Fund 12_Automation Testing Framework

Created by Pham Thuy Quynh, last modified by Đỗ Anh Ba just a moment ago

Status	APPROVED
Author	@Pham Thuy Quynh
Date	18 Sep 2024
Issue	https://milestones.projectcatalyst.io/projects/1
Content	

Related services	Approvers
Architecture Reviewer	@Pham Thuy Quynh
	@Ứng Hồng Quân
	@Lê Thị Tuyết

A. Problem and Solution Proposal

1. Problem Statement

- Testing Cardano smart contracts is a complex task due to the interactions between various blockchain components and the need for accurate transaction management.
- Manual testing is time-consuming and prone to human errors, leading to a need for an automated solution.
- This solution must support generating test scripts, running tests, building transactions and reporting results efficiently.

2. Solution Proposal

To address this challenge, we propose an automation testing framework that integrates:

- **Lucid** for transaction building and blockchain interaction.
- **Blockforst Provider** for querying blockchain data.
- **Bun** for running test scripts with its lightweight and high-performance Typescript runtime.
- **Typescript** to write structured and maintainable runtime.

This framework will streamline the testing process by automating test script generation, running tests, and reporting results.

B. High-Level design of Sub-Modules

1. Test Script Generator Module

- **Input**: Excel template containing test cases.
- **Functionality**: Convert test cases from Excel into Typescript-based test scripts. This involves mapping test case details (inputs, expected outputs) into executable scripts.
- **Output**: Typescript files compatible with the Bun testing framework (Bun Test documentation).

