DUY TÂN UNIVERSITY
International School

Capstone Project 2

CMU-SE 451

Code Standard

Version 1.0

Date: 22nd May, 2021

Food Care

Submitted by

Huynh Dac Vinh
Tran Quoc Trung
Ton That Minh Huy
Dang Van Duan

Approved by

Proposal Review Panel Representative:

Name	Signature	Date

Capstone Project 2-Mentor:

Name	Signature	Date
Dang Viet Hung		

PROJECT INFORMATION

Project acronym	FC		
Project Title	Food Care		
Start Date	01 Mar, 2021	**End Date**	25 May, 2021
Lead Institution	International School, Duy Tan University		
Project Mentor	Ph.D Hung, Dang Viet		
Scrum master / Project Leader & contact details	Vinh, Huynh Dac Email: dacvinh98@gmail.com Tel: 0347191925		
Partner Organization			
Project Web URL			
Team members	Name	Email	Tel
1	Vinh, Huynh Dac	dacvinh98@gmail.com	0347191925
2	Trung, Tran Quoc	quoctrung.tran210@gmail.com	0935420530
3	Huy, Ton That Minh	tonthatminhh@gmail.com	0935432561
4	Duan, Dang Van	dangvanduan755@gmail.com	0769710126

REVISION HISTORY

Version	Date	Comments	Author	Approval
1.1	22/05/2021	Final	Vinh, Huynh Dac	

Document Approval

The following signatures are required for approval of this document

Mentor	Dang Viet Hung	Date:
Scrum Master	Huynh Dac Vinh	Date:
Product Owner	Tran Quoc Trung	Date:
Team Member	Ton That Minh Huy	Date:
Team Member	Dang Van Duan	Date:

Contents

1. Introduction:

Anybody can write code. With a few months of programming experience, you can write 'working applications'. Making it work is easy, but doing it the right way requires more work, than just making it work. Believe it, the majority of the programmers write 'working code', but not ‘good code'. Writing 'good code' is an art and you must learn and practice it.

Everyone may have different definitions for the term ‘good code’. In my definition, the following are the characteristics of a good code.

- Reliable
- Maintainable
- Efficient

Most of the developers are inclined towards writing code for higher performance, compromising reliability, and maintainability. But considering the long term ROI (Return On Investment), efficiency and performance come below reliability and maintainability. If your code is not reliable and maintainable, you (and your company) will be spending a lot of time identifying issues, trying to understand code, etc. throughout the life of your application.

2. Purpose of coding standards and best practices:

To develop reliable and maintainable applications, you must follow coding standards and best practices.

The naming conventions, coding standards, and best practices described in this document are compiled from our own experience and by referring to various MDN and Google guidelines.

Several standards exist in the programming industry. None of them are wrong or bad and you may follow any of them. What is more important is, selecting one standard approach and ensuring that everyone is following it.

3. How to follow the standards across the team:

If you have a team of different skills and tastes, you are going to have a tough time convincing everyone to follow the same standards. The best approach is to have a team meeting and develop your own standards document. You may use this document as a template to prepare your document.

Distribute a copy of this document (or your coding standard document) well ahead of the coding standards meeting. All members should come to the meeting prepared to discuss the pros and cons of the various points in the document. Make sure you have a manager present in the meeting to resolve conflicts.

Discuss all points in the document. Everyone may have a different opinion about each point, but at the end of the discussion, all members must agree upon the standard you are going to follow. Prepare a new standards document with appropriate changes based on the suggestions from all of the team members. Print copies of it and post it in all workstations.

After you start the development, you must schedule code review meetings to ensure that everyone is following the rules. 3 types of code reviews are recommended:

1. Peer review – another team member reviews the code to ensure that the code follows the coding standards and meets requirements. This level of review can include some unit testing also. Every file in the project must go through this process.

2. Architect review – the architect of the team must review the core modules of the project to ensure that they adhere to the design and there are no "big" mistakes that can affect the project in the long run.

