
- az okos szerződés vagy okos aláírás hívásakor megadott paraméterek

6.3. AVM adattípusok

A stack-en csak `uint64` és `[]byte` tárolható, ahol a `[]byte` vektor max. hossza 4096. Egyes AVM utasítások ennél korlátozottabb értéktartománnyal rendelkeznek. Pl. a bigint műveletekben értelemszerűen csak a bigint adattípus használható.

Néhány ilyen AVM adattípus:

- Előjeltelen egész szám: `uint64` x, $0 \leqslant x \leqslant 2^{64}-1$. A stack-en `uint64`

adattípusként tárolódik.

- Byte vektor (string): `[]byte x`, ahol x hossza nem nagyobb 4096-nál. A stack-en `[]byte` adattípusként tárolódik.
- bigint: min. 1 byte, max. 64 byte hosszúságú nem negatív egész szám. A stacken `[]byte` adattípusként tárolódik.
- address, 32 byte-os bigint. A stack-en `[32]byte` adattípusként tárolódik.

  Megjegyzés: Az Algorand címek 58 karakterből állnak. Ezek a karakterek 32-es alapú számrendszerben kódolják a számlához tartozó nyilvános kulcsot és egy 4 byte-os ellenőrző kódot. A TEAL az `addr` pszeudo utasítás után álló 58 karakteres Algorand címet 32 byte-os byte tömbbé alakítja át, amely a számla nyilvános kulcsának felel meg.
- method: 4 byte-os metódus kiválasztó érték, lásd ARC-4. A stack-en `[4]byte` adattípusként tárolódik.

Referenciák:

The Algorand Virtual Machine (AVM) and TEAL.

ARC-4: Application Binary Interface (ABI)

TEAL specification

go-algorand/data/transactions/logic/opcodes.goopcodes.go

6.3.1. Alulcsordulás és túlcsordulás kezelés

Ha egy `uint64` típusú számból kivonunk egy nála nagyobb `uint64` típusú számot, akkor az AVM bytekód végrehajtása hibajelzéssel azonnal megszakad.

Ha két `uint64` típusú szám közötti művelet végrehajtása során az eredmény $2^{64} - 1$-nél nagyobb, akkor hasonló módon a bytekód végrehajtása

hibajelzéssel azonnal megszakad.

1. TEAL példa: adattípusok

```
lipi@lipi-VirtualBox:~/n_beta1$ nano ex1.teal
#pragma version 10        // max. TEAL version on mainnet
int 123456789                // uint64
int 0x1234567812345678        // uint64
byte "Hello"                // [5]char
byte "world"
byte 0x123456789ABCDEF1234567 // bigint
addr RGW4Q2Q2SKKTGZJ2XG3GTHTTTTLXO7IJAXKEMOCAFOII6DTGSHNZTMGVXI        //
↪  address
==        // compare 11 byte bigint with 32 byte address, expected: 0
pop        // pop result from stack
==        // compare "Hello" with "world", expected: 0
pop        // pop result from stack
==        // compare 12345678 with 0x1234567812345678, expected: 0
CTRL/X
lipi@lipi-VirtualBox:~/n_beta1$ ./goal clerk compile ex1.teal
ex1.teal: 2V4EUCTZJY7KB663Z5EEYJPDRVR3KAGPQJK7PPC4DTOXGKOP3O4UFBYZ3Y
lipi@lipi-VirtualBox:~/n_beta1$ hexdump -C ex1.teal.tok
00000000  0a 81 95 9a ef 3a 81 f8  ac d1 91 81 cf 95 9a 12
↪  |.....:..........|
00000010  80 05 48 65 6c 6c 6f 80  05 77 6f 72 6c 64 80 0b
↪  |..Hello..world..|
00000020  12 34 56 78 9a bc de f1  23 45 67 80 20 89 ad c8  |.4Vx....#Eg.
↪  ...|
00000030  6a 1a 92 95 33 65 3a b9  b6 69 9e 73 9c d7 77 7d
↪  |j...3e:..i.s..w}|
00000040  09 05 d4 46 38 40 2b 90  8f 0e 66 91 db 12 48 12
↪  |...F8@+...f...H.|
00000050  48 12                                             |H.|
00000052
lipi@lipi-VirtualBox:~/n_beta1$ ./goal clerk compile -D ex1.teal.tok
#pragma version 10
pushint 123456789
pushint 1311768465173141112
pushbytes 0x48656c6c6f // "Hello"
pushbytes 0x776f726c64 // "world"
pushbytes 0x123456789abcdef1234567 // 0x123456789abcdef1234567
```

```
32 pushbytes
   ↪  0x89adc86a1a929533653ab9b6699e739cd7777d0905d44638402b908f0e6691db //
   ↪  addr RGW4Q2Q2SKKTGZJ2XG3GTHTTTTLXO7IJAXKEMOCAFOII6DTGSHNZTMGVXI
33 ==
34 pop
35 ==
36 pop
37 ==
38 lipi@lipi-VirtualBox:~/n_beta1$
```

Az 2. sorban lévő `#pragma version` a használni kívánt AVM változatot adja meg. Értéke jelenleg 1 és 11 között lehet. A 3. sorban egy `uint64` típusú egész szerepel decimális alakban megadva. A 4. sor azt mutatja, hogy a 0x prefix használatával az uint64 hexadecimális alakban is megadható. Az 5. és 6. sorban egy-egy byte vektor (string) lett megadva. A 7. sorban egy 11 byte-os bigint szerepel, végül a 8. sorban egy Algorand cím lett megadva az addr pszeudo-utasítás segítségével.

A TEAL program a `goal clerk compile` segítségével fordítható le bytekódra, lásd 15. sor. A lefordított program a `.tok` kiterjesztést kapja. A lefordított program hexadecimális dump-ja a 18..24. sorban látható.

Végül a 25. sorban a `-D` kapcsolóval disassembláltuk a bytekódot. Látható, hogy a TEAL fordító a 10-es AVM változatban a `pushint` és `pushbytes` AVM kódot használja a konstansok stack-en történő tárolásához. A 32. sorban látható, hogy az Algorand címből valóban 32 byte-os byte tömb keletkezett.

A bytekód végrehajtásakor a 33. sorban lévő == leemeli a stack tetején lévő két értéket, jelen esetben egy címet (32. sor) és egy bigint-et (31. sor), és összehasonlítja őket. Mivel nem egyenlők, egy 0 értéket tesz a stackre. Ezt az értéket a pop (34. sor) „eldobja”. A 35. sorban lévő == leemeli a stack tetején lévő két értéket, jelen esetben egy byte tömböt (30. sor) és egy másik byte tömböt (29. sor), és összehasonlítja őket. Mivel nem egyenlők, egy 0 értéket tesz a stackre. Ezt a pop (36. sor) „eldobja”. Végül a 37. sorban álló == leemeli a stack tetején lévő két értéket, jelen esetben egy uint64 típusú

értéket (28. sor) és egy másik uint64 típusú értéket (27. sor). Mivel nem egyenlők, egy 0 értéket tesz a stackre.

@todo Jó lenne, ha egy szimulátorban futtatva láthatnánk a bytekód működését. Ahhoz, hogy egy bytekódot meghívhassuk, be kell ágyazni a kódot egy okos aláírásba vagy egy okos szerződésbe. A következő rész ennek a módját ismerteti.

6.4. Algorand Smart Signatures

@todo

6.5. Példa az ASS használatára

@todo

6.6. Algorand alkalmazások

6.6.1. A bytekód maximális mérete

A bytekód okos aláírások (smart signatures) esetén max. 1000 byte hosszú lehet. A paraméterek megadása tovább csökkenti ezt a méretet. Okos szerződések (smart contracts) esetén az engedélyező (approval) és a feltétlen kiszállást végző (clear) programok együttes mérete alapéretelmezésben 2 Kbyte lehet. A méret 2 Kbytes-os lépésekben növelhető, és mind az engedélyező, mind a feltétlen kiszállást végző program max. 8 Kbyte méretű lehet.

6.6.2. A bytekód maximális végrehajtási költsége

A bytekód végrehajtása során az egyes műveletek költségének (cost) összege okos aláírások esetén max. 20 000 egység lehet. A legtöbb bytekód művelet költsége 1. Vannak olyan műveletek, melyeknek lényegesen nagyobb a költsége. Pl. az `sha256` művelet költsége 35 egység, az `ed25519verify` költsége 1900 egység. Egy adott művelet költsége az egyes műveletek leírásánál található meg: v10 műveleti kódok

Okos szerződések esetén az engedélyező és a feltétlen kiszállást végző programban lévő műveletek költségének összege tranzakciónként max. 700 egység lehet. Ezt a maximum értéket tranzakciós csoportok és belső tranzakciók használatával jelentősen megnövelhetjük. Pl. ha a tranzakciós csoportban 10 tranzakció van, akkor a maximum érték 10*700=7000 egységre módosul. Hasonló módon, minden egyes belső tranzakció 700-zal növeli a maximum értékét. Mivel a tranzakciós csoportban max. 16 tranzakció lehet, a belső tranzakciók száma pedig max. 256 lehet, a műveletek költsége maximum 700*(16+256)=190 400 lehet. Ilyenkor a tranzakció végrehajtási költsége (fee) is megnő, flat fee esetén max. (16+256)*0,001 Algo = 0,272 Algo lesz.[3]

Hivatkozás: Az opkódok végrehajtási költsége

Hivatkozás: Minimális tx díjként ne adj meg fixen 1000 microAlgó-t

6.7. Példa az Algorand alkalmazások használatára: "Hello, World"

Az Algorand "HelloWorld" alkalmazása a legegyszerűbben az `algokit init` parancs segítségével hozható létre. A szükséges lépések:

- Lépj be a `https://github.com` alatti számládra
- Hozz létre egy új repository-t, pl. `asc`
- Hozz létre a repository-ban egy README.MD fájlt
- Nyomd meg a "Code" gombot. Menj a "Codespaces" fülre. Indítsd el a "Codespaces"-t.
- A megjelenő VS Code böngésző ablakban installáld az algokit-et: `pipx install algokit`
- Indítsd el a localnet-et: `algokit localnet start`

[3] A tranzakciós díj a jövőben változhat. Érdemes az SDK konstansokat használni, pl. a JS SDK-ban ezt: `const minFee = algosdk.ALGORAND_MIN_TX_FEE`

- A `docker ps` paranccsal vizsgáld meg, hogy elindult és fut-e már az öt darab container
- A ports fülön tedd publikussá a 4001, 4002 és 8980 portokat
- Futtasd az `algokit init` parancsot a következőképpen:
  - válaszd ki: `Smart Contracts & DApp Frontend`
  - válaszd ki: `TypeScript`
  - könyvtárnévnek add meg: `ex`
  - válaszd ki a következő mintát: `Starter`
  - Hagyd meg az alapértéket a smart contract névnek: `HelloWorld`
  - Arra a kérdésre, hogy "Futtassa az 'algokit project bootstrap' parancsot?" `Yes`

```
@A-Maugli → /workspaces/asc (main) $ algokit init
? Which of these options best describes the project you want to build?
↪  Smart Contracts & DApp Frontend
? Which language would you like to use for the smart contract? TypeScript
? Name of project / directory to create the project in: ex
Starting template copy and render at /workspaces/asc/ex...
   Name of the template preset to use.
   Starter - for a simpler starting point ideal for prototyping
   Name of the default smart contract app.
   HelloWorld
==== Checking compatibility with the cli ===
==== 1/4 - Initializing base template ====
Starting template copy and render at /workspaces/asc/ex...
Template render complete!
   Project initialized at `ex`! For template specific next steps, consult
   ↪  the documentation of your selected template
Your selected template comes from:
→  https://github.com/algorandfoundation/algokit-base-template
Your template includes a README.md file, you might want to review that as
↪  a next step.
==== 2/4 - Initializing frontend template ====
```

```
Starting template copy and render at /workspaces/asc/ex/projects/ex-frontend...
Template render complete!
   Project initialized at `ex-frontend`! For template specific next steps, consult the documentation of your selected template
Your selected template comes from:
→ https://github.com/algorandfoundation/algokit-react-frontend-template
Your template includes a README.md file, you might want to review that as a next step.
==== 3/4 - Initializing backend template ====
Starting template copy and render at /workspaces/asc/ex/projects/ex-contracts...
Template render complete!
   Project initialized at `ex-contracts`! For template specific next steps, consult the documentation of your selected template
Your selected template comes from:
→ https://github.com/algorand-devrel/tealscript-algokit-template
Your template includes a README.md file, you might want to review that as a next step.
==== 4/4 - Finalizing setup ====
Template render complete!
? Do you want to run `algokit project bootstrap` for this new project? This will install and configure dependencies allowing it to be run immediately. Yes
Installing npm dependencies
npm:
npm: added 568 packages, and audited 569 packages in 49s
npm:
npm: 137 packages are looking for funding
npm: run `npm fund` for details
npm:
npm: found 0 vulnerabilities
npm: npm notice
npm: npm notice New minor version of npm available! 10.5.0 -> 10.7.0
npm: npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.7.0>
npm: npm notice Run `npm install -g npm@10.7.0` to update!
npm: npm notice
Copying /workspaces/asc/ex/projects/ex-frontend/.env.template to /workspaces/asc/ex/projects/ex-frontend/.env and prompting for empty values
```

```
Installing npm dependencies
npm: npm WARN deprecated @walletconnect/types@1.8.0: WalletConnect's v1 SDKs are now deprecated. Please upgrade to a v2 SDK. For details see: https://docs.walletconnect.com/
npm: npm WARN deprecated @walletconnect/client@1.8.0: WalletConnect's v1 SDKs are now deprecated. Please upgrade to a v2 SDK. For details see: https://docs.walletconnect.com/
npm: npm WARN deprecated @motionone/vue@10.16.4: Motion One for Vue is deprecated. Use Oku Motion instead https://oku-ui.com/motion
npm:
npm: added 367 packages, and audited 369 packages in 1m
npm:
npm: 41 packages are looking for funding
npm: run `npm fund` for details
npm:
npm: found 0 vulnerabilities
   Project initialized at `ex`! For template specific next steps, consult the documentation of your selected template
Your selected template comes from:
→  https://github.com/algorandfoundation/algokit-fullstack-template
Directory is already under git revision control, skipping git setup
VSCode configuration detected in project directory, and 'code' command is available on path, attempting to launch VSCode
@A-Maugli → /workspaces/asc (main) $
```

Az `algokit init` parancs a következő könyvtárszerkezetet hozza létre:

```
asc
  + ex
      + projects
          + ex-contracts
              + __test__
                  HelloWorld.test.ts
              + contracts
                  + clients
                  + artifacts
                  Helloworld.algo.ts
              package.json
          + ex-frontend
              + src
```

```
                    + assets
                    + components
                    + contracts
                    + interfaces
                    + styles
                    + utils
                        App.tsx
                        Home.tsx
                        main.tsx
                .env
                .env.template
                package.json
```

6.7.1. A backend

Az okos szerződés

Az `algokit init` parancs létrehozza egy Typescript-ben megírt Algorand mintaszerződés kódját az `ex-contracts/contracts/HelloWorld.algo.ts` fájlban:

```
import { Contract } from '@algorandfoundation/tealscript';

export class HelloWorld extends Contract {
  /**
   * Calculates the sum of two numbers
   *
   * @param a
   * @param b
   * @returns The sum of a and b
   */
  private getSum(a: uint64, b: uint64): uint64 {
    return a + b;
  }

  /**
   * Calculates the difference between two numbers
   *
   * @param a
   * @param b
```

```
20      * @returns The difference between a and b.
21      */
22     private getDifference(a: uint64, b: uint64): uint64 {
23       return a >= b ? a - b : b - a;
24     }
25
26     /**
27      * A method that takes two numbers and does either addition or
        ↪ subtraction
28      *
29      * @param a The first uint64
30      * @param b The second uint64
31      * @param operation The operation to perform. Can be either 'sum' or
        ↪ 'difference'
32      *
33      * @returns The result of the operation
34      */
35     doMath(a: uint64, b: uint64, operation: string): uint64 {
36       let result: uint64;
37
38       if (operation === 'sum') {
39         result = this.getSum(a, b);
40       } else if (operation === 'difference') {
41         result = this.getDifference(a, b);
42       } else throw Error('Invalid operation');
43
44       return result;
45     }
46
47     /**
48      * A demonstration method used in the AlgoKit fullstack template.
49      * Greets the user by name.
50      *
51      * @param name The name of the user to greet.
52      * @returns A greeting message to the user.
53      */
54     hello(name: string): string {
55       return 'Hello, ' + name;
56     }
57   }
```

Az szerződés két meghívható metódusa a `doMath` és a `hello`. A szerződés a következőképpen fordítható le TEAL kódra:

```
cd ex/projects/ex-contracts

npm run build
```

A fordítás során keletkező fájlok

Az eredmény fájlok a `contracts/artifacts` könyvtárban képződnek:

- `HelloWorld.approval.teal`, az okos szerződést létrehozó TEAL kód + az okos szerződés metódusai
- `HelloWorld.arc4.json`, az alkalmazási hívási felület leíró JSON fájl (ABI fájl, Application Binary Interface)
- `HelloWorld.arc32.json`, az alkalmazást leíró JSON fájl, tartalmazza az ARC4 ABI leírást is
- `HelloWorld.arc56.json`, az alkalmazást még részletesebben leíró JSON file. Tartalmazza pl. a source map-et JSON formátumbam.
- `HelloWorld.clear.teal`, az opt-out feltétel nélküli végrehatásához tartozó TEAL kód
- `HelloWorld.src_map.json`, a Typscript forrássorok, a TEAL forrássorok és az AVM PC-k összerendelését végző map fájl

Az Algorand okosszerződést leíró TypeScript file a `contracts/clients` könyvtárban képződik:

- `HelloWorldClient.ts`, az alkalmazás hívását segítő TypeScript wrapper

A `HelloWorldClient.ts` TypeScript Wrapper feladata

A TypeScript Wrapper magas szintű interfészt biztosít egy Algorand okosszerződéshez. Az automatikusan generált kód elősegíti, hogy az okosszerződés műveleteit könnyen használhassuk a TypeScript alkalmazásokban.

