The Hitchhiker's Guide to Algorand

Written by Gábor Lipovszki

v0.5, Translated from the Hungarian version
dated from 05-Dec-2024 11:54

Contents

1 Introduction

This Guide is designed to help "hitchhikers", that is, developers, navigate the winding, labyrinthine world of the Algorand blockchain. It offers not only a theoretical overview but also illustrates various phenomena with examples.

Don't panic! – as the original Guide advises.

Stay far away from it! – my wife's opinion about Algorand.

Those who can, do; those who can't, teach. – a Hungarian proverb

1.1 A Few Words About Algorand

Algorand was created by Silvio Micali. He was motivated by the goal of ensuring that new blocks in the Algorand blockchain are created with minimal resource use, while maintaining robust security. To achieve this, he invented a new cryptographic primitive called the VRF (Verifiable Random Function). Using this innovation, consensus is established by selecting committee members at random during each phase with the help of the VRF. Every staked Algorand token (Algo) represents a vote in this selection process. Additionally, the VRF allows retrospective verification to ensure that the random selection was legitimate.

The name Algorand also comes from Professor Micali, formed by combining the words "algorithmic" and "random." A total of 10 billion Algo tokens were created simultaneously at the network's inception. Small investors can purchase Algo on various exchanges. Currently, approximately 8.3 billion Algo tokens are in circulation. The remaining supply is managed by the Algorand Foundation, which uses it to fund various programs, such as governance initiatives, development projects, support for DeFi (Decentralized Finance), and more.

The Algorand network began operating in 2019. As of October 2024, its maximum achievable speed is approximately 7 500 transactions per second, with transaction finality occurring in about 2.9 seconds. The probability of a blockchain fork is extremely low, at 10^{-18}, meaning a fork might occur only once in billions of years. The current transaction fee is 0.001 Algo, which, at the current Algo exchange rate, is roughly equivalent to 0.0005 USD.

Despite its excellent technical parameters, according to CoinCodex, Algorand currently ranks 72nd in market capitalization among digital currencies (as of October 2024). The community has not appreciated that the Algorand relay network is essentially operated by "insiders" even now, with the Algorand Foundation paying them several million dollars annually.

The CTO of the Algorand Foundation aims to address this issue in 2024 by introducing a reward system for block creation, effectively implementing a form of "mining" within the network. The expectation is that this change will allow the relay network to be gradually phased out, making Algorand a self-sustaining network. Since mining will be based on proof of stake, requiring the staking of Algo, the reward for block creation is anticipated to increase the current staking rate from around 15% to at least 25% this year. Additionally, the CTO hopes this initiative will help Algorand secure a position among the top 20 digital currencies.

Since transaction fees are extremely low and the aim is to keep them low in the future, the Algorand Foundation would subsidize mining rewards derived from transaction fees during the first

2–3 years. The CTO hopes that factors such as an increase in the Algorand token's value, growth in the number of network transactions, and a reasonable adjustment of transaction fees will make the mining process self-sustaining after this initial 2–3 year period.

The Algorand Foundation's 2024 development roadmap includes the following initiatives:

- Dynamic block time introduction: Excluding the slowest 5% of committee members during consensus, enabling further block time reduction.
- AlgoKit 2.0: Allowing Algorand smart contracts to be written natively in Python, which will eliminate the current developer bottleneck.
- Non-archival relay nodes: Reducing the amount of data stored on the network.
- Rewarding new block creation with "mining": Increasing the amount of staked Algo.
- Gradual transition from relay network topology to P2P (peer-to-peer) gossip topology: Creating a system similar to Bitcoin and other digital currencies.

1.2 Online Resources

- The Algorand Technologies website (formerly Algorand, Inc.)
- The Algorand Foundation website
- The Algorand Forum
- Invitation to the Algorand Discord server
- Algorand resource collection: Awesome Algo
- Algorand blockchain explorers: lora, allo', pera
- Hungarian website: Algorand.hu

1.2.1 YouTube Videos for Developers

YouTube videos for developers are available at `https://youtube.com/@algodevs`. By clicking on the "Playlists" tab, you can browse videos organized by topic. Some examples include:

- Algorand Development Environment Setup
- AVM Explained (Algorand Virtual Machine Explained)
- PyTeal Tutorial for Beginners (Full Course)
- Beaker for Beginners (Full Course)
- TealScript for Beginners (Full Course)
- The Differences Between Ethereum & Algorand
- Bonus Content: Algorand Foundation CTO John Woods (Interviews with John Woods, CTO of Algorand Foundation)
- Beginner Algorand Bootcamp [April 2023]

1.3 Algorand Wallets

Users interact directly with the Algorand blockchain through an Algorand wallet. The quality and usability of the wallet significantly influence the user experience.

Wallets can be either software or hardware-based.

Hardware wallets are characterized by storing the private key in cryptographically secure memory, making it unreadable directly from the device. Hardware wallets use specialized hardware for transaction signing, requiring explicit user intervention (such as pressing a button) to approve a transaction.

One of the most well-known hardware wallet manufacturers is Ledger. The Ledger Live application serves as the interface for managing the Ledger hardware wallet. The Ledger hardware wallet supports a wide range of digital currencies, including Algorand.

The official Algorand wallet is the Pera Wallet. Pera offers two types of wallets: a web-based wallet and wallets implemented as iOS and Android applications. However, for security reasons, the web-based wallet has been deprecated and can now only be used to view account balances. The iOS and Android applications can be downloaded from the App Store or Google Play.

There are several features of the Algorand blockchain that the official Pera Wallet does not yet support or only partially supports. Some of these limitations include:

- The Ledger Nano X hardware wallet cannot be used via USB
- Multisig accounts are not currently supported in Pera
- Algorand applications cannot be directly managed within the Pera Wallet
- Backing up the Pera Wallet is not very convenient

Another wallet is the A-Wallet, which is a web-based wallet. The A-Wallet supports the use of Ledger hardware wallets. It offers many interesting features, such as:

- Multisig (multiple signature) support
- Integration with Google Authenticator for two-factor signing
- Payment gateway support
- Support for voting on the blockchain, among other features

1.4 Nodes in the Algorand Blockchain

If we wish to develop an application that uses the Algorand blockchain, the application must be able to access one of Algorand's public networks or establish its own (private) network node. In the Algorand blockchain, several types of network nodes exist:

- Relay nodes, which provide high-speed network traffic
- Archival nodes, which store the complete history of the Algorand blockchain. These nodes enable fast searches within the blockchain using a database manager called Indexer, built on PostgreSQL.
- Non-archival nodes, which, for storage efficiency, only retain the last 1 000 transactions for

each Algorand account.

Archival nodes can be accessed through a standard REST API interface. For instance, `nodely.io` provides free Algorand endpoints via the playground link, allowing users to test the functionality of REST API commands. Instead of using low-level curl commands, developers can conveniently access various REST API functions through SDKs (System Development Kits).

A developer may also choose to operate their own network node, which can be either an archival or a non-archival node. The hardware requirements for running these nodes are detailed at the aforementioned hyperlink.

A custom node can be installed on Linux or within a Docker environment. For Linux, both binary installation packages and package-based installers are available.

The components of an installed Algorand node include:

- algod: The Algorand daemon, responsible for managing the blockchain.
- kmd: The Key Management daemon, responsible for wallet management.
- indexer: Provides a database for fast access to blockchain data.
- conduit: Enables the delivery of Algorand blockchain data to external applications.
- Command-line tools, such as goal and kmd.

To simplify development, the Algorand Foundation has created AlgoKit, which sets up an Algorand node and a development-friendly environment within a Docker interface.

During development, the following types of Algorand blockchains can be used:

- Private blockchain: A single-node blockchain running on the developer's machine.
- Betanet: A test blockchain for initial experiments. It uses its own beta test Algo, which can be downloaded for free from https://bank.betanet.algodev.network/.
- Testnet: A test blockchain that closely resembles the mainnet. It uses its own test Algo, which can be downloaded for free from https://bank.testnet.algorand.network/.
- Mainnet: The live, production blockchain.

1.5 Interaction with the Algorand Blockchain

- Command-line Interface: If you operate your own node, the `algod`, `kmd`, and `indexer` processes can be managed using shell commands. The most commonly used commands are `goal` and `kmd`.
- REST API: The services of your own node are also accessible via a REST API interface. The `goal`, `kmd` and `indexer` processes each provide a REST API. To access these services, API keys are required. If these keys are made public, others can also use your node.
- SDKs: Algorand SDKs "wrap" the REST API interfaces, providing convenient access to the REST APIs of the processes from various programming environments (e.g., JavaScript, PHP, Java, Rust, etc.).

Operating your own node can be achieved in several ways:

- Setting up an Algorand blockchain node on Linux: Refer to Install a node for details, including using the update script, as described in Installing on Linux.
- Setting up a private Algorand blockchain node on Linux: Refer to Create a Private Network.
- Using Docker (and Docker Compose): Deploying an Algorand Sandbox. See Algorand Sandbox for instructions.

1.6 SDKs

The Algorand blockchain can be accessed from various programming environments using SDKs. The "official" SDKs provided by Algorand include:

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

Additional SDKs developed by the community include:

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

1.7 Algorand Development Tools

Algorand development tools are available in multiple layers, much like onion skins. This structure is partly due to historical reasons. The layers include:

- Binary level: The level of instructions for the Algorand Virtual Machine (AVM). The Algorand developer documentation provides a detailed description of the AVM architecture and the functionality of each instruction. See v10 opcodes.
- Assembly level: The TEAL assembly programming language (Transaction Execution and Approval Language). Assembly programs written in TEAL are compiled into AVM binary code using the TEAL compiler. The TEAL compiler is accessible both via the command-line interface and through the SDKs.
- Compiler level: The PyTeal compiler translates "pseudo"-Python code written in the Python language into TEAL. Its usability can be challenging due to the mix of standard Python code and the "pseudo"-Python code intended for the PyTeal compiler. However, compared to TEAL, it allows for much more readable code thanks to its support for control structures.
- Compiler level: The Beaker compiler is an advanced version of PyTeal that enables applications for the Algorand blockchain to be developed as Python objects. On the Algorand blockchain, an application is roughly equivalent to an Ethereum smart contract, though with

numerous restrictions designed to ensure the Algorand blockchain remains fast. Since Algorand blockchain applications only emerged 1–2 years after the "core" AVM, Beaker is necessarily a later "invention."

- Compiler level: The PuyaPy compiler converts pure Python code into TEAL, compiling directly to the assembly language level of the Algorand Virtual Machine.
- Compiler level: The TealScript compiler translates from a subset of TypeScript into TEAL.

The development tools are organized into frameworks, with AlgoKit being one of the most well-known. AlgoKit provides developers with:

- Access to the Algorand blockchain, which can be private, Betanet, Testnet, or Mainnet. Blockchain access is managed using Docker and Docker Compose via Algorand Sandbox.
- Access to PyTeal, Beaker, and PuyaPy development tools, creating a virtual environment for them and handling dependencies.
- Access to the TealScript development tool.
- Sample projects for reference.
- Automatic user interface generation based on the Algorand application.

In addition to AlgoKit, the LORA blockchain explorer is recommended, as it can display not only transactions on Betanet, Testnet, and Mainnet but also those on private blockchains managed by Algorand Sandbox.

Another framework is TealCraft, which enables direct use of TealScript from within a browser.

2 Installing AlgoKit

AlgoKit is a framework developed by the Algorand Foundation, designed to facilitate the following:

- Setting up and managing a local (private) network: This includes starting, stopping, or connecting to the Algorand Betanet, Testnet, or Mainnet. See the `algokit local` command.
- Initializing development frameworks and loading sample projects: See the `algokit init` command.
- Creating a virtual environment for development: See the `algokit bootstrap all` command.

Prerequisites for Installing AlgoKit on Windows 10 or Windows 11:

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

A helpful summary of the installation steps can be found on YouTube. See Ryan Fox's video titled "Setup Your Algorand Development Environment on Windows in 10 Minutes".

Users registered on Github can access 120 free hours of virtualized environment per month under Codespaces. This environment is accessible via a web browser. AlgoKit can be installed in this environment with minimal steps since Codespaces automatically provides the following components:

- VS Code editor
- Git version control
- Docker and Docker Compose
- Node.js

2.1 Installing AlgoKit in Codespaces

The necessary steps are as follows:

- Log in to your account on github.com.
- Create a new repository.
- Navigate to the new repository. Click the Code button and select the Codespaces tab.
- Click the "Create codespace on main" button. This will create a new development environment with a VS Code editor.
- If the terminal tab doesn't appear, click on "+" and choose `Terminal | New Terminal`.
- In the terminal, run: `pipx install algokit`
- In the terminal, run: `algokit localnet start`

Within a minute or two, a local Algorand network node will be installed and running under Docker.

That's it! You can skip the Windows installation section, and proceed to the section demonstrating the use of AlgoKit with some examples.

2.2 Installing AlgoKit on Windows

2.2.1 Check Windows Version

Go to `Control Panel | System`. Check the OS version number. The minimum supported version is 19045.[1]

2.2.2 Install VS Code

Visit the VS Code Download page. Click the "Download, Windows 10, Windows 11" button. Run the downloaded file and follow the installation instructions. Launch VS Code.

2.2.3 Install Git

In VS Code, open a terminal by clicking on 'View | Terminal'. In the terminal, run:

```
git --version
```

If you receive an error message, run:

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

Verify the installation by running:

```
git --version
```

You should see output similar to:

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

2.2.4 Installing WSL (Windows Subsystem for Linux)

Installing WSL is necessary because Docker Desktop uses it to run Linux containers.

Open the terminal window in VS Code and run:

```
wsl -l -v
```

If you receive an error message, run:

```
wsl --install
```

When a new terminal window appears, you'll be prompted to set up a new UNIX username and password:

```
Enter new UNIX username: username
New password: password
Retype password: password again
```

In the Windows Start menu, type Ubuntu and launch the Ubuntu application.

[1] docker.desktop 4.35.0 now requires Windows version 19045 or later

Back in the VS Code terminal window, run the command again:

```
wsl -l -v
```

You should see output similar to:

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

2.2.5 Installing Docker Desktop

In the VS Code terminal window, run:

```
docker version
```

If you receive an error message, run:

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

Once the installation is complete, restart Windows.

Click the Docker Desktop icon to launch Docker. Accept the terms and conditions by clicking "Accept". Note: The "Accept" button may be hidden behind the startup window.

In the VS Code terminal window, run again:

```
docker version
```

This time, you should not see any error messages.

2.2.6 Installing Node.js

Installing Node.js is required for TealScript and React frontend functionality. In the VS Code terminal window, run:

```
node --version
```

If you receive an error message, run:

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

To verify the installation, run the following commands:

```
node --version

# => v21.7.3

npm --version

# => 10.5.0
```

2.2.7 Installing Python

Python installation is required for the PuyaPy compiler, as well as for earlier tools like PyTeal and Beaker, which also use Python code. In the VS Code terminal window, run:

```
python --version
```

If you receive an error message, run:

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

Note: It is recommended to install the latest version of Python, but at a minimum, you should install Python 3.11.

2.2.8 Installing pipx

Close VS Code, then restart it to ensure the python command is recognized in the PATH. In the VS Code terminal window, run:

```
pipx --version
```

If you receive an error message, run:

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

In the VS Code terminal window, run again:

```
pipx --version
```

```
# => 1.5.0
```

This time, you should not see any error messages.

2.2.9 Installing AlgoKit

In the VS Code terminal window, run:

```
pipx install algokit
```

2.3 Using AlgoKit

Verify that the installation was successful. In the Codespaces or VS Code terminal window, run:

```
algokit --version
```

```
# => algokit, version 2.0.3
```

Start the local Algorand network environment:

```
algokit localnet start
```

Check if the containers have successfully started under the Docker environment. On Windows: Open the Docker Desktop window to view the containers and running images. In the Codespaces environment: Run the following command:

```
docker container ls
```

You should see four images: `algorand/indexer:latest`, `algorand/conduit:latest`, `postgres:13-alpine`, `algorand/algod:latest`.

These containers run the indexer, the conduit (an event manager), the Postgres database manager, and the algod Algorand daemon.

The proper functioning of the Docker images can also be verified with the following command:

```
algokit localnet status
```

Example:

```
@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_postgres  Creating
docker: Container algokit_sandbox_algod  Created
docker: Container algokit_sandbox_postgres  Created
docker: Container algokit_sandbox_conduit  Creating
docker: Container algokit_sandbox_conduit  Created
docker: Container algokit_sandbox_indexer  Creating
docker: Container algokit_sandbox_indexer  Created
docker: Container algokit_sandbox_algod  Starting
docker: Container algokit_sandbox_postgres  Starting
docker: Container algokit_sandbox_postgres  Started
docker: Container algokit_sandbox_algod  Started
docker: Container algokit_sandbox_conduit  Starting
docker: Container algokit_sandbox_conduit  Started
docker: Container algokit_sandbox_indexer  Starting
docker: Container algokit_sandbox_indexer  Started
docker: Container algokit_sandbox_postgres  Waiting
docker: Container algokit_sandbox_indexer  Waiting
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
0c7f8f8fa8ef   algorand/indexer:latest   "docker-entrypoint.s..."   5 minutes ago   Up 5 minutes
↪  0.0.0.0:8980->8980/tcp, :::8980->8980/tcp
↪                                                                algokit_sandbox_indexer
2c45e5017538   algorand/conduit:latest   "docker-entrypoint.sh"   5 minutes ago   Up 5 minutes
↪
↪                                                               algokit_sandbox_conduit
0b590c17237f   postgres:13-alpine        "docker-entrypoint.s..."   5 minutes ago   Up 5 minutes
↪  0.0.0.0:5443->5432/tcp, :::5443->5432/tcp
↪                                                                algokit_sandbox_postgres
```

```
ea05aa8270b7   algorand/algod:latest     "/node/run/run.sh"       5 minutes ago   Up 5 minutes
↪  4160/tcp, 9100/tcp, 0.0.0.0:9392->9392/tcp, :::9392->9392/tcp, 0.0.0.0:4002->7833/tcp,
↪  :::4002->7833/tcp, 0.0.0.0:4001->8080/tcp, :::4001->8080/tcp   algokit_sandbox_algod
@A-Maugli → /workspaces (main) $ algokit explore
Opening localnet in https://app.dappflow.org using default browser
@A-Maugli → /workspaces (main) $ algokit localnet status
# algod status
Status: Running
Port: 4001
Last round: 0
Time since last round: 0.0s
Genesis ID: dockernet-v1
Genesis hash: sl0ctEZ9Qm2gqYLLQnz9+KTTihi0gwFXCLCXtsEUobo=
Version: 3.23.1
# conduit status
Status: Running
# indexer-db status
Status: Running
# indexer status
Status: Running
Port: 8980
Last round: 0
Version: 3.4.0
@A-Maugli → /workspaces (main) $
```

The operation of the local Algorand node can be verified in a browser using the following command:

```
algokit explore
```

In the Codespaces environment, before running this command: Go to the Ports tab. In the Visibility column, right-click on the entries for ports 4001, 4002, and 8980, then select Port visibility | Public. After this, switch to the Terminal tab and run `algokit explore` In the browser window that opens, click the OK button!

