BBM 384 – SOFTWARE ENGINEERING LABORATORY

SOCIAL INTEREST E-CLUB
CODING STANDARDS

COPY-PASTERS

HAZAL UZAR	**21727834**
HACI KERİM ARSLAN	**21726928**
İBRAHİM KOZ	**21786303**
BAŞAK ŞÜKRAN MELAHAT ÇONTU	**21827282**
FATMA NUR DEMİRBAŞ	**21727116**

Social Interest e-Club
Coding Standards

1 Introduction

This document contains the coding standards of the programming language used in the source code of the Social Interest e-Club project. Coding standards ensure that all developers working on the project follow certain guidelines, maintaining the easy understanding, suitability, and consistency of the code. Hence it provides speed and quality. Since everyone has a developer role in this project, coding standards are needed.

2 Description

The coding standard is a guide that ensures that all developers working on the project follow certain guidelines. While determining the guidelines, care should be taken to follow the different levels of the system homogeneously and not to contradict each other. Finished program code should appear as if it was written by a single developer in a single session. If coding standards are not defined, developers may be using any of their own methods, which could lead to some adverse effects. Security concerns, such as performance issues. When coding standards are implemented, these issues can be easily overcome. It can provide us with a safe program with minimal or no performance issues.

Some advantages of coding standards:

- Applying coding standards can help the team detect problems early or even avoid them altogether. This will increase efficiency throughout the software process.
- Implementing coding standards reduces many problems and the risk of project failures.
- Coding standards help develop less complex software programs, thereby reducing errors.
- If coding standards are followed, the code is consistent and can be easily maintained. This is because anyone can understand it and change it at any time.
- If the source code is written consistently, it becomes easy to find and fix bugs in the software.
- A clear view of how the code fits into the larger application, or how it fits the company, if the source code remains consistent.
- An open code gives developers the opportunity to reuse code as needed. This, along with the effort put into development, can radically reduce the cost.

In summary, coding standards play a vital role in any successful software development.

Since we developed this project as a team, the codes should be understandable, readable and changeable by everyone; codes work smoothly, can grow easily; It is very important to complete the tasks in a short time. The standards described in this town are applied by all team members, they are general and high-level standards.

3 Coding Standards Specifications

a. Naming Standards

- **Namespaces:**

We used namespaces for two purposes. First, it allowed us to group and organize classes. Secondly, it helped us to define scopes of classes and methods.

```
namespace LoginAuth.Models.ClubModels
```

- **Classes:**

We named classes with generally nouns and used pascal case. Pascal casing means that the first letter of each word, including the first word, is capitalized. Class names must be short enough but also understandable.

```
public class ApplicationUser
public class SubClub
```

- **Interfaces:**

Since we have many types of response way such as sometimes just a Json file or a view, we need to make them extend from the same interface so that we can use the same actions with respect to the different developments.

The naming of interfaces is done using Pascal casing and prefix the interface names with the letter **I** to indicate that the type is interface. Their interfaces are named by adjective phrases or occasionally by nouns or noun phrases. Interface names must be short enough and understandable.

```
public async Task<IActionResult> Index()
```

- **Methods:**

Method names are usually a combination of verb and verb nouns, and Pascal casing model is used for naming. Method names should be sufficiently short and understandable.

```
public async Task<IActionResult> Index()
public async Task<IActionResult> Index(QuestionnaireIndexViewModel
                                                    questionnaireIndexViewModel)
```

- **Variables:**

Local variables and method arguments are named using the Camel casing model where words are adjacent, and the first letters are capitalized.

```
var role = await _roleManager.FindByIdAsync(viewModel.Id);
```

b. File Organization Standards

- **File Naming:** All file names must be understandable and not too long. Every file name must tell the aim of the file and what it contains.
- The front-end and back-end source codes are in different folders. In front-end, the components (which are Vue files) are organized by using different folders according to the component tree that has been designed. The component file names also obey to the file naming rule and uses pascal casing. The Vue files contains template, script and style tags in order. Style tag is not a must, and the style of the pages can be coded in external CSS files. Those CSS files are kept in a different folder.

c. Comment Standards

Comment lines are bits of text that you add to a program to explain why and how the work is done, increasing the readability of code. It can then help the written code to be understood by the person who wrote the codes or someone else. In projects developed as a team, comment lines are used at least as often as the lines of code; In this way, it becomes easier for the whole software team to work on the same codes at different times. In addition, comments lines provide information about the code when the person who wrote the code looks at the code after a while. Thus, the things to be done in the continuation of the code at different times or the necessary corrections in the code can be easily performed.

- **Single Line Comments:**

  To start a single line comment, you use two forward slashes "//". Everything that comes after is considered a comment.

  ```
  // This is a comment
  ```

- **Multi-Line Comments:**

  Multi-line comments are comments that span multiple lines. You start such a comment with the "/*" delimiter and end it with the "/*" delimiter. Everything inside the delimiters is a comment.

  ```
  /* These
      are
      the
     comment
     lines. */
  ```

d. White Spaces

- **Whitespace:**

  Spaces, tabs, and newlines are considered "whitespaces". White space is used to make the program more readable for the programmer; the compiler is indifferent. It should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

  Some example usages of space:

  - Between parameters of a function after commas,
  - Between variables and the operations between them like '=', '==', '+=' etc.

- **Empty-Lines:**

  Empty lines improve readability by setting off sections of code that are logically related.

  Two blank lines should always be used in the following circumstances:
  - Between sections of a source file
  - Between class and interface definitions

  One blank line should always be used in the following circumstances:
  - Between methods
  - Between the local variables in a method and its first statement
  - Before a block or single-line comment
  - Between logical sections inside a method to improve readability

e. Coding Conventions

- Every expression should locate on a line. One line cannot hold two expressions.
- Some long expression may be divided into two lines.
- Visual Studio Code should be used while coding to provide convenience.