Testing policy Document

TeamTuring:

Darius Scheepers, Kyle Pretorius, Richard Dixie, Sewis van Wyk,

Tristan Rothman & Ulrik de Muelenaere

ERP-Coin

1 Testing process

We have a policy that feature branches may only be merged into the main development branch if all tests pass, and similarly for merging the development branch into master. This is enforced by Travis CI, which automatically runs tests on each pull request.

Functional requirements tests

Our back end is based on the Node.js framework and for this reason will be tested using Mocha. Due to the nature of our system, all requests will run through the central server, which allows us to create a central point of testing. All use cases can be expressed in terms of requests that must be sent to the server, hence our tests will be request-based and can be divided into modules that are formed by the APIs provided by our server.

As stated above, any changes made to the server will be extensively tested before they are merged with the main branch. The smallest units of the server can be seen as features, hence we will perform unit tests on each feature to ensure that it works as intended before merging it into the main development branch. The features will be tested again as part of the larger system in the development branch as part of integration testing. Once these tests pass as well, then only may the development branch be merged into the main branch and testing may start again on new features.

Non-functional requirements test

- Security:
  These tests are focused on the requirement that no unauthorized user may have access to the system. The system achieves this by testing that a user is logged in with every request. This creates the following two possible outcomes:

  1. The user is logged in, and their request may be processed.
  2. The user is not logged in, and the request must be rejected.

- Validation:
  This testing focuses that our system will correctly accept requests that contain valid data and will reject requests that contain invalid data. This prevents our system from entering an invalid state or storing invalid information on the database.

- Correctness:
  These tests focus on ensuring that the server performs as expected when a request to is made by a user. This includes testing that data is actually inserted into the database when a request is made, such as adding a new user or logging points where users have been. Another correctness test performed is to ensure that data returned by the server to the ERP Coin mobile application is valid.

2 Testing tools

For this project the main testing tool used is Mocha, which is a JavaScript testing framework running on Node.js. This framework allows us to extensively test our back end, which is our system's main point of computation through which all use cases run. The server runs on Node.js, making Mocha ideal for testing its functionality. We use SuperTest in conjunction with Mocha, since most of our tests take the form of HTTP requests. SuperTest is a tool to simulate HTTP requests in unit tests.

Our Ethereum smart contracts are created using the Truffle framework. This framework also integrates with Mocha, which we use to test the methods provided by our smart contract. It is particularly important to test the smart contract, since it is stored immutably on the blockchain, so mistakes cannot be fixed once the contract is deployed.

To improve code quality, ESLint is used for ensuring that the code conforms to the coding conventions for the project. JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it. ESLint is configured by use of a `.eslintrc.json` file. The configuration file used for this project can be found here: `https://github.com/TeamTuringCOS301/back-end/blob/master/.eslintrc.json`

Since the user interfaces are written in TypeScript, we use TSLint instead of ESLint in these repositories.

Finally, we use Travis CI to automatically run these tests on all pull requests. This allows us to enforce the policy that branches may not be merged if the tests do not pass.

The tools we use can be found at the following locations:

- Mocha: `https://mochajs.org/`
- SuperTest: `https://github.com/visionmedia/supertest`
- Truffle: `https://truffleframework.com/`
- ESLint: `https://eslint.org/`
- TSLint: `https://palantir.github.io/tslint/`
- Travis CI: `https://travis-ci.org/`

3 Test cases

The tests for the HTTP API are in the following directory:
`https://github.com/TeamTuringCOS301/back-end/tree/master/test`

This directory contains the following files:

- `mock-config.js`, `mock-db.js`, `mock-coins.js`, `mock-email.js`: These files mock the interfaces provided by the configuration file, database, Ethereum blockchain and SMTP server, respectively. This allows us to write tests that do not depend on these external services.

The rest of the files each test the HTTP API for one of our subsystems:

- `user-api.js`: This tests the user API, which is responsible for user management. It provides routes for users to create accounts, log in and out, and update their account details. It also keeps track of the number of coins earned by each user.
- `superadmin-api.js`: This tests the API provided for the main admin users. These users can add new conservation areas and conservation area admins.
- `admin-api.js`: This tests the API provided for conservation area admins. These admins can view alerts posted in their assigned areas, and suggest rewards to be made available to users.
- `area-api.js`: This tests the conservation area API. It provides routes to add and remove conservation areas, update the information of conservation areas, and keep track of the conservation area border.
- `point-api.js`: This tests the point API. It allows users to record their position within the conservation area border, and aggregates the points so that users can see which areas have been visited most frequently.