3 Examples of Using Algorand Commands

The command-line environment can be accessed with the following command:

```
algokit localnet console
```

When using Codespaces, the following commands must be run beforehand:

```
pipx install algokit
```

```
algokit localnet start
```

```
algokit localnet stop
```

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

Example:

```
@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':
```

```
33 Created new account with address RTKECLZXEQLG2DRSSG2KWVHNE53UVH5F2MYLUENKKQUEOUI4HPJQTJGXPM
34 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) $
```

Explanation of the Example:

- Line 3: The `goal wallet list` command lists the available wallets, showing the default wallet.
- Line 8: The `goal account list` command lists the accounts associated with the default wallet.
- Line 12: A new wallet can be created using the `goal wallet new` command. In this case, the wallet is named `w1`.
- Line 24: A new account can be created with the `goal account new` command. The account is created in the default wallet.
- Line 31: To create a new account in the w1 wallet, use the command: `goal account new --wallet w1`
- Line 34: The `goal account list` command displays the accounts in the default wallet.
- Line 39: Listing the accounts in the `w1` wallet.
- Line 41: To send Algó from one account to another, use the `goal clerk send` command. Provide the source account address with `--from`, the destination account address with `--to`, and the amount to send in micro-Algos using `--amount`. In this case, 1 Algo (1 000 000 microAlgos) was sent from `NMA5H...IOZQM` to `RTKEC...JGXPM`.
- Line 44: To check an account's balance, use the `goal account balance` command. Specify the account's address with `--address`.

An interesting note for those coming from the Bitcoin and Ethereum World: When creating a new wallet named `w1`, we output the "backup phrase", or recovery phrase. Notice that it consists of 25

words. These 25 words encode 256 bits of entropy and include a checksum to secure the recovery phrase. This 256-bit entropy aligns with the BIP39 standard's 256-bit entropy, but it is not encoded as specified by the BIP39 standard. Instead, it follows Algorand's 11-bit chunk format. For more details, refer to the forum article titled What's the rationale behind the bespoke 25-word mnemonic standard?

Of course, it is entirely possible to convert Algorand's 25-word mnemonic to the 24-word mnemonic used in BIP39, and vice versa. Hierarchical address generation is derived from this entropy, as described, for example, on the BIP39 website. In this context, the 24 words following the BIP39 standard or the 25 words used in Algorand do not encode a private address but instead represent 256 bits of entropy.

Adding to potential confusion is the ability to export the private key associated with individual accounts. These are also output in a 25-word mnemonic format. For wallets, the 25 words encode entropy, which can then be used to generate any number of accounts (public keys and private keys).

For true enthusiasts, it's worth mentioning that in the case of the Ledger hardware wallet, the implementation of the Algorand App on the hardware wallet uses only 24 words for encoding entropy and employs a completely different encryption algorithm. A handy utility is available at algorand.oortnet.com, which can generate addresses, keys, and mnemonics for individual hierarchical Algorand accounts based on the Ledger hardware wallet's mnemonic (recovery phrase). This means that even if you lose access to your Ledger hardware wallet (due to damage, theft, etc.), you can still access the Algorand accounts it encodes.

Note: The Algorand currency is not selectable on the BIP39 website. However, a standalone version that works for Algorand can be downloaded from `https://github.com/Coinomi/bip39-coinomi/releases`.

4 Examples of Using the Python SDK

The usage of the Python SDK is demonstrated with examples from Ryan Fox's "Algorand Bootcamp for Beginners" lectures. Python sample code can be found in the `https://github.com/A-Maugli/akt02` repository, under the hellow/playground/python_api directory.

4.1 Preparation Steps

Launch your Codespaces workspace. Once the workspace has loaded, start the private blockchain:

```
@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
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_conduit  Created
docker: Container algokit_sandbox_indexer  Created
docker: Container algokit_sandbox_algod  Starting
docker: Container algokit_sandbox_postgres  Starting
docker: Container algokit_sandbox_postgres  Started
docker: Container algokit_sandbox_algod  Started
docker: Container algokit_sandbox_conduit  Starting
docker: Container algokit_sandbox_conduit  Started
docker: Container algokit_sandbox_indexer  Starting
docker: Container algokit_sandbox_indexer  Started
docker: Container algokit_sandbox_algod  Waiting
docker: Container algokit_sandbox_conduit  Waiting
docker: Container algokit_sandbox_postgres  Waiting
docker: Container algokit_sandbox_indexer  Waiting
docker: Container algokit_sandbox_algod  Healthy
docker: Container algokit_sandbox_postgres  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.
```

Install the dependencies for the project using the following command:
`algokit project bootstrap all`

```
@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 pycparser (2.21)
poetry: - Installing sniffio (1.3.0)
poetry: - Installing typing-extensions (4.9.0)
poetry: - Installing anyio (4.3.0)
poetry: - Installing certifi (2024.2.2)
poetry: - Installing cffi (1.16.0)
poetry: - Installing h11 (0.14.0)
poetry: - Installing httpcore (0.16.3)
poetry: - Installing msgpack (1.0.7)
poetry: - Installing pycryptodomex (3.20.0)
poetry: - Installing pynacl (1.5.0)
poetry: - Installing rfc3986 (1.5.0)
poetry: - Installing wrapt (1.16.0)
poetry: - Installing deprecated (1.2.14)
poetry: - Installing docstring-parser (0.14.1)
poetry: - Installing executing (1.2.0)
poetry: - Installing httpx (0.23.3)
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
```

Set the Ports to global visibility: In the Ports tab, click on the lock symbol to make the ports global. The lock will open once clicked.

Activate the Virtual Environment: Use the graphical interface to create a new `python_api` directory within the playground directory. After creating the directory, navigate into it.

```
@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 Creating an Algorand Account

Use the File Explorer or in the terminal enter the command `nano 01-account_generation.py` to create the following file:

```
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()
```

Run the `01-make_account.py` file:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api (main) $ python
↪  01-account_generation.py
My address: 3API2XIBSH7IKMSZ3SNRGOINISSNXVAYT2OV6BLL4GBZRAVWTMYAWCXMPI
My private key:
↪  y1mWx1K1ZMGbHojwG0SWKtMTs0XuCIKOATJ9PFc2AyrYHo1dAZH+hTJZ3JsTOQ1EpNvUGJ6dXwVr4YOYgrabMA==
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
```

Congratulations! You successfully created an account number and its corresponding private key using a Python program, and you also exported the private key as a 25-word mnemonic!

4.3 Displaying Account Balance

To create the next file, use the File Explorer or the terminal with the command:
`nano 02-account_balance.py`

```
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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")

main()
```

Run the `2-account_balance.py` file:

```
(playground-py3.10) @A-Maugli → .../akt02/hellow/playground/python_api (main) $ python
↪  02-account_balance.py
DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE balance: 4000000000000000 microAlgos

DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ balance: 2000000000000000 microAlgos

ICNK7LCUJZEULNYG3SEKZ5LGM5J3H2TTIBKYAAUPXACUEKHW33DGXVQA74 balance: 4000000000000000 microAlgos
```

You can see, that wallet he wallet initially contains three account numbers, with a total of 10^{16} microAlgo, equivalent to 10 billion Algo. (10 billion Algo = 10 000 million Algo = 10^{10} Algo).

4.4 Creating a Payment Transaction

To create the next file, use the File Explorer or the terminal with the command:
`nano 03-payment_transaction.py`

```
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 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()
```

Run the `nano 03-payment_transaction.py` file:

```
(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": "ZDb6f6E30kybPW6KBF7gaosEQBpAZCXIMLgeYSG23Bbg/YQnUJz8ZsFz8/R7nTsYf1im1H509umuKjUxoc
        ↪  OkAA==",
        "txn": {
            "amt": 1000000,
            "fee": 1000,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "lv": 1000,
            "note": "SGVsbG8gV29ybGQ=",
            "rcv": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
            "snd": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
            "type": "pay"
        }
```

```
    }
}
Decoded note: Hello World
```

You can see that the transaction was successfully executed, and the transaction ID is `2ICYN...F4HSQ`. The transaction record is also displayed, including its details: `txn`: transaction, `amt`: amount, `fee`: transaction fee, `fv`: first block value, `gen`: Genesis, `gh`: Genesis hash, `lv`: last block value, `note`: pptional note, `rcv`: receiver, `snd`: sender, `pay`: payment transaction type.

The `note` field was also displayed in its decoded form.

4.5 Displaying Account Balances

To create the next file, use the File Explorer or in the terminal use the following command: `nano 04-account_info.py`

```
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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 details
    account_info = algod_client.account_info(addr3)
    print("Account information: {}".format(
        json.dumps(account_info, indent=4)))
    account_info = algod_client.account_info(addr2)
    print("Account information: {}".format(
        json.dumps(account_info, indent=4)))
    account_info = algod_client.account_info(addr1)
    print("Account information: {}".format(
        json.dumps(account_info, indent=4)))
```

```
main()
```

Run the `nano 04-account_info.py` file:

```
(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+7WeFuYOFAIcaO7snXb68TdI5b+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": 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/ew56tJGd9rODmXFSK01To=",
        "state-proof-key": "uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1wo5U9cPx6t+tBK7iwmNgL8
        ↪  OIaMHP3eUkisdIpQ==",
        "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
}
```

We received detailed information for all three accounts. The "assets" field is currently empty. ASA (Algorand Standard Assets) refers to non-Algorand tokens created on the blockchain. With ASA, it's possible to create any type of asset, even something like a *wooden nickel*.

4.6 Creating an Asset

To create the next file, use the File Explorer or in the terminal use the following command: `nano 05-asset_create.py`

```
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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()
    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

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

    # write the asset index to an environment file
    f = open('asset.index', 'w+')
    f.write(f'{confirmed_txn["asset-index"]}')
    f.close()

main()
```

The new asset is named "Fun Token", the name of one unit is FUNTOK. A total of 10 000 units will be created initially, and 100 units represent one token due to the decimals = 2 setting. This effectively means that 100.00 FUNTOK tokens will be created.

Run the `nano 05-asset_create.py` file:

```
Transaction information: {
    "asset-index": 1002,
    "confirmed-round": 2,
    "pool-error": "",
    "txn": {
        "sig": "9KHcv6feb9b1fkX6Wg6oRvlqj1/ttyaDjX95X0QWmv2ITe/c6zt3KHFPMBgbNZ9BMaoNkFIY0yrOa6XF5n
        ↪  e+Aw==",
        "txn": {
            "apar": {
                "an": "Fun Token",
                "au": "https://path/to/my/fungible/asset/metadata.json",
                "dc": 2,
                "m": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
                "t": 10000,
                "un": "FUNTOK"
            },
            "fee": 1000,
            "fv": 1,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "lv": 1001,
            "snd": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
            "type": "acfg"
        }
    }
}
```

The new "Fun Token" asset has been successfully created. The index of the created asset, or "asset-index," is 1002. The "apar" field contains the asset's parameters:

- an: Asset name
- au: Asset URL

- dc: Decimals
- m: Manager address
- t: Total units
- un: Unit name
- acfg: Asset configuration (transaction type)

4.7 Opting In to an Asset

In Algorand, ASA tokens cannot be sent to an account unless the account explicitly indicates its willingness to receive the asset. This willingness is referred to as "opt-in", meaning the account "opts in" to receive the given asset.

Opting in is achieved by sending 0 units of the ASA to the address of the account that wants to opt in. This process is abstracted by the `AssetOptInTxn` function.

Create file `06-asset_opt_in.py` using the File Explorer | New File, or using the terminal command `nano 06-asset_opt_in.py`

```
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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()
```

Run the `nano 06-asset_opt_in.py` file:

```
(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/KYubPT40B186gasqHXnRoK3mkeex4s+L
        ↪  7PAw==",
        "txn": {
            "arcv": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
            "fee": 1000,
            "fv": 2,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "lv": 1002,
            "snd": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
            "type": "axfer",
            "xaid": 1002
        }
    }
}
```

The transaction type is "axfer" (asset transfer), with "xaid" referring to the transfer asset ID, and "arcv" indicating the receiver address, which opted in to the ASA.

4.8 Swapping Assets Using an Atomic Transaction Group

In Algorand, ASAs can represent many things. For this example, let's assume the ASA represents another type of asset. To sell the ASA in exchange for Algorand (Algo), we must ensure that two transactions occur simultaneously or not at all:

1. The buyer pays the seller in Algo for the ASA.
2. The seller sends the ASA to the buyer.

In Algorand, this is handled using the concept of a transaction group, which ensures atomicity for the swap.

Create file `07-atomic_transaction.py` using the File Explorer | New File, or using the terminal command `07-atomic_transaction.py`:

```
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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)

    # build unsigned asset transfer transaction
    sender = addr1
    receiver = addr2
    amount = 100 # remember this ASA has 2 decimal places, so this is 1.00 FUNTOK
    index = get_asset_index(default_index = 1010) # ensure this matches the asset-index returned
    ↪  by asset_create.py
    txn_2 = transaction.AssetTransferTxn(sender, params, receiver, amount, index)

    # group transactions
    gid = transaction.calculate_group_id([txn_1, txn_2])
    txn_1.group = gid
    txn_2.group = gid

    # sign transaction
    stxn_1 = txn_1.sign(wallet.export_key(addr2))
    stxn_2 = txn_2.sign(wallet.export_key(addr1))

    # assemble transaction group
    signed_group = [stxn_1, stxn_2]

    #submit atomic transaction group
    txid = algod_client.send_transactions(signed_group)
    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()
```

Run the `07-atomic_transaction.py` file:

```
(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+XytRtzrV42VAUswRcemomUEbV3vKm4IaZIDEVJr4sIQifM3
        ↪  XBCQ==",
        "txn": {
            "amt": 1000000,
            "fee": 1000,
            "fv": 3,
            "gen": "dockernet-v1",
            "gh": "d2lGnqyAVFyDA61oZLE5NmjU64Ig2vQr1jG3C1nnQ6I=",
            "grp": "SBs5jPxhxYFk7y/WhsILxTXn7QlBByaETrD5xTHECPI=",
            "lv": 1003,
            "rcv": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
            "snd": "DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ",
            "type": "pay"
        }
    }
}
```

We printed only the first part of the transaction, the "pay" (payment transaction) that handles the payment in Algo. However, there is another transaction in the transaction group: the "axfer" (asset transfer transaction) that deals with the ASA transfer.

To examine the details of this transaction group, we can use the `04-account_info.py` script to assist us.

```
(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+7WeFuYOFAIcaO7snXb68TdI5b+Fz2hGNl8
        ↪  hE8qbAQSsNhopp/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+HYgyQvn8456ImNm7vMMT28Xg
        ↪  BSY4em3Hl4bOIi4A==",
        "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": 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": [],
```

```
87      "created-assets": [
88          {
89              "index": 1002,
90              "params": {
91                  "creator": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
92                  "decimals": 2,
93                  "default-frozen": false,
94                  "manager": "DPO7AIGM6OH2UREMX2ZPS6SBMRAVEGAAHV43BCTX4SIAOKQBWFFJRZYYOE",
95                  "name": "Fun Token",
96                  "name-b64": "RnVuIFRva2Vu",
97                  "total": 10000,
98                  "unit-name": "FUNTOK",
99                  "unit-name-b64": "RlVOVE9L",
100                 "url": "https://path/to/my/fungible/asset/metadata.json",
101                 "url-b64": "aHR0cHM6Ly9wYXRoL3RvL215L2Z1bmdpYmxlL2Fzc2V0L21ldGFkYXRhLmpzb24="
102             }
103         }
104     ],
105     "min-balance": 200000,
106     "participation": {
107         "selection-participation-key": "tSvVZrUkGrQbS4YTJ7//K/ew56tJGd9rODmXFSK01To=",
108         "state-proof-key": "uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1wo5U9cPx6t+tBK7iwmNgL8
        ↪  OIaMHP3eUkisdIpQ==",
109         "vote-first-valid": 0,
110         "vote-key-dilution": 10000,
111         "vote-last-valid": 30000,
112         "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) $
```

You can see in lines 42..48 of the example, that 1.00 FUNTOK from the asset with asset-id 1002 was successfully transferred to the address `DUWWU...2WRUQ`. Due to the 2 decimal places, this appears as 100 units. Similarly, in lines 79..85, we see that the balance of the issuing account `DPO7A...ZYYOE` decreased by 100 units, leaving only 9900 units of FUNTOK. With the 2 decimal places, this corresponds to 99.00 FUNTOK.

5 Development with the JavaScript SDK

The Algorand JavaScript SDK enables development in various environments:

- In a Node.js environment using the console.
- In a Node.js environment within a web browser.
- Directly in a web browser using bundled JS library/libraries.

In the following section, we will revisit the examples previously demonstrated, but this time using the Algorand JavaScript SDK in a Node.js environment via the console.

The JavaScript examples can be found in the repository `https://github.com/A-Maugli/akt02`, under the hellow/playground/js_api directory.

5.1 Preparation Steps

Launch your last used `Codespaces` workspace. Once the workspace has loaded, start the private blockchain with the following command:

```
@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
```

In the Ports tab, click the lock symbol with the mouse. The lock will open, setting the ports to global visibility.

Create a new directory named `js_api` under `/workspace/akt02/hellow/playground`. Navigate into this directory:

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

Initialize a `package.json` file:

```
npm init -y
```

Add the `algosdk` to the `package.json` file and install the algosdk module by running:

```
npm install algosdk
```

5.2 Creating an Algorand Account

Use the File Explorer or the nano text editor in the terminal to create the following file: `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();
```

Install the dependencies using the following command:

```
algokit project bootstrap all
```

```
(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
npm: npm notice
npm: npm notice New minor version of npm available! 10.2.4 -> 10.5.0
npm: npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.5.0>
npm: npm notice Run `npm install -g npm@10.5.0` to update!
npm: npm notice
Finished bootstrapping /workspaces/akt02/hellow/playground/js_api
```

Run the `01-account_generation.js` file:

```
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 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_handle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Run the `02-account_balance.js` file:

```
(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) $
```

Initially, the wallet contains three account numbers, with a total of 10^{16} microAlgos, equivalent to 10 billion Algos. (10 billion Algo = 10,000 million Algo = 10^{10} Algo).

