LinkedHU_CENG
Coding Standards

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. 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.

There are following three terminologies are used to declare C# and .NET naming standards:

- **Camel Case (camelCase):** In this standard, the first letter of the word is always in small letters and after that each word starts with a capital letter.
- **Pascal Case (PascalCase):** In this the first letter of every word is in capital letter.
- **Underscore Prefix (_underScore):** For underscore (__), the word after _ use camelCase terminology.s

2. Description

This document expresses the software development team's agreed understanding of the rules and practices that they will adhere to when building software. There are standards and norms that dictate how each team member will put out their code and comments. Developers must follow these guidelines in order to have a highly collaborative environment and shared understanding of the code's style and structure, as well as reducing the number of flaws in the developed program. The coding standard is a result of team consensus.

3. Coding Standards Specifications

The subsections that follow define the specifications that must be followed during the software's development. For cases that left unmentioned in this document, refer to C# Coding Guide for C# and JavaScript Style Guide for JavaScript.

3.1. Naming Standards

3.1.1. C#

- Use PascalCasing for **class names and method names.** For example: public class ClientActivity or public void ClearStatistics()

- Use camelCasing for **method arguments and local variables.** For example: public void Add(LogEvent logEvent) or int itemCount

- When naming an **interface**, use pascal casing in addition to prefixing the name with an I. This clearly indicates to consumers that it's an interface.

- Do not use Screaming Caps for **constants** or **readonly variables**.

- Use meaningful names for **variable**s. For example: seattleCustomers for customers who are located in Seattle

- Avoid using abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.

- Do use implicit type *var* for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names. Because it removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.

  These constraints are taken into account mostly because they are consistent with Microsoft's .NET Framework and easy to read.

3.1.2. JavaScript

- 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". Use descriptive names that convey the capabilities of the class. Components, which are used in frontend frameworks follow the same rules.

- Start **function** names with a letter, use camelCase for names. Use descriptive names, usually verbs in the imperative form. Use common prefixes like get, make, apply etc. Class methods follow the same rules.

- Start **variable** names with a letter, use camelCase for names. Variable names should be self-descriptive, describing the stored value. Boolean variables are usually prefixed with is or has.

- Define **constants** at the top of your file, function or class. Sometimes UPPER_SNAKE_CASE is used, while other times plain camelCase.

3.2. File Organization

The rules for structuring the source code repository's file structure and source file content are listed below.

3.2.1. C#

- The structure and organization of a **C# program** is radically different from the structure and organization of both C programs and Java programs.

- The **Main** method is the entry point of a C# application. When the application is started, the Main method is the first method that is invoked. There can only be one entry point in a C# program.

- C# programs are organized in **namespaces**. Namespace can contain types and - recursively - other namespaces.

- C# programs consist of one or more files. Each file contains zero or more namespaces. A namespace contains types such as classes, structs, interfaces, enumerations, and delegates, or other namespaces.

- The **using** directives import the types of a namespace. Imports should be at the start of the file. Importing a namespace N implies that types T in N can be used without qualification.

3.2.2. JavaScript

- 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.

- 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.

- Imports should be at the start of the file, followed by the functions and/or classes. Exports should reside at the end of the line instead of inline representation.

3.3. Comment Standards

The following are the rules to follow while documenting source code. A consistent approach to comments increases readability and can help with the ability to generate documentation from code.

3.3.1. C#

- Place the comment on a separate line, not at the end of a line of code.

- Begin comment text with an uppercase letter.

- End comment text with a period.

- Insert one space between the comment delimiter (//) and the comment text. For example:

```
// The following declaration creates a query. It does not run
// the query.
```

- Don't create formatted blocks of asterisks around comments.

- Ensure all public members have the necessary comments providing appropriate descriptions about their behavior.

3.3.2. JavaScript

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

- Line comments in the source code should start with an 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 "//".

- Comments documenting an interface of a method should include JSDocs. 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.

3.4. Coding Conventions

Coding conventions are style guidelines for programming. Coding conventions secure quality.

3.4.1. C#

- Use implicit typing for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.
- Don't use *var* when the type is not apparent from the right side of the assignment. Don't assume the type is clear from a method name.
- Don't use implicit typing to determine the type of the loop variable in *foreach* loops.
- In general, use *int* rather than unsigned types. The use of int is common throughout C#, and it is easier to interact with other libraries when you use *int*.
- Use the concise syntax when you initialize arrays on the declaration line. In the following example, note that you can't use *var* instead of *string[].*

```
string[] vowels1 = { "a", "e", "i", "o", "u" };
```

- If you specify an array size, you have to initialize the elements one at a time.
- Use a *try-catch* statement for most exception handling.
- In C# source code, [datatype]? annotation should be used if the field can be null.
- Design should be questioned if there are more than 3 nested loops in a code.

3.4.2. JavaScript

- For readability, avoid lines longer than 80 characters**.**
- If a JavaScript statement does not fit on one line, the best place to break it is after an operator or a comma.
- When declaring variables and constants, use the *let* and *const* keywords, not *var*.
- Ternary operators should be put on a single line.

3.5. Formatting

3.5.1. C#

Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Microsoft examples and samples conform to the following conventions:

- Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces).
- Write only one statement per line.
- Write only one declaration per line.
- If continuation lines are not indented automatically, indent them one tab stop (four spaces).

- Add at least one blank line between method definitions and property definitions.
- Use parentheses to make clauses in an expression apparent.
- Every statement should be ended with a semicolon.

3.5.2. JavaScript

- Indentation should be made using white spaces and should be 4 whitespaces long.

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

General rules for complex (compound) statements:

- Put the opening bracket at the end of the first line.
- Use one space before the opening bracket.
- Put the closing bracket on a new line, without leading spaces.
- Do not end a complex statement with a semicolon.

General rules for object definitions:

- Place the opening bracket on the same line as the object name.
- Use colon plus one space between each property and its value.
- Use quotes around string values, not around numeric values.
- Do not add a comma after the last property-value pair.
- Place the closing bracket on a new line, without leading spaces.
- Always end an object definition with a semicolon.