A TypeScript Wrapper automatikusan elkészíti az okosszerződés híváshoz szükséges tranzakciókat és segítséget nyűjt a komplex adatstruktúrák felhasználó-barát kezelésében:

- absztrakciót nyújt: Nem szükséges mélyrehatóan ismerni a TEAL-t, a kliens magasabb szintű interfészt biztosít.
- automatizálja a tranzakciókat: automatikusan kezeli a tranzakciók összeállítását, az aláírásokat és a hálózati kommunikációt.
- típusbiztonságot biztosít: a generált kód TypeScript típusokat használ, ami segíti a hibák elkerülését a fejlesztés során.

A TypeScript Wrapper részei:

- Okosszerződés specifikáció: Az APP_SPEC objektum tartalmazza az okosszerződés specifikációját, beleértve a szerződésben elérhető különböző metódusokat (doMath, hello, createApplication) és azok paramétereit, viselkedését. Ez az információ alapvető a kliens működéséhez, mivel meghatározza, hogy milyen műveletek végezhetők az okosszerződésen.
- HelloWorldClient osztály: a szerződés metódusainak hívásait magasabb szinten kezelő osztály. Az okosszerződés különböző funkcióit egyszerű TypeScript függvényhívásként érhetjük el, mint például a doMath() vagy a hello() függvények.
- HelloWorldCallFactory osztály: ez az osztály előre definiált függvényeket tartalmaz, amelyek tranzakciókat hoznak létre az okosszerződés meghívásához. Például a doMath() metódus egy tranzakciót készít, amely két szám összeadását vagy kivonását végzi el a szerződés segítségével.
- OnCompletion típusok és állapot kezelés: az alkalmazás hívásához szükséges típusok definiálása, valamint a szerződés (globális) állapotában tárolt adatok kezelésére szolgáló IntegerState és BinaryState típusok.

- deploy: A deploy() metódus segítségével az okosszerződést idempotens módon telepíthetjük. Az idempotens viselkedés azt jelenti, hogy ugyanazt az eredményt kapjuk, ha a telepítést többször hajtjuk végre.

A TypeScript Wrapper egyszerűsíti az Algorand okosszerződések meghívását. Elrejti az alacsony szintű működést (tranzakciók, tranzakció csoportok, aláírások kezelése), és lehetővé teszi decentralizált alkalmazások egyszerű és gyors fejlesztését.

A szerződés tesztelése

A szerződés tesztelése a Jest keretrendszerre támaszkodik. A jest és a ts-jest npm modulok installálása után a TypeScript-ben megírt tesztek közvetlenül a Jest keretrendszerben végrehajthatók. A Jest részletesebb leírása a Nathan Rozentals *Mastering TypeScript* című könyvének 10. fejezetében található: *Test-Driven Development.*

A Jest tesztek az `ex-contracts/__test__/Helloworld.test.ts` fájlban találhatók:

```
1  import { describe, test, expect, beforeAll, beforeEach } from
   ↪  '@jest/globals';
2  import { algorandFixture } from
   ↪  '@algorandfoundation/algokit-utils/testing';
3  import * as algokit from '@algorandfoundation/algokit-utils';
4  import { HelloWorldClient } from '../contracts/clients/HelloWorldClient';
5
6  const fixture = algorandFixture();
7  algokit.Config.configure({ populateAppCallResources: true });
8
9  let appClient: HelloWorldClient;
10
11 describe('HelloWorld', () => {
12   beforeEach(fixture.beforeEach);
13
14   beforeAll(async () => {
15     await fixture.beforeEach();
16     const { testAccount } = fixture.context;
17     const { algorand } = fixture;
```

```
18
19      appClient = new HelloWorldClient(
20        {
21          sender: testAccount,
22          resolveBy: 'id',
23          id: 0,
24        },
25        algorand.client.algod
26      );
27
28      await appClient.create.createApplication({});
29    });
30
31    test('sum', async () => {
32      const a = 13;
33      const b = 37;
34      const sum = await appClient.doMath({ a, b, operation: 'sum' });
35      expect(sum.return?.valueOf()).toBe(BigInt(a + b));
36    });
37
38    test('difference', async () => {
39      const a = 13;
40      const b = 37;
41      const diff = await appClient.doMath({ a, b, operation: 'difference'
        ↪  });
42      expect(diff.return?.valueOf()).toBe(BigInt(a >= b ? a - b : b - a));
43    });
44
45    test('hello', async () => {
46      const diff = await appClient.hello({ name: 'world!' });
47      expect(diff.return?.valueOf()).toBe('Hello, world!');
48    });
49  });
```

Magyarázata:

- az 1. sorban beimportált Jest globálisok feladata:
  - describe, a Jest teszt csoportok leírására szolgál
  - test, az egyes Jest tesztek leírására szolgál

  - expect, a tesztek eredményének vizsgálatára szolgál
  - beforeAll, az egyes teszt csoportok előtt futatott teszt setup
  - beforeEach, az egyes tesztek előtt futtatott teszt setup
- a 2. sorban importált algorandFixture állítja elő a 6. sorban a legfontosabb Algorand paramétereket, pl. a 17. sorban található algorandClient típusú algorand változót, amely az algorand.client.algod property-ben tartalmazza az algod (algorand démon) hivatkozását.
- a 11. sorban történik a teszt csoport megadása
- a 12. sor írja elő, hogy minden egyes teszt előtt le kell futtatni a fixture.beforeEach függvényt
- a 14. sor írja elő a teszt csoportra vonatkozó, egyszer futtatandó teszt beállításokat:
  - 15. sor: vár az Algorand fixture-re
  - 16. sor: kiveszi a testAccount-ot a fixture-ből
  - 17. sor: kiveszi az algorand: algorandClient property-t a fixture-ből
  - 19..26. sor: létrehoz egy új appClient-et
  - 28. sor: létrehoz egy új app-ot, azaz okosszerződést a blokkláncon. Megjegyzés: Az app, application és smart contract (app, alkalmazás, okosszerződés) egymás szinonimái.
- 31..36. sor: egy teszt definiálása
  - 34. sor: a blokkláncon élő szerződés doMath metódusának meghívása, az appClient-en keresztül. Az appClient intézi az app-ot hívó tranzakció összeállítását, a tranzakció aláírását, elküldését, és az eremény struktúrált formában történő előállítását.
  - 35. sor: annak a vizsgálata, hogy az eredmény egyezik-e $a+b$-vel.

- 38..43. sor: egy újabb teszt definiálása, amely az app doMath() metódusát hívja, majd ellenőrzi, hogy az app metódusa a megfelelő eredmény szolgáltatta-e
- 45..48. sor: egy újabb teszt definiálása, amely az app hello() metódusát hívja, majd ellenőrzi az app-pal végeztetett string összefűzés eredményét

```
npm run test
```

paranccsal futtathatók. A futtatás eredménye:

```
 PASS  __test__/HelloWorld.test.ts (45.45 s)
  HelloWorld
    ✓ sum (9982 ms)
    ✓ difference (10250 ms)
    ✓ hello (10407 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        45.863 s
Ran all test suites.
```

Ha a Jest TypeScript kód debug-olásásra van szükség, akkor a VS Code "Run and Debug" gombjának megnyomása után indítsunk el egy debug terminált. A debug terminálban a `npm run test` paranccsal indítsuk el a futást. A megadott töréspontokon megáll a futás, és lépésenként folytathatjuk a végrehajtást.

6.7.2. A frontend

Az `ex-frontend` egy React frontend-et tartalmaz. Menjünk a `ex-frontend` könyvtárba, és futtassuk a kliens generálást és a packer programot:

```
cd ex/projects/ex-frontend
```

```
npm run dev
```

A `package.json` fájlból látszik, hogy mi megy ekkor végbe:

```
cat package.json
...
"scripts": {
    "generate:app-clients": "algokit project link --all",
    "dev": "npm run generate:app-clients && vite",
    "build": "npm run generate:app-clients && tsc && vite build",
    "preview": "vite preview"
  },
...
```

Az `algokit project link --all` parancs határásra „legenerálódik” az összes szerződéshez tartozó typed client, a `ex-frontend/src/contract` könyvtárba.

A `vite` parancs elindítja a vite packer-t dev módban (gyors, szinte azonnali fordítás), és egy hyperlink-en keresztül elérhetővé teszi e React alkalmazást.

Ha a `codespaces`-t használjuk, akkor módosítani kell a `.env` fájlt is, a következőképpen: a Ports fülről ki kell másolni a 4001 portnál szereplő URL-t, és be kell írni a `localhost` helyébe. A 4001 portot pedig 443-ra kell átírni:

régi:

```
VITE_ALGOD_SERVER=http://localhost
VITE_ALGOD_PORT=4001
```

új:

```
VITE_ALGOD_SERVER=https://(a ports fülről átmásolt érték)
VITE_ALGOD_PORT=443
```

Hasonló módon, a `VITE_KMD_SERVER`, `VITE_KMD_PORT` és a `VITE_INDEXER_SERVER`, `VITE_INDEXER_PORT` soroknál is módosítani kell a `.env` fájlt.

Ezután a következő parancsokat kell kiadnunk az `ex-frontend` könyvtárban:

```
# létrehozza a vite.env fájlt és a .env fájlt, és betölti a node_modules
↪ könyvtárba a node.js modulokat
algokit project bootstrap all
# lokális Algorand hálózati csomópont indítása
algokit localnet start
# Az öt db container vizsgálata
docker ps
# többször érdemes kiadni a docker ps parancsot, mert jelenleg a conduit
↪ hajlamos az elszállásra. Ilyenkor próbálkozni lehet a
algokit localnet reset
# paranccsal.

# A 4001, 4002, és 8980 portok public-ká tétele a ports fülön

# Végül
npm run dev
# és CTRL + kattintás segítségével elindítani a böngészőben a React
↪ alkalmazást.
  → Local:   http://localhost:5173/
# A böngészőben F12-vel fejlesztői környezet indítása.
```

6.7.3. Az ABI

Az ABI (Application Binary Interface) azt írja le, hogy milyen adattípusok használhatók egy okos szerződés valamelyik metódusának meghívásakor. A használható adattípusok definiálása lehetővé teszi, hogy az okos szerződéseket erősen típusos (strongly typed) paraméterekkel lehessen meghívni.

Eredetileg az Ethereum-ban definiálták először az ABI-t.

Az Algorandban az arc-0004 leírás definiálja az ABI-t, azaz az Algorand fejlesztői környezetben használható adattípusokat. A read-only paraméterek definícióját az arc-0022, a napló eseményekét az arc-0028 tartalmazza.

Az ABI réteg és az TEAL (AVM) közötti típuskonverziót a különféle fejlesztő rendszerek – TypeScript, PuyaPy – automatikusan elvégzik. Ennek megfelelően az ABI-t használó okos szerződések hosszabbak, bonyolultab-

bak a szükséges adatkonverziók miatt. Például a `string` ABI adattípus esetén az első két byte-on a string hossza tárolódik, ezt követi maga a string. Az Algorand `[]byte` adattípusa viszont implicit módon tartalmazza a string hosszúságát, a stack-re feltett adatbyte-ok száma révén.

6.7.4. "HelloWorld" mintaprogram ABI használata nélkül

Egy "HelloWorld" mintaprogram ABI használata nélkül, PuyaPy Pythonban a következő:

contract.py

```
1  from algopy import Contract, Txn, log
2
3  class HelloWorldContract(Contract):
4      def approval_program(self) -> bool:
5          name = Txn.application_args(0)
6          log(b"Hello, " + name)
7          return True
8
9      def clear_state_program(self) -> bool:
10         return True
```

A Python kód magyarázata: A 3. sorban a Contract osztályból származtatjuk a HelloWorldContract szerződést. Az `approval_program` metódus a tranzakció végrehajtását engedélyezi vagy tiltja, 1 vagy 0 visszatérési érték esetén. A `clear_state_program` metódus a szerződésbe való benevezés esetén feltétlen kiszállást tesz lehetővé. Az 5. sor beolvassa az alkalmazás meghívására használt tranzakció első paraméterét (a paraméterek számozása a TEAL-ben 0-tól kezdődik), a 6. sor pedig naplózza a `"Hello, "+name` eredményét.

A PuyaPy compiler hívásakor a két függvény külön TEAL állományba kerül. A fenti Python kód lefordításával előálló engedélyező (approval) TEAL program a következő lesz:

HelloWorldContract.approval.teal

```
#pragma version 10

examples.hello_world.contract.HelloWorldContract.approval_program:
    byte "Hello, "
    txna ApplicationArgs 0
    concat
    log
    int 1
    return
```

Magyarázat: A TEAL program a 4. sorban a `"Hello, "` stringet teszi fel az AVM stack-re, az 5. sorban pedig az alkalmazás első paraméterét. Figyelem, az paraméterek indexelése 0-tól kezdődik. A 6. sorban összefűzi a két stringet, a 7. sorban pedig az eredményül kapott stringet leveszi a stack-ről, és naplóbejegyzésként rögzíti a blokkláncon. Végül a 8. sorban az 1 értéket teszi fel a stack-re. Ez lesz az engedélyező program visszatérési értéke, az 1 érték azt jelenti, hogy engedélyezzük a tranzakció végrehajtását.

6.7.5. "HelloWorld" mintaprogram ABI használatával

Egy "HelloWorld" mintaprogram ABI használatával, PuyaPy Pythonban megírva a következő:

contract.py

```
from algopy import ARC4Contract, String, arc4

class HelloWorldContract(ARC4Contract):
    @arc4.abimethod
    def hello(self, name: String) -> String:
        return "Hello, " + name
```

Az engedélyező (approval) TEAL program a következő:

HelloWorldContract.approval.teal

```
#pragma version 10
```

```
examples.hello_world_arc4.contract.HelloWorldContract.approval_program:
    txn NumAppArgs
    bz main_bare_routing@5
    method "hello(string)string"
    txna ApplicationArgs 0
    match main_hello_route@2
    err // reject transaction

main_hello_route@2:
    txn OnCompletion
    !
    assert // OnCompletion is NoOp
    txn ApplicationID
    assert // is not creating
    txna ApplicationArgs 1
    extract 2 0
    callsub hello
    dup
    len
    itob
    extract 6 2
    swap
    concat
    byte 0x151f7c75
    swap
    concat
    log
    int 1
    return

main_bare_routing@5:
    txn OnCompletion
    !
    assert // reject transaction
    txn ApplicationID
    !
    assert // is creating
    int 1
    return

```

```
43
44  // examples.hello_world_arc4.contract.HelloWorldContract.hello(name:
    ↪  bytes) -> bytes:
45  hello:
46      proto 1 1
47      byte "Hello, "
48      frame_dig -1
49      concat
50      retsub
```

A program magyarázata: A TEAL program 4. sora a paraméterek számát teszi fel az AVM stack-re. Az 5. sor a `main_bare_routing@5` címkére ugrik, ha nem adtak meg az alkalmazás hívásakor egyetlen paramétert sem, lásd 33. sor. A tranzakció OnCompletion mezője a következő értékeket veheti fel:

- NoOp (0)
  - ha az az AppId nulla, akkor most jött létre az alkalmazás.
  - ha az AppId nem nulla, akkor az alkalmazás hívása.
- OptIn (1)
  - ha az az AppId nulla, akkor érvénytelen
  - ha az AppId nem nulla, akkor benevezés (opt-in) az alkalmazásba.
- CloseOut (2)
  - ha az az AppId nulla, akkor érvénytelen
  - ha az AppId nem nulla, akkor kiszállás (opt-out) az alkalmazásból.
- ClearState (3) - nem fordulhat elő az approval programban
- UpdateApplication (4)
  - ha az az AppId nulla, akkor érvénytelen

  - ha az AppId nem nulla, akkor az alkalmazás módosítása

- DeleteApplication (5)
  - ha az az AppId nulla, akkor érvénytelen
  - ha az AppId nem nulla, akkor az alkalmazás törlése

A 34. sor az OnCompletion értékét teszi a stack-re. A 35. sor egy negálás, vagyis 0 értéket tesz a stack-re, ha a stack-en lévő érték nem nulla, és 1 értéket tesz a stack-re, ha a stack-en lévő érték nulla. A 36. sorban lévő `assert` azonnal megszakítja a program futását, ha a stack tetején lévő érték nulla. Vagyis: a program végrehajtása megszakad, ha az OnCompletion értéke nem NoOp-pal egyenlő. Hasonló módon, ha az ApplicationID értéke nem nulla, akkor a program futása azonnal megszakad (39. sor). Vagyis a két `assert` biztosítja, hogy most jött létre az alkalmazás, és NoOp a kód értéke. A 40. sorban lévő 1 érték stack-re helyezésével engedélyezzük az alkalmazás létrehozását. Minden más esetben – vagyis ha a completion kód nem NoOp vagy az alkalmazás nem éppen most jön létre – hibával megszakítjuk a hívást.

