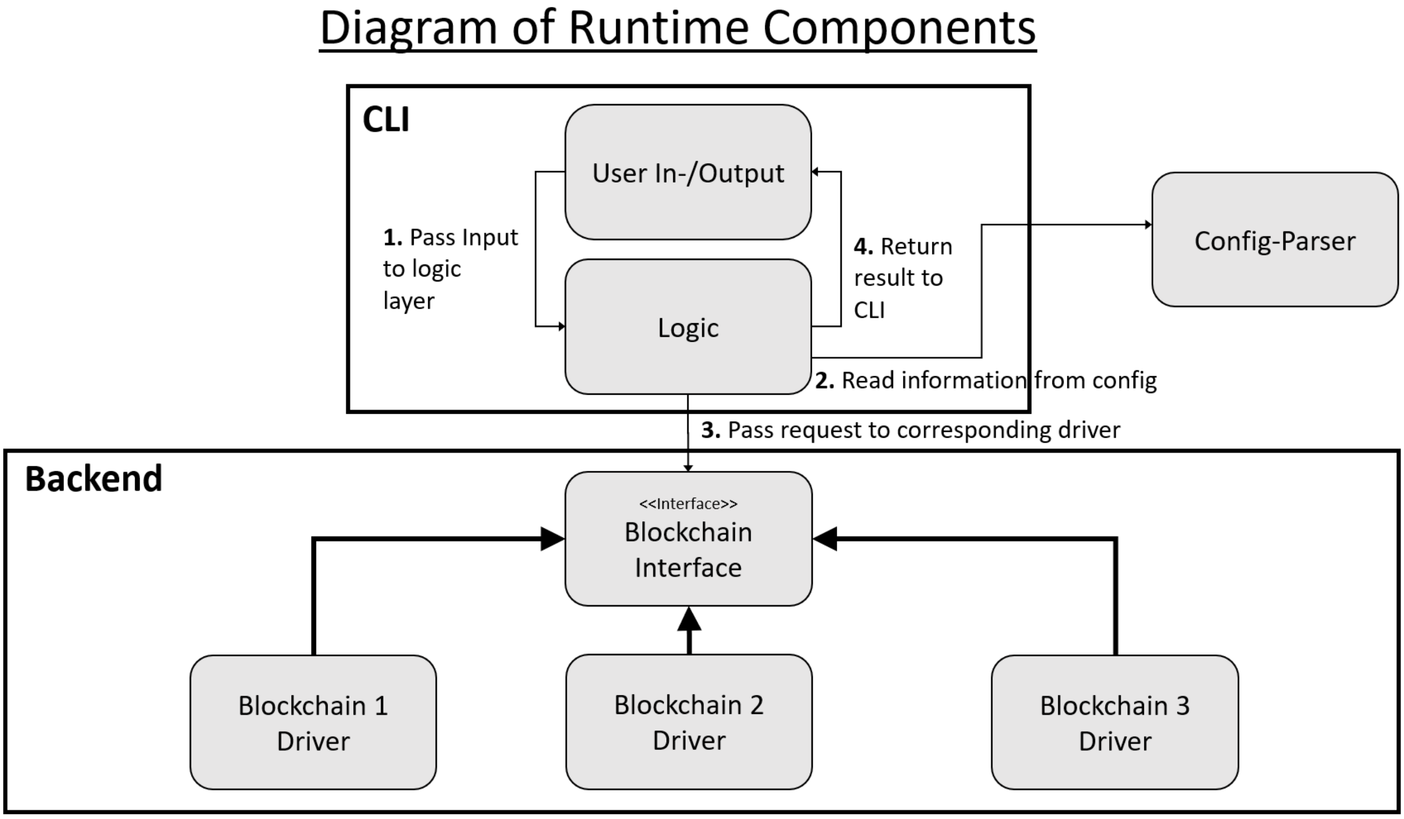
NFT Playbook
AMOS 2022 | Project 7

Software Architecture Document

Schwarzmann Sebastian
Schlinger Johanna
Al-Sheikh Tawfeek
Schilling Johannes
Dreesens Philipp
Rotsching Lukas
Wolfrum Lukas
Diktov Hristo
Kurz Noah

Table of contents

1. Overview diagram of runtime components



Runtime Component:

