COS 301

GENERAL SOFTWARE DEVELOPMENT STANDARDS AND GUIDELINES

May 31, 2018

Brogrammers

Imagine Interactive Systems Carpool App

Revision History

Date	Version	Description
09/04/2018	1.0	Initial Version
31/05/2018	1.1	Updated standards to be more in line with our current framework

Contents

Introduction

Well written software offers many advantages. It will contain fewer bugs and will run more efficiently than poorly written programs. Since software has a life cycle and much of which revolves around maintenance, it will be easier for the original developers and future keepers of the code to maintain and modify the software as needed. This will lead to increased productivity of the developers. The overall cost of the software is greatly reduced when the code is developed and maintained according to software standards.

Following a consistent coding standard helps improve the quality of the overall software system. The key to a good coding standard is consistency. This consistency needs to be found within the standard itself (in other words, you need to make sure that guidelines don't contradict one another) but also within the source code that uses the standard. Completed source code should reflect a harmonized style, as if a single developer wrote the code in one session.

Coding Standards

General Standards

File Header

A header to be included in each file that specifies useful information regarding that file, this information consists of, but is not limited to:

- **File Type** - A comment to indicate what the file is (*Component, Store, Stylesheet, Test Script, API endpoint, etc...*) and should be written in the format "*// File Type: ?*" (where the '?' represents the corresponding file type) followed by a blank line.
- **Imports** - All file imports should be listed at the head of a file with the NPM imports appearing first, followed by a blank line and all relative imports, followed by another blank line and all 'No-Object' imports. These imports should be listed in ascending alphabetical order. Example:

  import React from 'react';

  import LoginPage from "./components/landing/LoginPage";
  import SettingsPage from "./components/Settings";

  import './../css/style.css';

  If more than one object is imported from a single source, these should be listed in ascending alphabetical order on that line. Example:

  import { BrowserRouter, Route, Switch } from 'react-router-dom';

Descriptions

Purpose

A statement of the purpose of the Class/React Component/API Endpoint/etc... that should appear just above it's declaration and should be written in a multi-line comment.

Description of Methods

A brief description of the purpose of each method, the parameters, return type, and other input and output such as files, database tables and text fields accessed and updated by the method that should appear just above it's declaration and should be written in a multi-line comment.

Description of Fields

A brief description of each field that indicates it's purpose and data-type that should appear just above it's declaration and should be written in a single-line comment.

In-code comments

Comments should be inserted in places where the purpose of the code cannot be easily discerned, as well as to facilitate understanding to complex areas of code.

Naming Conventions

- Variable names should be descriptive, make use of multi-word identifiers if necessary
- Spaces in variable names should be indicated with underscores
- Avoid variable name collision
- Variable naming should make use of camel casing
- File names should be descriptive
- Component names should start with capital letters
- Class names should start with capital letters

Formatting Conventions

- There should be a blank line after the file header
- Tab spacing should be size 4
- Child Elements should be indented once more than their parent elements. Example:
  * *
  <h1> Hello World </h1>
  * *
- Child elements should have no blank line between themselves and their parents.
- Conventional operators surrounded by a white space *e.g. a = (b + c) * d;*
- Commas followed by a white space *e.g. someFunction(a, b, c);*
- Semicolons followed by a space character, if there is more on a line *e.g. for (int a = 0; b < c; d++)*
- Reserved words separated from opening parentheses by a white space *e.g. while (true)*
- Have a space before a leading brace *e.g. function test() {}*
- Braces and parentheses should always use the following format:
  function() {

  }

  if() {

  }else{

  }
- Variable declarations should be grouped together and followed by a blank line.
- Conditional statements, Loops and Function/Class calls/declarations should always be preceded and followed by a blank line. Example:
  var x = "";

  if(x) {
  x = "Hello World";
  }

  return x;

In-code comment Conventions

- Start all comments with a space to make it easier to read
- All comments should be complete sentences
- The first word should be capitalized, unless it is an identifier that begins with a lower case letter
- Large comments, should be commented out using block format (/* == */)
- Comments should be applied when the purpose of functions/methods cannot be immediately discerned
- If the comment is to explain a function/method, it should be placed just above the line of the function declaration
- Multi-line comments must be vertically aligned

JavaScript(ES6) Standards

References

- Use 'const' for all your references and avoid using var.
- If you need to reassign values, make use of let instead of var.

Objects

- Use the literal syntax for object creation:
  const item = {};
- Use property shorthand when listing object properties:

```
const item = "Hello";

const object = {
    item,       //Not item: item,
};
```

- Group shorthand properties at the beginning of the object declaration.