A 6. sorban a `method "hello(string)string"` a hello metódus 4 byte-os HASH lenyomatát hozza létre és helyezi a stack-re, az ABI szabványnak megfelelően. A 7. sor az alkalmazás 1. paraméterét teszi a stack-re. Ha a két érték egyezik, akkor a `main_hello_route@2` címkénél folytatódik a végrehajtás, egyébként pedig a végrehajtás hibajelzéssel azonnal megszakad.

A 12..16. sor ellenőrzi, hogy az Oncompletion == NoOp és az ApplicationID != 0 feltételek teljesülnek-e. Ha nem, akkor a végrehajtás hibával megszakad.

A 17. sorban feltesszük az AVM stackre a második hívási paramétert, ami a `name` ABI string. A 18. sorban levágjuk róla a string hosszát kódoló 2 byte-ot, ezzel megkapjuk az ABI string típusból az AVM elemi stringet. A 19. sorban meghívjuk a `hello` szubrutint, amelyik a `"Hello, "` stringet összefűzi a call frame-en lévő AVM stringgel.

Innentől az eredmény ABI szabványnak megfelelő visszakódolása történik, vagyis az AVM eredménystringből ABI eredménystringet állít elő az okos szerződés, a következőképpen:

A 20. sor megduplázza az AVM eredmény stringet. A 21. sor megállapítja az eredmény string hosszát. A 22. sorban lévő `itob` parancs az uint64 típusú 8 byte-os egész számból, ami a string hossza, egy 8 byte-os stringet készít. A 23. sor ennek az utolsó 2 byte-ját szedi ki. A 24. sorban lévő `swap` megcseréli a stack 2 fölső elemét, vagyis a stack tetején az AVM eredmény string lesz, eggyel beljebb a string hossza, 2 byte-on ábrázolva. Pl. ha az eredmény hossza 17, akkor ez a 2 byte a 0x31 0x37 lesz. A 25. sor összefűzi a 2 byte-os hosszt és az eredmény stringet.

Ezután az eredmény előállítása történik, az arc-0028-nak megfelelően. A 4 byte-os prefix, amit az eredmény elé kell tenni, a következőképpen állítható elő:

```
1  const { sha512_256 } = require("js-sha512");
2  sig = "return";
3  //sig = "Swapped(uint64,uint64)";
4  hash = sha512_256(sig);
5  prefix = hash.slice(0, 8);
6  console.log("Prefix: ", prefix);
```

Egészen ellentétes az intuícióval, hogy az event log-hoz a 4 byte-os prefixet nem a "hello(string)" hash-elésével, hanem ehelyett egyszerűen a "return" hash-elésével kell előállítani. A 26. sor az eredmény hash-ének első 4 byte-ját tartalmazza, 0x151f7c75. A 27. és 28. sor ezt a hash értéket elölről hozzáfűzi az eddigi stringhez, ami eddig 2 byte-on a hosszt tartalmazza, majd magát az AVM eredménystringet. A 29. sor ezt az eredményt naplózza a blokkláncban. Végül a 30. sor az 1 érték stack-re helyezésével engedélyezi a tranzakciót, és a 31. sorban véget ér ennek az ágnak a végrehajtása.

Látható, hogy mennyivel bonyolultabb lett a "HelloWorld" szerződés az ABI használatával, mint anélkül.

Hivatkozások:

Contract ABI Specification - Solidity 0.8.26 documentation

ABI details - Algorand Developer Portal

arc-0004, Algorand Transaction Calling Conventions

arc-0022, Add ‘read-only‘ annotation to methods

arc-0028, Algorand Event Log Spec

arc-0032, Application Specification

TEAL V10 opcodes

Könyvek:

Nathan Rozentals *Mastering TypeScript*, 4. kiadás

6.7.6. A TEAL kód debug-olása

A TEAL kód debug-olása a VS Code-on belül lehetséges. Ehhez szükség van arra, hogy installáljuk az “AlgoKit AVM Debugger” bővítést a VS Code-on belül.

Az AVM a byte kód végrehajtásakor képes a végrehajtási lépések szimulációjára. Az ennek során keletkező információk az `algokit-utils-ts` segítségével egy nyomkövetési fájlba, vagy másképpen trace file-ba irathatók ki. Az “AlgoKit AVM Debugger” ezeknek a `.trace.avm.json` kiterjesztésű trace file-oknak a megjelenítését végzi.

A nyomkövetési fájlok előállítása érdekében a következő sorokat kell hozzáadni a `__test__/HelloWorld.test.ts` fájlhoz:[4]

```
1 import { Config } from '@algorandfoundation/algokit-utils';
2 import { registerDebugEventHandlers } from
  ↪  '@algorandfoundation/algokit-utils-debug';
3
```

[4] Feltéve, hogy az algokit-utils V8 változatát használjuk.

```
Config.configure({ debug: true, traceAll: true });
registerDebugEventHandlers();
```

A `__test__/HelloWorld.test.ts` fájl eleje a sorok hozzáadása után a következő lesz:

```
import { describe, test, expect, beforeAll, beforeEach } from
↪  '@jest/globals';
import { algorandFixture } from
↪  '@algorandfoundation/algokit-utils/testing';
import { Config } from '@algorandfoundation/algokit-utils';
import { registerDebugEventHandlers } from
↪  '@algorandfoundation/algokit-utils-debug';
import { HelloWorldClient, HelloWorldFactory } from
↪  '../contracts/clients/HelloWorldClient';

const fixture = algorandFixture();
Config.configure({ populateAppCallResources: true });
Config.configure({ debug: true, traceAll: true });
registerDebugEventHandlers();

let appClient: HelloWorldClient;
...
```

Installálni kell az `algokit-utils-debug` modult is:

```
$ npm i @algorandfoundation/algokit-utils-debug
```

Ezután újból le kell futtatni a Jest teszteket:

```
$ npm run test
```

Ekkor a `debug-traces` könyvtárban minden egyes végrehajtott tranzakcióhoz egy `.trace.avm.json` fájl is létrejön.

Ha szeretnénk végiglépkedni a tranzakcióhoz tartozó AVM utasításokon, akkor:

- a `VS Code` file explorerrel válasszunk ki egy trace fájlt,

- az ablak jobb felső sarkában kattintsunk a “Debug” ikonra,
- map fájlként a `.algokit/sources/HelloWorld/approval.teal.map` fájlt adjuk meg,
- a képernyő tetején, középen megjelenő Debug ikonok közül válasszuk a “Step Into” ikont, és kattintsunk rá. Két kattintás után megjelenik a TEAL forrásprogram.
- kattintsunk a bal oldali ikon oszlopban lévő “Debug” ikonra, ekkor megjelenik a debug panel.
- a képernyő tetején lévő “Step Into” ikonra történő ismételt kattintással lépkedjünk végig a programon.

A TEAL kódban beállíthatók töréspontok is, és a végrehajtás eddig a pontig a “Run” megnyomásával lehetséges.

A bal oldali debug panelben látható az AVM-hez tartozó PC, a stack, a scratch regiszterek és az alkalmazás állapota (app state (global, local, box)).

Nem látjuk a call stack-et, vagy a frame pointer értékét, de látjuk a stack-en lévő call frame-eket.

Nagyon tanulságos, ha végiglépkledünk a HelloWorld program mind a négy app hívásához tartozó trace file-okon, és megfigyeljük, hogyan változik a stack az egyes utasítások végrehajtása során. Hamarosan érteni fogjuk, hogy az egyes TEAL utasítások hogyan működnek, és hogyan valósítják meg a magas szintű nyelven (TealScript vagy PuyaPy) leírt app működését.

Hivatkozások:

algokit-avm-vscode-debugger

algokit-utils-ts README.md

algokit-utils-ts debugging.md

algokit-utils-ts writeAVMDebugTrace.ts

AVM v10 opcodes

AVM v8 new features

7. Esettanulmány: tulajdonrész opciós vételi jog

7.1. A feladat leírása

A „Bizalmi Kör” nevű társaság tíz vevő számára értékesíteni szeretné a tulajdonrészének 10*0,1%-át. Ehhez a társaság egy blokkláncon 10 db zsetont bocsát ki, és egy okos szerződés segítségével értékesíti a zsetonokat. A zsetonok a kibocsátást követő 30 nap során vásárolhatók meg. A zsetonok mindegyike egy-egy opciós vételi jogot testesít meg. Egy vevő csak egy zsetont vásárolhat meg.

Ahhoz, hogy a zseton tulajdonosa élhessen az opciós vételi jogával, meg kell jelennie a Bizalmi Kör titkárságán, és ott be kell mutatnia

- a pénztárcájában lévő, megvásárolt zsetont,
- a személyi igazolványát,
- valamint egy igazolást arról, hogy a Bizalmi Kör bankszámlájára átutalta a 0,1% tulajdonrész vételárát.

Ekkor megkapja a tulajdonrész-szerzést igazoló hivatalos iratokat, és bevonásra kerül tőle a zseton.

Amennyiben a zseton tulajdonosa a zseton megvásárlástól számított 4 napon belül nem él az opciós vételi jogával, akkor a zseton automatikusan visszaszáll az okos szerződés tulajdonába, és ismét eladható.

Megjegyzés: a folyamat jelentősen egyszerűsödne, ha a zseton nem csupán opciós vételi jogot biztosítana, hanem magát a tulajdonrészt testesítené meg. De a jelenlegi jogi szabályozás miatt ekkor is szükség lenne a személyes adatok megadására.

7.2. A feladat elvi megoldása

A „Bizalmi Kör” tulajdonrész opciós vételi jogot megtestesítő zsetonok az Algorand blokkláncon egy ASA, azaz Algorand Standard Asset formájában

állíthatók elő. Az ASA paraméterei:

- Rövid név: BKTOVJ
- Hosszú név: Bizalmi Kör Zseton
- Darabszám: 10 db
- Web lap címe: https://algorand.hu/bk/bktovj.html

Az ASA egy okos szerződéssel értékesíthető. Az okos szerződés feladatai:

- ASA eladása, "first come, first served" módon, fix áron. Az okos szerződés biztosítja, hogy a zseton vételárának kiegyenlítése után a zseton a vevő pénztárcájába kerüljön. A zseton birtoklása révén a vevő opciós vételi joghoz jut.
- „lejárt" ASA automatikus visszavétele: amennyiben a zseton tulajdonosa adott időn belül nem él az opciós vételi jogával, a zseton az okos szerződés számlájára kerül vissza. Ehhez a szerződés "clawback" műveletet hajt végre, és eggyel növeli az eladható ASÁ-k számát.
- opciós vételi jog érvényesítése: ha a vevő a „Bizalmi Kör" titkárságán bemutatta a zsetont és az egyéb szükséges igazolásokat, akkor a „Bizalmi Kör" ügyintézője „bevonja" a zsetont. Ehhez a szerződés "clawback" műveletet hajt végre, de nem növeli az eladható ASÁ-k számát.

Számla címek:

- az okos szerződés létrehozójának Algorand címe
- az okos szerződéshez tartozó Algorand cím
- a vevő számlájának Algorand címe

Az ASA vételt a vevő kezdeményezi oly módon, hogy egy web lapon keresztül meghívja az okos szerződés ASA eladást végző metódusát. Ekkor egy WalletConnect felület összekapcsolja a vevő pénztárcáját az okos szerződéssel, és a vevő az okos szerződés által előállított tranzakció aláírásával

képes megvenni a zsetont.

Ahhoz, hogy egyszerű legyen a “clawback” feltétel vizsgálata, az okos szerződés a zseton eladásakor „befagyasztja” a zsetont a vevő pénztárcájába. Így elég az aktuális idő és a vétel időpontjának különbségét figyelni, és ha az nagyobb egy adott időnél, jelen esetben 4 napnál, akkor végrehajtható a “clawback”.

Megjegyzés: a szerződés nem tud „magától” tranzakciókat kezdeményezni. A “clawback” funkció megvalósításához egy ütemezett folyamatnak naponta ki kell gyűjtenie a lejárt érméket birtokló számlákat, és meg kell hívnia a szerződés “clawback” funkcióját.

7.3. Az okos szerződés magyarázata

A feladatot megoldó Algorand alkalmazás vagy Algorand okos szerződés a BizKor.algo.ts fájlban található. Az okos szerződés TypeScript-ben készült, és a TealScript fordítóprogram segítségével fordítható le TEAL kóddá.

Hivatkozások:

- TEALScript forráskód
- TEALScript dokumentáció
- TEALScript példa programok

7.3.1. Az okos szerződés osztályának definiálása

```
1  import { Contract } from '@algorandfoundation/tealscript';
2  // verzió történet ...
```

```
31 // eslint-disable-next-line no-unused-vars
32 class BizKor extends Contract {
33 // BizKor állapotváltozók és metódusok...
34 }
```

A 1. sorban történik a `Contract` osztály beimportálása. A 32. sorban ebből

származtatjuk a `BizKor` okos szerződés osztályt.

7.3.2. A globális állapotváltozók definiálása

```
31 // eslint-disable-next-line no-unused-vars
32 class BizKor extends Contract {
33   appVersion = GlobalStateKey<string>({ key: 'apv' });
34
35   appCreatorAddress = GlobalStateKey<Address>({ key: 'apca' });
36
37   assetAmountInitial = GlobalStateKey<uint64>({ key: 'asa_total' });
38
39   assetAmount = GlobalStateKey<uint64>({ key: 'asa_amt' });
40
41   assetPrice = GlobalStateKey<uint64>({ key: 'asa_price' });
42
43   asset = GlobalStateKey<AssetID>({ key: 'asa_id' });
44
45   sellPeriodEnd = GlobalStateKey<uint64>({ key: 'end' });
46
47   assetValidityPeriod = GlobalStateKey<uint64>({ key: 'asa_v' });
```

A 33..47. sorokban a globális állapotváltozókat definiáljuk. Ezek kulcs-érték párok, és egy okos szerződésben 64 darab lehet belőlük.

A `GlobalStateKey` után meg kell adni a kulcs típusát: `string`, `uint64`, `Address`, `AssetID` stb. Ezt követően opcionálisan egy `{key: 'rövid-key-name'}` adható meg. Ennek a szerepe csupán a helytakarékosság, t. i. az okos szerződésen belül, az AVM kód ezzel a kulcsnévvel hivatkozik a kulcs-érték párra. Egy globális állapotváltozónak a következőképpen tudunk értéket adni: `this.kulcsnév.value = érték`. Egy globális allapotváltozó értékére a `this.kulcsnév.value` kifejezéssel tudunk hivatkozni.

7.3.3. `createApplication` – az okos szerződés létrehozása után meghívva

```
49  /**
50   * Init the values of global keys
51   */
52  createApplication(): void {
53    this.appVersion.value = 'v1.3';
54    this.appCreatorAddress.value = globals.creatorAddress;
55    this.assetAmountInitial.value = 0;
56    this.assetAmount.value = 0;
57    this.assetPrice.value = 0;
58    this.asset.value = AssetID.zeroIndex;
59    this.sellPeriodEnd.value = 0;
60    this.assetValidityPeriod.value = 0;
61  }
```

Az okos szerződés létrehozásakor a `createApplication` metódus is lefut. Ennek 53..60. soraiban történik meg a globális állapotváltozók inicializálása.

