ICES4HU
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 <name>. The arrangement and framework of the document are elucidated in the subsequent subsection. The justification and inspiration behind this coding guideline and its explanations are presented in the document's second section. The specific regulations and practices to adhere to are outlined in the document's third section.

These are used to declare Java and React 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. (This is compulsory for react functions.)

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 represents the collective agreement of the software development team regarding the rules and practices they will adhere to during software development. It establishes standards and conventions that govern how each team member should write their code and comments. By following these guidelines, developers foster a highly collaborative environment and ensure a shared understanding of the code's style and structure, while also minimizing the occurrence of errors in the developed software. The coding standard is a product of consensus among the team members.

3 Coding Standards Specifications

3.1 Naming Standards

3.1.1 C#

- Classes: Use **PascalCase**. For example: public class DepartmentManager
- Methods and Functions: Use **PascalCase**. For example: public void UserDeleteBL(User p)
- Variables: Use **camelCase.** For example: public int loginID
- Constants: Use **uppercase letters and underscores (_) for separation**.
  For example: int STRING_LENGHT
- Interface names usually begin with an "I" followed by a name that describes the interface's purpose. For example: public interface IRepository<T>

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

3.1.2 React

- File names are in PascalCase and have names such as “ManageCourse”. These names are descriptive and give a clue about the context of that class.
- Main function of each class carries the same name as the class name. Other helper functions may be used in the class but they are used in the main function which carries the class name. Functions use camelCase.
- Variables also use camelCase and are descriptive.
- Constants use UPPER_SNAKE_CASE for clearance and are defined in a separate file.

3.2 File Organization

3.2.1 C#

- Each file should include a comment structure at the beginning, describing the purpose and contents of the file.
- To maintain organization and structure among relevant files and classes, folders can be used.
- C# programs are composed of one or more files. Each file can contain namespaces. A namespace can include types such as classes, structures, interfaces, enumerations, and delegates, as well as other namespaces.
- Usage directives import namespace types. Import statements should be placed at the beginning of the file.

3.2.2 React

- All files regarding frontend are in the /frontend folder. In this folder there is an extra /src folder. There are different folders such as /components, /styled, /utils.
- All folders use lowercase and their names are descriptive.
- /components folder has each component for different use cases and the component's corresponding styling is in the /styled folder. If the component is named "ManageCourse.js" for example, its style file is named "ManageCourse.css" or scMamanegCourse.js".
- /utils folder has "constans.js" which has constant values such as color codes etc.
- Imports are always made at the beginning of the file.
- All of the components and pages are routed in index.js.

3.3 Comment Standards

3.3.1 C#

- Add sufficient and descriptive comments to enhance code readability.
- Comments should explain the purpose, functionality, and complexity of the code.
- Include comments that provide information on pointers, important points, considerations, and potential errors.

3.3.2 React

- In JavaScript, single line comments are made by // and multiple-line comments are made by /* */.
- Comments are short and clear. They are used to explain the basic goal of functions, the purpose of a variable, how a function works etc.

3.4 Coding Conversions

3.4.1 C#

- Follow the recommended standards for the language you are using. For C#, referring to the C# coding standards provided by Microsoft can be helpful.
- Avoid code duplication and unnecessary complexity.
- Group related code blocks and use consistent indentation.
- Implement error handling and exception mechanisms.

3.4.2 React

- Variables are declared using "const", rather than "var".
- Long lines and comments are broken into multiple lines for readability.

3.5 Formatting

3.5.1 C#

- Use consistent formatting to improve code readability. Apply proper spacing, indentation, and line wrapping.
- Use a separate line for each code block or statement.
- Aim for lines to be around 80-120 characters long.
- Leave spaces around operators.

3.5.2 React

- After imports, variable declarations and function declarations, semicolon is used.
- When passing multiple parameters or importing multiple objects from the same file, a single space is added before and after variable names.
- White Spaces are added between different functions, variable declarations etc.