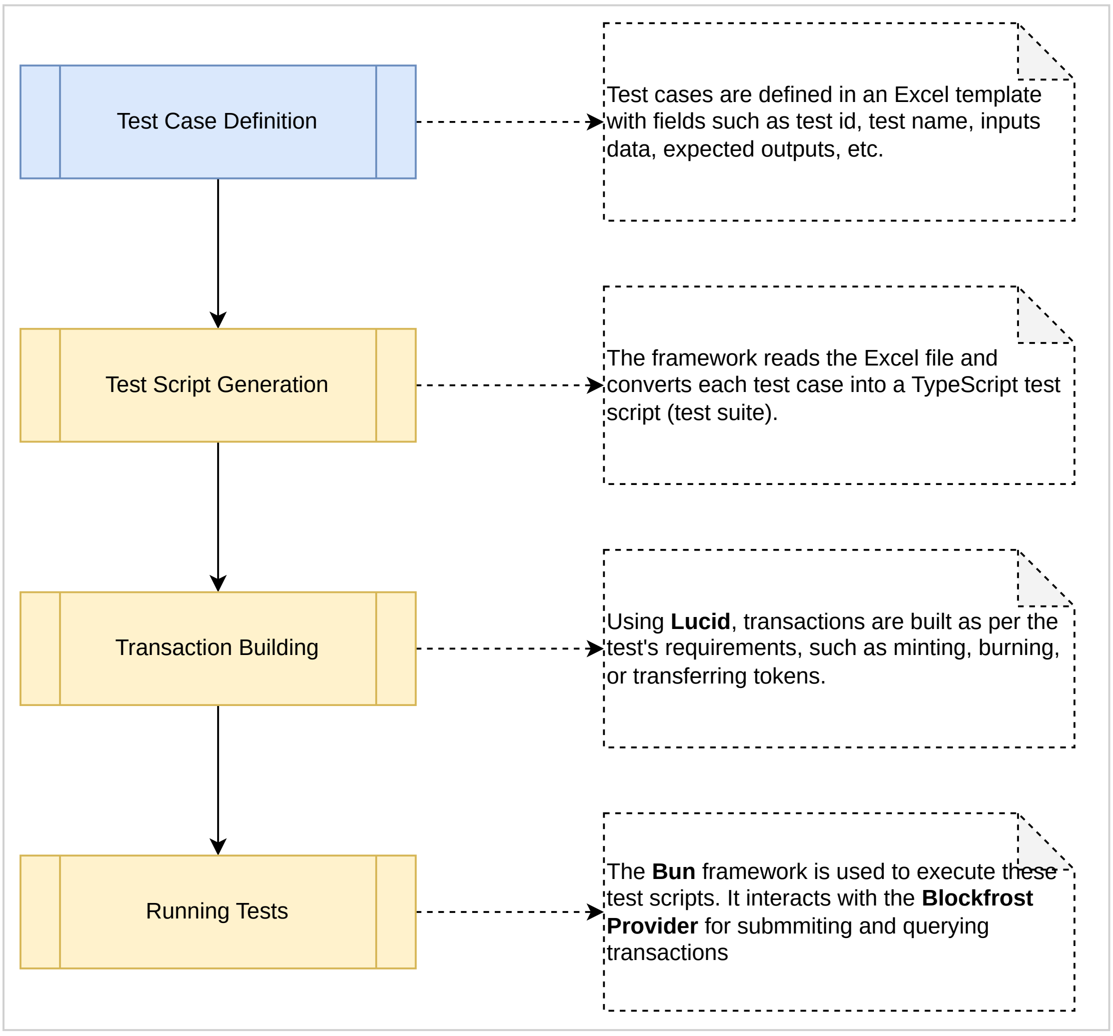
2. Transaction Builder Module

- **Input**: Test script data and Smart contract logic details.
- **Functionality**: Use Lucid to create and sign Cardano transactions for smart contract interaction, such as minting, burning, or transfering tokens.
- **Output**: Transactions ready for submission to the blockchain network.

3. Test Runner Module

- **Input**: Generated TypeScript test scripts.
- **Functionality**: Execute the test scripts using Bun's test library. This includes sending transactions to the blockchain and verifying results.
- **Output**: Test results with pass/fail status.

C. Smart Contract Development workflow with Automation testing


Test Case Definition
Test cases are defined in an Excel template with fields such as test id, test name, inputs data, expected outputs, etc.
Test Script Generation
The framework reads the Excel file and converts each test case into a TypeScript test script (test suite).
Transaction Building
Using Lucid, transactions are built as per the test's requirements, such as minting, burning, or transferring tokens.
Running Tests
The Bun framework is used to execute these test scripts. It interacts with the Blockfrost Provider for subbmiting and querying transactions

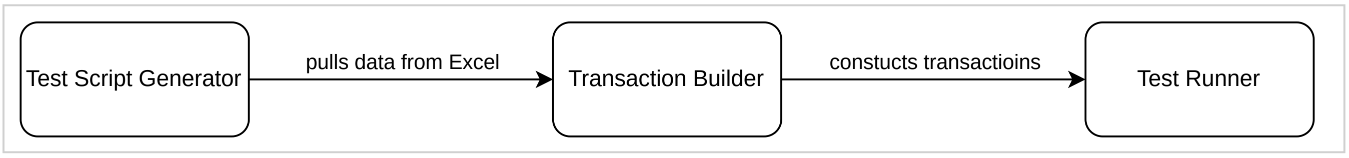
D. Technologies Involved

1. **Lucid** : For interacting with the Cardano blockchain to create and manage transactions.
2. **Blockforst Provider**: For querying blockchain data and monitoring transaction status.
3. **Bun**: A fast runtime for executing Typescript test scripts.
4. **Typescript**: For writing test scripts, framework logic.
5. **Excel Integration**: For reading test case templates and converting them into Typescript test scripts.

The Excel file will serve as the basis for creating Typescript test scripts. It should include the following columns:

- **Test ID:** A unique identifier for each test case.
- **Test Name**: A descriptive name for the test.
- **Steps**: A brief description of the test scenario.
- **Smart Contract**: The name of the smart contract is being tested.
- **Transaction**: The name of the transaction use the smart contract script.
- **Inputs**: Specific inputs or parameters for the transaction (e.g. asset names, token amount,...)
- **Expected Output**: The expected result, such as successful transaction completion or specific data returned from the contract.