7.3.4. `bootstrap` – kezdeti paraméterek beállítása

```
63  /**
64    * create ASA, set global key values
65    * @param assetPrice ASA price in microAlgos
66    * @param assetAmount ASA initial amount in pieces
67    * @param sellPeriodLength sell period length in secs
68    * @param assetValidityPeriod asset validity in secs, after that time
      ↪  it can be clawbacked
69    */
70  bootstrap(assetPrice: uint64, assetAmount: uint64, sellPeriodLength:
    ↪  uint64, assetValidityPeriod: uint64) {
71    /// allow only the app creator to call this method
72    verifyAppCallTxn(this.txn, { sender: globals.creatorAddress });
73
74    /// assert bootstrap hasn't been called yet
75    assert(this.assetAmountInitial.value === 0);
76
```

```
77      // create asset
78      const asset = sendAssetCreation({
79        configAssetTotal: assetAmount,
80        configAssetDecimals: 0,
81        configAssetName: 'Bizalmi Kör Zseton',
82        configAssetUnitName: 'BKTOVJ1',
83        configAssetURL: 'https://algorand.hu/bk/bktovj.html',
84        configAssetDefaultFrozen: 0,
85        configAssetManager: globals.currentApplicationAddress,
86        configAssetReserve: globals.currentApplicationAddress,
87        configAssetFreeze: globals.currentApplicationAddress,
88        configAssetClawback: globals.currentApplicationAddress,
89      });
90
91      // set global values
92      this.assetAmountInitial.value = assetAmount;
93      this.assetAmount.value = assetAmount;
94      this.assetPrice.value = assetPrice;
95      this.asset.value = asset;
96      this.sellPeriodEnd.value = globals.latestTimestamp + sellPeriodLength;
97      this.assetValidityPeriod.value = assetValidityPeriod;
98    }
```

Az okos szerződés `bootstrap` metódusa felel az okos szerződés kezdeti paramétereinek beállításáért. A 70. sorban látható, hogy a metódus paraméterei szabályozzák a zseton árát, a létrehozandó zsetonok darabszámát, az értékesítési periódus hosszát másodpercekben megadva, és a zseton lejárati periódust, szintén másodpercekben megadva.

A 72. sorban látható ellenőrzés biztosítja, hogy csak a szerződés létrehozója hívhassa meg ezt az eljárást.

A 75. sorban végzett ellenőrzés biztosítja, hogy a `bootstrap` metódust ne lehessen kétszer meghívni. Talán helyesebb lett volna egy külön csak erre a célra szolgáló, `already_bootstrapped` globális állapotváltozót felvenni erre a célra.

Az ASA létrehozása a 78..89. sorokban történik, egy belső tranzakció használatával. A 87. és 88. sorokban megadtuk a befagyasztási (“freeze”)

és visszavonási (“clawback”) művelet végzésére szolgáló címet is, ami jelen esetben nem más, mint az okos szerződés Algorand címe.

A globális állapotváltozók beállítása a 92..97. sorokban történik. A 95. sorban a `this.asset.value = asset` értékadásban nem csupán az asset azonosítóját, hanem az asset egyéb jellemzőit is tároljuk.

7.3.5. A globális állapotváltozók visszaolvasása

```
100  /**
101    * get app creator address
102    * @returns app creator address
103    */
104   getAppCreatorAddress(): Address {
105     return this.appCreatorAddress.value;
106   }
107
108   /**
109    * get app version
110    * @returns app version
111    */
112   getAppVersion(): string {
113     return this.appVersion.value;
114   }
115
116   /**
117    * get asa initial amount
118    * @returns asa amount minted initially
119    */
120   getAssetAmountInitial(): uint64 {
121     return this.assetAmountInitial.value;
122   }
123
124   /**
125    * get asa amount
126    * @returns sellable asa amount
127    */
128   getAssetAmount(): uint64 {
129     return this.assetAmountInitial.value;
130   }
```

```
131
132   /**
133    * get asa price
134    * @returns asa price in microAlgos
135    */
136   getAssetPrice(): uint64 {
137     return this.assetPrice.value;
138   }
139
140   /**
141    * get asa id
142    * @returns asa id
143    */
144   getAssetID(): AssetID {
145     return this.asset.value;
146   }
147
148   /**
149    * get end of sell period
150    * @returns end of sell period as absolute time in sec, from 01-Jan-1970
151    */
152   getSellPeriodEnd(): uint64 {
153     return this.sellPeriodEnd.value;
154   }
```

A globális állapotváltozók visszaolvasására külön metódusok szolgálnak, melyek az ABI szabályainak megfelelően elvégzik az adatkonverziót is. Ugyanakkor a `appClient.getGlobalState()` hívással „nyers” formában az összes állapotváltozó egyszerre beolvasható az app külön hívása nélkül is, vagyis a `getGlobalState()` hívás „sokkal olcsóbb”. Az `assetValidityPeriod` lekérdezésére szolgáló getter függvény véletlenül kimaradt a sorból.

7.3.6. `buyAsset` – zseton vétele

```
156   /**
157     * Buy 1 piece of the asset
158     * @param payment txn, where amount is equal to assetPrice, receiver is
        ↪  app address
```

```
159     */
160   buyAsset(payment: PayTxn): void {
161     /// Ensure asset selling period hasn't ended yet
162     assert(globals.latestTimestamp <= this.sellPeriodEnd.value, 'Sell
        ↪  period ended');
163
164     /// Ensure that buyer hasn't bought earlier this asset
165     assert(this.txn.sender.assetBalance(this.asset.value) === 0, 'Asset
        ↪  already bought');
166
167     /// Verify payment transaction: receiver is the app, amount is the
        ↪  asset price
168     verifyPayTxn(payment, {
169       sender: this.txn.sender,
170       receiver: globals.currentApplicationAddress,
171       amount: { greaterThanEqualTo: this.assetPrice.value,
          ↪  lessThanEqualTo: this.assetPrice.value },
172     });
173
174     /// Is there still an asset to sell?
175     assert(this.assetAmount.value > 0, 'No more ASA to sell');
176
177     /// Opt into asset, unconditionally
178     sendAssetTransfer({
179       xferAsset: this.asset.value,
180       assetAmount: 0,
181       assetReceiver: this.app.address,
182     });
183
184     /// Unfreeze asset
185     sendAssetFreeze({
186       freezeAsset: this.asset.value,
187       freezeAssetAccount: this.txn.sender,
188       freezeAssetFrozen: false,
189     });
190
191     /// Send asset to the buyer
192     sendAssetTransfer({
193       xferAsset: this.asset.value,
194       assetReceiver: this.txn.sender,
195       assetAmount: 1,
```

```
196       });
197
198       /// Freeze the asset at the buyer's address
199       sendAssetFreeze({
200         freezeAsset: this.asset.value,
201         freezeAssetAccount: this.txn.sender,
202         freezeAssetFrozen: true,
203       });
204
205       // Decrease asset amount
206       this.assetAmount.value = this.assetAmount.value - 1;
207     }
```

A zseton vételt a `buyAsset` metódus valósítja meg. Ennek egyetlen paramétere van, egy fizetési tranzakció, amely a zseton megvételére szolgál. A generált TEAL kód vizsgálatából látszik, hogy erre a tranzakcióra a metóduson belül mint egy tranzakciós csoport tranzakciójára történik hivatkozás. A tranzakciós csoport végrehajtásakor vagy az összes tranzakció végbemegy atomi módon, vagy egyetlen egy sem megy végbe. Az Algorand szerződés így biztosítja, hogy ha nem lép fel semmilyen hiba, akkor a fizetési tranzakció ellenében a zsetont megkapja a vevő, de ha valamilyen hiba történt, akkor nem megy végbe a fizetési tranzakció.

A `buyAsset` metódusban számos ellenőrzés történik:

- a 162. sor azt ellenőrzi, hogy a vétel időpontja az értékesítési periódus vége előtti van-e. Ha a feltétel nem áll fönn, akkor a metódus végrehajtása hibajelzéssel megszakad.
- a 165. sor azt ellenőrzi, hogy a vevőnek az eladandó zsetonból még egy sincs a birtokában (azaz „0 db van a birtokában”).
- a 168..172. sorok az ellenőrzik, hogy
  - a fizetési tranzakció küldője ugyanaz a cím-e, mint az okos szerződést meghívó tranzakció küldőjének a címe, lásd 169. sor,
  - a fizetési tranzakció címzettje megegyezik-e az okos szerződés

Algorand címével, lásd 170. sor,

  - a fizetési tranzakcióban küldött összeg nem kisebb-e vagy nagyobb-e, mint a globális állapotváltozóban szereplő ár, lásd 171. sor

- a 175. sor azt ellenőrzi, hogy van-e még eladható zseton

Ezt követően a 178..182. sorokban a metódus „benevez” (opt-in) az ASÁ-ba egy belső tranzakció segítségével. Nem sikerült megoldani, hogy ne legyen szükség a metódus meghívása előtt külön benevezésre, „xxx” hibát kaptam, ezért ezek a sorok fölöslegesek. (Talán a 165. sorban lévő assertation miatt az opt-in műveletet az elé kellene rakni.)

A 185..189. sorok először feloldják a zseton befagyasztását, majd a 192..196. sorok elküldik a vevőnek a zsetont, végül a 199..203. sorok befagyasztják a vevő címén lévő zsetont.

Joggal kérdezheti az Olvasó, hogy mi szükség van a zseton befagyasztásának a feloldására, ha a vevő még nem is rendelkezik a zsetonnal? Erre a később ismertetett visszavonási folyamat miatt van szükség: ha a vevő vett egy zsetont, majd azt a lejárata miatt visszavontuk, akkor a vevő címe „befagyasztott” állapotban marad, befagyasztott címre pedig nem lehet zsetont küldeni.

A `buyAsset` végül a 206. sorban csökkenti az eladható zsetonok számát nyilvántartó globális állapotváltozó értékét.

Megjegyzés: mivel a `buyAsset` négy belső tranzakciót használ, a metódus meghívásakor ezeknek a belső tranzakcióknak is ki kell fizetni a díját, vagyis a `buyAsset` metódust a minimális tranzakciós díj ötszörösével kell meghívni.

7.3.7. `sendAlgosToCreator` – zseton vételár visszaküldés

```
209 /**
210   * Send Algos from the app address to the app creator address
211   */
212  sendAlgosToCreator(): void {
```

```
213     /// Allow only the creator to call this method
214     verifyAppCallTxn(this.txn, { sender: globals.creatorAddress });
215
216     /// Send back all the Algos above minAmount to the app creator
217     const minAmount = 600_000; // uAlgos
218     const balance = globals.currentApplicationAddress.balance;
219     if (balance > minAmount) {
220       sendPayment({
221         receiver: globals.creatorAddress,
222         amount: balance - minAmount,
223       });
224     }
225   }
```

A zsetonok vételárát a `sendAlgosToCreator` metódus hívásával lehet elküldeni az okos szerződésből az okos szerződés létrehozójának. Mint a `bootstrap` metódusnál, itt is ellenőrzi a metódus, hogy ki a hívó (lásd 214. sor), és csak az okos szerződés létrehozójának engedi meg a metódus futtatását.

Az okos szerződés címén szeretnénk, ha ott maradna 0.6 Algó, és csak az ezt meghaladó összeget utaljuk át az okos szerződés létrehozójának, lásd 219..223. sorok.

7.3.8. `clawback` – zseton visszavétele

```
227 /**
228    * Clawback asset to app & inc amount
229    * @param addr address from which to clawback asset
230    */
231   clawback(addr: Address): void {
232     /// Allow only the app creator to call this method
233     verifyAppCallTxn(this.txn, { sender: globals.creatorAddress });
234
235     /// Clawback assets to app
236     sendAssetTransfer({
237       xferAsset: this.asset.value,
238       assetAmount: 1,
239       assetSender: addr,
240       assetReceiver: globals.currentApplicationAddress,
```

```
241     });
242
243     /// Inc asset amount
244     this.assetAmount.value = this.assetAmount.value + 1;
245   }
```

Egy zseton visszavétele a `clawback` metódus hívásával valósítható meg. Paraméterként meg kell adni, hogy melyik címről szeretnénk visszavenni a zsetont. A 236..241. sorokban történik meg a visszavétel: egy belső tranzakcióval elküldjük a zsetont a megadott címről a szerződésnek. Erre azért van lehetőségünk, mert a szerződés címét adtuk meg a zseton létrehozáskor a visszavonási címnek. A 244. sorban növeljük az eladható zsetonok számát.

Megjegyzés: egy külső, időzített eljárás ellenőrzi, hogy melyek a „lejárt" zsetonok, és a `clawback` metódus segítségével visszaszedi őket. Célszerű lenne a clawback-et kiegészíteni egy olyan ellenőrzéssel, hogy valóban lejárt-e a zseton, azaz az utolsó művelet (vásárlás) és a mostani művelet között eltelt-e már az `assetValidityPeriod` értéknek megfelelő időtartam.

7.3.9. `clawbackNoIncAmount` – zseton bevonása

Egy tulajdonrész sikeres megszerzésekor a `clawbackNoIncAmount` metódus meghívásával lehetséges egy zseton bevonása. Ez egyezik a `clawback` metódussal, de nem növeli az eladható zsetonok számát.

7.3.10. `deleteAsset` – az ASA törlése

```
265   /**
266    * Delete asset within app
267    */
268   deleteAsset(): void {
269     /// Allow only the app creator to call this method
270     verifyAppCallTxn(this.txn, { sender: globals.creatorAddress });
271     // assert(this.txn.sender === this.app.creator, 'Allow only the app
      ↪  creator to call this method');
272
273     /// Delete asset
```

```
274       sendAssetConfig({
275         configAsset: this.asset.value,
276       });
277     }
```

Az okos szerződés törlésének előfeltétele, hogy az okos szerződés már ne rendelkezzen a létrehozott ASÁ-val. A `deleteAsset` metódus törli az ASÁ-t, a 274..276. sorokban látható belső tranzakcióval. Igazán érdekes, hogy épp ilyen `sendAssetConfig` hívással törölhető az ASA.

7.3.11. `deleteApplication` – az okos szerződés törlése előtt meghívva

```
279   /**
280    * Delete app with ABI method
281    */
282   deleteApplication(): void {
283     /// Allow only the app creator to call this method
284     verifyAppCallTxn(this.txn, { sender: globals.creatorAddress });
285
286     /// Send back Algos to app creator account
287     sendPayment({
288       receiver: globals.creatorAddress,
289       amount: 0,
290       closeRemainderTo: globals.creatorAddress,
291     });
292   }
```

A szerződés törlése során a `deleteApplication` metódus kerül meghívásra. Ahhoz, hogy ez hibátlanul lefusson, a `deleteAsset` előzetes meghívása szükséges, a `deleteAsset` hibátlan lefutásának előfeltétele pedig az, hogy az okos szerződés címén legyen az összes zseton. A `deleteApplication` 287..291. sorában látható belső tranzakció elküldi a szerződés létrehozójának a címére a maradék Algorand-ot.

7.4. Az okos szerződés tesztelése

Az okos szerződés tesztelését végző Jest fájl a BizKor.test.ts.

Az Algokit Typescript segédprogramok között található algorandFixture.ts return értékei:

```
126  return {
127    get context() {
128      return context
129    },
130    get algorand() {
131      return algorandClient
132    },
133    beforeEach,
134  }
```

ahol a `context` a következő:

```
110  context = {
111      algod: transactionLoggerAlgod,
112      indexer: indexer,
113      kmd: kmd,
114      testAccount,
115      generateAccount: async (params: GetTestAccountParams) => {
116        const account = await getTestAccount(params,
           ↪  transactionLoggerAlgod, kmd)
117        algorandClient.setSignerFromAccount(account)
118        return { ...account, signer:
           ↪  algorandClient.account.getSigner(account.addr) }
119      },
120      transactionLogger: transactionLogger,
121      waitForIndexer: () => transactionLogger.waitForIndexer(indexer),
122      waitForIndexerTransaction: (transactionId: string) =>
         ↪  runWhenIndexerCaughtUp(() =>
         ↪  lookupTransactionById(transactionId, indexer)),
123    }
```

Ezt azt jelenti, hogy a `context`-ben megtalálható:

- egy `algod` algorand kliens

- egy `kmd` key management démon kliens
- egy `testAccount` teszt számlaszám
- egy `genarateAccount` függvény
- egy `transactionLogger` függvény stb., stb.