5.3 Creating a Payment Transaction

Use the File Explorer or the nano text editor in the terminal to create the following file:
`nano 03-payment_transaction.js`

```
const 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 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, '');
    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);
    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);

    // build unsigned transaction
    params = await algod_client.getTransactionParams().do();
```

```
    params['fee'] = 0;
    if (DEBUG) console.log('params:', params);
    unsigned_txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
        from: addr1,
        to: addr2,
        amount: algosdk.algosToMicroalgos(0.15),
        note: algosdk.encodeObj("Hello World"),
        suggestedParams: params
    });
    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: %s", 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);

    // 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_handle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

Run the `03-payment_transaction.js` file:

```
(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) $
```

It can be seen that the transaction was successfully completed, and the transaction ID is `MIPIZ...TNUPQ`. The transaction record was also displayed, including the following details:

- txn: Transaction
- amt: Amount
- fee: Transaction fee
- fv: First block value
- gen: Genesis
- gh: Genesis hash
- lv: Last block value
- note: Optional note field
- rcv: Receiver
- snd: Sender
- pay: Payment transaction type

The `note` field was also displayed in its decoded form.

5.4 Displaying Account Balances

Use the File Explorer or the nano text editor in the terminal to create the following file: `04-account_info.js`

```
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_handle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

```

```
main();
```

Run the script `04-account_info.js`

```
(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':
    'uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1wo5U9cPx6t+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':
    '/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: 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': '2m3w5viSs8ZXPhW0+Mns+z8oX3dkJEH3M+DbJQWtN/U=',
    'state-proof-key':
    ↪  'e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFAIcaO7snXb68TdI5b+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) $
```

We received comprehensive information for all three accounts. In lines 8 and 36, the previously created "assets" (using the python_api) are visible. ASA (Algorand Standard Assets) represents non-Algorand tokens created on the blockchain. With ASA, any type of asset can be created – even something like a "wooden nickel."

5.5 Creating an Asset

Use the File Explorer or the nano text editor in the terminal to create the following file:
`05-asset_create.js`

```
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;
}

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: %s", 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_handle_token']);
    if (DEBUG) console.log('wallet handle released, hr:', hr)
}

main();
```

The new asset is named “Fun Token", with the unit name FUNTOK. A total of 10 000 units will be created initially. Since 100 units represent one token due to the `decimals = 2` setting, this effectively creates 100.00 FUNTOK tokens.

Run the `05-asset_create.js` script to create the asset:

```
(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) $
```

The new “Fun Token” asset has been successfully created. The “asset-index” of the created asset is 1007. The “apar” field contains the asset’s parameters, although they are not displayed in the list. These parameters include:

- an: Asset name
- au: Asset URL
- dc: Decimals
- m: Manager address
- t: Total units
- un: Unit name
- acfg: Asset configuration (transaction type)

5.6 Opting In to an Asset

In Algorand, an account cannot receive ASA tokens until it explicitly indicates its willingness to accept them. This process, called opt-in, essentially means "signing up" for a particular ASA.

The opt-in process is performed by sending 0 units of the ASA to the address of the account that wants to opt-in. This can be implemented using the `makeAssetTransferTxnWithSuggestedParamsFromObject` method.

To create the script for opting in, use the File Explorer or the nano text editor in the terminal to create the following file:

`06-asset_opt_in.js`

```
const algosdk = require('algosdk');
const fs = require('fs');
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 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,
        from: addr2,
        to: addr2,
        assetIndex: asset_index,
        amount: 0
    });
    if (DEBUG) console.log('unsigned_txn:', unsigned_txn);

    // sign transaction
    addr2_sk = await kmd_client.exportKey(wallet_handle.wallet_handle_token, '', addr2);
    if (DEBUG) console.log('addr1_sk:', addr2_sk);
    signed_txn = unsigned_txn.signTxn(addr2_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: %s", 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');
```

```
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_handle_token']);
107     if (DEBUG) console.log('wallet handle released, hr:', hr)
108 }
109
110 main();
```

Run the script `06-asset_opt_in.js`

```
1  (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api (main) $ node
   ↪  06-asset_opt_in.js
2  Successfully sent transaction with tx_id: { txId:
   ↪  'MC4NDEDTIW7TDRIZ2HETCADG4JF7WR4RJCGUVUSSHDVHNYLFWHLA' }
3  tx_id["txId"]: MC4NDEDTIW7TDRIZ2HETCADG4JF7WR4RJCGUVUSSHDVHNYLFWHLA
4  Awaiting confirmation (this may take several seconds)...
5  Transaction is successfully completed
6  confirmed_txn: {
7    'confirmed-round': 7,
8    'pool-error': '',
9    txn: {
10     sig: Uint8Array(64) [
11        21,  72,  32,  49, 110, 132, 206,  97, 160,  12, 242,
12        37, 177,  24,   0, 138,   8, 106, 193,   4,  98,  70,
13       202, 194, 177, 178, 169,   3,  42,  32, 187, 228, 235,
14        79, 210, 248, 246,  23, 112, 177, 111, 164, 166,  59,
15       177,  71, 147, 120,  56, 122, 228, 224,  40, 186,  53,
16        70, 152, 218,  47, 242, 249,  78,  62,   9
17     ],
18     txn: {
19       arcv: [Uint8Array],
20       fee: 1000,
21       fv: 6,
22       gen: 'dockernet-v1',
23       gh: [Uint8Array],
24       lv: 1006,
25       snd: [Uint8Array],
26       type: 'axfer',
27       xaid: 1007
28     }
29   }
30 }
31 (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api (main) $
```

The transaction type is "axfer" (asset transfer), where "xaid" represents the transfer asset ID, and "arcv" indicates the address receiver, which is the account that opted in to the ASA.

5.7 Swapping Assets Using an Atomic Transaction Group

In Algorand, ASAs can represent a variety of assets. For this example, let's assume the ASA represents another type of asset. To sell the ASA in exchange for Algo, it must be ensured that the following two transactions occur simultaneously, or not at all:

- The buyer pays the seller in Algo for the ASA.
- The seller sends the ASA to the buyer.

This atomicity is guaranteed in Algorand through the concept of a transaction group.

To create the script for implementing this swap, use the File Explorer or the nano text editor in the terminal to create the following file:

`07-atomic_transaction.js`

```
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) {
    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);
```

```
48
49      // connect to wallet
50      const wallet_name = 'unencrypted-default-wallet';
51      const wallet_pw = '';
52      wallet_id = await getWalletId(kmd_client, wallet_name);
53      wallet_handle = await kmd_client.initWalletHandle(wallet_id, wallet_pw);
54
55      // gather the first three accounts from the wallet
56      wallet_addresses = await kmd_client.listKeys(wallet_handle.wallet_handle_token);
57      addr1 = wallet_addresses.addresses[0];
58      addr2 = wallet_addresses.addresses[1];
59      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);
97      stxn_1 = await txn_1.signTxn(addr2_sk.private_key);
98
99      addr1_sk = await kmd_client.exportKey(wallet_handle.wallet_handle_token, wallet_pw, addr1);
100     if (DEBUG) console.log('addr1_sk:', addr1_sk);
101     stxn_2 = await txn_2.signTxn(addr1_sk.private_key);
102
103     // assemble transaction group
104     signed_group = [stxn_1, stxn_2];
```

```
105
106     // submit atomic transaction group
107     tx_id = await algod_client.sendRawTransaction(signed_group).do();
108     console.log("Successfully sent transaction gropup with tx_id: %s", tx_id);
109     console.log('tx_id["txId"]:', tx_id['txId']);
110
111     // wait for confirmation
112     console.log('Awaiting confirmation (this may take several seconds)...');
113     const roundTimeout = 7;
114     const confirmed_txn = await algosdk.waitForConfirmation(
115       algod_client,
116       tx_id['txId'],
117       roundTimeout
118     );
119     console.log('Transaction group is successfully completed');
120
121     // log confirmed transaction
122     if (DEBUG) console.log('confirmed_txn:', confirmed_txn);
123
124     // release wallet handle
125     hr = await kmd_client.releaseWalletHandle(wallet_handle['wallet_handle_token']);
126     if (DEBUG) console.log('wallet handle released, hr:', hr)
127 }
128
129 main();
```

Run the script `07-atomic_transaction.js`

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

Run the script `04-account_info.js` to verify the result of the atomic transaction group.

```
1  (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api (main) $ node
   ↪  04-account_info.js
2  account_info {
3    address: 'DUWWUSWZSLBPGVY6GJ7QM5RTUC54EPJ4273FN5QTFELSVOCAQGTF62WRUQ',
4    amount: 1999999999146000,
5    'amount-without-pending-rewards': 1999999999146000,
6    'apps-local-state': [],
7    'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
8    assets: [
9      { amount: 100, 'asset-id': 1002, 'is-frozen': false },
10     { amount: 100, 'asset-id': 1007, 'is-frozen': false }
11   ],
12   'created-apps': [],
13   'created-assets': [],
14   'min-balance': 300000,
15   participation: {
16     '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: 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':
    ↪  'uGQ3C1jRdub2CgTsqTNakL4fvDBliDiwgaUB39NDurQev6O1wo5U9cPx6t+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': [],
```

```
72    'apps-total-schema': { 'num-byte-slice': 0, 'num-uint': 0 },
73    assets: [],
74    'created-apps': [],
75    'created-assets': [],
76    'min-balance': 100000,
77    participation: {
78      'selection-participation-key': '2m3w5viSs8ZXPhW0+Mns+z8oX3dkJEH3M+DbJQWtN/U=',
79      'state-proof-key':
      ↪  'e9zi9f7feDLyGhG4RQoIjddrieRHGpRMqZQ+7WeFuYOFAIcaO7snXb68TdI5b+Fz2hGNl8hE8qbAQSsNhopp/Q==',
80      'vote-first-valid': 0,
81      'vote-key-dilution': 10000,
82      'vote-last-valid': 30000,
83      'vote-participation-key': 'ERZwnIHHihb37ERB93xwo6ejSbA3TyAtggzegnrylS8='
84    },
85    'pending-rewards': 0,
86    'reward-base': 0,
87    rewards: 0,
88    round: 8,
89    status: 'Online',
90    'total-apps-opted-in': 0,
91    'total-assets-opted-in': 0,
92    'total-created-apps': 0,
93    'total-created-assets': 0
94  }
95  (playground-py3.10) @A-Maugli → .../akt02/hellow/playground/js_api (main) $
```

In line 41, it can be seen that 1.00 FUNTOK from asset ID 1007 was successfully transferred to the address `DPO7A...YYOE`. Due to the 2 decimal places, this appears as 100 units.

5.8 Node.js Web Applications

Web development using the Algorand JavaScript SDK can be done with simple HTML or through frameworks like React, Angular, Vue, etc.

Regardless of whether you use plain HTML or a framework, a common approach in web development is to bundle numerous Node.js modules into a few larger files using a web packer tool. These bundled files are then made accessible to the browser via a web server.

Development Mode

In development mode, bundling the files and making them accessible through the web server happens immediately after every source code modification. This allows for rapid iteration and testing.

Production Mode

For the final version, the web packer performs a more thorough analysis and optimizes the size of the bundled libraries, ensuring better performance and efficiency for deployment.

5.8.1 Pera WalletConnect Example

The WalletConnect Node.js module facilitates communication between an application and a user's wallet, allowing the application to perform various operations with the user's permission.

Using the Pera Connect SDK, which is designed for the Algorand Pera Wallet, you can perform operations such as connect, disconnect, reconnect, payment transactions, and more. The documentation for Pera Connect is available at: `https://docs.perawallet.app/references/pera-connect`.

Pera also provides sample examples for Pera Connect. The following small example is a modified version of the plain vanilla JavaScript sample and is available in the repository: `https://github.com/A-Maugli/pera-connect-vanillajs-demo_mod5`.

Developement

The example was developed in Node.js. Below is the content of the `package.json` file:

```
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"
  }
}
```

NPM Modules and Tools Used

The example uses the following NPM modules:

- @perawallet/connect: For connecting to the Pera Wallet.
- algosdk: For interacting with the Algorand blockchain.
- jquery: For DOM manipulation.

The web packer used is Parcel, version 2.11.0.

Implemented NPM scripts are:

- `npm run start` – Starts an interactive web packer along with a web server.
- `npm run build` – Creates an optimized build using the web packer and launches a separate

web server.

- `npm run clean` – Removes unnecessary files from the project.

Example Workflow

Here is a possible workflow for using this setup:

```
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!      │
   │                                          │
   └──────────────────────────────────────────┘
```

```
46
47  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /
48  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 47 ms
49  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /index.runtime.7745c7dc.js
50  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 3 ms
51  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /index.0ed18bef.js
52  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 8 ms
53  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 GET /App-94e9365e.cefc48d1.js
54  HTTP  2/28/2024 3:55:57 PM 127.0.0.1 Returned 304 in 4 ms
55 ^C
56  INFO  Gracefully shutting down. Please wait...
```

The demo app running within the browser can be seen on Fig. 1.

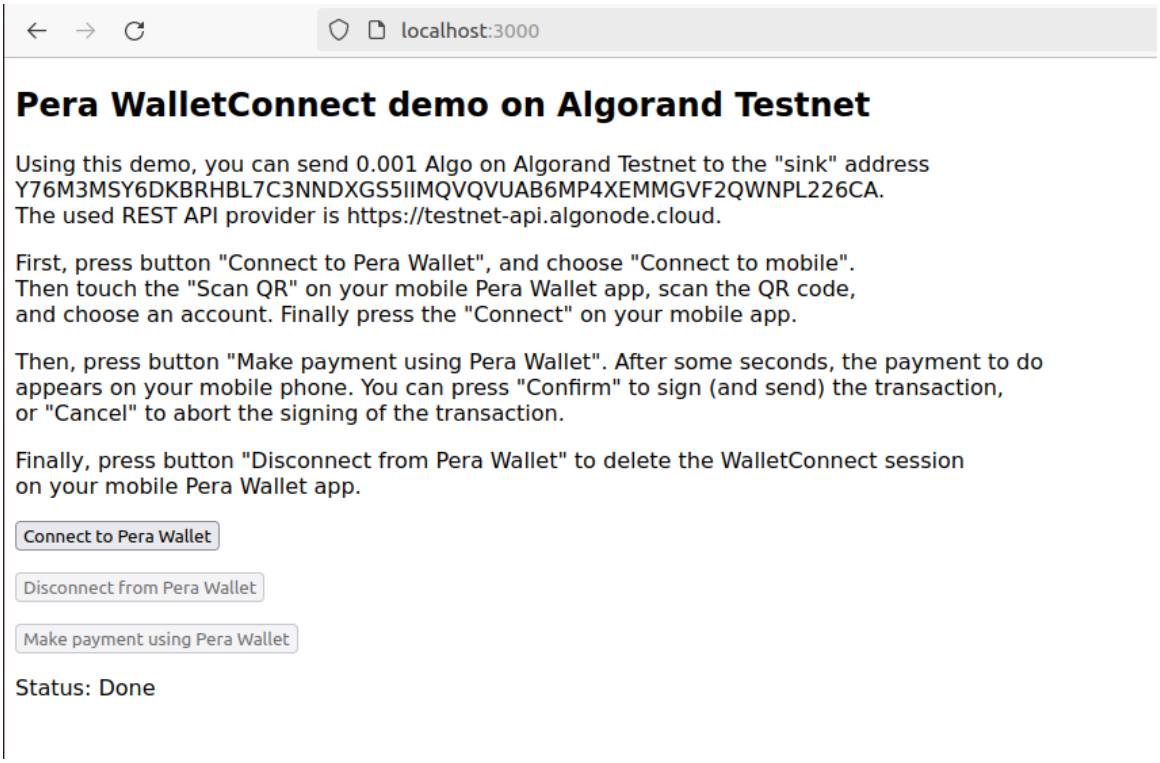

Figure 1: The Pera WalletConnect demo app after starting

Workflow Explanation

- Line 1: Clean up everything unnecessary.
- Line 11: Install the Node.js modules into the `node_modules` directory.
- Line 22: Display how much disk space the project uses. It consumed a significant amount, totaling 391 MB.
- Line 24: Build the optimized, bundled application and start a web server.

Demo Application in the Browser

The demo application launched in the browser is shown in Figure 1. Its functions include:

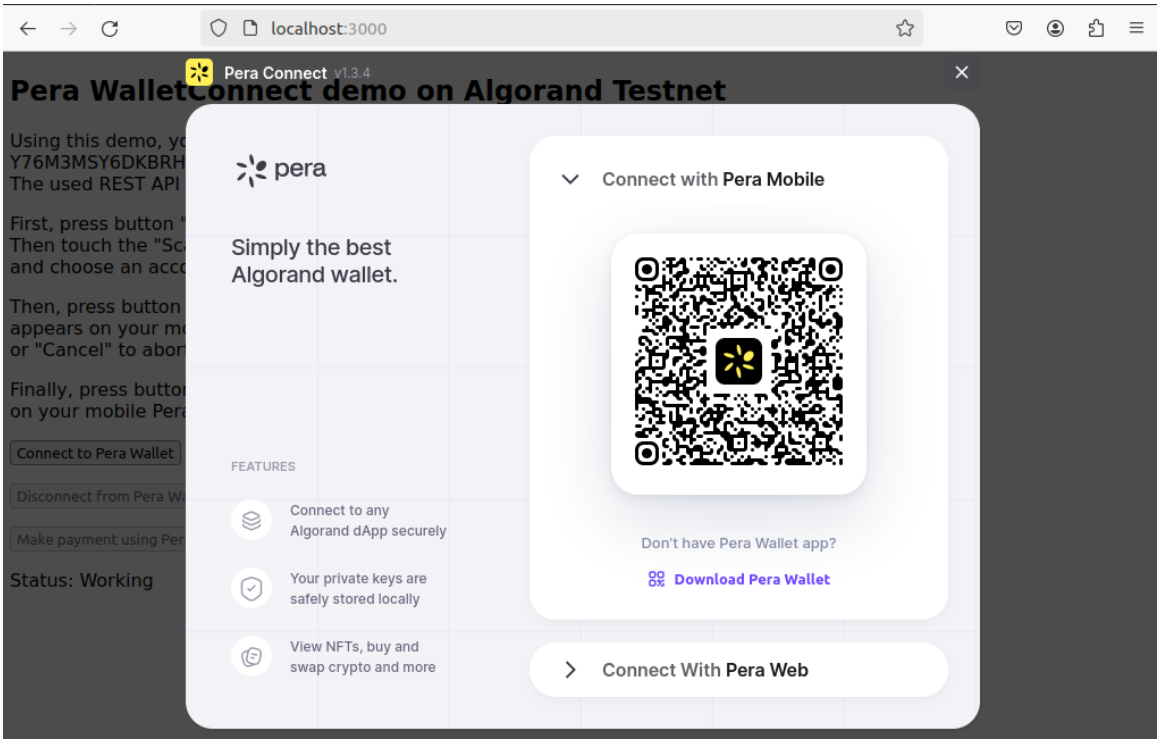

Figure 2: Connection to the mobile Pera wallet

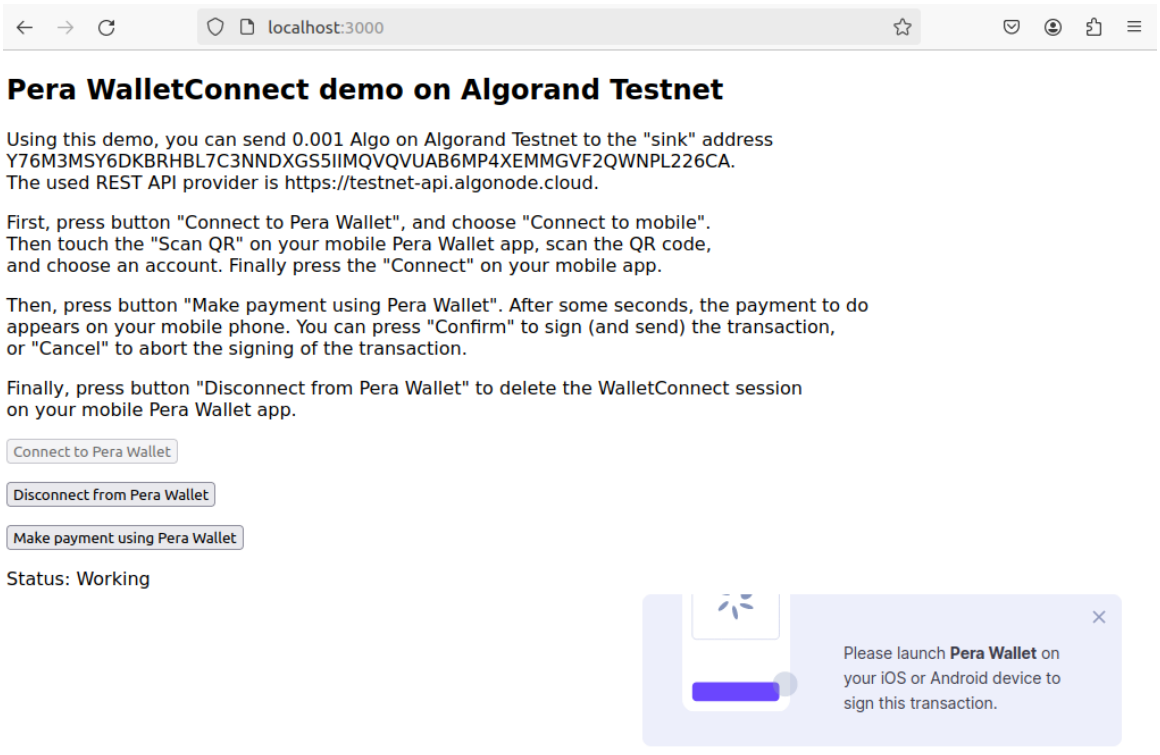

Figure 3: Initiating a payment transaction

- "Connect to Pera Wallet": Connects the demo application to a Pera wallet. After clicking the button, you can choose whether to connect the demo app to a web-based wallet or an iOS/Android wallet. Select the "Connect to mobile" option.

  At this point, a QR code will appear on the screen (see Figure 2).

  - Open the Pera Wallet on your mobile device.
  - Scan the QR code using the wallet.
  - Select an account on your mobile device that you want to link to the Pera WalletConnect demo application.

- "Make payment using PeraWallet": Performs a payment transaction, sending 0.001 Algo to Algorand's "all-absorbing" address (see Figure 3).

  - After pressing the button, a payment transaction screen will appear on your mobile device within a few seconds.
  - The transaction can be signed and sent by pressing the "Confirm" button or canceled by pressing the "Cancel" button.

- "Disconnect from Pera Wallet": Ends the WalletConnect session. The session can also be terminated from the mobile app under Settings | WalletConnect Sessions.

Demo Application Source Code

The source code for the demo application is located in the `src` directory. Below is the content of `src/index.html`:

```
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>
  <p>
    Using this demo, you can send 0.001 Algo on Algorand Testnet to the "sink" address<br/>
    Y76M3MSY6DKBRHBL7C3NNDXGS5IIMQVQVUAB6MP4XEMMGVF2QWNPL226CA. <br/>
    The used REST API provider is https://testnet-api.algonode.cloud.
```

```
  </p>
  <p>
    First, press button "Connect to Pera Wallet", and choose "Connect to mobile". <br/>
    Then touch the "Scan QR" on your mobile Pera Wallet app, scan the QR code, <br />
    and choose an account. Finally press the "Connect" on your mobile app.
  </p>
  <p>
    Then, press button "Make payment using Pera Wallet". After some seconds, the payment to do
    ↪  <br/>
    appears on your mobile phone. You can press "Confirm" to sign (and send) the transaction, <br/>
    or "Cancel" to abort the signing of the transaction.
  </p>
  <p>
    Finally, press button "Disconnect from Pera Wallet" to delete the WalletConnect session <br/>
    on your mobile Pera Wallet app.
  </p>
  <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>
```

Explanation of `index.html`

The `index.html` file describes the structure of the webpage. Informational text is included within `<p>` tags, followed by the creation of three buttons using the `<button>` tag, each corresponding to a specific function. Figure 1 shows the webpage rendered in the browser.

JavaScript Logic

The JavaScript logic for handling button actions is located in the `src/pera_walletconnect_demo.js` file. This file manages interactions such as connecting to the wallet, making payments, and disconnecting.

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

import algosdk from "algosdk";
//import { PeraWalletConnect } from "@perawallet/connect";
import * as pwcsdk from "@perawallet/connect";
import $ from "jquery";

const working = "Status: Working";
const done = "Status: Done";

```

```
const peraWallet = new pwcsdk.PeraWalletConnect();

const token = "";
const server = "https://testnet-api.algonode.cloud";
const port = 443;
const client = new algosdk.Algodv2(token, server, port);

let accountAddress = "";
const sinkAddress = "Y76M3MSY6DKBRHBL7C3NNDXGS5IIMQVQVUAB6MP4XEMMGVF2QWNPL226CA";


$(window.document).ready(function () {
  console.log("document ready");
  addEventListeners();
  reconnectSession();
});

function addEventListeners() {
$("#connect_pera_wallet").on("click", function () {
  handleConnectWallet();
});

$("#disconnect_pera_wallet").on("click", function () {
  handleDisconnectWallet();
});

$("#make_payment_pera_wallet").on("click", function () {
  handlePayment();
})
}

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();
```

```
126   peraWallet
127     .disconnect()
128     .then(() => {
129       setButtonState(false);
130       setStatus(done);
131     })
132     .catch((error) => {
133       console.log('Error in handleDisconnectWallet: ' + error.name + ' ' + error.message);
134       setStatus(done);
135       setError(error);
136     });
137   accountAddress = "";
138 }
139
140 async function handlePayment() {
141   setStatus(working);
142   clearError();
143   const params = await client.getTransactionParams().do();
144   const txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
145     from: accountAddress,
146     to: sinkAddress,
147     amount: algosdk.algosToMicroalgos(0.001),
148     suggestedParams: params
149   });
150   let txGroup = [{ txn, signers: [accountAddress] }];
151   try {
152     const signedTxnGroup = await peraWallet.signTransaction([txGroup]);
153     const { txId } = await client.sendRawTransaction(signedTxnGroup).do();
154     console.log('Info in handlePayment: ', 'Payment txn sent to Algorand network');
155     console.log('Info in handlePayment: ', 'txId: ', txId);
156     setStatus(done);
157   } catch (error) {
158     console.log('Error in handlePayment: ' + error.name + ' ' + error.message);
159     setStatus(done);
160     setError(error);
161   }
162 }
```

JavaScript Logic Overview

- **Event Handlers Setup:**
  When the DOM is fully loaded, event handlers for the buttons are set up (line 25, lines 29–41) and the last WalletConnect session is restored (line 26, lines 43–67).
- **Button Enabling/Disabling:**
  The logic for enabling or disabling buttons is implemented in lines 69–80, based on whether a successful connection to the wallet has been established.
- **Status Messages:**
  Status messages are displayed in lines 82–84, while error messages, if any, are handled in lines 86–97.
- **Connecting to the Wallet:**
  The logic for connecting to the wallet is in lines 103–121.

- **Disconnecting from the Wallet:**
  Handling the disconnection is covered in lines 123–138.
- **Processing Payment Transactions:**
  Payment transaction handling is implemented in lines 140–162.

5.9 Developing Applications Directly within the Web Browser

This type of application development bypasses the Node.js layer entirely. Developers work directly with HTML, CSS, and JavaScript files, running them in the web browser. JavaScript libraries used by the application are referenced in their bundled forms. During debugging, the browser's developer tools (accessible via F12) are used.

The `algosdk` JavaScript library is available in bundled form from CDNs and is approximately 300 KB in size. The following section explains how to bundle the `@perawallet/connect Node.js` module into a format usable by web browsers.

5.9.1 Bundling `@perawallet/connect`

The `@perawallet/connect` module is only available for Node.js and is not provided in a bundled form consumable by web browsers. To use it for browser-based development, the module must be bundled using a known web packer, such as webpack.

A solution for this task is provided in the repository `https://github.com/A-Maugli/pera-conect-vanillajs-demo-webpack`.

In this setup, the `@perawallet/connect` Node.js module is referenced in the `src/pwc.js` file.

```
1  //import {PeraWalletConnect} from "@perawallet/connect";
2  const pwcsdk = require("@perawallet/connect");
3  exports.pwcsdk = pwcsdk;
4
5  const algosdk = require("algosdk");
6  exports.algosdk = algosdk;
```

The `src/pwc.js` file exports both the Pera Wallet Connect SDK (referenced as `pwcsdk`) and the Algorand JavaScript SDK (referenced as `algosdk`).

Webpack Configuration

Below is the content of the `webpack.config.js file`:

```
1  const path = require('path');
2
3  module.exports = {
4    mode: 'production',
5    entry: './src/pwc.js',
6    output: {
7      filename: 'perawalletconnect.min.js',
8      path: path.resolve(__dirname, 'dist/browser'),
9      library: {
10       type: 'umd',
11       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: {
          configFile: path.resolve(__dirname, 'tsconfig-browser.json'),
        },
      },

      // All output '.js' files will have any sourcemaps re-processed by 'source-map-loader'.
      ////{ test: /\.js$/, loader: 'source-map-loader' },
    ],
    // Don't parse tweetnacl module –
    ↪  https://github.com/dchest/tweetnacl-js/wiki/Using-with-Webpack
    noParse: [/[\\/]tweetnacl[\\/]/, /[\\/]tweetnacl-auth[\\/]/],
  },
};
```

Explanation of the Webpack Configuration file

- The entry point for the library is defined in ./src/pwc.js (line 5).
- The library is configured as a umd (Universal Module Definition) module, making it accessible across various environments. Its name is set as pwc, short for Pera Wallet Connect (lines 10 and 11).
- The bundled library is output in the dist/browser directory (line 8).

The bundled library can be tested using the `public/test.html` file.

```
 <!DOCTYPE html>
<html>

<head>
  <title>Test for @perawallet/connect packed JavaScript library</title>
  <meta charset="UTF-8">
  <!--
  <script type="module" src="./browser/algosdk.min.js"></script>
  -->
  <!-- Note: perawalletconnect.min.js contains both pwcsdk and algosdk -->
  <!--       but algosdk version is v2.1 -->
  <script type="module" src="./jslib/perawalletconnect.min.js"></script>
</head>

<body>
<h2>Test for @perawallet/connect packed JavaScript library</h2>
<p>Use F12 to see Console</p>
```

```
<p>Expected:</p>
<p>success: Pera wallet connection created</p>
<p>success: algod client created</p>
<script>
window.onload = function(e) {
  const pwcsdk = pwc.pwcsdk;
  const algosdk = pwc.algosdk;

  // Create pera wallet connection
  try {
    const peraWallet = new pwcsdk.PeraWalletConnect();
    console.log('success: Pera wallet connection created');
  }
  catch (e) {
    console.log('error: ',  e);
  }

  // Create algod client
  const token = "";
  const server = "https://testnet-api.algonode.cloud";
  const port = 443;
  try {
    const client = new algosdk.Algodv2(token, server, port);
    console.log('success: algod client created');
  }
  catch (e) {
    console.log('error: ', e);
  }
}
</script>
</body>
</html>
```

The bundled library is referenced in the browser in the usual way using the `<script>...</script>` tags (line 12). The testing of the library is handled by the JavaScript code between lines 21 and 47.

We wait for the DOM to fully load, placing further test code inside the `window.onload` event handler. From the `pwc` object, the `pwcsdk` and `algosdk` objects are created (lines 23 and 24). A `PeraWalletConnect()` call is made (line 28), and the success or failure is logged in the console.

The Algorand JS SDK version 2.1 is included in `pwc` due to its inclusion as a dependency of `@perawallet/connect`. The test also demonstrates that an Algorand client can be created using the bundled library (line 40).

5.9.2 Example: Pera Wallet Connect Demo using Packed Libraries

This example revisits the full Pera Wallet Connect Demo with two associated files:

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

The bundled library is referenced in `public/index.html` as follows:

```
  <script type="module" src="./jslib/perawalletconnect.min.js"
    integrity="sha384-1/BfpY6oNlkbLhIQ2HqXVz2NzZb2zw5D6iDz6Qkdi1E6dCxZyJm1+NN9SMLamlXm"
```

```
    crossorigin="anonymous"></script>
```

This includes the SHA-384 hash for verifying the integrity of the library. The hash ensures the library hasn't been tampered with.

Only the beginning of the `public/pera_walletconnect_demo.js` file differs from the Node.js version.

```
"use strict";

//import algosdk from "algosdk";
////import { PeraWalletConnect } from "@perawallet/connect";
//import * as pwcsdk from "@perawallet/connect";
//import $ from "jquery";

const working = "Status: Working";
const done = "Status: Done";

const pwcsdk = pwc.pwcsdk;
const peraWallet = new pwcsdk.PeraWalletConnect();
...
```

5.9.3 Recording Foosball Match Results on the Algorand Testnet Blockchain

The note field of an Algorand transaction allows for up to 1 Kbyte of data. Each transaction costs 0.001 Algo[2], which, at the current exchange rate, is roughly equivalent to 0.0005 USD (0.05 cent). Transaction fees only increase in cases of network congestion.

The Algorand network currently supports a speed of 7500 transactions per second (tps). Since the typical traffic is only 50–100 tps, it is highly unlikely for fees to exceed 0.001 Algo. Moreover, this application runs on the Testnet blockchain, where free test Algos are readily available.

The demo application is available in the repository `https://github.com/A-Maugli/csocso-vanillajs-pwc`. The source code is located in the `src` directory. The `index.html` file references the following JavaScript libraries:

- algosdk.min.js
- jquery-3.7.1.min.js
- perawalletconnect.min.js

The HTML file consists of two main parts:

- Store Game Result (lines 23–35)
- Read Game Results (lines 41–47)

The user interface allows players to input their names and the match result.

[2] Transaction fees may change in the future. It is recommended to use constants provided by the JavaScript SDK, e.g. `const minFee = algosdk.ALGORAND_MIN_TX_FEE`

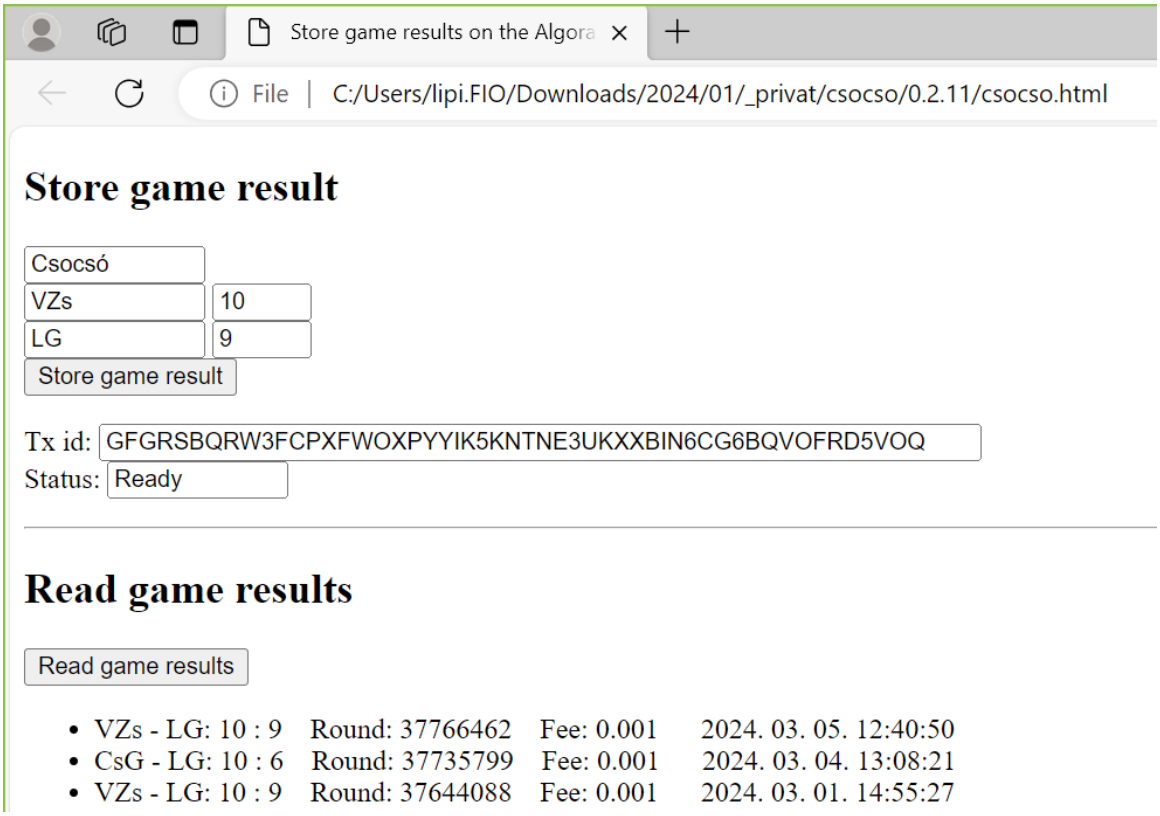

Figure 4: The User Interface of the foosball demo application

Storing Game Results

When the Store game result button is clicked: The application sends 0.001 Algo from the account `JW6L2Z...FNKILM` to the "universal sink" account `UUOB7Z...C7Q62E`. The match result is written in JSON format into the `note` field of the transaction.

Reading Games Results

When the Read game results button is clicked:

- A search operation is sent to the indexer.
- Transactions are retrieved where the source account is `JW6L2Z...FNKILM`.
- A filter is applied to display only those transactions where the note field contains a JSON structure in the correct format.

Here is the contents of `csocso.js` file, implementing this demo:

```
1  "use strict";
2
3  //const algod = require('algosdk');
4  //const pwcsdk = require('@perawallet/connect');
5  //const $ = require('jquery');
6
7  const ver = "0.2.12"; // Use PeraWalletConnect();
```

```
8
9   /* Algonode.io API */
10  const baseServer = "https://testnet-api.algonode.cloud";
11  const baseServerIdx = "https://testnet-idx.algonode.cloud";
12  const port = 443;
13  const token = "";
14
15  const working = "Working";
16  const ready = "Ready";
17  var addr_src, addr_dst;
18  var account_addr = '';  // address selected on PeraWallet
19
20  const testing = false;
21  if (testing) {
22    debugger;
23    addr_src = 'CDRLCYZICK7XEBZ7M4HKYWNB7ZZLO4BFOQ5RGZNMHQ5ZH7EZWUDFQ6Z32U';  // Test account 1
24    addr_dst = 'UUOB7ZC2IEE4A7JO4WY4TXKXWDFNATM43TL73IZRAFIFFOE6ORPKC7Q62E';
25  }
26  else {
27    addr_src = 'JW6L2ZCQT3UIQH5AFM3CVW3C7M3QFYHXA3EU4WHREOATRWMXP6MBFNKILM';  // Csocsó account
28    addr_dst = 'UUOB7ZC2IEE4A7JO4WY4TXKXWDFNATM43TL73IZRAFIFFOE6ORPKC7Q62E';
29  }
30
31  const algod_client = new algosdk.Algodv2(token, baseServer, port);
32  console.log('algod_client created');
33  const indexer_client = new algosdk.Indexer(token, baseServerIdx, port);
34  console.log('indexer_client created');
35
36  const pwcsdk = pwc.pwcsdk;
37  const peraWallet = new pwcsdk.PeraWalletConnect();
38  console.log('peraWallet created');
39
40  async function reconnectSessionA() {
41    let accounts = await peraWallet.reconnectSession();
42    if (peraWallet.connector !== null) {
43      peraWallet.connector.on("disconnect", handleDisconnectWalletA);
44    }
45    return accounts;
46  }
47
48  async function handleConnectWalletA() {
49    let newAccounts = await peraWallet.connect();
50    peraWallet.connector.on("disconnect", handleDisconnectWalletA);
51    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();
```

```
    if (accounts.length == 0) {
      accounts = await handleConnectWalletA();
    }
    if (accounts.length == 0) {
      throw ('Wallet connect error');
    }
    if (accounts[0] !== addr_src) {
      await handleDisconnectWalletA();
      throw ('Please connect to ' + addr_src);
    }
    else {
      account_addr = accounts[0];
    }

    //  get params from algod
    let params = await algod_client.getTransactionParams().do();

    let obj = {
      "from": account_addr,
      "to": addr_dst,
      "amount": 1,
      "note": algosdk.encodeObj(note),
      "suggestedParams": params
    };
    let txn = algosdk.makePaymentTxnWithSuggestedParamsFromObject(obj);
    // sign transaction
    let txGroup = [{ txn, signers: [account_addr] }];
    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 };
    try {
      if (typeof (tx.transactions[i].note) !== 'undefined') {
        //console.log('i:'+' note:'+ tx.transactions[i].note);
        const buff = Buffer.from(tx.transactions[i].note, 'base64');   // Node.js Buffer.from
        //const buff = base64js.toByteArray(tx.transactions[i].note);
        note1 = algosdk.decodeObj(buff);
      }
    }
    catch (e) {/*alert(e);*/ };

    if ((tx.transactions[i]['tx-type'] === "pay") && (note1.game === "Csocsó") &&
      (note1.goal1 !== undefined) && (note1.goal2 !== undefined)) {
      list += "<li>" + note1.user1 + ' - ' + note1.user2 + ': ' +
        note1.goal1 + ' : ' + note1.goal2;
      list += "   ";
      list += "\t Round: " + tx.transactions[i]['confirmed-round'];
      list += "   ";
      list += "\t Fee: " + tx.transactions[i].fee / 1000000.0;
      list += "   ";
      if (typeof note1.date !== "undefined") {
        list += "   ";
        let date = new Date(note1.date);
        let date1 = date.toLocaleDateString("hu-HU");
        let time1 = date.toLocaleTimeString("hu-HU");
        list += date1 + " " + time1;
      }
      list += "</li>";
    }
  }
  return list
}


function algo_get_tx() {
  if ($('#get_tx_status').val() !== working) {
    (async () => {
      $('#get_tx_status').val(working);
      $('#game_list').empty();

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

      while (num_tx > 0) {
        let tx = await indexer_client
```

```
179           .lookupAccountTransactions(addr_src)
180           .limit(tx_limit)
181           .nextToken(next_token).do();
182         num_tx = tx.transactions.length;
183         list += make_list_from_tx(tx);
184         next_token = tx['next-token'];
185       }
186       $('#get_tx_status').val(ready);
187       $("#game_list").append(list);
188       $("#game_list").show();
189     })().catch(e => {
190       console.log(e);
191     });
192   }
193 }
194
195 $(window.document).ready(function () {
196   console.log("document ready");
197   const game_name = "Csocsó";
198   const user1 = "CsG";
199   const user2 = "LG";
200   $('#game').val(game_name);
201   $('#user1').val(user1);
202   $('#user2').val(user2);
203   $('#goal1').val('0');
204   $('#goal2').val('0');
205   $('#tx_id').val('');
206   $('#send_tx_status').val('');
207
208   $('#game_list').empty();
209   $('#get_tx_status').val('');
210
211   $('#store_game_result').on("click", function () {
212     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 });
```

The Algod and Indexer REST endpoints are specified in lines 10 and 11. The `algod_client` and `indexer_client` are initialized in lines 31 and 33. The `peraWallet` instance is created in line 37. Execution continues within the `$(document).ready` block (line 195) after the DOM is fully loaded. HTML fields are initially populated in lines 197–209. Event handlers are set for the buttons: for `id="store_game_result"` in lines 211–214, for `id="read_game_results"` in lines 216–219. The functions called at button press are `algo_send_tx_outer()` and `algo_get_tx()` respectively.