E. Module Interactions



1. Test script Generator

- Once the Excel data is parsed, the next step is to generate TypeScript test scripts. This can be archieved by iterating over the test cases and dynamically creating TypeScript files that define the test logic.
- For each test case, the script would replace placeholders with the actual values from the Excel file. A file generation function can be implemented using TypeScript.
- Script Generation Flow:
  - **Read Excel:** Load the Excel file containing test cases.
  - **Parse Test Cases:** Convert each row into an object containing test details.
  - **Generate Scripts:** Use a template to generate TypeScript test files for each test case.
  - **Save files:** Store the generated Typescript files in a designated directory (e.g., /tests/).

2. Transaction Builder

Lucid is a library designed to help QC/developers interact with the Cardano blockchain, including building and signing transactions. In the testing framework, Lucid will be used to create and submit transactions that are part of the test cases.

2.1 Lucid integration in Test scripts

Each test case will interact with the Cardano blockchain through Lucid to:

- Build transactions: Lucid will be responsible for creating Cardano transactions(e.g., minting, burning, transferring tokens,....) as defined in each test case.
- Sign transactions: The framework will need to sign the transactions using the correct private keys, which can be handled through Lucid's signing capabilities.
- Submit transactions: Once the transactions are built and signed, Lucid will submit them to the blockchain, potentially using the Blockforst Provider to interact with Cardano nodes.

2.2 Lucid in Test Cases

Once the Build Transaction function is in place, the test scripts (generated from the Excel sheet) will call this function, providing the necessary inputs. Each test case can define specific parameters such as the recipient address, the amount of ADA of tokens to transfer and other transaction details.

2.3 Handling Smart Contracts in Lucid

For smart contract-based tests (e.g. minting or burining tokens using Plutus scripts), we can use Lucid to attach smart contract scripts to transactions.

3. Test Runner

Bun is a fast Javascript runtime, and its built-in testing library makes it a good choice for running Typescript test casses. In this section, we will discuss how the testing framework will execute test scripts using **Bun** and validate the results.

3.1 Executing Test scripts Using Bun

Once the test scripts are generated from the Excel template, we can use Bun to run them. Bun has a built-in test runner that automatically finds and executes test files that follow a particulr naming convention (e.g., files ending with .ts in ./tests/ folder).

3.2 Running Tests with Bun

Here is how to set up and run tests using Bun:

- install Bun: First, ensure that Bun is installed in the test environment. Install Bun by following the Official Bun Test documentation.
- Run the Tests: Once the test scripts are generated, run them by executing Bun's test command: **bun test ./tests --watch**

This command will automatically search for test files in the project, execute them, and display the results.

./tests : this is directory where all generated test scripts are saved.

--watch : this option ensures that Bun will re-run the tests whenever there is a change in the test files, which is helpful during development.

3.3 Validating Test results

After running the tests, the framework needs to validate whether each transaction was successfull or not. The validation will compare the expected output from the test cases (provided in the Excel template) with the actual results of test execution.

3.4 Handling transaction status checks

Using Lucid, the framework can fetch the transaction status to validate its outcome. This will allow us to confirm whether a transaction was successful, failed, or included specific outputs.

3.5 Error Handling and Failures

In case of errors (e.g., failed transactions, wrong outputs), the test framework will catch these issues and mark the test as **failed**. Bun's test framework will provide detailed error logs, including which test cases failedd and why.

3.6 Bun Test Output

Once the tests have run, Bun will output the results in a simple console format:

```
test-mint-token.ts:
✓ mint token successfully [0.82ms]

Summary:
  Total: 1 tests
  Passed: 1 tests
  Failed: 0 tests
```

No labels