Hivatkozások:

- Jest dokumentáció
- Algokit Typescript segédprogramok
- algorandFixture.ts

7.4.1. Az egyes Jest tesztek előtti beállítások

```
1  /* eslint-disable no-console */
2  import { describe, test, expect, beforeAll, beforeEach } from
   ↪  '@jest/globals';
3  import { algorandFixture } from
   ↪  '@algorandfoundation/algokit-utils/testing';
4  import * as algokit from '@algorandfoundation/algokit-utils';
5  import algosdk, { Transaction } from 'algosdk';
6  import { TransactionSignerAccount } from
   ↪  '@algorandfoundation/algokit-utils/types/account';
7  import { BizKorClient } from '../contracts/clients/BizKorClient';
8
9  const fixture = algorandFixture();
10 algokit.Config.configure({ populateAppCallResources: true });
11
12 let appClient: BizKorClient;
13
14 describe('BizKor', () => {
15   const log = false; // skip console.log() calls
16   const paramAppVersion = 'v1.3'; // app version
17   const paramAssetPrice = 1_000_000; // microAlgos
18   const paramAssetAmountInitial = 10; // pieces
19   const paramSellPeriodLength = 1000; // sec
20   const paramAssetValidityPeriod = 100; // sec
21
```

```
22  let acc1: algosdk.Account;
23  let signer1: TransactionSignerAccount;
24  let acc2: algosdk.Account;
25
26  beforeEach(fixture.beforeEach);
27
28  beforeAll(async () => {
29    await fixture.beforeEach();
30    const { algod, kmd } = fixture.context;
31
32    acc1 = await algokit.getOrCreateKmdWalletAccount(
33      {
34        name: 'Buyer of Biz.Kör. token',
35        fundWith: algokit.algos(100),
36      },
37      algod,
38      kmd
39    );
40    if (log) console.log('acc1.addr (token buyer):', acc1.addr);
41    // signer1 = algosdk.makeBasicAccountTransactionSigner(sender1);
42    signer1 = {
43      addr: acc1.addr,
44      // eslint-disable-next-line no-unused-vars
45      signer: async (txnGroup: Transaction[], indexesToSign: number[]) =>
      ↪  {
46        return txnGroup.map((tx) => tx.signTxn(acc1.sk));
47      },
48    };
49
50    acc2 = await algokit.getOrCreateKmdWalletAccount(
51      {
52        name: 'App creator',
53        fundWith: algokit.algos(100),
54      },
55      algod,
56      kmd
57    );
58    if (log) console.log('acc2.addr (app creator):', acc2.addr);
59
60    appClient = new BizKorClient(
61      {
```

```
62          sender: acc2,
63          resolveBy: 'id',
64          id: 0,
65        },
66        algod
67      );
68
69      await appClient.create.createApplication({});
70    });
```

A 2. sor a Jest teszteléshez szükséges függvények importját tartalmazza.

A 3. sor a Jest teszteléshez szükséges algod és kmd klienseket előállító `algorandFixture` importját tartalmazza.

A 4. sor az Algorand segédrutinokat tartalmazó `algotkit` importját tartalmazza.

Az 5. sor az `algosdk` API-t importálja.

A 6. sor a `TransactionSignerAccount` típust importálja.

A 7. sor a `BizKorClient`-et importálja, amely a `BizKor` okos szerződéshez definiálja a TypeScript típusokat.

A 9. sor a Jest fixture-t mint az algorandFixture()-t definiálja.

A 10. sor egy konfigurációs beállítás az okos szerződések hívásához.

A 12. sor definiálja az okos szerződés klienst, ami egy típusos kliens, a TEALScript fordítóprogram hozza létre az okos szerződés forrásából. A típusos kliens sokkal könnyebbé teszi az okos szerződés kezelését, metódusainak hívását, mert a TypeScript nyelv típusdefinícióival írja le az okos szerződést.

A 14. sorban kezdődik a teszthalmaz leírása. A tesztek futtatása során a naplózás (`console.log` hívások) a 15. sorban tiltható vagy engedélyezhető. A 16..20. sorban az okos szerződés paramétereket adjuk meg. A 22. sorban szereplő `acc1` lesz a vevő számlája, a 24. sorban szereplő `acc2` pedig az a

számla, amely az okos szerződést létrehozza. A 23. sorban szereplő `signer1` az `acc1`-hez tartozó, aláírást végző szerkezet.

A 26. sorban `beforeEach(fixture.beforeEach)` azt jelenti, hogy *minden egyes teszt* előtt le kell futtatni a `fixture`-t, ami az `algorandFixture` futtatását jelenti.

A Jest tesztek előtt *csak egyetlen egyszer* fut le a 28..70. sor közötti rész. A `fixture.beforeEach()` (29. sor) előállítja a `context`-ben az `algod, kmd` stb. változókat.

A 30. sorban a `fixture.context`-ből visszanyerjük az `algod` algod kliens és `kmd` kmd kliens változókat.

A 32..39. sorban létrehozunk vagy visszanyerünk egy kmd számlát, és feltöltjük 100 Algóval. Ez az `acc1` számla lesz a vevőnek a számlája, aki a zsetont megveszi.

A 42..48. sorban készítünk egy `signer1` tranzakció aláírót a vevő számára. A `signer1` tranzakció aláíró tartalmazza a `acc1` számla címét (`acc1.addr`), valamint egy `signer` aláíró függvényt, mellyel az `acc1` nevében tranzakció csoportok írhatók alá.

Az 50..57. sorokban létrehozunk vagy visszanyerünk egy kmd számlát, és feltöltjük 100 Algóval. Ez az `acc2` számla lesz az a számla, amely az okos szerződést létrehozza.

A 60..67. sorokban létrehozzuk az okos szerződéshez tartozó klienst. Mivel a megadott `id` paraméter 0, mindig új `appClient` jön létre.

Végül a 69. sorban létrehozzuk az okos szerződést a blokkláncon, és a szerződés létrehozása után meghívjuk a `createApplication` metódust.

7.4.2. `bootstrap` teszt

```
72  test('bootstrap', async () => {
73    await appClient.appClient.fundAppAccount(algokit.microAlgos(600_000));
74    const assetPrice = paramAssetPrice;
```

```
75      const assetAmount = paramAssetAmountInitial;
76      const sellPeriodLength = paramSellPeriodLength;
77      const assetValidityPeriod = paramAssetValidityPeriod;
78      // fee must be paid for 2 transactions, due to the inner transaction
79      await appClient.bootstrap(
80        { assetPrice, assetAmount, sellPeriodLength, assetValidityPeriod },
81        { sendParams: { fee: algokit.transactionFees(2) } }
82      );
83      const globalState = await appClient.getGlobalState();
84      expect(globalState.asa_total?.asNumber()).toBe(assetAmount);
85      expect(globalState.asa_amt?.asNumber()).toBe(assetAmount);
86      expect(globalState.asa_price?.asNumber()).toBe(assetPrice);
87    });
```

A 79. sorban történik az alkalmazás/okos szerződés `bootstrap` metódusának hívása. Az első `{ }` között a metódus paramétereit kell megadni, a második `{ }` között pedig az okos szerződést hívó tranzakció paramétereit, itt a tranzakciós díjat. A tranzakciós díj a szokásos díj kétszerese, a metódusban lévő belső tranzakció miatt.

7.4.3. `getAppVersion` teszt

```
89    test('getAppVersion', async () => {
90      const version = await appClient.getAppVersion({});
91      expect(version.return).toBe(paramAppVersion);
92    });
```

7.4.4. `getAppCreatorAddress` teszt

```
94    test('getAppCreatorAddress', async () => {
95      const appCreatorAddress = await appClient.getAppCreatorAddress({});
96      expect(appCreatorAddress.return).toBe(acc2.addr);
97    });
```

7.4.5. getAssetAmountInitial teszt

```
99  test('getAssetAmountInitial', async () => {
100   const assetAmountInitial = await appClient.getAssetAmountInitial({});
101   expect(assetAmountInitial.return).toBe(BigInt(paramAssetAmountInitial
      ↪ ));
102 });
```

7.4.6. getAssetAmount teszt

```
104 test('getAssetAmount', async () => {
105   const assetAmountInitial = await appClient.getAssetAmount({});
106   expect(assetAmountInitial.return).toBe(BigInt(paramAssetAmountInitial
      ↪ ));
107 });
```

7.4.7. getAssetPrice teszt

```
109 test('getAssetPrice', async () => {
110   const assetPrice = await appClient.getAssetPrice({});
111   expect(assetPrice.return).toBe(BigInt(paramAssetPrice));
112 });
```

7.4.8. getAssetId teszt

```
114 test('getAssetId', async () => {
115   const assetId = await appClient.getAssetId({});
116   expect(assetId.return).toBeGreaterThan(BigInt(1_000));
117 });
```

7.4.9. getSellPeriodEnd teszt

```
119 test('getSellPeriodEnd', async () => {
120   const sellPeriodEnd = await appClient.getSellPeriodEnd({});
121   // get date/time
122   const now = new Date();
```

```
123     // get msec since 1970
124     const millisecondsSinceEpoch = now.getTime();
125     // get sec from msec
126     const secondsSinceEpoch = Math.floor(millisecondsSinceEpoch / 1000);
127     // check sellPeriodEnd
128     if (log) console.log('sellPeriodEnd: ', sellPeriodEnd.return);
129     expect(sellPeriodEnd.return).toBeGreaterThan(BigInt(secondsSinceEpoch
        ↪  )); // "algokit localnet reset" may be
        ↪  required
130     expect(sellPeriodEnd.return).toBeLessThan(BigInt(secondsSinceEpoch +
        ↪  paramSellPeriodLength));
131   });
```

7.4.10. getGlobalState teszt

```
133   test('getGlobalState', async () => {
134     const globalState = await appClient.getGlobalState();
135     const apv = globalState.apv!.asByteArray();
136     const apca = globalState.apca?.asByteArray();
137     const asaTotal = globalState.asa_total?.asNumber();
138     const asaAmt = globalState.asa_amt?.asNumber();
139     const asaPrice = globalState.asa_price?.asNumber();
140     const asaId = globalState.asa_id?.asNumber();
141     const end = globalState.end?.asNumber();
142     const asaV = globalState.asa_v?.asNumber();
143     // console.log('globalState:', globalState);
144
145     // get apvGood, i.e. without the length (first 2 bytes)
146     const apvGood = Buffer.from(apv).slice(2).toString('utf-8'); // get
        ↪  rid of length
147     if (log) console.log('apvGood: ', apvGood);
148     expect(apvGood).toBe(paramAppVersion);
149     // get apcaGood. i.e. encode 32 byte Algorand address as string
150     const bufferApca = Buffer.from(apca!);
151     if (log) console.log('bufferApca: ', bufferApca);
152     if (log) console.log('bufferApca.length: ', bufferApca.length);
153     const apcaGood = algosdk.encodeAddress(bufferApca); // encode as
        ↪  Algorand address
154     if (log) console.log('apcaGood: ', apcaGood);
155     expect(apcaGood).toBe(acc2.addr);
```

```
156
157       if (log) console.log('getGlobalState apv (appVersion):', apv);
158       if (log) console.log('getGlobalState apca (appCreatorAddress):',
          ↪  apca);
159       if (log) console.log('getGlobalState asa_total
          ↪  (assetAmountInitial):', asaTotal);
160       expect(asaTotal).toBe(paramAssetAmountInitial);
161       if (log) console.log('getGlobalState asa_amt (assetAmount):', asaAmt);
162       expect(asaAmt).toBe(paramAssetAmountInitial);
163       if (log) console.log('getGlobalState asa_price (assetPrice):',
          ↪  asaPrice);
164       expect(asaPrice).toBe(paramAssetPrice);
165       console.log('getGlobalState asa_id (asset):', asaId);
166       if (log) console.log('getGlobalState end (sellPeriodEnd):', end);
167       if (log) console.log('getGlobalState asa_v (assetValidityPeriod):',
          ↪  asaV);
168       expect(asaV).toBe(paramAssetValidityPeriod);
169     });
```

A 134. sorban a `getGlobalState()` hívással megtörténik az összes állapotváltozó beolvasása.

A 135. sor az `apv` (appVersion) értékét állítja elő, byte tömbként. A tömb elején lévő két byte az ABI string hosszát tárolja. Ettől a 146. sorban szabadulunk meg, és a 148. sorban ellenőrizzük, hogy azt kaptuk-e, amit a tesztek elején lévő paraméterek tartalmaznak.

A 136. sor az `apca` (appCreatorAddress) értékét állítja elő, byte tömbként. Az ebben lévő 32 byte az Algorand számlához tartozó publikus kulcs. Ebből az ASCII alakú cím kiformázását a 153. sor végzi.

A 137..142. sor a többi globális állapotváltozó értékét állítja elő, az `asNumber()` getter rutin segítségével. Ezekben az esetekben nincs átformázási feladat.

7.4.11. `opt in to asset` teszt

```
171  test('opt in to asset', async () => {
172    const { algod } = fixture.context;
173    const params = await algod.getTransactionParams().do();
174    const globalState = await appClient.getGlobalState();
175    const asset = globalState.asa_id!.asNumber();
176    if (log) console.log('Try to opt in to asset: ', asset, acc1.addr);
177    const txn1 =
       ↪  algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
178      from: acc1.addr,
179      to: acc1.addr,
180      amount: 0,
181      assetIndex: asset,
182      suggestedParams: params,
183    });
184    const stxn1 = txn1.signTxn(acc1.sk);
185    const txn2 = await algod.sendRawTransaction(stxn1).do();
186    await algosdk.waitForConfirmation(algod, txn2.txId, 4);
187  });
```

A 177..183. sorokban történik meg az `acc1.addr` cím benevezése (opt in) a zsetonba (ASA). A zseton azonosítóját a 175. sorban, a globális állapotból állítjuk elő.

7.4.12. `buyAsset` teszt

```
189  test('buyAsset', async () => {
190    const { algod, testAccount } = fixture.context;
191    const params = await algod.getTransactionParams().do();
192    // Make a payment tx, to buy asset
193    const appRef = await appClient.appClient.getAppReference();
194    // const appAddres = await
       ↪  algosdk.getApplicationAddress(appRef.appId);
195    if (log) console.log('buyAsset: testAccount.addr ', testAccount.addr);
196    if (log) console.log('buyAsset: appRef.appAddress ',
       ↪  appRef.appAddress);
197    if (log) console.log('buyAsset: appCreatorAddr ', acc2.addr);
198    const tx1 = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
```

```
199       from: acc1.addr,
200       to: appRef.appAddress,
201       amount: paramAssetPrice,
202       suggestedParams: params,
203     });
204
205     // Buy asset
206     const globalState = await appClient.getGlobalState();
207     const asset = globalState.asa_id!.asNumber();
208     const compose = appClient.compose().buyAsset(
209       {
210         payment: tx1,
211       },
212       {
213         sender: signer1,
214         sendParams: {
215           fee: algokit.transactionFees(5),
216         },
217         assets: [Number(asset)],
218       }
219     );
220     // atc, build group, sign, send
221     const atc = await compose.atc();
222     const txs = atc.buildGroup().map((tx) => tx.txn);
223     const signed = await signer1.signer(
224       txs,
225       Array.from(Array(txs.length), (_, i) => i)
226     );
227     const txg = await algod.sendRawTransaction(signed).do();
228     await algosdk.waitForConfirmation(algod, txg.txId, 4);
229   });
```

190. sor, `algod` betöltése, 191. sor, `params` tranzakciós paraméterek betöltése, 193. sor, `appRef` betöltése. 198..203. sor, fizetési tranzakció előállítása, a küldő cím az `acc1.addr`, a fogadó cím az `appRef.appAddress`.

206. sor, globális állapot beolvasása. 207. sor, a globális állapotból az `asa_id` előállítása. 208..219. sor, a `buyAsset` metódus hívás a compose-on keresztül. Figyelje meg a 215. sorban az 5-szörös tranzakciós díjat, és a

217. sorban az `assets` tömbben megadott `asset_id`-t. Ez utóbbi megadása nélkül "Unavailable asset" hibaüzenetet kapnánk. A 221. sor a tranzakciókat nyeri vissza, a 222. sor csoport azonosítóval látja el a tranzakciókat. A 223..226. sor aláírja a tranzakciókat.

A 225. sorban lévő konstrukció magyarázata, a ChatGPT4 szerint:

> Az `Array.from(Array(txs.length), (_, i) => i)` egy tömböt hoz létre, amely a tranzakciók indexeit tartalmazza. Ez azt jelzi, hogy melyik tranzakciókat kell aláírni.
>
> Az `Array(txs.length)` létrehoz egy új tömböt, amelynek hossza megegyezik a `txs` tömb hosszával. Kezdetben minden eleme `undefined`. Az `Array.from()` egy új tömböt hoz létre az adott tömbből. Ebben az esetben az első paraméter a létrehozott `undefined` elemekkel rendelkező tömb. A második paraméter egy függvény, amely minden elemre (itt `undefined`) és az indexére (`i`) meghívásra kerül. Ebben az esetben a függvény egyszerűen visszaadja az indexet (`i`), így a végeredmény egy tömb lesz, amely 0-tól kezdődően tartalmazza a tranzakciók indexeit.

Megjegyzés: a `buyAsset` hívása egyszerűsíthető, ha az `execute` függvénnyel végeztetjük el a tranzakciós csoport létrehozását, aláírását és elküldését. Ekkor a `buyAsset` teszt vége megváltozik:

```
208     const result = await appClient.compose().buyAsset(
209         { payment: tx1, },
210         {
211           sender: signer1,
212           sendParams: {
213             fee: algokit.transactionFees(5),
214           },
215           assets: [Number(asset)],
216         }
217       )
218       .execute();
219     await algokit.waitForConfirmation(result.txIds[0], 4, algod);
```

```
220   });
```

Figyelje meg az Olvasó a 218. sorban lévő `execute()` függvényt.