The key asynchronous functions for handling Pera Wallet Connect are:

- reconnectSessionA (Async, lines 40–46) attempts to re-establish a connection to an existing wallet session. If successful, it returns the wallet's usable account numbers in the `accounts`

array.

- handleConnectWalletA (Async, lines 48–52) establishes a new connection to a wallet. On success, it returns the usable account numbers for wallet operations in the `newAccounts` array.
- andleDisconnectWalletA (Async, lines 54–58) terminates an existing wallet connection.

The `algo_send_tx` function

The `algo_send_tx` function sends the content of the `note` to the blockchain. The function attempts to use an existing wallet connection (line 64). If this connection does not offer the address from which we want to send the transaction, it throws an error (lines 71–73); otherwise, it places the account number into the `account_addr` variable (line 76).

It then reads the parameters from the Algorand network (e.g., starting block, genesis block, etc., line 80) and creates a payment transaction (lines 82–89). The transaction is signed with the Pera wallet (lines 91–92) and the signed transaction is sent to the Algorand network (line 94). The function writes the transaction ID to the HTML form (line 96) and starts waiting (for a maximum of 5 rounds) until the transaction is added to the blockchain (line 98).

If the transaction is written to the blockchain, the form is updated to indicate that the transaction was successfully sent (line 99). Otherwise, it catches the exception and writes the error to the form (line 102).

The `algo_send_tx_outer` function

The `algo_send_tx_outer` function reads parameters from the form (lines 109–114), constructs the `note` record (lines 115–122), and calls the previously explained `algo_send_tx` function.

The `algo_get_tx` function

The `algo_get_tx` function, using the indexer, retrieves transactions from the blockchain in a loop. These transactions have the source address specified by the user (lines 177–185).

Transactions are fetched in batches of `tx_limit` (50) at a time. The retrieved transactions are converted into a displayable format using the `make_list_from_tx` function (called at line 183, defined in lines 128–163).

Noteworthy Detail: `.do()` for Network Communication

An interesting detail to note is that communication with the network is performed by the client JS API functions when the `.do()` method is invoked. Examples include:

- Line 80: Fetching transaction parameters.
- Line 94: Sending the signed transaction.
- Line 181: Retrieving transactions using the indexer.

This `.do()` invocation ensures the client executes the required operations against the Algorand network.

5.10 Summary of the First Part

In the first part, we covered the following topics:

- Installing AlgoKit
- Using the Algorand `goal` command
- Using the Python API
- Using the JavaScript API
- Managing the Pera Wallet via the WalletConnect interface
- Bundling Node.js modules for use in a web browser
- Using the `note` field for record-keeping tasks

In the next part, we will discuss the programmability of the Algorand blockchain. Topics will include:

- Some instructions of the AVM (Algorand Virtual Machine)
- Using TEAL in a command-line environment
- Writing smart signatures or Algorand contracts using PyTeal
- Using the LORA blockchain explorer for testing Algorand contracts
- Using Beaker to write, deploy, and test Algorand contracts
- Using TealScript to write, deploy, and test Algorand contracts
- Debugging TEAL code
- The PuyaPy native Python compiler, which allows programs written in Python to be directly compiled into TEAL contract

6 TEAL

TEAL (Transaction Execution Approval Language) is the assembly language for the Algorand Virtual Machine (AVM). A compiled TEAL program is converted into bytecode, which the AVM can directly interpret.

6.1 Elements of a TEAL Program

A TEAL program consists of the following elements:

- `#pragma version`, which specifies the AVM version to be used. Currently, versions between 1 and 11 can be specified.
- labels, which can be the targets of jump insttructions. Examples: `l10:`, `subr_hello:`
- symbolic operation codes, which represent operations with mnemonic names to indicate their function. Examples: `dup`, `callsub`, `return`
- parameters for symbolic operation codes, e.g. for jump instructions a label, or the name of the transaction parameter in case of a transaction. Example: `asset_param_get AssetUnitName`
- comments. Example: `// this is a comment`

TEAL assigns a symbolic code to each operation in the AVM. he parameters for these operations can be stored within the bytecode itself, or Located in other memory areas defined by the AVM architecture, such as the stack or temporary storage area.

For example, the AVM byte code of the `dup` instruction is `0x49`. This instruction duplicates the value on the top of the stack, i.e. pushes the value onto the stack again.

6.2 Architecture of the AVM

The AVM (Algorand Virtual Machine) executes instructions sequentially, as indicated by the Program Counter (PC). Each instruction has access to the following resources:

- Stack
  - Capacity: 1000 elements
  - Data types:
    - 64-bit unsigned integer
    - byte[] string with a maximum length of 4096
- Temporary Storage
  - Capacity: 256 slots
  - Data types: see at Stack above
- Tranasction fields, accessible fields of accounts or ASAs within the transaction, such as the sender's Algorand address, the receiver's Algorand address, the amount being transferred etc.
- Global parameters, like `MinTxFee` minimal transaction fee, `GenesisHash`, used for network identification etc.

- Global Key-Value Pairs
  - Capacity:
    - max. 64 key-value pairs
    - max. 128 byte for a key-value pair
    - max. 8 Kbyte total space occupied
  - Access: rw for the app. creator, read for other apps
  - Minimum balance requirements: see here
- Local Key-Value Pairs
  - Capacity:
    - max. 16 key-value pairs
    - max. 128 byte for a key-value pair
    - max. 2 Kbyte total space occupied
  - Access: rw for the app. opted into, read for other apps
  - Minimum balance requirements: the opted-in account must fund local storage, see here
- Mailboxes
  - Capacity:
    - max. 32 Kbyte/mailbox
    - max. number: unlimited
  - Access: rw for the app, read for other apps
  - Minimum balance requirements: see here
- Parameters for Smart Contract or Smart Signature Calls. These are the parameters provided when invoking the smart contract or smart signature.

6.3 AVM Data Types

In the AVM, the stack can store only uint64 and []byte data types, where the maximum length of a []byte vector is 4096. Some AVM instructions have further restrictions on the range or types of values they accept. For example, bigint operations can only use the bigint data type.

Common AVM datatypes are:

- Unsigned integer, `uint64` $x, 0 \leq x \leq 2^{64} - 1$. Stored on the Stack as `uint64`
- Byte vector (string), `[]byte` x, where the length of x is not greater than 4096. Stored on the Stack as `[]byte`
- bigint, a non-negative integer with a length between 1 and 64 bytes. Stored as `[]byte` on the stack.
- address, a 32 byte bigint. Stored on the stack as `[32]byte`

Note: Algorand addresses consist of 58 characters. These characters encode:

  - The public key associated with the account, in base-32 encoding
  - A 4-byte checksum in base-32 encoding.

  In TEAL, the `addr` pseudo-instruction converts the 58-character Algorand address into a 32-byte array, which corresponds to the account's public key.

- method selector, a 4-byte value used to select a method, as defined in ARC-4.

References:

- The Algorand Virtual Machine (AVM) and TEAL.
- ARC-4: Application Binary Interface (ABI)
- TEAL specification
- go-algorand/data/transactions/logic/opcodes.goopcodes.go

6.4 Handling Underflow and Overflow

If an `uint64` number is subtracted from a smaller `uint64` number, the execution of the AVM bytecode terminates immediately with an error.

If the result of an operation between two `uint64` numbers exceeds $2^{64} - 1$, the execution of the AVM bytecode similarly terminates immediately with an error.

TEAL Example 1: data types

```
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
```

```
28 pushint 1311768465173141112
29 pushbytes 0x48656c6c6f // "Hello"
30 pushbytes 0x776f726c64 // "world"
31 pushbytes 0x123456789abcdef1234567 // 0x123456789abcdef1234567
32 pushbytes 0x89adc86a1a929533653ab9b6699e739cd7777d0905d44638402b908f0e6691db // addr
   ↪  RGW4Q2Q2SKKTGZJ2XG3GTHTTTTLXO7IJAXKEMOCAFOII6DTGSHNZTMGVXI
33 ==
34 pop
35 ==
36 pop
37 ==
38 lipi@lipi-VirtualBox:~/n_beta1$
```

Explanation of the example:

- Line 2 contains `#pragma version`, which specifies the AVM version to be used. Its value must be between 1 and 11.
- Line 3 defines a `uint64` number in decimal form.
- Line 4 demonstrates that an `uint64` can also be expressed in hexadecimal using the `0x` prefix.
- Lines 5 and 6 contain byte vectors (`[]byte`).
- Line 7 specifies an 11-byte bigint.
- Line 8 uses the `addr` pseudo-instruction to define an Algorand address, which is converted into a 32-byte array representing the public key.
- Line 15 shows the compile command. The TEAL program is compiled into bytecode using the `goal clerk compile` command. The compiled program has a `.tok` file extension.
- A hexadecimal dump of the bytecode is shown in lines 18–24.
- Line 25, disassembly with the -D flag reveals that in AVM version 10, the compiler uses `pushint` and `pushbytes` AVM codes to store constants on the stack. Line 32 demonstrates that the Algorand address is correctly converted into a 32-byte array.
- Line 33: The `==` operation removes the top two stack elements – a 32-byte address (line 32) and an 11-byte bigint (line 31) – and compares them. Since they are not equal, a `0` is pushed onto the stack.
- Line 35: The `==` operation compares two byte arrays (lines 30 and 29). Again, they are not equal, so a `0` is pushed onto the stack. The `pop` operation (line 36) discards this value.
- Line 37: The `==` operation compares two `uint64` values (lines 28 and 27). Since they are not equal, a `0` is pushed onto the stack.

@todo Running the bytecode in a simulator would help visualize its behavior step-by-step. To execute bytecode on the blockchain, it must be embedded into either a smart signature or a smart contract. The next section will explain how to achieve this.

6.5 Algorand Smart Signatures

@todo

6.6 Example: Using Algorand Smart Signatures

@todo

6.7 Algorand applications (smart contracts)

6.7.1 Maximum Bytecode Size

The bytecode for smart signatures can be a maximum of 1000 bytes long. Specifying parameters further reduces this size. For smart contracts, the combined size of the approval program and the clear program is limited to 2 KB by default. The size can be increased in 2 KB increments, and both the approval and clear programs can have a maximum size of 8 KB each.

6.7.2 Maximum Bytecode Execution Cost

During bytecode execution, the total cost (sum of the cost of individual operations) is capped at 20 000 units for smart signatures. Most bytecode operations have a cost of 1 unit. However, some operations are significantly more expensive. For example:

- `sha256` operation costs 35 units.
- `ed25519verify` operation costs 1900 units.

The cost of each operation is detailed in the documentation: TEAL Opcodes v10.

For smart contracts, the total cost of operations in the approval and clear programs is capped at 700 units per transaction. This maximum value can be significantly increased by using transaction groups and inner transactions. For example, if a transaction group contains 10 transactions, the maximum value is adjusted to $10 \times 700 = 7000$ units. Similarly, each inner transaction adds 700 units to the maximum value.

Since a transaction group can contain up to 16 transactions, and the number of inner transactions can be a maximum of 256, the total operation cost can reach $700 \times (16 + 256) = 190\,400$. In this case, the transaction execution fee also increases. For a flat fee, the maximum fee becomes $(16 + 256) \times 0.001$ Algo $= 0.272$ Algo. [3]

Reference: Operational Costs of TEAL Opcodes

Reference: Avoid Hard-Coding 1000 MicroAlgos as Minimum Fee

6.8 Example of Using Algorand Applications: "Hello, World"

The Algorand "HelloWorld" application can be created most easily using the `algokit init` command. The necessary steps are as follows:

- Log in to your account at `https://github.com`.
- Create a new repository, e.g., `asc`.
- Create a `README.MD` file in the repository.
- Click the "Code" button. Go to the "Codespaces" tab and launch Codespaces.

[3] Transaction fees may change in the future. It is recommended to use SDK constants, such as: `const minFee = algosdk.ALGORAND_MIN_TX_FEE` in the JavaScript SDK.

- In the displayed VS Code browser window, install `algokit`, by entering `pipx install algokit`.
- Start the localnet: `algokit localnet start`.
- Use the `docker ps` command to ensure that the four containers have started and are running.
- Make the ports 4001, 4002, and 8980 public on the Ports tab.
- Run the `algokit init` command as follows:
  - Select: `Smart Contracts & DApp Frontend`.
  - Select: `TypeScript`.
  - Set the directory name to: `ex`.
  - Choose the following template: `Starter`.
  - Keep the default name for the smart contract: `HelloWorld`.
  - When prompted, "Run the 'algokit project bootstrap' command?", select `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) $
```

The `algokit init` command creates the following directory structure:

```
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
```

6.8.1 The Backend

The Smart Contract

The `algokit init` command generates the code for a sample Algorand smart contract written in TypeScript in the file `ex-contracts/contracts/HelloWorld.algo.ts`:

```
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
   * @returns The difference between a and b.
   */
  private getDifference(a: uint64, b: uint64): uint64 {
    return a >= b ? a - b : b - a;
  }

  /**
   * A method that takes two numbers and does either addition or subtraction
   *
   * @param a The first uint64
   * @param b The second uint64
   * @param operation The operation to perform. Can be either 'sum' or 'difference'
   *
   * @returns The result of the operation
   */
  doMath(a: uint64, b: uint64, operation: string): uint64 {
    let result: uint64;

```

```
    if (operation === 'sum') {
      result = this.getSum(a, b);
    } else if (operation === 'difference') {
      result = this.getDifference(a, b);
    } else throw Error('Invalid operation');

    return result;
  }

  /**
   * A demonstration method used in the AlgoKit fullstack template.
   * Greets the user by name.
   *
   * @param name The name of the user to greet.
   * @returns A greeting message to the user.
   */
  hello(name: string): string {
    return 'Hello, ' + name;
  }
}
```

The contract has two callable methods: `doMath` and `hello`. The contract can be compiled into TEAL code as follows:

```
cd ex/projects/ex-contracts
```

```
npm run build
```

Files Generated During Compilation

The resulting files are generated in the `contracts/artifacts` directory:

- `HelloWorld.approval.teal`: The TEAL code for the smart contract, including its methods.
- `HelloWorld.arc4.json`: A JSON file describing the application's callable interface (ABI file, Application Binary Interface).
- `HelloWorld.arc32.json`: A JSON file describing the application, including the ARC4 ABI description.
- `HelloWorld.arc56_draft.json`: A JSON file with a more detailed description of the application, including the source map in JSON format. The `ARC56_draft` is not yet approved and can be found in Joe Polny's GitHub repository.
- `HelloWorld.clear.teal`: The TEAL code for unconditional execution during opt-out.
- `HelloWorld.src_map.json`: A map file that correlates TypeScript source lines, TEAL source lines, and AVM program counters (PCs).

The TypeScript file describing the Algorand smart contract is generated in the `contracts/clients` directory:

- `HelloWorldClient.ts`: A TypeScript wrapper that facilitates application calls.

Purpose of the `HelloWorldClient.ts` TypeScript Wrapper

The TypeScript Wrapper provides a high-level interface to an Algorand smart contract. The automatically generated code simplifies the use of smart contract operations in TypeScript applications.

The TypeScript Wrapper automates the creation of transactions required to interact with the smart contract and makes handling complex data structures more user-friendly:

- **Provides Abstraction:** It eliminates the need for in-depth knowledge of TEAL, offering a higher-level interface through the client.
- **Automates Transactions:** Handles transaction composition, signing, and network communication automatically.
- **Ensures Type Safety:** The generated code uses TypeScript types, reducing the likelihood of errors during development.

Components of the TypeScript Wrapper:

- **Smart Contract Specification:** The `APP_SPEC` object contains the specification of the smart contract, including the available methods (e.g., `doMath`, `hello`, `createApplication`) and their parameters and behavior. This information is fundamental for the client's functionality as it defines the operations that can be performed on the smart contract.
- **HelloWorldClient Class:** A class that manages high-level interactions with the smart contract's methods. The smart contract's various functions are accessible as simple TypeScript function calls, such as `doMath()` or `hello()`.
- **HelloWorldCallFactory Class:** This class provides predefined functions for creating transactions to interact with the smart contract. For instance, the `doMath()` method constructs a transaction to perform addition or subtraction of two numbers via the smart contract.
- **OnCompletion Types and State Management:** This includes the definition of types required for application calls and the management of data stored in the contract's (global) state. It utilizes `IntegerState` and `BinaryState` types for handling state data.
- **Deploy:** The `deploy()` method allows the smart contract to be deployed in an idempotent manner. Idempotent behavior ensures that the same result is achieved even if the deployment is executed multiple times.

The TypeScript Wrapper simplifies the process of invoking Algorand smart contracts. It abstracts the low-level details (e.g., handling transactions, transaction groups, and signatures) and enables the rapid and straightforward development of decentralized applications.

Testing the Contract

The contract testing relies on the Jest framework. After installing the `jest` and `ts-jest` npm modules, tests written in TypeScript can be executed directly within the Jest framework. A more detailed description of Jest can be found in Chapter 10, *Test-Driven Development*, of Nathan Rozentals' book *Mastering TypeScript*.

The Jest test associated with the contract is located in the file `ex-contracts/__test__/Helloworld.test.ts`:

```
import { describe, test, expect, beforeAll, beforeEach } from '@jest/globals';
import { algorandFixture } from '@algorandfoundation/algokit-utils/testing';
import * as algokit from '@algorandfoundation/algokit-utils';
import { HelloWorldClient } from '../contracts/clients/HelloWorldClient';

const fixture = algorandFixture();
algokit.Config.configure({ populateAppCallResources: true });

let appClient: HelloWorldClient;

describe('HelloWorld', () => {
  beforeEach(fixture.beforeEach);

  beforeAll(async () => {
    await fixture.beforeEach();
    const { testAccount } = fixture.context;
    const { algorand } = fixture;

    appClient = new HelloWorldClient(
      {
        sender: testAccount,
        resolveBy: 'id',
        id: 0,
      },
      algorand.client.algod
    );

    await appClient.create.createApplication({});
  });

  test('sum', async () => {
    const a = 13;
    const b = 37;
    const sum = await appClient.doMath({ a, b, operation: 'sum' });
    expect(sum.return?.valueOf()).toBe(BigInt(a + b));
  });

  test('difference', async () => {
    const a = 13;
    const b = 37;
    const diff = await appClient.doMath({ a, b, operation: 'difference' });
    expect(diff.return?.valueOf()).toBe(BigInt(a >= b ? a - b : b - a));
  });

  test('hello', async () => {
    const diff = await appClient.hello({ name: 'world!' });
    expect(diff.return?.valueOf()).toBe('Hello, world!');
  });
});
```

Explanation:

- **Line 1: Importing Jest Globals**
  - `describe`: Used to describe Jest test groups.

- `test`: Used to define individual Jest tests.
- `expect`: Used to assert the results of the tests.
- `beforeAll`: Runs setup code before all tests in a group.
- `beforeEach`: Runs setup code before each individual test.

- **Line 2: Importing `algorandFixture`** The `algorandFixture` initializes key Algorand parameters, such as the `algorandClient` variable (line 17), which contains a reference to `algod` (Algorand daemon) in the `algorand.client.algod` property.
- **Line 11: Defining a Test Group**
- **Line 12: Pre-test Setup for Each Test** The `fixture.beforeEach` function is executed before every test.
- **Line 14: Test Setup for the Group**
  - **Line 15:** Waits for the Algorand fixture to initialize.
  - **Line 16:** Extracts `testAccount` from the fixture.
  - **Line 17:** Extracts `algorand: algorandClient` from the fixture.
  - **Lines 19–26:** Creates a new `appClient`.
  - **Line 28:** Deploys a new app (smart contract) on the blockchain. *Note:* `app`, `application`, and `smart contract` are synonymous terms.
- **Lines 31–36: Defining a Test**
  - **Line 34:** Invokes the `doMath` method of the live contract via `appClient`. The `appClient` manages transaction creation, signing, submission, and provides the structured result.
  - **Line 35:** Asserts that the result matches $a + b$.
- **Lines 38–43: Defining Another Test** This test invokes the `doMath()` method of the app and verifies that the method returns the correct result.
- **Lines 45–48: Defining Yet Another Test** This test invokes the `hello()` method of the app and verifies the result of the string concatenation performed by the app.

The tests can be run by

```
npm run test
```

The result of the test execution:

```
 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.
```

If debugging of Jest TypeScript code is required, after pressing the "Run and Debug" button in VS Code, start a debug terminal. In the debug terminal, run the command `npm run test` to start execution. The execution will halt at the specified breakpoints, allowing you to step through the execution step by step.

6.8.2 The Frontend

The `ex-frontend` directory contains a React frontend. Navigate to the `ex-frontend` directory and run the client generation and packer program:

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

```
npm run dev
```

The `package.json` file reveals what happens during this process:

```
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"
  },
...
```

The `algokit project link -all` command generates typed clients for all contracts in the `ex-frontend/src/contract` directory.

The `vite` command starts the `vite` packer in development mode (fast, nearly instant compilation) and makes the React application accessible via a hyperlink.

If using `codespaces`, the `.env` file must also be modified as follows: Copy the URL associated with port 4001 from the Ports tab and replace `localhost` with it. Change the port 4001 to 443:

Old:

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

New:

```
VITE_ALGOD_SERVER=https://(value copied from the Ports tab)
VITE_ALGOD_PORT=443
```

Similarly, update the `VITE_KMD_SERVER`, `VITE_KMD_PORT`, and `VITE_INDEXER_SERVER`, `VITE_INDEXER_PORT` lines in the `.env` file.

Then, execute the following commands in the `ex-frontend` directory:

```
# To create vite.env file, and to create and install node_modules, enter:
algokit project bootstrap all
# Start an Algorand local node
algokit localnet start
```

```
# Check docker containers
docker ps
# It is recommended to run the `docker ps` command multiple times,
# as the conduit service currently tends to crash.
# In such cases, you can try using the `algokit localnet reset` command.

# Make ports 4001, 4002, and 8980 public on the "Ports" tab

# Last,
npm run dev
# and then start the React app in the browser with CTRL + click
  →  Local:   http://localhost:5173/
# In the browser, press F12 in order to start a development environment.
```

6.8.3 The ABI

The ABI (Application Binary Interface) defines the data types that can be used when invoking a method of a smart contract. Defining usable data types enables strongly typed parameters to be used for calling smart contracts.

The concept of ABI was originally defined in Ethereum, as described in the ABI Specification.

In Algorand, the arc-0004 document defines the ABI, specifying the data types usable within the Algorand development environment. The definition of read-only parameters can be found in arc-0022, and log events are described in arc-0028.

Type conversion between the ABI layer and TEAL (AVM) is automatically handled by various development environments, such as TypeScript and PuyaPy. Consequently, smart contracts that use the ABI are longer and more complex due to the required data conversions. For example, the `string` ABI data type stores the string length in the first two bytes, followed by the string itself. In contrast, Algorand's `[]byte` data type implicitly contains the string length based on the number of data bytes pushed onto the stack.

6.8.4 "HelloWorld" Example Program Without Using ABI

A "HelloWorld" example program implemented in PuyaPy Python *without* using the ABI is as follows:

contract.py

```
from algopy import Contract, Txn, log

class HelloWorldContract(Contract):
    def approval_program(self) -> bool:
        name = Txn.application_args(0)
        log(b"Hello, " + name)
        return True

    def clear_state_program(self) -> bool:
        return True
```

Explanation of the Python Code:

In line 3, the `HelloWorldContract` contract is derived from the `Contract` class. The `approval_program` method either allows or denies the execution of the transaction, returning 1 or 0, respectively. The `clear_state_program` method enables an unconditional exit in the case of opting out of the contract. In line 5, the first parameter of the transaction used to invoke the application is read (parameters in TEAL are indexed starting from 0), and line 6 logs the result of `"Hello, " + name`.

When the PuyaPy compiler is invoked, the two methods are compiled into separate TEAL files. The approval TEAL program generated from the above Python code is as follows:

HelloWorldContract.approval.teal

```
1 #pragma version 10
2
3 examples.hello_world.contract.HelloWorldContract.approval_program:
4     byte "Hello, "
5     txna ApplicationArgs 0
6     concat
7     log
8     int 1
9     return
```

Explanation:

In line 4, the TEAL program pushes the string `"Hello, "` onto the AVM stack. In line 5, it pushes the first parameter of the application (note that parameters are indexed starting from 0). In line 6, it concatenates the two strings, and in line 7, the resulting string is removed from the stack and recorded as a log entry on the blockchain. Finally, in line 8, the value 1 is pushed onto the stack. This serves as the return value of the approval program, with the value 1 indicating that the execution of the transaction is approved.

6.8.5 `"HelloWorld"` Example Program Using ABI

A `"HelloWorld"` example program implemented in PuyaPy Python using the ABI is as follows:

contract.py

```
1 from algopy import ARC4Contract, String, arc4
2
3 class HelloWorldContract(ARC4Contract):
4     @arc4.abimethod
5     def hello(self, name: String) -> String:
6         return "Hello, " + name
```

The approval TEAL program is as follows:

HelloWorldContract.approval.teal

```
1 #pragma version 10
2
3 examples.hello_world_arc4.contract.HelloWorldContract.approval_program:
4     txn NumAppArgs
5     bz main_bare_routing@5
```

```
6       method "hello(string)string"
7       txna ApplicationArgs 0
8       match main_hello_route@2
9       err // reject transaction
10
11  main_hello_route@2:
12      txn OnCompletion
13      !
14      assert // OnCompletion is NoOp
15      txn ApplicationID
16      assert // is not creating
17      txna ApplicationArgs 1
18      extract 2 0
19      callsub hello
20      dup
21      len
22      itob
23      extract 6 2
24      swap
25      concat
26      byte 0x151f7c75
27      swap
28      concat
29      log
30      int 1
31      return
32
33  main_bare_routing@5:
34      txn OnCompletion
35      !
36      assert // reject transaction
37      txn ApplicationID
38      !
39      assert // is creating
40      int 1
41      return
42
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
```

Explanation of the Program:

In line 4, the TEAL program pushes the number of parameters onto the AVM stack. Line 5 jumps to the label `main_bare_routing@5` if no parameters are provided during the application call, as seen in line 33. The transaction's `OnCompletion` field can take the following values:

- `NoOp (0)`
  - If the `AppId` is zero, the application is being created.

  - If the `AppId` is non-zero, the application is being called.
- `OptIn (1)`
  - If the `AppId` is zero, it is invalid.
  - If the `AppId` is non-zero, it opts into the application.
- `CloseOut (2)`
  - If the `AppId` is zero, it is invalid.
  - If the `AppId` is non-zero, it opts out of the application.
- `ClearState (3)` - cannot occur in the approval program.
- `UpdateApplication (4)`
  - If the `AppId` is zero, it is invalid.
  - If the `AppId` is non-zero, it updates the application.
- `DeleteApplication (5)`
  - If the `AppId` is zero, it is invalid.
  - If the `AppId` is non-zero, it deletes the application.

In line 34, the `OnCompletion` value is pushed onto the stack. Line 35 performs a negation, pushing 0 onto the stack if the value is non-zero, or 1 if the value is zero. In line 36, the `assert` immediately halts the program execution if the value at the top of the stack is zero. Thus, the execution is terminated if `OnCompletion` is not equal to `NoOp`. Similarly, the program halts immediately if `ApplicationID` is not zero (line 39). These two `assert` instructions ensure that the application is being created and the completion code is `NoOp`. By pushing the value 1 onto the stack in line 40, the application creation is approved. For all other cases—if the completion code is not `NoOp` or the application is not being created—the call terminates with an error.

In line 6, the instruction `method "hello(string)string"` generates a 4-byte hash of the `hello` method and places it onto the stack, following the ABI standard. In line 7, the first parameter of the application is pushed onto the stack. If the two values match, execution continues at the label `main_hello_route@2`; otherwise, the execution immediately terminates with an error.

Lines 12 to 16 check if the conditions `OnCompletion == NoOp` and `ApplicationID != 0` are met. If not, execution terminates with an error.

In line 17, the second call parameter, which is the `name` ABI string, is pushed onto the AVM stack. Line 18 strips off the first two bytes encoding the string length, converting the ABI string type to a native AVM string. In line 19, the `hello` subroutine is called, which concatenates the `"Hello, "` string with the AVM string present in the call frame.

From this point onward, the ABI-compliant encoding of the result is performed, converting the AVM result string to an ABI result string in the following manner:

Line 20 duplicates the AVM result string. Line 21 determines the length of the result string. The `itob` instruction in line 22 converts the 8-byte `uint64` integer, representing the string length, into an 8-byte string. Line 23 extracts the last 2 bytes of this string. The `swap` instruction in line

24 swaps the top two elements of the stack, placing the AVM result string on top and its length in 2-byte format just below. For instance, if the result length is 17, the 2-byte representation would be `0x31 0x37`. Line 25 concatenates the 2-byte length and the result string.

Next, the result is prepared according to arc-0028. The 4-byte prefix that must be prepended to the result is generated as follows:

```
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);
```

Counterintuitively, the 4-byte prefix for the event log is not derived from hashing "hello(string)" but instead from hashing the word "return". Line 26 obtains the first 4 bytes of the hash, which is `0x151f7c75`. Lines 27 and 28 prepend this hash value to the current string, which already includes the 2-byte length followed by the AVM result string. Line 29 logs this result to the blockchain. Finally, line 30 pushes the value 1 onto the stack to approve the transaction, and line 31 concludes the execution of this branch.

It is evident how much more complex the "HelloWorld" contract has become with the use of ABI compared to its simpler counterpart without ABI.

References:

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

Books:

Nathan Rozentals *Mastering TypeScript*, 4th Edition

7 Case Study: Optional Buying Right for Ownership Share

7.1 Task Description

The "Circle of Trust" company aims to sell 10 tokens, each representing 0.1% of its ownership share, to ten buyers. For this purpose, the company issues 10 tokens on a blockchain and sells them through a smart contract. The tokens can be purchased within 30 days of issuance. Each token represents an optional buying right for a 0.1% ownership share. A buyer may only purchase one token.

To exercise the optional buying right associated with the token, the token holder must visit the Circle of Trust's secretariat and present:

- the purchased token stored in their wallet,
- their personal identification document,
- and proof of payment for the purchase price of the 0.1% ownership share transferred to the Circle of Trust's bank account.

At this point, the token holder will receive official documentation certifying the acquisition of the ownership share, and the token will be revoked.

If the token holder does not exercise the optional buying right within 4 days of purchasing the token, the token will automatically revert to the smart contract's ownership and become available for resale.

Note: The process would be significantly simplified if the token not only granted an optional buying right but directly represented the ownership share itself. However, due to current legal regulations, personal data submission would still be required.

7.2 Conceptual Solution to the Task

The tokens representing the optional buying rights for ownership shares in the "Circle of Trust" can be created as an ASA, or Algorand Standard Asset, on the Algorand blockchain.

The ASA parameters are as follows:

- Short Name: BKTOVJ
- Long Name: Circle of Trust Token
- Quantity: 10 units
- Website URL: `https://algorand.hu/bk/bktovj.html`

The ASA can be sold via a smart contract. The tasks of the smart contract are:

- **Selling the ASA:** Tokens are sold on a "first come, first served" basis at a fixed price. The smart contract ensures that upon full payment of the token price, the token is transferred to the buyer's wallet. By owning the token, the buyer gains an optional buying right.
- **Automatic retrieval of expired tokens:** If the token holder does not exercise the optional buying right within the specified time frame, the token is returned to the smart contract's account. To achieve this, the contract executes a "clawback" operation and increases the amount

of available tokens for sale by one.

- **Exercising the optional buying right:** If the buyer presents the token along with the required documents at the "Circle of Trust" secretariat, the token is "revoked" by the administrator. To perform this operation, the contract executes a "clawback" operation but does not increase the amount of tokens available for sale.

Account Addresses:

- The Algorand address of the smart contract creator
- The Algorand address associated with the smart contract
- The Algorand address of the buyer's account

The ASA purchase is initiated by the buyer through a web interface that invokes the smart contract's ASA selling method. A WalletConnect interface links the buyer's wallet to the smart contract, allowing the buyer to sign the transaction generated by the smart contract to complete the purchase.

To simplify the conditions for executing a "clawback," the smart contract freezes the token in the buyer's wallet at the time of sale. This enables the contract to check the time difference between the current time and the purchase time, and if it exceeds a specified threshold (e.g., 4 days), the "clawback" can be executed.

Note: The smart contract cannot autonomously initiate transactions. To implement the "clawback" functionality, a scheduled process must regularly identify wallets holding expired tokens and invoke the contract's "clawback" function.

7.3 Explanation of the Smart Contract

The Algorand application or smart contract that solves the task can be found in the BizKor.algo.ts file. The smart contract is written in TypeScript and is compiled into TEAL code using the TealScript compiler.

References:

- TEALScript Source Code
- TEALScript Documentation
- TEALScript Example Programs

7.3.1 Defining the Smart Contract Class

```
1  import { Contract } from '@algorandfoundation/tealscript';
2  // version history ...
```

```
31 // eslint-disable-next-line no-unused-vars
32 class BizKor extends Contract {
33 // BizKor state variables and methods...
34 }
```

In line 1, the `Contract` class is imported.

In line 32, the `BizKor` smart contract class is derived from the `Contract` base class.

7.3.2 Defining Global State Variables

```
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' });
48 }
```

In lines 33–47, the global state variables are defined.
These are key-value pairs, with a maximum of 64 pairs allowed in a smart contract.

After `GlobalStateKey`, the key's type must be specified, such as `string`, `uint64`, `Address`, or `AssetID`. Optionally, a short key name can be provided using `{key: 'short-key-name'}` for space optimization.
This short key name is used internally by the AVM code to reference the key-value pair.

To assign a value to a global state variable, use: `this.keyName.value = value`.

To access the value of a global variable, use: `this.keyName.value`.

7.3.3 `createApplication` – Called After Smart Contract Creation

```
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   }
```

When the smart contract is created, the `createApplication` method is also executed. In lines 53–60, the global state variables are initialized.

7.3.4 `bootstrap` – Setting Initial Parameters

```
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  }
```

The `bootstrap` method of the smart contract is responsible for setting the initial parameters. As seen in line 70, the method parameters define the token price, the number of tokens to be created, the sales period duration in seconds, and the token expiration period, also in seconds.

The check in line 72 ensures that only the contract creator can call this procedure.

The validation in line 75 prevents the `bootstrap` method from being called more than once. It might have been more appropriate to include a dedicated global state variable, `already_bootstrapped`, for this purpose.

The creation of the ASA (Algorand Standard Asset) is carried out in lines 78–89 using an inner transaction. In lines 87 and 88, the freeze and clawback addresses are specified, which, in this case, correspond to the smart contract's Algorand address.

The global state variables are set in lines 92–97. The assignment `this.asset.value = asset`

not only stores the asset ID but also other properties of the asset.

7.3.5 Reading Global State Variables

Specific methods are provided for reading global state variables, which perform data conversion according to ABI standards. However, the `appClient.getGlobalState()` call allows retrieving all state variables in raw form at once without making individual calls to the app. This makes the `getGlobalState()` call "much cheaper."

The methods:

- `getAppCreatorAddress(): Address`, line 104
- `getAppVersion(): string`, line 112
- `getAssetAmountInitial(): uint64`, line 120
- `getAssetAmount(): uint64`, line 128
- `getAssetPrice(): uint64`, line 136
- `getAssetID(): AssetID`, line 144
- `getSellPeriodEnd(): uint64`, line 152

The getter function for querying `assetValidityPeriod` was accidentally omitted from the list.

7.3.6 `buyAsset` – Purchasing a Token

```
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? (this can be optimized away)
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 (this can be optimized away)
199     sendAssetFreeze({
200       freezeAsset: this.asset.value,
201       freezeAssetAccount: this.txn.sender,
202       freezeAssetFrozen: true,
203     });
204
205     // Decrease asset amount (this can be optimized away)
206     this.assetAmount.value = this.assetAmount.value - 1;
207   }
```

The token purchase is implemented by the `buyAsset` method. This method has a single parameter, a payment transaction that serves to buy the token. From examining the generated TEAL code, it is evident that this transaction is referenced within the method as part of a transaction group. During the execution of a transaction group, either all transactions are executed atomically, or none are executed. This ensures that, if no errors occur, the buyer receives the token in exchange for the payment transaction, but if an error occurs, the payment transaction does not proceed.

The `buyAsset` method performs several checks:

- Line 162 checks if the purchase time is before the end of the selling period. If this condition is not met, the method execution terminates with an error.
- Line 165 verifies that the buyer does not already own any of the tokens (i.e., "0 tokens are owned").
- Lines 168–172 check the following conditions:
  - The sender of the payment transaction is the same as the sender of the transaction invoking the smart contract (line 169).
  - The recipient of the payment transaction matches the smart contract's Algorand address (line 170).
  - The amount sent in the payment transaction is neither less than nor greater than the price stored in the global state variable (line 171).
- Line 175 verifies that there are tokens still available for sale.

Subsequently, lines 178–182 opt into the ASA using an inner transaction. This step was included

because earlier attempts without it resulted in an "xxx" error. These lines may be redundant if the opt-in step is performed before the assertion in line 165.

Lines 185–189 first unfreeze the token, then lines 192–196 transfer the token to the buyer, and finally, lines 199–203 freeze the token under the buyer's address.

The reader may reasonably ask why the unfreezing of the token is necessary if the buyer does not yet own the token. This is required for the later revocation process: if a buyer purchases a token, and it is subsequently revoked due to expiration, the buyer's address remains in a "frozen" state, and a token cannot be sent to a frozen address.

Finally, `buyAsset` reduces the value of the global state variable tracking the number of tokens available for sale (line 206).

Note: Since `buyAsset` uses four inner transactions, invoking this method requires covering the transaction fees for these inner transactions as well. Therefore, the `buyAsset` method must be invoked with at least five times the minimum transaction fee.

7.3.7 `sendAlgosToCreator` – Returning Token Sale Proceeds

