Coding Standards Document

TeamTuring:

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

Tristan Rothman & Ulrik De Muelenaere

ERP-Coin

1 Detailed system design

To illustrate the overall System design, a complete Component Diagram (Figure 1), together with a Domain Model (Figure 2) is given for reference.

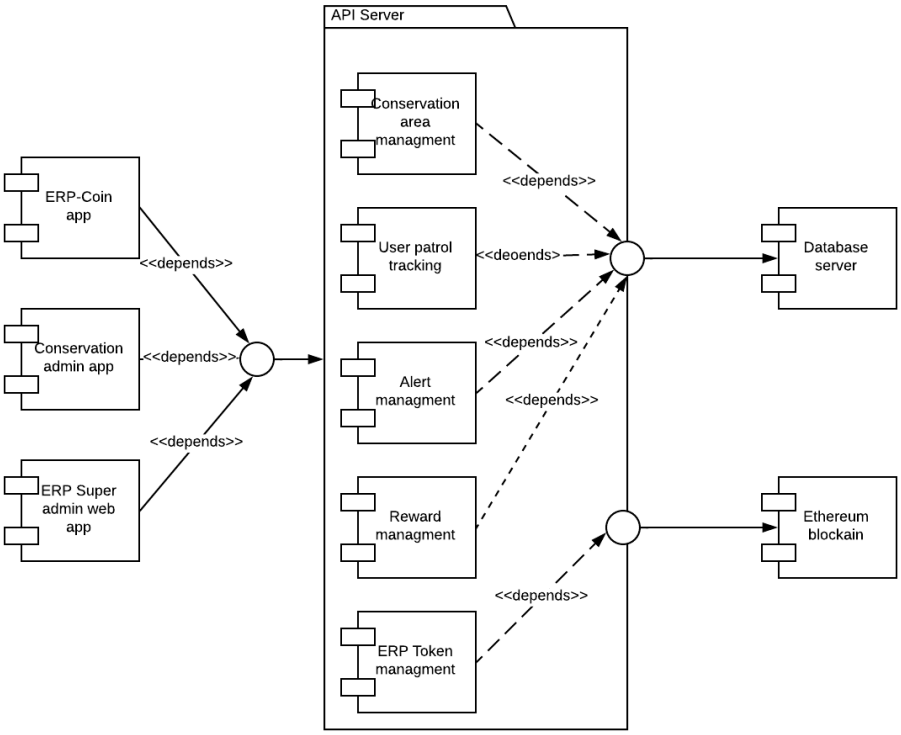

Figure 1: Component Diagram

2 Coding conventions

2.1 File Names

Ionic

- The name of a page should clearly state it's purpose.
- The name should be one or two words, where no caps are used, and a '-' is used to separate words.

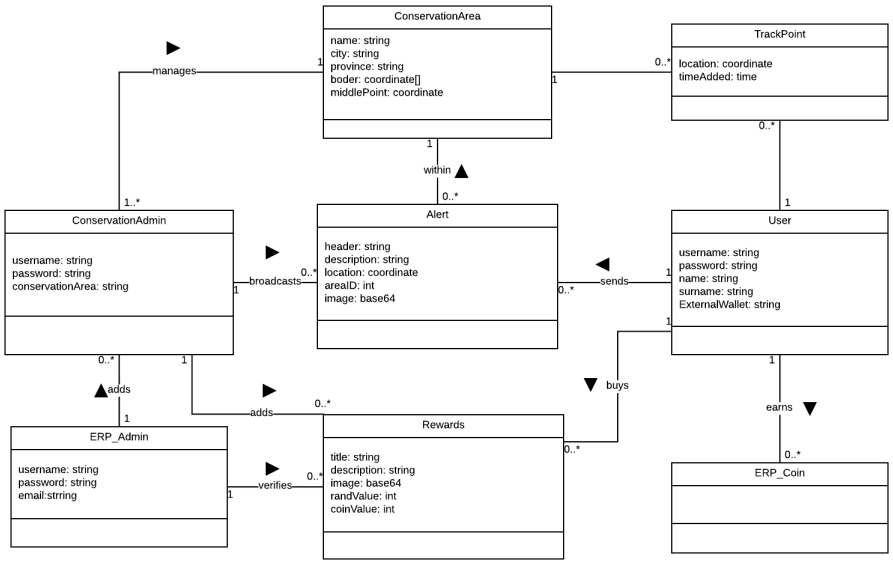

Figure 2: Domain Model

NodeJS

- Interface API's are separated into domain concepts, where each concept's accompanying API's are in separate files.
- The file names should be one or two words, where no caps are used, and a '-' is used to separate words.
- The name of the page should clearly state the area of the system it contains API's for.

2.2 File Headers

All files should be accompanied by a header on the top of the file, stating the author and describing all technologies used, functionality, as well as a basic description.

2.3 Naming Conventions

All Class, Method and Variable names must be self-explanatory and clearly state its purpose in the code, while adhering to the following guidelines:

- Class names should be nouns, and the first letter must be Capitalized, together with the first accompanying word, without using spaces.

```
1  ------------------------------------------------------------------------
2  Project:        ERP-Coin
3  Version Number: 1.0.0 / 2018
4
5  Company:        EPI-USE
6
7  Author/s:       Kyle Pretorius
8                  Ulrik de Muelenaere
9                  Tristan Rothman
10                 Sewis van Wyk
11                 Richard Dixie
12                 Darius Scheepers
13
14 @Nodejs Version 10.0.2 (or Ionic version, or SQL, etc.)
15
16 Purpose Statement:     (one or two line description)
17 ------------------------------------------------------------------------
```

Figure 3: Example File Header

- Method names should be verbs, and start with a lower case letter, only capitalizing the first letter of the following words, without using spaces.
- Variable names should use the same convention as Methods in terms of capitalization. Underscores may be used to substitute a space.
- Constant names and variables should be in all capital letters.