A 213. sorban az `algokit.transactionFees` függvénnyel összesen 5 tranzakció díját kellett megfizetnünk: az app `buyAsset` metódusának hívása 1 tranzakció, a `buyAsset` metóduson belül pedig további 4 belső tranzakció van.

7.4.13. `buyAsset 2nd time` teszt

```
231   test('buyAsset 2nd time', async () => {
232     const { algod, testAccount } = fixture.context;
233     const params = await algod.getTransactionParams().do();
234
235     // Make a payment tx, to buy asset
236     const appRef = await appClient.appClient.getAppReference();
237     // const appAddres = await
      ↪  algosdk.getApplicationAddress(appRef.appId);
238     if (log) console.log('buyAsset: testAccount.addr ', testAccount.addr);
239     if (log) console.log('buyAsset: appRef.appAddress ',
      ↪  appRef.appAddress);
240     if (log) console.log('buyAsset: appCreatorAddr ', acc2.addr);
241     const tx1 = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
242       from: acc1.addr,
243       to: appRef.appAddress,
244       amount: paramAssetPrice,
245       suggestedParams: params,
246     });
247
248     // Buy asset
249     const globalState = await appClient.getGlobalState();
250     const asset = globalState.asa_id!.asNumber();
251     const compose = appClient.compose().buyAsset(
252       {
253         payment: tx1,
254       },
255       {
256         sender: signer1,
```

```
257         sendParams: {
258           fee: algokit.transactionFees(5),
259         },
260         assets: [Number(asset)],
261       }
262     );
263
264     const atc = await compose.atc();
265     const txs = atc.buildGroup().map((tx) => tx.txn);
266     const signed = await signer1.signer(
267       txs,
268       Array.from(Array(txs.length), (_, i) => i)
269     );
270     try {
271       await algod.sendRawTransaction(signed).do();
272     } catch (err) {
273       console.log('this test should fail, as the buyer already has a
      ↪  coin', err); // err.response.body.data.pc);
274     }
275   });
```

Egyezik a `buyAsset` teszttel, de itt az app hibát ad, mert a vevő már rendelkezik egy zsetonnal. A hiba kezelése a 272..274. sorban történik.

7.4.14. `sendAlgosToCreator` teszt

```
277   test('sendAlgosToCreator', async () => {
278     await appClient.sendAlgosToCreator({}, { sendParams: { fee:
      ↪  algokit.transactionFees(2) } });
279   });
```

7.4.15. `clawback` teszt

```
281   test('clawback', async () => {
282     await appClient.clawback({ addr: acc1.addr }, { sendParams: { fee:
      ↪  algokit.transactionFees(2) } });
283   });
```

7.4.16. buyAsset after clawback teszt

```
285   test('buyAsset after clawback', async () => {
286     const { algod, testAccount } = fixture.context;
287     const params = await algod.getTransactionParams().do();
288     // Make a payment tx, to buy asset
289     const appRef = await appClient.appClient.getAppReference();
290     // const appAddres = await
      ↪  algosdk.getApplicationAddress(appRef.appId);
291     if (log) console.log('buyAsset: testAccount.addr ', testAccount.addr);
292     if (log) console.log('buyAsset: appRef.appAddress ',
      ↪  appRef.appAddress);
293     if (log) console.log('buyAsset: appCreatorAddr ', acc2.addr);
294     const tx1 = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
295       from: acc1.addr,
296       to: appRef.appAddress,
297       amount: paramAssetPrice,
298       suggestedParams: params,
299     });
300
301     // Buy asset
302     const globalState = await appClient.getGlobalState();
303     const asset = globalState.asa_id!.asNumber();
304     const compose = appClient.compose().buyAsset(
305       {
306         payment: tx1,
307       },
308       {
309         sender: signer1,
310         sendParams: {
311           fee: algokit.transactionFees(5),
312         },
313         assets: [Number(asset)],
314       }
315     );
316     // atc, build group, sign, send
317     const atc = await compose.atc();
318     const txs = atc.buildGroup().map((tx) => tx.txn);
319     const signed = await signer1.signer(
320       txs,
321       Array.from(Array(txs.length), (_, i) => i)
```

```
322       );
323       const txg = await algod.sendRawTransaction(signed).do();
324       await algosdk.waitForConfirmation(algod, txg.txId, 4);
325     });
```

7.4.17. clawback again teszt

```
327   test('clawback again', async () => {
328     await appClient.clawback({ addr: acc1.addr }, { sendParams: { fee:
        ↪  algokit.transactionFees(2) } });
329   });
```

7.4.18. 'opt out buyer from asset' teszt

```
331   test('opt out buyer from asset', async () => {
332     const { algod } = fixture.context;
333     const params = await algod.getTransactionParams().do();
334     const globalState = await appClient.getGlobalState();
335     const asset = globalState.asa_id!.asNumber();
336     const appRef = await appClient.appClient.getAppReference();
337     if (log) console.log('Try to opt out from asset: ', acc1.addr);
338     const txn1 =
        ↪  algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
339       from: acc1.addr,
340       to: appRef.appAddress,
341       closeRemainderTo: appRef.appAddress,
342       amount: 0,
343       assetIndex: asset,
344       suggestedParams: params,
345     });
346     const stxn1 = txn1.signTxn(acc1.sk);
347     const txn2 = await algod.sendRawTransaction(stxn1).do();
348     await algosdk.waitForConfirmation(algod, txn2.txId, 4);
349   });
```

A 338..345. sorban elküldi a vevő összes zsetonját az okos szerződésnek, úgy, hogy a `closeRemainderTo` mezőbe is az okos szerződés címét teszi. Ez

egyenértékű azzal, hogy a vevő „kiszáll” (“opt out”) az adott zsetonból.

7.4.19. ’deleteAsset’ teszt

```
351   test('deleteAsset', async () => {
352     await appClient.deleteAsset({}, { sendParams: { fee:
        ↪  algokit.transactionFees(2) } });
353   });
```

7.4.20. ’deleteApplication’ teszt

```
355   test('deleteApplication', async () => {
356     await appClient.delete.deleteApplication({}, { sendParams: { fee:
        ↪  algokit.transactionFees(2) } });
357   });
```

7.5. A tesztek futtatása

A tesztek Codespaces alatt a következő parancsokkal futtathatók:

```
1  algokit --version
2  cd biz_kor/projects/biz_kor-contracts
3  npm install
4  algokit localnet start
5  npm run build
6  npm run test
```

A parancsok eredménye a következő:

```
1  Welcome to Codespaces! You are on our default image.
2     - It includes runtimes and tools for Python, Node.js, Docker, and
      ↪  more. See the full list here: https://aka.ms/ghcs-default-image
3     - Want to use a custom image instead? Learn more here:
      ↪  https://aka.ms/configure-codespace
4
5  To explore VS Code to its fullest, search using the Command Palette
   ↪  (Cmd/Ctrl + Shift + P or F1).
6
```

```
Edit away, run your app as usual, and we'll automatically make it
↪  available for you to access.

@A-Maugli → /workspaces/akt02 (main) $ algokit --version
algokit, version 2.4.2
@A-Maugli → /workspaces/akt02 (main) $ ls
README.md  biz_kor  hellow
@A-Maugli → /workspaces/akt02 (main) $ cd biz_kor
@A-Maugli → /workspaces/akt02/biz_kor (main) $ ls
README.md  biz_kor.code-workspace  projects
@A-Maugli → /workspaces/akt02/biz_kor (main) $ cd projects
@A-Maugli → /workspaces/akt02/biz_kor/projects (main) $ ls
biz_kor-contracts  biz_kor-frontend
@A-Maugli → /workspaces/akt02/biz_kor/projects (main) $ cd
↪  biz_kor_contracts
bash: cd: biz_kor_contracts: No such file or directory
@A-Maugli → /workspaces/akt02/biz_kor/projects (main) $ cd
↪  biz_kor-contracts/
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ ls
README.md  __test__  contracts  jest.config.js  package-lock.json
↪  package.json  tsconfig.json
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ npm
↪  install

added 567 packages, and audited 568 packages in 13s

137 packages are looking for funding
  run `npm fund` for details

2 vulnerabilities (1 moderate, 1 high)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
npm notice
npm notice New minor version of npm available! 10.8.2 -> 10.9.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.9.0
npm notice To update run: npm install -g npm@10.9.0
npm notice
```

```
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ npm audit
# npm audit report

braces  <3.0.3
Severity: high
Uncontrolled resource consumption in braces - https://github.com/advisories/GHSA-grv7-fg5c-xmjg
fix available via `npm audit fix`
node_modules/braces

micromatch  <4.0.8
Severity: moderate
Regular Expression Denial of Service (ReDoS) in micromatch - https://github.com/advisories/GHSA-952p-6rrq-rcjv
fix available via `npm audit fix`
node_modules/micromatch

2 vulnerabilities (1 moderate, 1 high)

To address all issues, run:
  npm audit fix
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ npm audit fix

changed 3 packages, and audited 568 packages in 2s

137 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

Node.js v20.17.0
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ algokit localnet start
indexer has a new version available, run `algokit localnet reset --update` to get the latest version
algod has a new version available, run `algokit localnet reset --update` to get the latest version
Starting AlgoKit LocalNet now...
docker: algod Pulling
```

```
docker: conduit Pulling
...
docker: Container algokit_sandbox_algod  Healthy
docker: Container algokit_sandbox_conduit  Healthy
docker: Container algokit_sandbox_indexer  Healthy
Started; execute `algokit explore` to explore LocalNet in a web user interface.
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ algokit localnet status
# container engine
Name: docker (change with `algokit config container-engine`)
# algod status
Status: Running
Port: 4001
Last round: 0
Time since last round: 0.0s
Genesis ID: dockernet-v1
Genesis hash: Uwfli0j9XZXEKs3GGWBebEtbEOOzZPxGNVOXGC5za1w=
Version: 3.26.0
# conduit status
Status: Running
# indexer-db status
Status: Running
# indexer status
Status: Running
Port: 8980
Last round: 0
Version: 3.5.0
# proxy status
Status: Running
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ npm run compile-contract

> biz_kor-contracts@0.0.0 compile-contract
> tealscript contracts/*.algo.ts contracts/artifacts

@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ npm run test

> biz_kor-contracts@0.0.0 test
> npm run build && jest
```

```
113
114
115 > biz_kor-contracts@0.0.0 build
116 > npm run compile-contract && npm run generate-client
117
118
119 > biz_kor-contracts@0.0.0 compile-contract
120 > tealscript contracts/*.algo.ts contracts/artifacts
121
122
123 > biz_kor-contracts@0.0.0 generate-client
124 > algokit generate client contracts/artifacts/ --language typescript
    ↪  --output contracts/clients/{contract_name}Client.ts
125
126 Generating TypeScript client code for application specified in
    ↪  /workspaces/akt02/biz_kor/projects/biz_kor-contracts/contracts/artifa⌋
    ↪  cts/BizKor.arc32.json and writing to
    ↪  contracts/clients/BizKorClient.ts
127
128 ...
129
130  PASS  __test__/BizKor.test.ts (42.277 s)
131   BizKor
132     ✓ bootstrap (1498 ms)
133     ✓ getAppVersion (1445 ms)
134     ✓ getAppCreatorAddress (1380 ms)
135     ✓ getAssetAmountInitial (1529 ms)
136     ✓ getAssetAmount (1541 ms)
137     ✓ getAssetPrice (1410 ms)
138     ✓ getAssetId (1442 ms)
139     ✓ getSellPeriodEnd (1400 ms)
140     ✓ getGlobalState (1351 ms)
141     ✓ opt in to asset (1304 ms)
142     ✓ buyAsset (1835 ms)
143     ✓ buyAsset 2nd time (1381 ms)
144     ✓ sendAlgosToCreator (1139 ms)
145     ✓ clawback (1398 ms)
146     ✓ buyAsset after clawback (1518 ms)
147     ✓ clawback again (1408 ms)
148     ✓ opt out buyer from asset (1364 ms)
149     ✓ deleteAsset (1369 ms)
```

```
150       ✓ deleteApplication (1414 ms)
151
152   Test Suites: 1 passed, 1 total
153   Tests:       19 passed, 19 total
154   Snapshots:   0 total
155   Time:        42.389 s
156   Ran all test suites.
157   @A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $
```

7.6. A frontend

Az `algokit` által generált frontend alkalmazás a React keretrendszer használatát feltételezi.

A generált React alkalmazás tartalmazza pl. a különféle pénztárcák kezelését végző modult (walletconnect interface-t használva), illetve példát ad egy Algorand app létrehozására, ill. metódusainak meghívására. Nem végzi el a fejlesztő helyett a grafikus felület megtervezését, és a grafikus felületen belül az Algorand app-ok megfelelő metódusainak helyes meghívását sem.

Szerencsére a `contract` kifejlesztése során megírt tesztek jól használhatók a React komponensek kifejlesztése során is, mert a megírandó kódot már nagyrészt tartalmazzák. Mi több, a TealScript contract fejlesztése során lehetőség van a frontend React komponensek vázának automatikus generálására is, az `npm run generate-components` parancs futtatásával. Megjegyzés: Ezt a generátort a jövőben nem kívánják tovább fejleszteni, kivezetésre fog kerülni.

Természetesen előfordul, hogy a tesztekhez képest a kód átszervezésére vagy jelentős bővítésére van szükség. A tesztek során pl. külön történt az Algorand app létrehozása (createApplication) és felparaméterezése (bootstrap), míg a React alkalmazásban mindez egy lépésbe vonható össze. Egy másik példa: a Biz. Kör. érme visszahívásakor (clawback) a tesztek során csak egy érmét kezeltünk. A React frontend viszont végignézi, hogy (1) milyen számlaszámok rendelkeznek az érmével, és (2) ezen számlaszámok között van-e olyan, amelyre már lejárt a türelmi idő. Végül egy ciklusban visszahívja azokat az érméket, melyek túl régóta nem lettek felhasználva a tulajdonrész megvételére.

A React frontend megértéséhez szükség van a React alapjainak ismeretére. A React keretrendszerről a fejlesztők többek között a következő forrásokból tudnak tájékozódni:

- Nathan Rozentals: Mastering TypeScript. Build enterprise-ready, modular web applications using TypeScript 4 and modern frameworks.

Packt, 4th edition, Chapter 12: React

- Adam Boduch, Roy Derks: React and React Native. A complete hands-on guide to modern web and mobile development with React.js, Packt, 3rd edition
- Ûdemy, Complete React Developer (w/ Redux, Hooks, GraphQL), 36 rész, 42 óra videó, lásd `udemy.com`

7.6.1. A frontend oldali npm scriptek

A `package.json` fájlban található script-ek:

```
$ cd biz_kor/projects/biz_kor-frontend
$ cat package.json
...
  "scripts": {
    "generate:app-clients": "algokit project link --all",
    "dev": "npm run generate:app-clients && vite",
    "build": "npm run generate:app-clients && tsc && vite build",
    "test": "jest --coverage --passWithNoTests",
    "playwright:test": "playwright test",
    "lint": "eslint src --ext ts,tsx --report-unused-disable-directives
    ↪  --max-warnings 0",
    "lint:fix": "eslint src --ext ts,tsx
    ↪  --report-unused-disable-directives --max-warnings 0 --fix",
    "preview": "vite preview"
  },
...
```

A nekik megfelelő `npm` parancsok:

- `npm run generate:app-clients`, ami a `contract`-okhoz tartozó "typed client wrapper" file-okat generálja le, és teszi a `src/contracts` könyvtárba. Emlékeztetjük rá az Olvasót, hogy ez egy TypeScript-ben megírt wrapper réteg, amely sokkal könnyebbé teszi az Algorand okosszerződések kezelését: az app létrehozását, módosítását, a metódusok meghívását, a tranzakciók vagy tranzakció csoportok létrehozását,

aláírását stb. Mindezt már láttuk az okosszerződés tesztelésekor.