Arrays

- Use "Array.push()" instead of direct assignment to add items to the array.
- Use array spreads "..." to copy arrays:
  const itemsCopy = [...items];
- Use line breaks after open and before close array brackets if an array has multiple lines:

```
const objectInArray = [
  {
    id: 1,
  },
  {
    id: 2,
  },
];
```

Destructuring

- Use object destructuring when accessing and using multiple properties of an object. Use a white space after the first brace and before the last brace to better indicate destructuring:

  const { firstName, lastName } = user;
- Use array destructuring:

  const arr = [1, 2, 3, 4];

  const [first, second] = arr;

Functions

- Use arrow functions wherever possible so that the function is correctly bound within it's context:

  const someFunction = (parameter) => {

  //implementation

  }
- If the function body consists of only a single return statement, omit the braces and use the implicit return:

  [1, 2, 3].map(number => 'The number is: ' + number);

React Standards

Naming

- Use PascalCase for filenames *e.g. LoginPage.js*
- Use camelCase for instances of React components *e.g. const loginPage = <LoginPage />*
- Always include a single space in your self-closing tag
- Methods that return components are prefixed with 'render' *e.g. renderProfileHeader()*

Props

- Always use camelCase for prop names
- Avoid using DOM elements like class/style as prop names when their function is for a purpose not related to that DOM element
- Do not add a space before or after the '=' when declaring a prop

Tags

- Always self close tags that have no children
- If your component has multi-line properties, close it's tag on new line. Example:

```
<LoginPage
    bar="bar"
    foo="foo"
/>
```

Ordering

1. Constructor
2. componentWillMount
3. componentDidMount
4. componentWillUnmount
5. Optional methods
6. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
7. Optional render methods like renderProfileHeader()
8. render

CSS Standards

Naming

- Always use hyphens in class names. Do not use underscores or camelCase.

Selectors

- Selectors should be on a single line, with a space after the selector, followed by an opening brace. A selector should end with a closing brace on the next line. Next selector related the the previous one should be on the next line with one additional line space between them. Example:
  .nav li {
  }

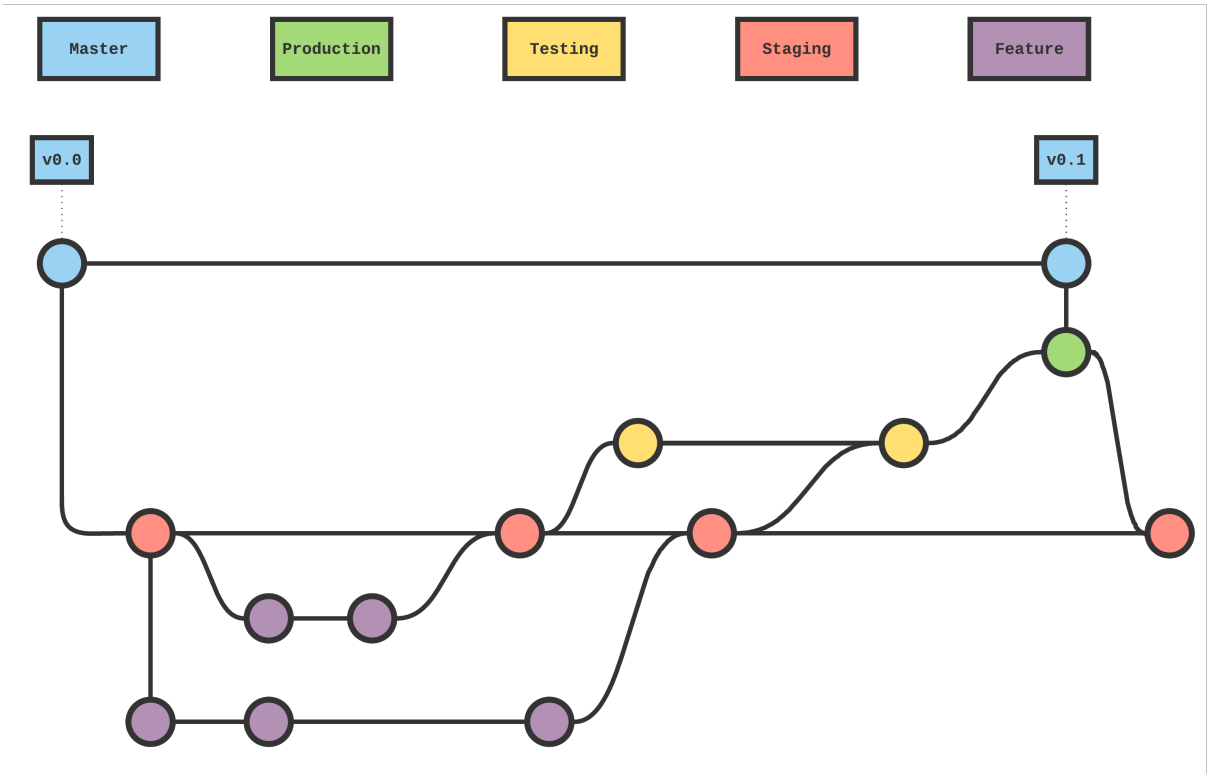
  .nav a {
  }

- Multiple selectors should each be on a single line, with no space after each comma.Example:
  .nav a.open,
  .nav a.close {
  }

Properties

- Each property should appear on it's own line prefixed by one soft tab and a single space before the property value. Example:
  .nav {
  position: absolute;
  top: 0;
  }
- Properties should be ordered in ascending alphabetical order
- Properties with multiple values should have those values separated by a space

Git Repository Structure



The git structure is made up of 5 levels of branching, the first level being all new features that are being added to the system. A new feature should be branched off of the staging branch so that the developer of the new feature is using the latest iteration of all components in the system, as the staging branch is where all completed features get merged to once they are completed.

The next level of branching is the testing branch, this is where a stable version of staging that is going to go into a release is pushed to. This is to allow development of new features to still continue while not interfering with a release. This is also the level where extensive testing for bugs occurs.

The production branch is where stable code from the testing branch is put into a production state, once this is done and the system is running it can be put into a full release and pushed to the master branch.