2.4 Indentation

A tab character should be used for indentation.

Indentation should not be used other than within the guidelines stipulated below, in order to avoid confusion.

- Indentation is used for any sub-code, for example all code within a class should be indented, all code in a loop should be indented and all code in a function should be indented.
- If a single instruction is wrapped to more than one, the accompanying lines should be indented.

```
this.http.post("/admin/login", jsonArr).subscribe
(
  (data) =>
  {
    var jsonResp = JSON.parse(data.text());
    if(jsonResp.success)
    {
      this.presentToast("Logged in!");
      this.navCtrl.push(TabsPage);
    }
    else
    {
      alert("Invalid Login. Try Again.");
    }
  },
```

Figure 4: Example Indentation

2.5 Blank Lines and Blank Spaces

Blank lines and blank spaces should be used extensively to increase readability. In addition to the areas to be mentioned, blank lines and blank spaces may be used anywhere the author feels readability can be improved without sacrificing functionality.

- Blank lines must be used between sections of a file, between classes and definitions, between methods and before a block or a single-line comment
- Blank spaces should be used before and after all operators, before and after parenthesis (except in a function call) and after commas.

2.6 Line length and Wrapping lines

Lines should not contain lines much relatively longer than the average of the rest of the code. If lines are too long it should be wrapped and indented to increase readability, preferably after a comma or before an operator.

Lines should contain one instruction only, new instructions should start on a new line.

2.7 Comments

All files should be accompanied by a header as stipulated in the section on File Headers. Comments giving information on an instruction should appear in the same line, but after the instruction. Comments giving information on a block of code should appear on its own line, before the block.

If code is pushed to Github that contains errors or is incomplete, the author should clearly leave a comment at the suspected block of code in order to carry the task over to the next sprint.

3 File structure

3.1 Github

Since the project is separated into multiple architectural pieces that are inherently independent on one-another, there is a separate repository on Github for each part of the project.

The repositories are as follows:

- User Front-End (Ionic and Cordova)
- Admin Front-End (Ionic and Cordova)
- Conservation Admin Front-End (Ionic and Cordova)
- Back-End (NodeJS and MySQL)

Each of these repositories have 2 brances, namely a "master" branch and a "dev" branch.

The "master" branch contains a working prototype of the system.

The "dev" branch is where new requirements are added, tested and refactored.

After a specified set of requirements are added to the "dev" branch as per the sprint meeting consensus, and said requirements are tested and refactored, then the new changes are pushed to the "master" branch to become a part of the working prototype.

3.2 Ionic Front-End Repositories

The file structure of Ionic projects are managed by the framework, and not modifiable by the development team. Compliance to the Ionic Framework file structure is essential for the system to compile and execute correctly.

Please refer to the following link for more information on the Ionic Framework file structure:

ionicframework.com/docs/intro/tutorial/project-structure/

3.3 Back-End Repository

On the NodeJS Back-End, there are 4 folders, of which each has the following name and function:

- "sql" - Containing the MySQL files for creating and maintaining the database.
- "src" - Containing the NodeJS server code, as well as all API's in a separate "apis" folder.
- "test" - Containing the source code for all the automated tests.
- "token" - Containing the code for using smart contracts on the blockchain.

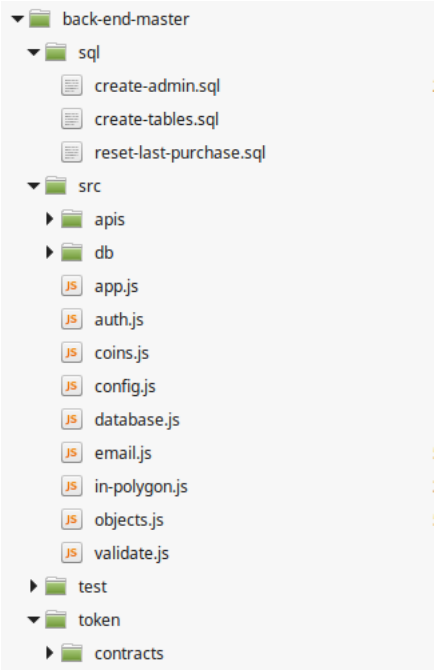

Figure 5: Example Back-End Repository file structure

4 Code review process

4.1 TSLint

TSLint is a static code analysis tool used in software development for checking Typescript code quality, if TypeScript source code complies with coding rules. TSLint checks your TypeScript code for readability, maintainability, and functionality errors.

Since an enormous part of the project development is in TypeScript, the use of TSLint is compulsary to all team members, as well as strict compliance thereof.

4.2 Refactoring

Referring to the section on Github (under File Structure), every team member must conduct extensive refactoring on all the code he/she has created before it is pushed to the master branch.

Refactoring encapsules the compliance to all guidelines as stipulated in this document in the previous sections on Coding Standards, and modifying code readability in line with these guidelines.

The idea is to prioritize working code in the development phase on the "dev" branch, and thereafter refactoring before adding new code to the working

prototype on the "master" branch.

4.3 Peer Reviews

At least once (but often more) before every deliverable to either client or department, peer reviews are conducted. This is usually done after the refactoring phase.

In a peer review, team members are randomly assigned another team member's source code of the deliverable they worked on since the last peer review.

During the review the source code is briefly assessed for common errors, inefficiencies and compliance with the coding standards as stipulated in this document.