- `npm run dev`, ami gerálja a "typed client wrapper" fájlokat, majd elindítja a vite frontend fejlesztői rendszert. Ebben a „fejlesztői módban" a React alkalmazáshoz tartozó React forrás fájlok változásakor azonnal megtörténik az alkalmazás újrafordítása, és a React alkalmazást a vite szerver a `http://localhost:5137` url-en teszi elérhetővé a böngészőből.
- `npm run build`, ami gerálja a "typed client wrapper" fájlokat, majd elidítja a vite build parancsát, ami a rollup.js web packer segítségével a kész React alkalmazás kismillió fájlját néhány nagyobb `js` és `css` fájlba pakolja össze. Az eredmény a `dist` könyvtárba kerül.
- `npm run test`, ami a *.spec.tx kiterjesztésű Jest teszt fájlokat futtatja. Egy ilyen van pl. a `biz_kor-frontend/src/utils` könyvtárban, az `ellipseAddress.ts` tesztelésére, mintaként.
- `npm run playwright.test`, ami a böngésző szintjént teszteli az alkalmazást. A minta teszt file a `biz_kor-frontend/test/example.spec.ts` fájlban található. A futtatáshoz Python környezet is szükséges. Jelenleg ennek hiánya miatt elszáll.
- `npm run lint`, ami a formázási és szintaktikai hibákat ellenőrzi.
- `npm run lint:fix`, ami a formázási és szintaktikai hibákat ellenőrzi és javítja.
- `npm run preview`, ami elindítja a vite web szervert oly módon, hogy a `dist` jönyvtárban lévő, összepakolt React alkalmazást teszi elérhetővé a böngésző számára a `http://localhost:4173/` url-en.

7.6.2. A fronend fejlesztés menete

A `bizkor-frontend` fejlesztésekor mind a `contract`, mind a `frontend` használja a generált "typed client" fájlt. A wrapper generálása a `frontend` oldalon minden script parancsba be van építve:

```
"scripts": {
    "generate:app-clients": "algokit project link --all",
    "dev": "npm run generate:app-clients && vite",
    "build": "npm run generate:app-clients && tsc && vite build",
    "preview": "vite preview"
  }
```

A `generate:app-clients` hatására történik meg a wrapper átmásolása a `frontend/src/contract` könyvtárba. A többi parancs is meghívja ezt a parancsot.

A TealScript `contract`-ok fejlesztésekor lehetőség van a `frontend` React komponensek automatikus generálására is, az `npm run generate-components` parancs futtatásával. Megjegyzés: Ezt a generátort a jövőben nem kívánják tovább fejleszteni, kivezetésre fog kerülni.

7.6.3. Algorand specifikus részek

- pénztárca kezelés
- számlaszám kezelés
- algod, kmd, indexer kezelés

A fejlesztés során felmerült problémák és kérdések:

- Verziókezelés: Az `@algorandfoundation/tealscript` csomag `"latest"` verzióval szerepel a `package.json` fájlban. Csak a `package-lock.json` fájl tartalmazza a konkrétan használt verziót. Vigyázat: ha a package-lock.json fájlt töröljük, akkor előfordulhat, hogy az `npm install` már nem ugyanazt a verziójú TealScript-et installálja. Ez gondokat okozhat, ha pl. a hibakezelés a pc alapján szeretne felhasználóbarát hibaüzeneteket kiírni, hiszen a TealScript újabb verziója másképpen generálhatja a TEAL kódot. Megoldás: (1) A `package-lock.json` file megtartása, vagy (2) az adott esetben használt TealScript verzió beírása a `package.json` fájlba.

- Generátor formázási probléma: Ha az Algorand TealScript szerződés `ABC` alakú, csupa nagybetűt tartalazó név, akkor a generált typed client fájlneve `ABCClient.ts` lesz, de beimportálni `AbcClient` névvel tudjuk.
- Generátor formázási probléma: Az Algorand TealScript app-ban használt `abc_def` alakú, aláhúzás karaktert tartalmazó (globális) változók a `frontend` typed clientben `abcDef` változókként hivatkozhatók.
- React komponens tervezési kérdés: A "call bootstrap" gombra ill. komponensre külön nincs szükség, a "create DAO" komponensbe be lehet tenni a "bootstrap" metódus hívását is.
- React specifikus feladat: A "call bootstrap" hívása után frissíteni kell az "egy zseton ára", ill. "zsetonok száma" állapotváltozókat, amik a szülő komponensben vannak tárolva. Megoldása: a szülő komponens event handler-jét lett props-ként átadtam a gyermek komponensnek, majd meghívtam a gyermek komponensből. A szülő komponens event handler-jében az állapot megfrissíthető.

```
// src/components/BizKorBootstrap.tsx
type Props = {
  buttonClass: string
  //...
  onClick?: React.MouseEventHandler<HTMLButtonElement>
}
//...
    if (props.onClick) {
      props.onClick(event);
    }
```

```
// src/Home.tsx
  // Get the price of tokens from app and store in state
  const getPrice = async () => {
    try {
      const state = await typedClient.getGlobalState()
      setPrice(state.asaPrice!.asNumber())
    } catch (e: any) {
```

```
      if (e.message !== "Couldn't find global state") {
        console.warn(e)
      }
      setPrice(0)
    }
  }

  const handleBootstrapButtonClick = async () => {
    console.log('handleCalwbackButtonClick is called')
    await getPrice();
  };

  //...
  <BizKorBootstrap
    //...
    onClick={handleBootstrapButtonClick}
```

- Algorand specifikus kérdés: hogyan lehet egy algod kliens hivatkozást szerezni?

  Megoldás:

```
import { getAlgodConfigFromViteEnvironment } from
↪  '../utils/network/getAlgoClientConfigs'
import * as algokit from '@algorandfoundation/algokit-utils'

    const algodConfig = getAlgodConfigFromViteEnvironment()
    const algod = algokit.getAlgoClient({
      server: algodConfig.server,
      port: algodConfig.port,
      token: algodConfig.token,
    })
```

- Algorand specifikus kérdés: Hogyan lehet az optIn tranzakciót elküldeni?

  Megoldás: a globális állapotból megállapítjuk az asset ID-t (4.sor), előállítunk egy olyan tranzakciót, amely 0 egységet küld az adott asset-ből (5..11. sor), és a sendTransaction-nal aláírjuk és elküldjük a

tranzakciót (12. sor).

```
1     console.log(`Opt in to asset`)
2     const params = await algod.getTransactionParams().do();
3     const globalState = await props.typedClient.getGlobalState();
4     const asset = globalState.asaId!.asNumber();
5     const txn1 =
      ↪  algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
6         from: activeAddress!,
7         to: activeAddress!,
8         amount: 0,
9         assetIndex: asset,
10        suggestedParams: params,
11      });
12    algokit.sendTransaction({transaction: txn1, from: sender },
      ↪  algod);
13    //await algokit.waitForConfirmation(txn1.txID(), 4, algod);
```

Megjegyzés: az, hogy az `algokit.sendTransaction` várakozik-e a txn elküldése után, vagy sem, az algokit beállításoktól függ.

- Algorand specifikus kérdés: Hogyan lehet a buyAsset txn-t elküldeni?

  Megoldás: a typed client "compose" metódusának meghívása szükséges, mert egy *tranzakciós csoport* előállítására van szükség. Ebben az első tranzakció a `payment` tranzakció, a második maga az Algorand okosszerződés `buyAsset` metódusának a hívása (1..12. sor). Az `execute` metódus (12. sor) aláírja a tranzakciós csoportban lévő tranzakciókat. Megjegyzés: meg kell adnunk a foreign assets tömbben (11. sor, `assets: [...]`) a Biz.Kör. érme asset ID-jét is, különben a hívás hibaüzenettel elszáll: "Asset not found".

```
1       const result = await props.typedClient.compose().buyAsset(
2         {
3           payment: tx1,
4         },
5         {
6           sender: sender,
7           sendParams: {
```

```
            fee: algokit.transactionFees(5),
          },
          assets: [Number(asset)],
        }
      ).execute();
      const waitRoundsToConfirm = 4;
      await algokit.waitForConfirmation(result.txIds[0],
      ↪  waitRoundsToConfirm, algod);
```

- React specifikus kérdés: Hogyan adhatunk át egy payment txn-t propként?

  Megoldás: @todo

- Algorand specifikus kérdés: Hogyan lehet az `appCreatorAddress`-t visszanyerni?

  Megoldás:

```
    const appRef = await
    ↪  props.typedClient.appClient.getAppReference();
    const appAddr = appRef.appAddress;
```

- Algorand specifikus kérdés: Hogyan lehet a `price`-t visszanyerni?

  Megoldás: a globális állapotból:

```
    const price = globalState.asaPrice!.asNumber();
```

- Algorand specifikus probléma: Hogyan lehet felhasználóbarát hibaüzeneteket küldeni?

  Megoldás: A map fájlból kikereshető, hogy melyik `assert`-hez milyen TEAL pc tartozik. Hiba felléptekor a pc értéke alapján felhasználóbarát hibaüzenet iratható ki:

```
    try {
      enqueueSnackbar('A vételi tranzakció elküldése...', {
      ↪  variant: 'info' })
      const result = await props.typedClient.compose().buyAsset(
```

```
4       //...
5       ).execute();
6       const waitRoundsToConfirm = 4;
7       await algokit.waitForConfirmation(result.txIds[0],
        ↪  waitRoundsToConfirm, algod);
8       enqueueSnackbar(`A vételi tranzakció elküldve:
        ↪  ${result.txIds[0]}`, { variant: 'success' })
9     } catch(e: any) {
10      const msg='Nem sikerült a tranzakció elküldése';
11      if (e.response.body.data.pc === 460) {
12        enqueueSnackbar(`${msg}, mert a tranzakció típusa nem
          ↪  fizetési tranzakció`, { variant: 'error' })
13      }
14      else if (e.response.body.data.pc === 475) {
15        enqueueSnackbar(`${msg}, mert véget ért az értékesítési
          ↪  időszak`, { variant: 'error' })
16      }
17      else if (e.response.body.data.pc === 486) {
18        enqueueSnackbar(`${msg}, mert Ön már rendelkezik ezzel a
          ↪  zsetonnal`, { variant: 'error' })
19      }
20      else if (e.response.body.data.pc === 494) {
21      //...
```

- elvi probléma: a .env file-ban az admin_mode és és az app_id változók szolgáltak arra, hogy szabályozzák, hogy (1) milyen szerepkörben vagyunk: a szerződés létrehozója esetén az `admin_mode="true"`, míg a normál felhasználó esetén `"false"`; (2) létre kell-e hozni a szerződést (`app_id=0`), vagy a szerződés már korábban létre lett hozva (`app_id=nnn`).

  Sajnos, a kész alkalmazásba ez a fájl is beépül, emiatt ez a megoldás nem használható.

  Megoldás: processz env változók használata?

  @todo

8. Fejlesztői környezet upgrade, 2024. okt. 29.

8.1. git upgrade

A `winget list` paranccsal állapítható meg, hogy van-e frissebb változat a korábban installált programokból, esetünkben a git-ből.

A git upgrade megkezdése előtt a VS Code-ban be kell csukni a nyitott munkaterületet. Az upgrade-et `PS`, Power Shell alatt kell elvégezni, mert a `bash` fogja a git-et. Az upgrade során a VS Code termináljában használt parancsok:

```
winget list --id Git.Git

winget upgrade --id Git.Git

git --version
```

Az upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id Git.Git
Name Id      Version Available Source
-------------------------------------
Git  Git.Git 2.44.0  2.47.0    winget
PS C:\Users\lipi.FIO> winget upgrade --id Git.Git
Found Git [Git.Git] Version 2.47.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Successfully verified installer hash
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\lipi.FIO> git --version
git version 2.47.0.windows.1
PS C:\Users\lipi.FIO>
```

8.2. Docker Desktop upgrade

A `Docker Desktop` upgrade-je során használt VS Code terminál parancsok:

```
winget list --id Docker.DockerDesktop
```

```
winget upgrade --exact --id Docker.DockerDesktop
```

A `Docker Desktop` upgrade 3–4 percig tart. Ezt követően a Windows újraindítása szükséges.

A `Docker Desktop` upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Available Source
Docker Desktop Docker.DockerDesktop 4.29.0  4.35.0    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id Docker.DockerDesktop
Found Docker Desktop [Docker.DockerDesktop] Version 4.35.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://desktop.docker.com/win/main/amd64/172550/Docker%20Des
↪  ktop%20Installer.exe
  [redacted]
  ↪  [redacted]  463 MB /  463
  ↪  MB
Successfully verified installer hash
Starting package install...
Successfully installed
```

A Windows újraindítása után el kell indítanunk a `Docker Desktop` ikonra történő dupla kattintással a `Docker Desktop`-ot. Ekkor a következő üzenetet kapjuk:

```
What's new in 4.35.0
- New installations of Docker Desktop for Windows now require a Windows
↪  version of 19045 or later
- Docker Desktop Support for Red Hat Enterprise Linux RHEL
- Volume Backup and Share is now generally available and can be found in
↪  the Volumes tab.
```

```
- Terminal support within Docker Desktop using system shells is now
↪  generally available
- Enables users to import and export a subset of multi-platform images
- Beta release of Docker VMM - the faster alternative to Apple's
↪  Virtualization Framework on MacOS
```

A verzószám ellenőrzéséhez adjuk ki a következő parancsot:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Available Source
Docker Desktop Docker.DockerDesktop 4.35.0  4.35.0    winget
```

8.3. node.js upgrade

A node.js upgrade-je során használt parancsok:

```
winget list --id OpenJS.NodeJS
winget upgrade --exact --id OpenJS.NodeJS
node --version
npm --version
```

A node.js upgrade naplója a következő:

```
PS C:\Users\lipi.FIO> winget list --exact --id  OpenJS.NodeJS
Name    Id            Version Available Source
----------------------------------------------
Node.js OpenJS.NodeJS 21.7.3  23.1.0    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id  OpenJS.NodeJS
Found Node.js [OpenJS.NodeJS] Version 23.1.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://nodejs.org/dist/v23.1.0/node-v23.1.0-x64.msi
  [redacted] ]
  ↪  [redacted]  29.5 MB / 29.5
  ↪  MB
Successfully verified installer hash
```

```
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\lipi.FIO\Downloads> node --version
v23.1.0
PS C:\Users\lipi.FIO\Downloads> npm --version
10.9.0
```

8.4. Python upgrade

A `winget list python.python.3` paranccsal állapítsuk meg az installált Python verziószámát. Ez a `Python 3.12.3` volt.

Nézzük meg a `https://www.python.org/downloads/` webhelyen, hogy mi az aktuálisan elérhető Python verziószáma. 2024. október 29-én a Python 3.13 volt elérhető.[5]

Az upgrade során használt parancsok:

`winget uninstall python.python.3.12`

`winget install python.python.3.13`

A `VS Code` újraindítása után:

`python --version`

`=> 3.13.0`

Megjegyzés: Hiba esetén érdemes a Path környezeti változót is ellenőrizni, mind a Felhasználó, mind a Rendszer részben, hogy nem maradt-e ott valahol az előző Python verzió útvonala.

8.5. pipx upgrade

`python -m pip uninstalll pipx`

A `c:\Users\Lipi.FIO\pipx` könyvtár törlése, majd:

[5] Ez nem stabil verzó. Stabil verzó telepítését javaslom.

```
python -m pip install pipx
```

8.6. algokit újrainstallálás

```
pipx install algokit
```

Ezt követően némelyik vírusellenőrző program, pl. a Bitdefender bizonyos file-okat fertőzöttnek vél, és karanténba helyez, pl.