```
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   }
```

The proceeds from token sales can be sent from the smart contract to its creator by invoking the `sendAlgosToCreator` method. Similar to the `bootstrap` method, this method verifies the identity of the caller (see line 214) and only allows execution by the smart contract creator.

The method ensures that 0.6 Algos remain in the smart contract's address, transferring only the excess amount to the smart contract creator (lines 219–223).

7.3.8 `clawback` – Token Clawback

```
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   }
```

A token can be repossessed by invoking the `clawback` method. As a parameter, the address from which the token is to be reclaimed must be specified. Lines 236–241 perform the clawback: using an internal transaction, the token is sent from the specified address back to the contract. This is possible because the contract's address was designated as the clawback address when the token was created. Line 244 increments the number of tokens available for sale.

Note: An external scheduled process checks which tokens have "expired" and reclaims them using the `clawback` method. It would be prudent to enhance the clawback logic by verifying that the token has indeed expired, i.e., the time elapsed since the last operation (purchase) exceeds the duration specified by the `assetValidityPeriod`.

7.3.9 `clawbackNoIncAmount` – Token Revocation

Revocation of a token upon successful acquisition of ownership shares can be done by invoking the `clawbackNoIncAmount` method. It is similar to the `clawback` method but does not increment the number of tokens available for sale.

7.3.10 `deleteAsset` – Deleting the ASA

```
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   }
```

Before the smart contract can be deleted, it must no longer hold the created ASA. The `deleteAsset` method deletes the ASA using an internal transaction, as shown in lines 274–276. Interestingly, such an ASA can be deleted with a `sendAssetConfig` call.

7.3.11 `deleteApplication` – Invoked Before Deleting the Smart Contract

```
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  }
```

The `deleteApplication` method is invoked during the deletion of the smart contract. For it to execute successfully, the `deleteAsset` method must be called beforehand, and `deleteAsset` requires that all tokens are already present in the smart contract's address. The internal transaction shown in lines 287–291 of `deleteApplication` transfers the remaining Algorand from the contract's address to the creator's address.

7.4 Testing the Smart Contract

The Jest file for testing the smart contract can be found at BizKor.test.ts.

Among the Algokit Typescript utilities, the return values of algorandFixture.ts are as follows:

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

where the `context` is defined as follows:

```
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)),
```

```
  }
```

This means that the `context` contains:

- an `algod` Algorand client
- a `kmd` key management daemon client
- a `testAccount` test account number
- a `generateAccount` function
- a `transactionLogger` function, among others.

References:

- Jest Documentation
- Algokit Typescript Utilities
- algorandFixture.ts

7.4.1 Pre-test setup for each Jest test

```
/* eslint-disable no-console */
import { describe, test, expect, beforeAll, beforeEach } from '@jest/globals';
import { algorandFixture } from '@algorandfoundation/algokit-utils/testing';
import * as algokit from '@algorandfoundation/algokit-utils';
import algosdk, { Transaction } from 'algosdk';
import { TransactionSignerAccount } from '@algorandfoundation/algokit-utils/types/account';
import { BizKorClient } from '../contracts/clients/BizKorClient';

const fixture = algorandFixture();
algokit.Config.configure({ populateAppCallResources: true });

let appClient: BizKorClient;

describe('BizKor', () => {
  const log = false; // skip console.log() calls
  const paramAppVersion = 'v1.3'; // app version
  const paramAssetPrice = 1_000_000; // microAlgos
  const paramAssetAmountInitial = 10; // pieces
  const paramSellPeriodLength = 1000; // sec
  const paramAssetValidityPeriod = 100; // sec

  let acc1: algosdk.Account;
  let signer1: TransactionSignerAccount;
  let acc2: algosdk.Account;

  beforeEach(fixture.beforeEach);

  beforeAll(async () => {
    await fixture.beforeEach();
    const { algod, kmd } = fixture.context;

    acc1 = await algokit.getOrCreateKmdWalletAccount(
      {
        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  });
```

The following lines detail the setup for each Jest test:

- Line 2 imports the Jest testing functions required for the tests.
- Line 3 imports the `algorandFixture`, which provides the clients and accounts needed for testing.
- Line 4 imports the Algokit utilities for working with Algorand.
- Line 5 imports the `algosdk` API.
- Line 6 imports the `TransactionSignerAccount` type.
- Line 7 imports the `BizKorClient`, a TypeScript wrapper for interacting with the `BizKor` smart contract.
- Line 9 defines the Jest fixture as `algorandFixture()`.
- Line 10 provides configuration options for smart contract calls.
- Line 12 defines the smart contract client. This strongly typed client, generated by the TEALScript

compiler, simplifies interactions with the contract.

- Line 14 begins the Jest test suite. Logging for tests can be toggled on or off in line 15.
- Lines 16–20 specify the parameters for the smart contract.
- Line 22 sets `acc1` as the account for the buyer.
- Line 24 defines `acc2` as the account for creating the smart contract.
- Line 23 defines `signer1`, the signing entity for `acc1`.

Setup before each test:

- Line 26 invokes `beforeEach(fixture.beforeEach)` to run the fixture before every test.
- Lines 28–70 are executed only once before all tests:
  - Line 29 runs `fixture.beforeEach()`, which initializes `algod`, `kmd`, and other values in the `context`.
  - Line 30 retrieves `algod` and `kmd` from the `fixture.context`.
  - Lines 32–39 create or retrieve `acc1`, a KMD account for the buyer, and fund it with 100 Algo.
  - Lines 42–48 configure `signer1`, the transaction signer for the buyer. The `signer1` object includes `acc1`'s address and a signing function for transaction groups.
  - Lines 50–57 create or retrieve `acc2`, the account used to create the smart contract, and fund it with 100 Algo.
  - Lines 60–67 initialize the smart contract client for interacting with the smart contract. A new `appClient` is created since the provided `id` parameter is 0.
  - Line 69 deploys the smart contract on the blockchain and invokes the `createApplication` method after deployment.

7.4.2 `bootstrap` test

```
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  });
```

In line 79, the `bootstrap` method of the application/smart contract is invoked.

- The first `{ }` block specifies the parameters for the method.
- The second `{ }` block specifies the parameters for the transaction that calls the smart contract, including the transaction fee.

The transaction fee is set to double the usual amount due to the internal transaction executed within the method.

7.4.3 `getAppVersion` test

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

7.4.4 `getAppCreatorAddress` test

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

7.4.5 `getAssetAmountInital` test

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

7.4.6 `getAssetAmount` test

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

7.4.7 `getAssetPrice` test

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

7.4.8 `getAssetId` test

```
114 test('getAssetId', async () => {
115   const assetId = await appClient.getAssetId({});
```

```
    expect(assetId.return).toBeGreaterThan(BigInt(1_000));
  });
```

7.4.9 getSellPeriodEnd test

```
  test('getSellPeriodEnd', async () => {
    const sellPeriodEnd = await appClient.getSellPeriodEnd({});
    // get date/time
    const now = new Date();
    // get msec since 1970
    const millisecondsSinceEpoch = now.getTime();
    // get sec from msec
    const secondsSinceEpoch = Math.floor(millisecondsSinceEpoch / 1000);
    // check sellPeriodEnd
    if (log) console.log('sellPeriodEnd: ', sellPeriodEnd.return);
    expect(sellPeriodEnd.return).toBeGreaterThan(BigInt(secondsSinceEpoch)); // "algokit localnet
    ↪  reset" may be required
    expect(sellPeriodEnd.return).toBeLessThan(BigInt(secondsSinceEpoch + paramSellPeriodLength));
  });
```

7.4.10 getGlobalState test

```
  test('getGlobalState', async () => {
    const globalState = await appClient.getGlobalState();
    const apv = globalState.apv!.asByteArray();
    const apca = globalState.apca?.asByteArray();
    const asaTotal = globalState.asa_total?.asNumber();
    const asaAmt = globalState.asa_amt?.asNumber();
    const asaPrice = globalState.asa_price?.asNumber();
    const asaId = globalState.asa_id?.asNumber();
    const end = globalState.end?.asNumber();
    const asaV = globalState.asa_v?.asNumber();
    // console.log('globalState:', globalState);

    // get apvGood, i.e. without the length (first 2 bytes)
    const apvGood = Buffer.from(apv).slice(2).toString('utf-8'); // get rid of length
    if (log) console.log('apvGood: ', apvGood);
    expect(apvGood).toBe(paramAppVersion);
    // get apcaGood. i.e. encode 32 byte Algorand address as string
    const bufferApca = Buffer.from(apca!);
    if (log) console.log('bufferApca: ', bufferApca);
    if (log) console.log('bufferApca.length: ', bufferApca.length);
    const apcaGood = algosdk.encodeAddress(bufferApca); // encode as Algorand address
    if (log) console.log('apcaGood: ', apcaGood);
    expect(apcaGood).toBe(acc2.addr);

    if (log) console.log('getGlobalState apv (appVersion):', apv);
    if (log) console.log('getGlobalState apca (appCreatorAddress):', apca);
    if (log) console.log('getGlobalState asa_total (assetAmountInitial):', asaTotal);
    expect(asaTotal).toBe(paramAssetAmountInitial);
    if (log) console.log('getGlobalState asa_amt (assetAmount):', asaAmt);
    expect(asaAmt).toBe(paramAssetAmountInitial);
    if (log) console.log('getGlobalState asa_price (assetPrice):', asaPrice);
    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   });
```

7.4.11 State Retrieval and Validation

In line 134, the `getGlobalState()` call retrieves all global state variables.

Line 135 processes the `apv` (appVersion) value as a byte array. The first two bytes in the array store the ABI string length. These two bytes are removed in line 146, and line 148 verifies that the retrieved value matches the parameters specified at the beginning of the tests.

Line 136 processes the `apca` (appCreatorAddress) value as a byte array. The array contains 32 bytes corresponding to the public key of the associated Algorand account. Line 153 formats this public key into its ASCII representation.

Lines 137–142 retrieve the remaining global state variable values using the `asNumber()` getter. In these cases, no additional formatting is required.

7.4.12 `opt in to asset` test

```
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   });
```

In lines 177–183, the `acc1.addr` address opts into the asset (ASA). The asset ID is retrieved from the global state in line 175.

7.4.13 `buyAsset` Test

```
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   });
```

Line 190: Loading `algod`. Line 191: Loading `params`, transaction parameters. Line 193: Loading `appRef`. Lines 198–203: Creating the payment transaction where the sender address is `acc1.addr`, and the recipient address is `appRef.appAddress`.

Line 206: Reading the global state. Line 207: Retrieving `asa_id` from the global state. Lines 208–219: Invoking the `buyAsset` method using `compose`. Note the following:

- In line 215, the transaction fee is set to 5 times the minimum fee.
- In line 217, the `assets` array specifies the `asset_id`. Without this, an "Unavailable asset" error would occur.

Line 221: Retrieving the transactions. Line 222: Assigning a group ID to the transactions. Lines 223–226: Signing the transactions.

Explanation for the construction in line 225, according to ChatGPT4:

> The expression `Array.from(Array(txs.length), (_, i) => i)` creates an array containing the indices of the transactions, indicating which transactions to sign.
>
> - `Array(txs.length)` creates a new array with a length equal to `txs.length`, ini-

tially filled with `undefined`.

- `Array.from()` generates a new array from the one provided as input. Here, the first parameter is the array filled with `undefined`.
- The second parameter is a function applied to each element (initially `undefined`) and its index (`i`). The function simply returns the index (`i`), resulting in an array containing indices from 0 up to `txs.length - 1`.

Note: The `buyAsset` invocation can be simplified by using the `execute` method to handle the creation, signing, and submission of the transaction group. In this case, the end of the `buyAsset` test would be modified as follows:

```
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 });
```

Please note the `execute()` method used in line 218.

In line 213, a total fee equivalent to 5 transactions was required, calculated using the `algokit.transactionFees` function. This is because:

- The invocation of the app's `buyAsset` method constitutes 1 transaction.
- Within the `buyAsset` method, there are 4 additional inner transactions.

7.4.14 `buyAsset 2nd time` test

```
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   });
```

This test is identical to the `buyAsset` test, but in this case, the application returns an error because the buyer already owns a token. The error handling occurs in lines 272–274.

7.4.15 `sendAlgosToCreator` test

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

7.4.16 `clawback` test

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

7.4.17 `buyAsset after clawback` test

```
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);
```

```
    if (log) console.log('buyAsset: testAccount.addr ', testAccount.addr);
    if (log) console.log('buyAsset: appRef.appAddress ', appRef.appAddress);
    if (log) console.log('buyAsset: appCreatorAddr ', acc2.addr);
    const tx1 = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
      from: acc1.addr,
      to: appRef.appAddress,
      amount: paramAssetPrice,
      suggestedParams: params,
    });

    // Buy asset
    const globalState = await appClient.getGlobalState();
    const asset = globalState.asa_id!.asNumber();
    const compose = appClient.compose().buyAsset(
      {
        payment: tx1,
      },
      {
        sender: signer1,
        sendParams: {
          fee: algokit.transactionFees(5),
        },
        assets: [Number(asset)],
      }
    );
    // atc, build group, sign, send
    const atc = await compose.atc();
    const txs = atc.buildGroup().map((tx) => tx.txn);
    const signed = await signer1.signer(
      txs,
      Array.from(Array(txs.length), (_, i) => i)
    );
    const txg = await algod.sendRawTransaction(signed).do();
    await algosdk.waitForConfirmation(algod, txg.txId, 4);
  });
```

7.4.18 clawback again test

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

7.4.19 'opt out buyer from asset' test

```
  test('opt out buyer from asset', async () => {
    const { algod } = fixture.context;
    const params = await algod.getTransactionParams().do();
    const globalState = await appClient.getGlobalState();
    const asset = globalState.asa_id!.asNumber();
    const appRef = await appClient.appClient.getAppReference();
    if (log) console.log('Try to opt out from asset: ', acc1.addr);
    const txn1 = algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
      from: acc1.addr,
```

```
      to: appRef.appAddress,
      closeRemainderTo: appRef.appAddress,
      amount: 0,
      assetIndex: asset,
      suggestedParams: params,
    });
    const stxn1 = txn1.signTxn(acc1.sk);
    const txn2 = await algod.sendRawTransaction(stxn1).do();
    await algosdk.waitForConfirmation(algod, txn2.txId, 4);
  });
```

In lines 338–345, all of the buyer's tokens are sent back to the smart contract, with the `closeRemainderTo` field set to the address of the smart contract. This is equivalent to the buyer "opting out" of the given asset.

7.4.20 'deleteAsset' test

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

7.4.21 'deleteApplication' test

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

7.5 Running the Tests

The tests can be executed under Codespaces using the following commands:

```
algokit --version
cd biz_kor/projects/biz_kor-contracts
npm install
npm audit
npm audit fix
npm run build
algokit localnet start
algokit localnet status
npm run build
# edit BizKor.algo.ts, see clawbackNoIncAmount
# edit BizKor.algo.ts, see line 229, 250
npm run build
npm run test
```

The output of the commands is as follows:

```
# Sample output for illustrative purposes
PASS  __tests__/BizKor.test.ts
  ✓ createApplication initializes correctly (200 ms)
  ✓ bootstrap sets initial parameters (300 ms)
```

```
  ✓ buyAsset works as expected (500 ms)
  ✓ clawback reclaims expired tokens (250 ms)
  ✓ deleteApplication cleans up properly (150 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.5 s
Ran all test suites.
```

```
Welcome to Codespaces! You are on our default image.
   - It includes runtimes and tools for Python, Node.js, Docker, and more. See the full list here: https://aka.ms/ghcs-default-image
   - Want to use a custom image instead? Learn more here: https://aka.ms/configure-codespace

To explore VS Code to its fullest, search using the Command Palette (Cmd/Ctrl + Shift + P or F1).

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
@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

Error when parsing tsdoc comment for clawback: Error: address is not an argument of clawback
Error when parsing tsdoc comment for clawbackNoIncAmount: Error: address is not an argument of clawbackNoIncAmount

/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modules/node-fetch/lib/index.js:1501
                        reject(new FetchError(`request to ${request.url} failed, reason: ${err.message}`, 'system', err));
                               ^
FetchError: request to http://localhost:4001/v2/teal/compile?sourcemap=true failed, reason:
    at ClientRequest.<anonymous> (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modules/node-fetch/lib/index.js:1501:11)
    at ClientRequest.emit (node:events:519:28)
    at emitErrorEvent (node:_http_client:108:11)
    at Socket.socketErrorListener (node:_http_client:511:5)
    at Socket.emit (node:events:519:28)
    at emitErrorNT (node:internal/streams/destroy:169:8)
    at emitErrorCloseNT (node:internal/streams/destroy:128:3)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  type: 'system',
  errno: 'ECONNREFUSED',
  code: 'ECONNREFUSED'
}

Node.js v20.17.0
@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
```

```
Error when parsing tsdoc comment for clawback: Error: Addr is not an argument of clawback
Error when parsing tsdoc comment for clawbackNoIncAmount: Error: Addr is not an argument of clawbackNoIncAmount

/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modules/node-fetch/lib/index.js:1501
                        reject(new FetchError(`request to ${request.url} failed, reason: ${err.message}`, 'system', err));
                               ^
FetchError: request to http://localhost:4001/v2/teal/compile?sourcemap=true failed, reason:
    at ClientRequest.<anonymous> (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modules/node-fetch/lib/index.js:1501:11)
    at ClientRequest.emit (node:events:519:28)
    at emitErrorEvent (node:_http_client:108:11)
    at Socket.socketErrorListener (node:_http_client:511:5)
    at Socket.emit (node:events:519:28)
    at emitErrorNT (node:internal/streams/destroy:169:8)
    at emitErrorCloseNT (node:internal/streams/destroy:128:3)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  type: 'system',
  errno: 'ECONNREFUSED',
  code: 'ECONNREFUSED'
}

Node.js v20.17.0
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $ algokit localnet status
# container engine
Name: docker (change with `algokit config container-engine`)
Error: LocalNet has not been initialized yet, please run 'algokit localnet start'
@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: indexer-db Pulling
docker: indexer Pulling
docker: proxy Pulling
docker: indexer Pulled
docker: conduit Pulled
docker: proxy Pulled
docker: algod Pulled
docker: indexer-db Pulled
docker: Network algokit_sandbox_default  Creating
docker: Network algokit_sandbox_default  Created
docker: Container algokit_sandbox_postgres  Creating
docker: Container algokit_sandbox_algod  Creating
docker: Container algokit_sandbox_algod  Created
docker: Container algokit_sandbox_postgres  Created
docker: Container algokit_sandbox_conduit  Creating
docker: Container algokit_sandbox_conduit  Created
docker: Container algokit_sandbox_indexer  Creating
docker: Container algokit_sandbox_indexer  Created
docker: Container algokit_sandbox_proxy  Creating
docker: Container algokit_sandbox_proxy  Created
docker: Container algokit_sandbox_algod  Starting
```

```
docker: Container algokit_sandbox_postgres  Starting
docker: Container algokit_sandbox_algod  Started
docker: Container algokit_sandbox_postgres  Started
docker: Container algokit_sandbox_conduit  Starting
docker: Container algokit_sandbox_conduit  Started
docker: Container algokit_sandbox_indexer  Starting
docker: Container algokit_sandbox_indexer  Started
docker: Container algokit_sandbox_proxy  Starting
docker: Container algokit_sandbox_proxy  Started
docker: Container algokit_sandbox_algod  Waiting
docker: Container algokit_sandbox_conduit  Waiting
docker: Container algokit_sandbox_postgres  Waiting
docker: Container algokit_sandbox_indexer  Waiting
docker: Container algokit_sandbox_proxy  Waiting
docker: Container algokit_sandbox_proxy  Healthy
docker: Container algokit_sandbox_postgres  Healthy
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: Not running
# indexer-db status
Status: Running
# indexer status
Status: Not running
# proxy status
Status: Running
Error: At least one container isn't running; execute `algokit localnet start` to start the LocalNet
@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