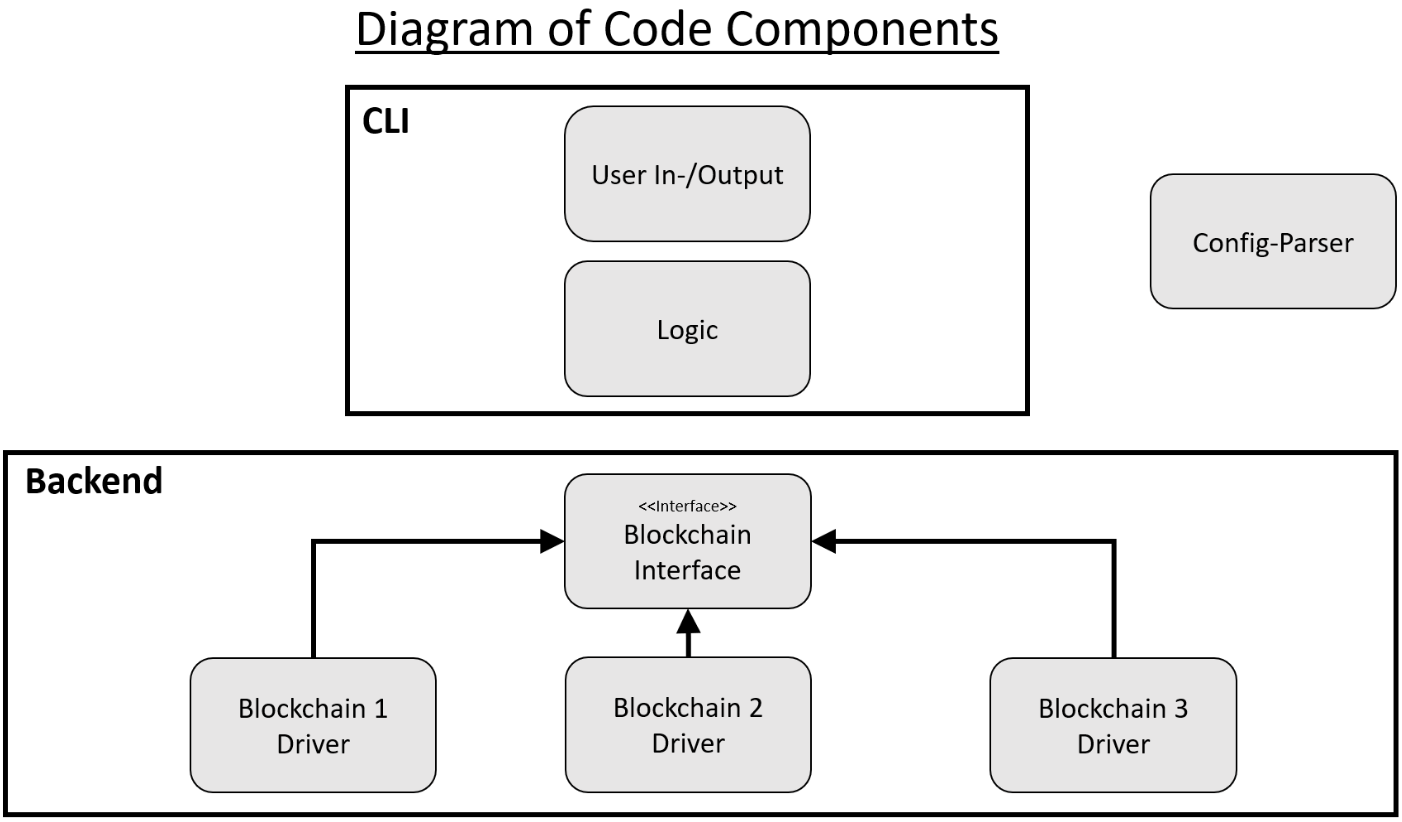
The first interaction is triggered by a user input via the command line. When the user executes a valid command, it is passed to the logic layer **(1).** In case the command is illegal or unsupported the CLI displays a help message.

The Logic-Layer then uses the necessary information from the Config-Parser **(2)** and forwards the request to one or more corresponding blockchain driver(s) **(3)**.

The Logic-Layer accesses the requested Blockchain(s) through the Blockchain Drivers, which implement the same Interface, so that they offer a consistent functionality. Those Drivers then interact with their specific blockchain so the request is permanently integrated in the chain.

After the driver(s) accomplished the work, the result is returned to the Logic-Layer, which itself forwards it to the CLI so the result can be shown to the user **(4)**.

2. Overview diagram of code components



Code Component:

The Code is separated into two aspects. The upper part consists of the CLI, which handles the interaction with the user and forwards the request to the backend. Furthermore, a Config-Parser is used to retrieve the persistent configurations from the config file.

The lower part is used to abstract from different Blockchains. The Backend provides one single interface, which can be used to access multiple Blockchains the same way. Furthermore this enables us to easily expand the project to new Blockchains, just by adding a new driver that is capable of handling that Blockchain. Considering the agile nature of this project, the so gained expandability might become very useful later on.

3. Summary of the underlying technology stack

The app is realized with by a native node application typed with type scripted. Within the node-project, we are using a mono-repo-setup build-system using NX, JEST as a test-framework and ESLINT as a linter. We also implemented a powerfull CI/CD-Pipeline using GH-Actions, which lints, tests and builds the whole project on every push. In the following listing you can find all dependencies from used packages with a short explanation:

- @nrwl/cli: "14.1.2", → taskexexutor for build system
- "@nrwl/eslint-plugin-nx": "14.1.2", → linter
- "@nrwl/jest": "14.1.2", → testing
- "@nrwl/js": "14.1.2", → JS-Adaption for mono-Repo
- "@nrwl/linter": "14.1.2", linter-Adaption for mono-Repo
- "@nrwl/node": "^14.1.2", node-Adaption for mono-Repo
- "@nrwl/workspace": "14.1.2", → mono-Repo
- "@types/jest": "27.4.1", → testing
- "@types/node": "16.11.7", → node-JS-framework
- "@typescript-eslint/eslint-plugin": "~5.18.0", → linter
- "@typescript-eslint/parser": "~5.18.0", → linter
- "eslint": "~8.12.0", → linter
- "eslint-config-prettier": "8.1.0", → linter
- "eslint-plugin-prettier": "^4.0.0", →linter
- "jest": "27.5.1", → testing
- "nx": "14.1.2", → build system / Mono-Repo
- "prettier": "^2.5.1", → Linting
- "ts-jest": "27.1.4", testing with typescript
- "ts-node": "9.1.1", node with typescript
- "typescript": "~4.6.2" typescript for java script