To run the tests described above, execute the command `npm test`.

The unit tests for the Ethereum smart contract can be found in the follwoing file:
`https://github.com/TeamTuringCOS301/back-end/tree/master/token/test/erp-coin.js`
To run these tests, execute the command `npm run truffle test`.

4 History

The full test logs can be found on the Travis CI website:
`https://travis-ci.org/TeamTuringCOS301/back-end`
Below are some excerpts from the latest test log.

The first excerpt shows that ESLint was run successfully, without producing any warnings:

```
1518 $ npm run lint                                                                    2.68s
1519
1520 > erp-coin-backend@1.0.0 lint /home/travis/build/TeamTuringCOS301/back-end
1521 > eslint src
1522
1523
1524
1525 The command "npm run lint" exited with 0.
```

The next excerpt shows the tests for one of the HTTP APIs, in particular the conservation area admin API. This includes both functional and non-functional tests:

- Lines 1538–1540, 1548, 1556, 1562, 1565, 1569 and 1573 are functional tests.
- Lines 1535, 1542, 1547, 1551, 1554, 1559, 1561, 1568 and 1571 are security tests.
- Lines 1536, 1537, 1543, 1545, 1546, 1555, 1560 and 1572 are validation tests.
- Lines 1541, 1549, 1552, 1557, 1563, 1566 and 1574 are correctness tests.

```
1533  Admin API
1534    POST /admin/add
1535      ✓ fails without a login session (66ms)
1536      ✓ fails on missing data (92ms)
1537      ✓ fails with invalid area ID
1538      ✓ succeeds with valid data (82ms)
1539      ✓ sends an email
1540      ✓ sends the password via email
1541      ✓ sets the correct password (89ms)
1542      ✓ hashes the password
1543      ✓ fails with an existing username
1544    GET /admin/login
1545      ✓ fails on missing data
1546      ✓ fails for a nonexistent admin
1547      ✓ fails with an incorrect password (77ms)
1548      ✓ succeeds with correct credentials (77ms)
1549      ✓ sets the session cookie
1550    GET /admin/info
1551      ✓ fails without a login session
1552      ✓ returns the correct information
1553    POST /admin/update
1554      ✓ fails without a login session
1555      ✓ fails on missing data
1556      ✓ succeeds with valid data
1557      ✓ updates the admin information
1558    POST /admin/password
1559      ✓ fails without a login session
1560      ✓ fails on missing data
1561      ✓ fails with an incorrect password (76ms)
1562      ✓ succeeds with the correct password (149ms)
1563      ✓ updates the password (76ms)
1564    GET /admin/logout
1565      ✓ returns successfully
1566      ✓ clears the session cookie
1567    GET /admin/list
1568      ✓ fails without a login session
1569      ✓ returns a list of admins
1570    GET /admin/remove/:admin
1571      ✓ fails without a login session
1572      ✓ fails for an invalid admin ID
1573      ✓ succeeds for a valid admin ID
1574      ✓ removes the admin from the database
                                                                    Top ▲
```

The final excerpt contains the results of the smart contract tests. As above, this includes both functional and non-functional tests.

```
1706  Contract: ERPCoin
1707    constructor
1708      ✓ sets the owner address (56ms)
1709      ✓ sets initial supply to zero (42ms)
1710      ✓ sets initial balances to zero (211ms)
1711      ✓ sets initial allowances to zero (1399ms)
1712    rewardCoins
1713      ✓ only works for the owner (64ms)
1714      ✓ updates the total supply
1715      ✓ updates the current balance
1716    buyReward
1717      ✓ is limited by the current balance (69ms)
1718      ✓ updates the total supply
1719      ✓ updates the current balance
1720    transfer
1721      ✓ is limited by the current balance
1722      ✓ returns true (38ms)
1723      ✓ does not change the total supply
1724      ✓ updates the current balances
1725    approve
1726      ✓ returns true
1727      ✓ updates the allowance
1728      ✓ does not affect the reverse allowance
1729    transferFrom
1730      ✓ is limited by the allowance
1731      ✓ returns true (38ms)
1732      ✓ does not change the total supply
1733      ✓ updates the current balances
1734      ✓ decreases the allowance
1735      ✓ is limited by the current balance (39ms)
```