Error when parsing tsdoc comment for clawback: Error: Addr is not an argument of clawback
Error when parsing tsdoc comment for clawbackNoIncAmount: Error: Addr is not an argument of
↪  clawbackNoIncAmount
@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


> biz_kor-contracts@0.0.0 build
> npm run compile-contract && npm run generate-client


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


> biz_kor-contracts@0.0.0 generate-client
> algokit generate client contracts/artifacts/ --language typescript  --output
↪  contracts/clients/{contract_name}Client.ts

Generating TypeScript client code for application specified in
↪  /workspaces/akt02/biz_kor/projects/biz_kor-contracts/contracts/artifacts/BizKor.arc32.json and
↪  writing to contracts/clients/BizKorClient.ts
Reading application.json file from path
↪  /workspaces/akt02/biz_kor/projects/biz_kor-contracts/contracts/artifacts/BizKor.arc32.json
Generating TS client for BizKor
Writing TS client to
↪  /workspaces/akt02/biz_kor/projects/biz_kor-contracts/contracts/clients/BizKorClient.ts
Operation completed successfully

  console.info
    LocalNet account 'Buyer of Biz.Kör. token' doesn't yet exist; created account
    ↪  ZKDY63B3QNQPZ6ZHD3HWTCG2MAYSQOQRWVY6CLQQSXXPZY533YYWRCDAQY with keys stored in KMD and
    ↪  funding with 100 ALGOs

      at Object.getOrCreateKmdWalletAccount
      ↪  (node_modules/@algorandfoundation/src/localnet/get-or-create-kmd-wallet-account.ts:51:17)

  console.debug
    Transferring 100000000µALGOs from LM7ZQJLPE6QJ4NYBNPMZHEZXFLCNSY32665QTX64MF2VKWJJKGYBTEIDJ4
    ↪  to ZKDY63B3QNQPZ6ZHD3HWTCG2MAYSQOQRWVY6CLQQSXXPZY533YYWRCDAQY
```

```
    at Object.transferAlgos
    ↪  (node_modules/@algorandfoundation/src/transfer/transfer-algos.ts:38:46)

console.info
  LocalNet account 'App creator' doesn't yet exist; created account
  ↪  J3U6O32GZ66RNMP7IP5OQCYPEHTYYS6YHTHXCHIWKUEDAIFVWRW7DDCL5U with keys stored in KMD and
  ↪  funding with 100 ALGOs

    at Object.getOrCreateKmdWalletAccount
    ↪  (node_modules/@algorandfoundation/src/localnet/get-or-create-kmd-wallet-account.ts:51:17)

console.debug
  Transferring 100000000µALGOs from LM7ZQJLPE6QJ4NYBNPMZHEZXFLCNSY32665QTX64MF2VKWJJKGYBTEIDJ4
  ↪  to J3U6O32GZ66RNMP7IP5OQCYPEHTYYS6YHTHXCHIWKUEDAIFVWRW7DDCL5U

    at Object.transferAlgos
    ↪  (node_modules/@algorandfoundation/src/transfer/transfer-algos.ts:38:46)

console.debug
  Created app 1004 from creator J3U6O32GZ66RNMP7IP5OQCYPEHTYYS6YHTHXCHIWKUEDAIFVWRW7DDCL5U

    at Object.createApp (node_modules/@algorandfoundation/src/app.ts:115:48)

console.debug
  Transferring 600000µALGOs from J3U6O32GZ66RNMP7IP5OQCYPEHTYYS6YHTHXCHIWKUEDAIFVWRW7DDCL5U to
  ↪  SW4BTZGCCNMSDYSANRLKFQOZNYRKH4B7J6DNSEUAYPSRGOYZRGKQEAW5EY

    at Object.transferAlgos
    ↪  (node_modules/@algorandfoundation/src/transfer/transfer-algos.ts:38:46)

console.log
  getGlobalState asa_id (asset): 1008

    at __test__/BizKor.test.ts:165:13

console.log
  this test should fail, as the buyer already has a coin URLTokenBaseHTTPError: Network request
  ↪  error. Received status 400 (Bad Request): TransactionPool.Remember: transaction
  ↪  PCY7D6URKBX5QS7JCN7T6K6NDIJGFZFACRPMVYN5FI6MO3OSOPNQ: logic eval error: assert failed
  ↪  pc=486. Details: app=1004, pc=486, opcodes=intc_1 // 0; ==; assert
      at Function.checkHttpError (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modu
      ↪  les/@algorandfoundation/src/types/urlTokenBaseHTTPClient.ts:129:11)
      at processTicksAndRejections (node:internal/process/task_queues:95:5)
      at Function.formatFetchResponse (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node
      ↪  _modules/@algorandfoundation/src/types/urlTokenBaseHTTPClient.ts:137:5)
      at AlgoHttpClientWithRetry.callWithRetry (/workspaces/akt02/biz_kor/projects/biz_kor-contr
      ↪  acts/node_modules/@algorandfoundation/src/types/algo-http-client-with-retry.ts:30:20)
      at AlgoHttpClientWithRetry.post (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node
      ↪  _modules/@algorandfoundation/src/types/algo-http-client-with-retry.ts:68:12)
      at HTTPClient.post (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_modules/algo
      ↪  sdk/src/client/client.ts:269:19)
      at SendRawTransaction.do (/workspaces/akt02/biz_kor/projects/biz_kor-contracts/node_module
      ↪  s/algosdk/src/client/v2/algod/sendRawTransaction.ts:53:17)
      ↪  {
    response: {
      body: {
```

```
          data: [Object],
          message: 'TransactionPool.Remember: transaction
          ↪  PCY7D6URKBX5QS7JCN7T6K6NDIJGFZFACRPMVYN5FI6MO3OSOPNQ: logic eval error: assert
          ↪  failed pc=486. Details: app=1004, pc=486, opcodes=intc_1 // 0; ==; assert'
        },
        status: 400,
        headers: {
          '291': 'content-length',
          'keep-alive': 'connection',
          'application/json; charset=UTF-8': 'content-type',
          'Thu, 17 Oct 2024 17:11:57 GMT': 'date',
          Origin: 'vary'
        },
        text: '{"data":{"app-index":1004,"eval-states":[{},{"stack":[0,0]}],"group-index":1,"pc":4
        ↪  86},"message":"TransactionPool.Remember: transaction
        ↪  PCY7D6URKBX5QS7JCN7T6K6NDIJGFZFACRPMVYN5FI6MO3OSOPNQ: logic eval error: assert failed
        ↪  pc=486. Details: app=1004, pc=486, opcodes=intc_1 // 0; ==; assert"}\n',
        ok: false
      },
      status: 400
    }

      at __test__/BizKor.test.ts:273:15
          at Generator.throw (<anonymous>)

 PASS  __test__/BizKor.test.ts (42.277 s)
  BizKor
    ✓ bootstrap (1498 ms)
    ✓ getAppVersion (1445 ms)
    ✓ getAppCreatorAddress (1380 ms)
    ✓ getAssetAmountInitial (1529 ms)
    ✓ getAssetAmount (1541 ms)
    ✓ getAssetPrice (1410 ms)
    ✓ getAssetId (1442 ms)
    ✓ getSellPeriodEnd (1400 ms)
    ✓ getGlobalState (1351 ms)
    ✓ opt in to asset (1304 ms)
    ✓ buyAsset (1835 ms)
    ✓ buyAsset 2nd time (1381 ms)
    ✓ sendAlgosToCreator (1139 ms)
    ✓ clawback (1398 ms)
    ✓ buyAsset after clawback (1518 ms)
    ✓ clawback again (1408 ms)
    ✓ opt out buyer from asset (1364 ms)
    ✓ deleteAsset (1369 ms)
    ✓ deleteApplication (1414 ms)

Test Suites: 1 passed, 1 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        42.389 s
Ran all test suites.
@A-Maugli → .../akt02/biz_kor/projects/biz_kor-contracts (main) $
```

7.6 The Frontend

The `algokit`-generated frontend application assumes the use of the React framework.

The generated React application includes components for handling various wallets (using the WalletConnect interface) and provides examples for creating and interacting with Algorand applications and invoking their methods. However, it does not design the graphical user interface for the developer or handle the correct invocation of application methods within the UI.

Fortunately, the tests written during the `contract` development process are highly reusable for creating React components. These tests already contain much of the necessary logic for app functionality. Moreover, during the development of the TealScript contract, the React component skeletons can be automatically generated by running the `npm run generate-components` command.

Note: This generator is scheduled for deprecation and will no longer be maintained in the future.

7.7 Adjustments and Expansions for React Frontend

Naturally, adjustments or significant expansions to the code might be necessary compared to the tests. For example, in the tests, the Algorand application creation (`createApplication`) and parameterization (`bootstrap`) were handled separately. In a React application, these can be consolidated into a single step. Another example is the handling of the "Biz. Kör." token clawback. During testing, only one token was managed at a time. The React frontend, however, iterates through all accounts holding the token, checks if the grace period has expired for any of these accounts, and, in a loop, recalls all tokens that have not been used to purchase ownership shares within the allowed timeframe.

Understanding the React frontend requires familiarity with React fundamentals. Developers can learn about the React framework from the following resources:

- 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 sections, 42 hours of video, see `https://www.udemy.com`.

7.7.1 Frontend npm Scripts

The scripts available in the `package.json` file are as follows:

```
$ 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"
  },
...
```

The corresponding `npm` commands are as follows:

- `npm run generate:app-clients`: Generates "typed client wrapper" files for the `contract`s and places them in the `src/contracts` directory. These TypeScript wrappers simplify the management of Algorand smart contracts, such as app creation, modification, method invocation, and transaction group handling.
- `npm run dev`: Generates the "typed client wrapper" files and then starts the vite frontend development system. In this "development mode," the React application automatically rebuilds upon changes to the React source files. The vite server makes the application accessible via `http://localhost:5137` in a browser.
- `npm run build`: Generates the "typed client wrapper" files and then invokes the vite build command, which uses rollup.js to bundle the many files of the React application into a few larger `js` and `css` files. The output is stored in the `dist` directory.
- `npm run test`: Runs Jest test files with the `*.spec.ts` extension. For example, a test for `ellipseAddress.ts` can be found in the `biz_kor-frontend/src/utils` directory.
- `npm run playwright.test`: Tests the application at the browser level. A sample test file is located at `biz_kor-frontend/test/example.spec.ts`. Running this requires a Python environment, and its absence may cause failures.
- `npm run lint`: Checks for formatting and syntactical errors.
- `npm run lint:fix`: Checks and automatically fixes formatting and syntactical errors.
- `npm run preview`: Launches the vite web server, serving the bundled React application from the `dist` directory at `http://localhost:4173/`.

7.7.2 Frontend Development Process

During the development of `bizkor-frontend`, both the `contract` and the `frontend` utilize the generated "typed client" file. Wrapper generation is integrated into every script command on the `frontend` side:

```
"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"
  }
```

The `generate:app-clients` script copies the wrapper files to the `frontend/src/contract` directory. Other commands also invoke this script.

When developing TealScript `contract`s, there is an option to automatically generate React components for the `frontend` by running the `npm run generate-components` command. Note: This generator will be deprecated in the future and is planned for removal.

7.7.3 Algorand-Specific Features

- Wallet management
- Account handling
- Management of `algod`, `kmd`, and `indexer`

Issues and Questions During Development

- **Version Management:** The `@algorandfoundation/tealscript` package appears as `"latest"` in the `package.json` file. The specific version in use is only recorded in the `package-lock.json` file. Caution: If the `package-lock.json` file is deleted, running `npm install` may install a different version of TealScript. This can lead to issues, especially if error handling depends on the program counter (PC) to generate user-friendly error messages, as a newer TealScript version might generate TEAL code differently.
  - **Solution:** (1) Retain the `package-lock.json` file, or (2) specify the exact TealScript version in the `package.json` file.
- **Generator Formatting Issue:** If the Algorand TealScript contract name is in uppercase (e.g., `ABC`), the generated typed client file will be named `ABCClient.ts`, but it must be imported as `AbcClient`.
- **Generator Formatting Issue:** Global variables in the Algorand TealScript app that use underscores (e.g., `abc_def`) are referenced as `abcDef` in the frontend typed client.
- **React Component Design Question:** A separate "call bootstrap" button or component is unnecessary; the bootstrap method can be included in the "create DAO" component.
- **React-Specific Task:** After invoking the "call bootstrap" method, state variables such as the "price of one token" or "number of tokens available" must be updated in the parent component.
  - **Solution:** The parent component's event handler was passed to the child component as a prop and invoked from the child component. The parent component's event handler then updates the state.

```
// 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 Framework-Specific Question:** How can one obtain a reference to an `algod` client?

  Solution:

```
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 Framework-Specific Question:** How can the `optIn` transaction be sent?

  Solution: Retrieve the asset ID from the global state (line 4), construct a transaction that sends 0 units of the specified asset (lines 5–11), and sign and send the transaction using `sendTransaction` (line 12).

```
   console.log(`Opt in to asset`)
   const params = await algod.getTransactionParams().do();
   const globalState = await props.typedClient.getGlobalState();
   const asset = globalState.asaId!.asNumber();
   const txn1 = algosdk.makeAssetTransferTxnWithSuggestedParamsFromObject({
       from: activeAddress!,
       to: activeAddress!,
       amount: 0,
       assetIndex: asset,
       suggestedParams: params,
     });
   algokit.sendTransaction({transaction: txn1, from: sender }, algod);
   //await algokit.waitForConfirmation(txn1.txID(), 4, algod);
```

  Note: Whether `algokit.sendTransaction` waits after sending the transaction depends on the `algokit` configuration settings.

- **Algorand Framework-Specific Question:** How can the `buyAsset` transaction be sent?

  Solution: The `compose` method of the typed client must be invoked to create a *transaction group*. In this group, the first transaction is the `payment` transaction, and the second is the call to the Algorand smart contract's `buyAsset` method (lines 1–12). The `execute` method (line 12) signs the transactions in the transaction group.

  Note: The Biz.Kör token's asset ID must be specified in the `foreign assets` array (line 11, `assets: [...]`) to avoid the error: "Asset not found."

```
1      const result = await props.typedClient.compose().buyAsset(
2        {
3          payment: tx1,
4        },
5        {
6          sender: sender,
7          sendParams: {
8            fee: algokit.transactionFees(5),
9          },
10         assets: [Number(asset)],
11       }
12     ).execute();
13     const waitRoundsToConfirm = 4;
14     await algokit.waitForConfirmation(result.txIds[0], waitRoundsToConfirm, algod);
```

- **React-Specific Question:** How can a payment transaction be passed as a prop?

  Solution: @todo

- **Algorand Framework-Specific Question:** How can the `appCreatorAddress` be retrieved?

  Solution:

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

- **Algorand Framework-Specific Question:** How can the `price` be retrieved?

  Solution: From the global state:

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

- **Algorand Framework-Specific Problem:** How can user-friendly error messages be sent?

  Solution: The map file can be used to determine which `assert` corresponds to a specific TEAL program counter (pc) value. When an error occurs, a user-friendly error message can be displayed based on the pc value:

```
1    try {
2      enqueueSnackbar('A vételi tranzakció elküldése...', { variant: 'info' })
3      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      //...
```

- **Conceptual Problem:** In the `.env` file, the variables `admin_mode` and `app_id` were used to control:
  - (1) The role: `admin_mode="true"` for the contract creator and `"false"` for regular users.
  - (2) Whether the contract needs to be created (`app_id=0`) or if it has already been created (`app_id=nnn`).

  Unfortunately, this file is embedded into the final application, making this solution unsuitable.

  Solution: Use process environment variables?

8 Algokit Update, October 29, 2024

8.1 Git Update

The `winget list` command can be used to determine if there is a newer version of previously installed programs, in this case, Git.

Before starting the Git update, the open workspace in VS Code must be closed. During the update, the commands executed in the VS Code terminal are:

```
winget list –id Git.Git
```

```
winget upgrade –id Git.Git
```

```
winget list –id Git.Git
```

```
git –version
```

The update log:

```
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> winget list --id Git.Git
Name Id      Version Source
---------------------------
Git  Git.Git 2.47.0  winget
PS C:\Users\lipi.FIO> git --version
git version 2.47.0.windows.1
PS C:\Users\lipi.FIO>
```

8.2 Docker Desktop Update

To determine the version of `Docker Desktop`, first launch the application. This can be done by double-clicking on the `Docker Desktop` icon.

The following commands are used in the VS Code terminal for the update:

```
winget list –id Docker.DockerDesktop
```

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

The `Docker Desktop` update takes approximately 3–4 minutes. After the update, a Windows restart is required.

The update log for `Docker Desktop`:

```
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 update --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%20Desktop%20Installer.exe
  ████████████████████████████████████████████████████████████████  463 MB /
  ↪  463 MB
Successfully verified installer hash
Starting package install...
Successfully installed
```

After restarting Windows, launch `Docker Desktop` by double-clicking on the `Docker Desktop` icon. At this point, you will see the following message:

```
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
```

To check the version number, run the following command:

```
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 Update

The commands used during the node.js update process:

```
winget list --id OpenJS.NodeJS

winget update --exact --id OpenJS.NodeJS

node --version

npm --version
```

The node.js update log is as follows:

```
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 update --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
```

```
   [                                                                ] 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 Update

To determine the installed Python version, use the command: `winget list python.python.3` The installed version was `Python 3.12.3`.

Check the Python Downloads website to find the latest available version. On October 29, 2024, `Python 3.13` was available.

Commands used during the update process:

`winget uninstall python.python.3.12`

`winget install python.python.3.13`

After the restart of `VS Code`:

`python --version`

`=> 3.13.0`

Note: In case of errors during the Python update, it is recommended to check the `Path` environment variable in both the **User** and **System** sections. Ensure that there are no leftover paths pointing to the previous Python version.

8.5 pipx update

`python -m pip uninstalll pipx`

Delete the directory `c:\Users\Lipi.FIO\pipx`, then:

`python -m pip install pipx`

8.6 Reinstalling `algokit`

`pipx install algokit`

`=> installed package algokit 2.4.3, installed using Python 3.13.0`

`=> These apps are now globally available`

`=> - algokit.exe`

After updating, some antivirus programs, such as Bitdefender, might incorrectly flag certain files as infected and move them to quarantine. For example:

```
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.
```

To restore these files from quarantine:

- Navigate to the antivirus program's **Protection** settings.
- Locate the **Quarantined Threats** section.
- Click **Manage Quarantine** (specific to Bitdefender).

In Bitdefender, the program displays the quarantined files. Two files were restored in this case:

- `pythoncom313.dll`
- `pywintypes313.dll`

These files can also be verified on the VirusTotal website. Only a small fraction (9/71) of security companies consider these files to be infected.

8.7 Post-Update Tasks

1. **`algokit init` fails to locate `python3.EXE` after Python upgrade:**

```
algokit init
...
Unhandled CalledProcessError: Command '"C:\Users\lipi.FIO\AppData\Local\Microsoft\WindowsApps\python3.EXE" pre_init.py' returned non-zero exit status 9009.
```

To resolve this issue, execute the following command in an Administrator mode PowerShell:

```
New-Item -ItemType SymbolicLink -Path "C:\Windows\System32\python3.exe" -Target "C:\Users\lipi.FIO\AppData\Local\Programs\Python\Python313\python.exe"
```

2. **Error during the first run of the `algokit init` generated frontend:**

```
npm run dev
...
file:///C:/Users/lipi.FIO/Downloads/2024/10/ALGO/Hackaton_FR/circle_of_trust/ct/projects/ct-frontend/tailwind.config.js:2
module.exports = {
^

ReferenceError: module is not defined
```

The solution is to rename the `tailwind.config.js` file to `tailwind.config.cjs`.

3. **Clear temporary files:** After the update, it is recommended to delete files in the `AppData\Local\Temp` directory. This can free up approximately 3.5 GB of disk space.