```
File C:\Users\Lipi.FIO\pipx\venvs\algokit\Lib\site-packages\pywin32_system32\pythoncom313.dll file is infected,
Gen:Variant.Tedy.659017
Bitdefender also quaranteened the pywintypes313.dll file.
```

A karanténból ezeket a fájlokat vissza kell tenni az eredeti helyükre. Bitdefender esetén például ez a következőképpen tehető meg:

„Védelem” | „Beállítások” | „Karanténbe helyezett fenyegetések”

A „Karantén kezelése” gomb megnyomásakor a Bitdefender kiírta, hogy milyen fájlok vannak karanténban.

Két file-t állítottam vissza: `pythoncom313.dll` és `pywintypes313.dll`.

A virustotal web helyen is ellenőrizhetők ezek a fájlok. A biztonsági cégeknek csupán töredéke (9/71) véli ezeket a fájlokat fertőzöttnek.

8.7. Utómunkák

1. Az `algokit init` a Python upgrade után nem találja a `python3.EXE` fájlt:

```
algokit init
...
Unhandled CalledProcessError: Command '"C:\Users\lipi.FIO\AppData\Local\Microsoft\WindowsApps\python3.EXE" pre_init.py' returned non-zero exit status 9009.
```

A helyes működéshez egy Admin módban elindított PowerShell-ben ki

kell adnunk a következő parancsot:

```
mklink
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python3.exe
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python.exe
```

2. Az `algokit init` által generált frontend első futtatása során a következő hibát kapjuk:

```
npm run dev
...
file:///C:/Users/lipi.FIO/Downloads/2024/10/ALGO/Hackaton_FR/circle_of_tr⌋
↪  ust/ct/projects/ct-frontend/tailwind.config.js:2
module.exports = {
^

ReferenceError: module is not defined
```

A megoldás az, hogy a `tailwind.config.js` fájlt átnevezzük oly módon, hogy a file kiterjesztése `cjs` legyen: `tailwind.config.cjs`.

3. Az upgrade után célszerű a `AppData\Local\Temp` könyvtárban lévő fájlok törlése. Ezzel kb. 3,5 GByte hely szabadítható fel.

9. Fejlesztői környezet upgrade, 2024. dec. 17.

9.1. git upgrade

A `winget list` paranccsal állapítható meg, hogy van-e frissebb változat a git-ből.

A git upgrade megkezdése előtt a VS Code-ban be kell csukni a nyitott munkaterületet. A git-et a VS Code-ban Power Shell alatt kell installálni, mert a VS Code Bash shell használja és lock-olja a git-et.

Az upgrade során a VS Code termináljában használt parancsok:

```
winget list --id Git.Git
```

```
winget upgrade --exact --id Git.Git
```

```
git --version
```

Az upgrade naplója:

```
1  PS C:\Users\lipi.FIO> winget list --id Git.Git
2  Name Id      Version Available Source
3  -------------------------------------
4  Git  Git.Git 2.47.0  2.47.1    winget
5  Found Git [Git.Git] Version 2.47.1
6  PS C:\Users\lipi.FIO> winget upgrade --exact --id Git.Git
7  Found Git [Git.Git] Version 2.47.1
8  This application is licensed to you by its owner.
9  Microsoft is not responsible for, nor does it grant any licenses to,
   ↪  third-party packages.
10 Successfully verified installer hash
11 Starting package install...
12 The installer will request to run as administrator, expect a prompt.
13 Successfully installed
14 PS C:\Users\lipi.FIO> git --version
15 git version 2.47.1.windows.1
```

9.2. Docker Desktop upgrade

A `winget list` paranccsal állapítható meg, hogy van-e frissebb változat a Docker Desktop-ból.

Az upgrade során használt VS Code terminálparancsok:

```
winget list --id Docker.DockerDesktop
```

```
winget upgrade --exact --id Docker.DockerDesktop
```

A `Docker Desktop` upgrade 3–4 percig tart. Ezt követően általában a Windows újraindítása szükséges.

A `Docker Desktop` upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Available Source
-----------------------------------------------------------
Docker Desktop Docker.DockerDesktop 4.35.0  4.36.0    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id Docker.DockerDesktop
Found Docker Desktop [Docker.DockerDesktop] Version 4.36.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://desktop.docker.com/win/main/amd64/175267/Docker%20Des
↪  ktop%20Installer.exe
  [redacted]
  ↪  [redacted]  494 MB /  494
  ↪  MB
Successfully verified installer hash
Starting package install...
Successfully installed
PS C:\Users\lipi.FIO>
```

Az installálás után automatikusan elindult a Docker Desktop, és már az új verziót futtatta:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Source
-----------------------------------------------------
```

```
Docker Desktop Docker.DockerDesktop 4.36.0  winget
```

9.3. node.js upgrade

A node.js upgrade-je során használt parancsok:

```
winget list --id OpenJS.NodeJS
```

```
winget upgrade --exact --id OpenJS.NodeJS
```

```
node --version
```

```
npm --version
```

A node.js upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id OpenJS.NodeJS
Name    Id            Version Available Source
-----------------------------------------------
Node.js OpenJS.NodeJS 23.1.0  23.4.0    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id OpenJS.NodeJS
Found Node.js [OpenJS.NodeJS] Version 23.4.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://nodejs.org/dist/v23.4.0/node-v23.4.0-x64.msi
  [                                                                  ]
  ↪  [redacted]  29.8 MB / 29.8
  ↪  MB
Successfully verified installer hash
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\lipi.FIO> node -v
v23.4.0
PS C:\Users\lipi.FIO> npm -v
10.9.2
PS C:\Users\lipi.FIO>
```

9.4. Python upgrade

A `winget list --id Python.Python` paranccsal vizsgálható meg, hogy van-e frissebb Python verzió.

Az upgrade során használt parancsok:

`winget list --id Python.Python`

`python --version`

Megjegyzés: Nem volt újabb Python verzió.

Megjegyzés: Ahhoz, hogy helyesen működjön a `python3 -m venv env` parancs, egy link-et kell létrehozni, lásd utómunkák.

A napló:

```
PS C:\Users\lipi.FIO> winget list --id Python.Python
Name                   Id                  Version Source
-----------------------------------------------------------
Python 3.13.1 (64-bit) Python.Python.3.13 3.13.1  winget
PS C:\Users\lipi.FIO> python --version
Python 3.13.1
PS C:\Users\lipi.FIO>
```

9.5. pipx upgrade

Nem volt újabb Python verzió, ezért nem volt szükség pipx upgrade-re. Ha szükséges lett volna, akkor:

`python -m pip uninstalll pipx`

A `c:\Users\Lipi.FIO\pipx` könyvtár törlése, majd:

`python -m pip install pipx`

9.6. algokit upgrade

A következő csomagok lettek a `pipx` segítségével upgrade-elve: `algokit`, `poetry`, `tealer`. A használt parancsok:

```
pipx upgrade algokit

pipx upgrade poetry

pipx upgrade tealer
```

A napló:

```
PS C:\Users\lipi.FIO> pipx list
venvs are in C:\Users\lipi.FIO\pipx\venvs
apps are exposed on your $PATH at C:\Users\lipi.FIO\.local\bin
manual pages are exposed at C:\Users\lipi.FIO\.local\share\man
   package algokit 2.4.3, installed using Python 3.13.0
    - algokit.exe
   package poetry 1.8.4, installed using Python 3.13.0
    - poetry.exe
   package tealer 0.1.2, installed using Python 3.13.0
    - tealer.exe
PS C:\Users\lipi.FIO> pipx upgrade algokit
Overwriting file C:\Users\lipi.FIO\.local\bin\algokit.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\algokit\Scripts\algokit.exe
upgraded package algokit from 2.4.3 to 2.5.1 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\algokit)
PS C:\Users\lipi.FIO> algokit --version
algokit, version 2.5.1
PS C:\Users\lipi.FIO> pipx upgrade poetry
Overwriting file C:\Users\lipi.FIO\.local\bin\poetry.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\poetry\Scripts\poetry.exe
upgraded package poetry from 1.8.4 to 1.8.5 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\poetry)
PS C:\Users\lipi.FIO> pipx upgrade tealer
Overwriting file C:\Users\lipi.FIO\.local\bin\tealer.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\tealer\Scripts\tealer.exe
tealer is already at latest version 0.1.2 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\tealer)
PS C:\Users\lipi.FIO>
```

9.7. Utómunkák

1. Az `python3 -m venv env` parancs helytelenül működött. A helyes működéshez egy Admin módban elindított PowerShell-ben ki kell adnunk a következő parancsot:

```
mklink
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python3.exe
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python.exe
```

2. Az upgrade után célszerű a `AppData\Local\Temp` könyvtárban lévő fájlok törlése, egy Admin módban elindított Total Commander segítségével.

3. Problémák esetén az `algokit doctor` paranccsal lehet kíratni a verziószámokat:

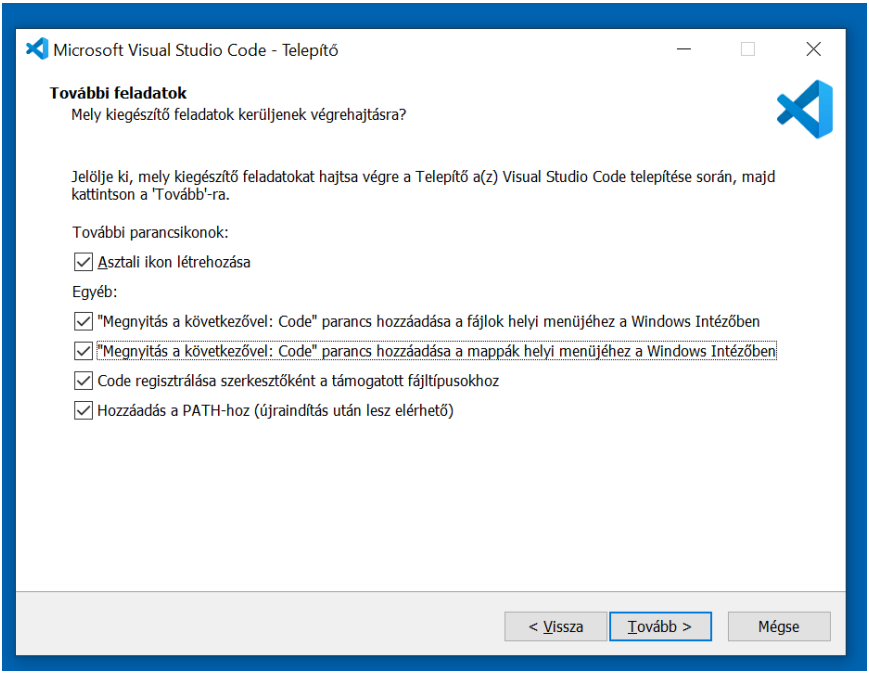
```
PS C:\Users\lipi.FIO> algokit  doctor
timestamp: 2024-12-17T02:22:24+00:00
AlgoKit: 2.5.1
AlgoKit Python: 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC
↪  v.1942 64 bit (AMD64)] (location:
↪  C:\Users\lipi.FIO\pipx\venvs\algokit)
OS: Windows-10-10.0.19045-SP0
docker: 27.3.1
docker compose: 2.30.3-desktop.1
git: 2.47.1.windows.1
python: 3.13.1 (location:
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python.EXE)
python3: 3.13.1 (location:
↪  C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python3.EXE)
pipx: 1.7.1
poetry: 1.8.5
node: 23.4.0
npm: 10.9.2
winget: 1.9.25200

If you are experiencing a problem with AlgoKit, feel free to submit an
↪  issue via:
https://github.com/algorandfoundation/algokit-cli/issues/new
```

```
Please include this output, if you want to populate this message in your
↪  clipboard, run `algokit doctor -c`
PS C:\Users\lipi.FIO>
```

10. Fejlesztői környezet upgrade, 2025. jan. 09.

10.1. `VS Code` újratelepítés

Az újratelepítés azért történt, mert nem működött az F5-tel a js programok debug-ja, csak a `node --inspect-brk js_script` és az attach `launch.json` segítségével. A problémát végül nem a `VS Code` újratelepítése, hanem a `node.js` újratelepítése oldotta meg.



5. ábra. VS Code install: kiegészítő feladatok

A `https://code.visualstudio.com/download` címről egy Windows x64 system installer letöltése. A régebbi változat eltávolítása: Win + R, `appwiz.cpl` futtatása után jobb egérgombbal "VS Code"-ra kattintás, eltávolítás.

Ezután a system installer-re kattintva megkezdhető a `VS Code` telepítés. A licencfeltételek elfogadása után érdemes bekattintani, hogy az installer hozzon létre egy ikont az asztalon, ill. hogy a VS Code legyen elindítható a `Code` paranccsal, lásd 5. ábra.

10.2. git upgrade

A `winget list` paranccsal állapítható meg, hogy van-e frissebb változat a git-ből.

A git upgrade megkezdése előtt a VS Code-ban be kell csukni a nyitott munkaterületet. A git-et a VS Code-ban Power Shell alatt kell installálni, mert a VS Code Bash shell használja és lock-olja a git-et.

Az upgrade során a VS Code termináljában használható parancsok:

```
winget list --id Git.Git
```

```
winget upgrade --exact --id Git.Git
```

```
git --version
```

Nem volt a Git-ből újabb változat. A napló:

```
1 PS C:\Users\lipi.FIO> winget list --id Git.Git
2 Name Id      Version Source
3 ---------------------------
4 Git  Git.Git 2.47.1  winget
5 PS C:\Users\lipi.FIO>
```

10.3. Docker Desktop upgrade

A `winget list` paranccsal állapítható meg, hogy van-e frissebb változat a Docker Desktop-ból.

Az upgrade során használt VS Code terminálparancsok:

```
winget list --id Docker.DockerDesktop
```

```
winget upgrade --exact --id Docker.DockerDesktop
```

A `Docker Desktop` upgrade 3–4 percig tart. Ezt követően általában a Windows újraindítása szükséges.

A `Docker Desktop` upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Available Source
-----------------------------------------------------------
Docker Desktop Docker.DockerDesktop 4.36.0  4.37.1    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id Docker.DockerDesktop
Found Docker Desktop [Docker.DockerDesktop] Version 4.37.1
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://desktop.docker.com/win/main/amd64/178610/Docker%20Des
↪  ktop%20Installer.exe
  [redacted]
  ↪  [redacted]  500 MB /  500
  ↪  MB
Successfully verified installer hash
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\lipi.FIO>
```

Az installálás után automatikusan elindult a Docker Desktop, és már az új verziót futtatta:

```
PS C:\Users\lipi.FIO> winget list --id Docker.DockerDesktop
Name           Id                   Version Source
-----------------------------------------------------
Docker Desktop Docker.DockerDesktop 4.37.1  winget
```

10.4. node.js upgrade

A node.js upgrade-je során használt parancsok:

```
winget list --id OpenJS.NodeJS
```

`winget upgrade --exact --id OpenJS.NodeJS`

`node --version`

`npm --version`

A node.js upgrade naplója:

```
PS C:\Users\lipi.FIO> winget list --id OpenJS.NodeJS
Name    Id            Version Available Source
-----------------------------------------------
Node.js OpenJS.NodeJS 23.4.0  23.6.0    winget
PS C:\Users\lipi.FIO> winget upgrade --exact --id OpenJS.NodeJS
Found Node.js [OpenJS.NodeJS] Version 23.6.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to,
↪  third-party packages.
Downloading https://nodejs.org/dist/v23.6.0/node-v23.6.0-x64.msi
  [                                                                    ]
  ↪  [          ]  29.9 MB / 29.9
  ↪  MB
Successfully verified installer hash
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\lipi.FIO> node -v
v23.6.0
PS C:\Users\lipi.FIO> npm -v
10.9.2
PS C:\Users\lipi.FIO>
```

10.5. Python upgrade

A `winget list --id Python.Python` paranccsal vizsgálható meg, hogy van-e frissebb Python verzió.

Az upgrade során használt parancsok:

`winget list --id Python.Python`

`python --version`

Megjegyzés: Nem volt újabb Python verzió.

A napló:

```
PS C:\Users\lipi.FIO> winget list --id Python.Python
Name                   Id                  Version Source
-----------------------------------------------------------
Python 3.13.1 (64-bit) Python.Python.3.13 3.13.1  winget
PS C:\Users\lipi.FIO> python --version
Python 3.13.1
PS C:\Users\lipi.FIO>
```

10.6. pipx upgrade

Nem volt újabb Python verzió, ezért nem volt szükség pipx upgrade-re. Ha szükséges lett volna, akkor:

```
python -m pip uninstalll pipx
```

A `c:\Users\Lipi.FIO\pipx` könyvtár törlése, majd:

```
python -m pip install pipx
```

10.7. algokit upgrade

A következő csomagok lettek a `pipx` segítségével upgrade-elve: `algokit`, `poetry`, `tealer`. A használt parancsok:

```
pipx upgrade algokit
```

```
pipx upgrade poetry
```

```
pipx upgrade tealer
```

A napló:

```
PS C:\Users\lipi.FIO> pipx list
venvs are in C:\Users\lipi.FIO\pipx\venvs
apps are exposed on your $PATH at C:\Users\lipi.FIO\.local\bin
manual pages are exposed at C:\Users\lipi.FIO\.local\share\man
   package algokit 2.5.1, installed using Python 3.13.0
```

```
    - algokit.exe
  package poetry 1.8.5, installed using Python 3.13.0
    - poetry.exe
  package tealer 0.1.2, installed using Python 3.13.0
    - tealer.exe
PS C:\Users\lipi.FIO> pipx upgrade algokit
Overwriting file C:\Users\lipi.FIO\.local\bin\algokit.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\algokit\Scripts\algokit.exe
upgraded package algokit from 2.5.1 to 2.5.2 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\algokit)
PS C:\Users\lipi.FIO> algokit --version
algokit, version 2.5.2
PS C:\Users\lipi.FIO> pipx upgrade poetry
Overwriting file C:\Users\lipi.FIO\.local\bin\poetry.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\poetry\Scripts\poetry.exe
upgraded package poetry from 1.8.5 to 2.0.0 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\poetry)
PS C:\Users\lipi.FIO> pipx upgrade tealer
Overwriting file C:\Users\lipi.FIO\.local\bin\tealer.exe with
↪  C:\Users\lipi.FIO\pipx\venvs\tealer\Scripts\tealer.exe
tealer is already at latest version 0.1.2 (location:
↪  C:\Users\lipi.FIO\pipx\venvs\tealer)
PS C:\Users\lipi.FIO>
```

10.8. Utómunkák

Az upgrade után célszerű a `AppData\Local\Temp` könyvtárban lévő fájlok törlése, egy Admin módban elindított Total Commander segítségével.