Coding Standards V.2

COS 301 Capstone Project: Demo 4

Members:

Maria Petronella Laura-Lee Strydom *u04974359*

LightBot Adaptive Traffic Control System

Contents

1. Introduction

This document will discuss the different coding standards, tools and language packages implemented to create the LightBot system. The purpose of implementing these coding standards is to make sure that the language is written according to the specified guidelines, which in turn simplifies the understanding and implementation of the code.

2. Languages

2.1 LightBot Application & Interface

The system interface (front-end) is coded in React, which is a JavaScript library, combined with SCSS.

2.2 LightBot Server

The system server is written in NodeJS and express, as the main functionality of this component is to serve as a web server.

2.3 Traffic Simulation

The traffic simulator is coded in Java, combined with multiple imported libraries.

3. Coding Standards

The coding standards used to develop LightBot:

3.1 Naming Conventions

- All variables will have meaningful and conventional names which follow the Camel Casing standard. This is done by declaring names without spaces or punctuations, indicating separate words by starting the first word with a lowercase letter and the second word with an uppercase letter, for example, 'userName'.
- Names are also chosen to clearly explain the function of each.
- Variable names are initialized before use.
- Class names are kept short, and also follows the Camel Casing convention.

3.2 Indentations

- Proper indentation is used throughout the code to maximise readability and maintainability of the components.

- Indentation is used to accentuate the bodies of conditional statements, control statements, functions, etc.
- Multiple packages and libraries are used to ensure clean code and reduce code review time. They automatically indent bodies of code and ensure all code is up to formatting standards.

3.3 Inline Comments

Inline comments are used to explain the functioning of multiple parts in the code, as well as highlight key aspects of the code.

3.4 Classes, Subroutines, Functions and Methods

- Classes, subroutines, functions and methods are kept reasonably sized.
- Each method is expected to contain only one function or action. This is to keep the module from growing too large in size and confusing the programmer and readers.
- The names for the classes, subroutines, functions and methods will have verbs or nouns in them to specify actions and meanings, for example, "getCode".

3.5 Source Files

- The names given to the source files represent its function in the relevant component.
- Each source file will only contain one class definition.

3.6 Use of Braces

- Braces are used to delimit the bodies of conditional statements, control statements, functions, etc.
- The use of braces simplifies the readability of the code.
- Braces are automatically implemented through the use of code formatting libraries.

4. Tools

4.1 LightBot Application & Interface

- The react-json-pretty npm library is used to format and "prettify" the JSON data.
- The eslint-config-prettier-react npm library is used to format the javascript and scss code.
- Redux is used as a predictable state container for the web application.
- An additional 20 helper tools have been installed to simplify and format the code.
- The IDE used to develop the front end is VSC - Visual Studio Code.

4.2 LightBot Server

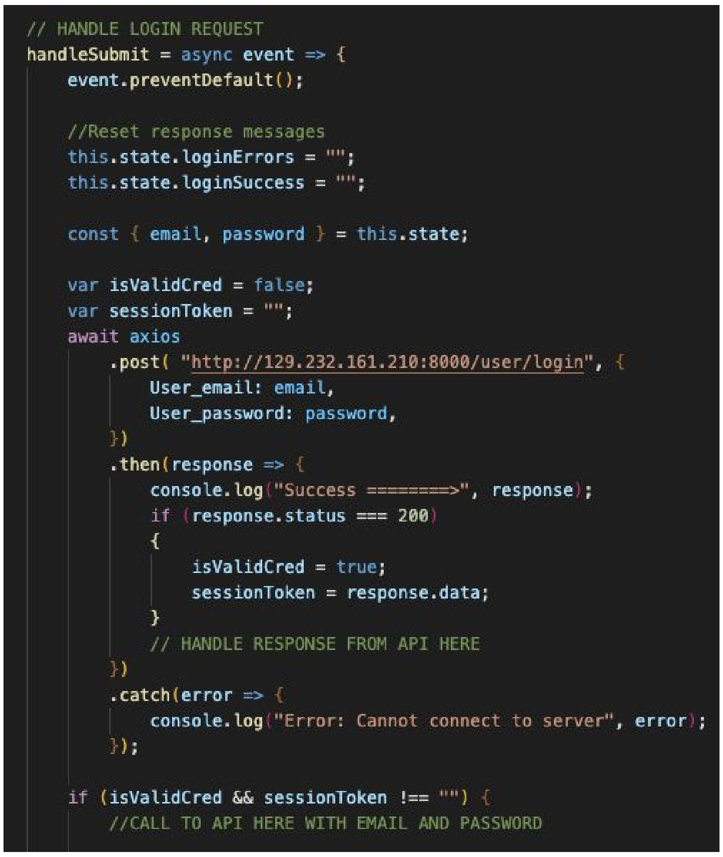
- NodeJS gives me the ability to use node modules which provides a vast array of utility for developing and testing scalable and secure web server applications.

