# 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.

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

- Validation:
  These 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 of 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 systems main point of computation through which all use cases run. The server runs on Node.js making Mocha ideal for testing its functionality.

For more information on the Mocha framework visit: `https://mochajs.org/`

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`

## 3 Test cases

## 4 History