CoSpace
Coding Standards

Table of Contents

Terms & Definitions

Term	Definition
PascalCase	PascalCase is a naming convention in which the first letter of each word in a compound word is capitalized. Note that abbreviations should also be lowercase with their first letter being uppercase.
camelCase	camelCase is a naming convention in which except the the first letter of the compound word, each word is capitalized. Note that abbreviations should also be lowercase with their first letter being uppercase.
CONSTANT_CASE	CONSTANT_CASE is a naming convention in which every letter of the each word is a capital letter compounded with underscores.

1 Introduction

This document describes the standards, rules and conventions each member of the software development team will follow during the software construction phase of the project CoSpace. The layout and the structure of the document is explained in the following subsection.

Document Overview

The rationale and the motivation behind this coding standard and its descriptions are provided in the second section of the document. The detailed rules and conventions to follow are described in the third section of the document.

2 Description

This document manifests the software development team's shared understanding of the rules and conventions they will follow when writing software. There rules and conventions govern how each team member will lay out their code as well as their comments. Developers shall adhere to these rules to share a common understanding of the style & structure of the code and help minimize the time required to understand and change the code, as well as to minimize defects of the developed software.

3 Coding Standards Specifications

Following subsections describe the specifications to be followed during the construction of the software. For cases that left unmentioned in this document, refer to *Java Style Guide*[1] for Java and *JavaScript Style Guide*[2] for JavaScript.

3.1 Naming Standards

Following lists are the rules to be used when naming variables in the source code, organized by programming languages used.

3.1.1 Java

3.1.1.1 Package names

Each package in the folder structure should be named in ***lowercase*** letters. Consecutive words should be concatenated together without any uppercase letters or underscores.

Example: `com.overengineers.cospace.controller`

3.1.1.2 Class names

Each class name should be in ***PascalCase***. If the class is a part of test code, it should end with the noun *"Test"*, for example *"*`class DeletePostTest`*"*.

3.1.1.3 Method names

Each method name should be in ***camelCase***.

3.1.1.4 Variable names

Each local or class variable name should be in ***camelCase***. Parameter names should also be in ***camelCase***.

3.1.1.5 Constant names

Each constant field should be named in ***CONSTANT_CASE***.

3.1.2 JavaScript

3.1.2.1 Class names

Each class name should be in ***PascalCase***. If the class is a part of test code, it should end with the noun *"Test"*, for example *"*`class DeletePostTest`*"*.

[1] https://google.github.io/styleguide/javaguide.html
[2] https://google.github.io/styleguide/jsguide.html

3.1.2.2 Class method names

Each method name should be in ***camelCase***.

3.1.2.3 Function names

Each each name should be in ***camelCase***. If the function returns a graphical user interface element or a *"component"* that describes a UI element using JSX[3], it should be named in ***PascalCase***.

3.1.2.4 Variable names

Each local or class variable name should be in ***camelCase***. Parameter names should also be in ***camelCase***.

3.1.2.5 Constant names

Each constant field should be named in ***CONSTANT_CASE***.

3.2 Organization of Source Files & File Content

Following lists are the rules to be used when organizing the file structure of the source code repository and the content of the source files, organized by programming languages used.

3.2.1 Java

3.2.1.1 File names

Each source file in the file system should contain only one class and should be named that class in ***PascalCase***.

3.2.1.2 Folder structure

Folders in the source file tree should conform to package names as per Java language specification. Organization of the packages should be *layer by layer*, meaning that, conceptually, source code that in the same layer should be put inside the same package.

3.2.2 JavaScript

3.2.2.1 File names

If the source file contains functions that are not related to *JSX components* or if they don't return GUI elements, the source file should be named in ***lowercase*** with underscores between words. If the source file contains components and/or classes, it should be named in ***PascalCase***.

3.2.2.2 Folder structure

All folders should be in ***lowercase***. Consecutive words should be concatenated together without any uppercase letters or underscores. Organization of folder structure should be *layer by layer*, meaning that, conceptually, source code that in the same layer should be put inside the same folder.

3.2.2.3 File content

*Import*s should be at start of the file, followed by the functions and/or classes. *Export*s should reside at the end of the line instead of inline representation.

3.3 Comment Standards

Following lists are the rules to be used when documenting the source code, organized by programming languages used.

3.3.1 Java

3.3.1.1 Inline comment

Inline comments in the source code should start with lowercase letter and there should be **1 whitespace** before and after of "**`/*`**" and "**`*/`**".

[3] JavaScript XML

3.3.1.2 Line comment

Line comments in the source code should start with a uppercase letter and there should be **1 whitespace** after "`//`". If the line comment is after a source code line, there should be **2 whitespaces** before "`//`".

3.3.1.3 Javadoc

Comments documenting an interface of a method should include *Javadoc*s. These multiline detailed documentation comments should be started with "`/**`" and ended with "`*/`", with "`*`" at the beginning of each line. The styling of these comments should adhere to *Oracle Style Guide for Documentation Comments*[4].

3.3.2 JavaScript

3.3.2.1 Inline comment

Inline comments in the source code should start with lowercase letter and there should be **1 whitespace** before and after of "`/*`" and "`*/`".

3.3.2.2 Line comments

Line comments in the source code should start with a uppercase letter and there should be **1 whitespace** after "`//`". If the line comment is added after a source code line, there should be **2 whitespaces** before "`//`".

3.3.2.3 JSDoc

Comments documenting an interface of a method should include *JSDoc*s. These multiline detailed documentation comments should be started with "`/**`" and ended with "`*/`", with "`*`" at the beginning of each line. JSDoc also allows the use of *Markdown*, use whenever appropriate. The styling of these comments should adhere to *Google Style Guide for JSDoc*[5].

3.4 Programming Practice Conventions

Following lists are the rules to be used when organizing the file structure of the source code repository. These conventions are mostly programming language independent. Programming languages are mentioned when needed.

3.4.1 Nestedness

3.4.1.1 If statements

Design should be questioned if there are more than 2 nested if clauses. Nested if clauses dramatically increase the cognitive complexity of the source code and the load on the reader.

3.4.1.2 Loops

Design should be questioned if there are more than 3 nested loops.

3.4.2 Notation

3.4.2.1 Overriden methods

In Java source code, **`@Override`** annotation should always be added if the method is overridden.

3.4.2.2 Nullable fields

In Java source code, **`@Nullable`** annotation should be used if the field can be null.

3.5 Formatting

Following lists are the rules to be used when organizing the file structure of the source code repository, organized by programming languages used.

[4] https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
[5] https://google.github.io/styleguide/jsguide.html#jsdoc

3.5.1 Java

3.5.1.1 Indentation

Indentation should be made using **spaces** and should be **4 whitespaces** long.

3.5.1.2 If clauses

There should be no line breaks before left bracket.
Example:

```
if (check) {
    ...
}
```

3.5.2 JavaScript

3.5.2.1 Indentation

Indentation should be made using **spaces** and should be **4 whitespaces** long.

3.5.2.2 Use of semicolon

Although JavaScript language specification allows statements without the use of semicolons, every statement should be ended with a semicolon.

3.5.2.3 If clauses

There should be no line breaks before left bracket.
Example:

```
if (check) {
    ...
}
```