4.3 Traffic Simulation

- JDK is used as a basic toolkit to develop Java applications, which includes a Java compiler, the Java Runtime Environment and the Java API's.
- JRAt is used to measure the applications' performance. Any type of problems can be identified before it can cause an impact on the overall performance of the system.

5. Code Snippets

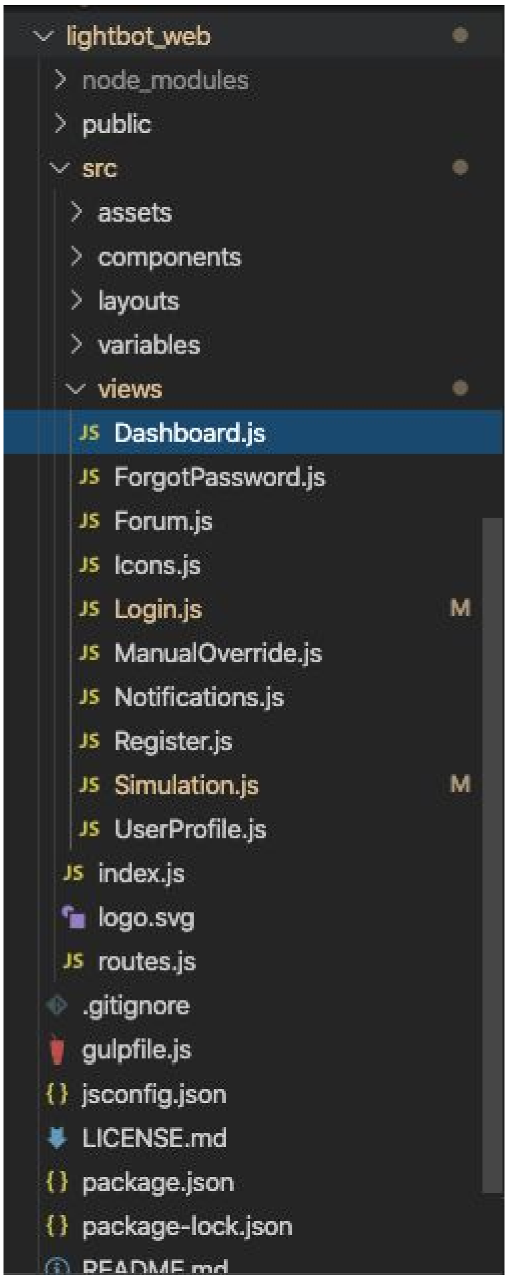
The implementation of indentations, inline comments and the use of brackets can be seen in Figure 1.



```
// HANDLE LOGIN REQUEST
handleSubmit = async event => {
    event.preventDefault();

    //Reset response messages
    this.state.loginErrors = "";
    this.state.loginSuccess = "";

    const { email, password } = this.state;

    var isValidCred = false;
    var sessionToken = "";
    await axios
        .post( "http://129.232.161.210:8000/user/login", {
            User_email: email,
            User_password: password,
        })
        .then(response => {
            console.log("Success ========>", response);
            if (response.status === 200)
            {
                isValidCred = true;
                sessionToken = response.data;
            }
            // HANDLE RESPONSE FROM API HERE
        })
        .catch(error => {
            console.log("Error: Cannot connect to server", error);
        });

    if (isValidCred && sessionToken !== "") {
        //CALL TO API HERE WITH EMAIL AND PASSWORD
```

Figure 1

The source files for each component are located in the relevant src folders, and all files have been given a descriptive and relevant name, see Figure 2.



```
lightbot_web
  node_modules
  public
  src
    assets
    components
    layouts
    variables
    views
      Dashboard.js
      ForgotPassword.js
      Forum.js
      Icons.js
      Login.js                M
      ManualOverride.js
      Notifications.js
      Register.js
      Simulation.js           M
      UserProfile.js
    index.js
    logo.svg
    routes.js
  .gitignore
  gulpfile.js
  jsconfig.json
  LICENSE.md
  package.json
  package-lock.json
  README.md
```

Figure 2