3. Group review – randomly select one or more files and conduct a group review once a week. Distribute a printed copy of the files to all team members 30 minutes before the meeting. Let them read and come up with points for discussion. In the group review meeting, use a projector to display the file content on the screen. Go through every section of the code and let every member give their suggestions on how that piece of code can be written in a better way. (Don't forget to appreciate the developer for the good work and also make sure he does not get offended by the "group attack"!).

4. ES6 Features:

JavaScript ES6 brings new syntax and new awesome features to make your code more modern and more readable. It allows you to write less code and do more. ES6 introduces us to many great features like arrow functions, template strings, class destruction, Modules… and more.

1. Const and Let

const is a new keyword in ES6 for declaring variables. const is more powerful than var. Once used, the variable can't be reassigned. In other words, it's an immutable variable except when it is used with objects.

This is really useful for targeting the selectors. For example, when we have a single button that fires an event, or when you want to select an HTML element in JavaScript, use const instead of var. This is because var is 'hoisted'. It's always preferable to use const when you don't want to reassign the variable.

let can be reassigned and take new value. It creates a mutable variable.

let is the same as const in that both are blocked-scope. It means that the variable is only available within its scope.

2. Arrow functions

The arrow function is really awesome and makes your code more readable, more structured, and looks like modern code.

Template Literals

Template literals or template strings are pretty cool. We don't have to use the plus (+) operator to concatenate strings, or when we want to use a variable inside a string.

3. Default parameters

In JavaScript, function parameters default to undefined. However, it's often useful to set a different default value. This is where default parameters can help.

In the past, the general strategy for setting defaults was to test parameter values in the function body and assign a value if they are undefined.

4. Array and object destructing

Destruction makes the assignment of the values of an array or object to the new variable easier.

With ES5, we have to assign each value to each variable. With ES6, we just put our values within curly brackets to get any property of the object.

5. Import and export

Using import and export in your JavaScript application makes it more powerful. They allow you to create separate and reusable components.

6. Promises

Promises are a new feature of ES6. It's a method to write asynchronous code. It can be used when, for example, we want to fetch data from an API, or when we have a function that takes time to be executed. Promises make it easier to solve the problem.

7. Rest parameter and Spread operator

The rest parameters are used to get the argument of an array and return a new array.

The spread operator has the same syntax as the rest parameter, but the spread operator takes the Array itself and not just the arguments. We can use the Spread parameter to get the values of an Array, instead of using a for loop or any other method.

8. Classes

Classes are the core of object-oriented programming (OOP). They make your code more secure and encapsulated. Using classes gives your code a nice structure and keeps it oriented.

5. Indentation:

The unit of indentation is four spaces. Use of tabs should be avoided because (as of this writing in the 21st Century) there still is not a standard for the placement of tab stops. The use of spaces can produce a larger file size, but the size is not significant over local networks, and the difference is eliminated by minification.

6. Comments:

Be generous with comments. It is useful to leave information that will be read at a later time by people (possibly yourself) who will need to understand what you have done. The comments should be well-written and clear, just like the code they are annotating. An occasional nugget of humor might be appreciated. Frustrations and resentments will not.

Comments must be kept up-to-date. Erroneous comments can make programs even harder to read and understand.

Make comments meaningful. Focus on what is not immediately visible. Don't

waste the reader's time with stuff like:

```
i = 0; // Set i to zero.
```

Generally, use line comments. Save block comments for formal documentation.

7. Variable Declarations:

1. Variable naming

For variable names use lowerCamelCasing and use concise, human-readable, semantic names where appropriate.

```
let playerScore = 0;

let speed = distance / time;
```

2. Declaring variables

When declaring variables and constants, use the let and const keywords, not var.

If a variable will not be reassigned, prefer const:

```
const myName = 'Chris';

console.log(myName);
```

8. Operators and comparison:

Ternary operators:

Ternary operators should be put on a single line:

let status = (age >= 18) ? 'adult' : 'minor';

Use strict equality:

Always use strict equality and inequality.

name === 'Chris';

age !== 25;

Use shortcuts for boolean tests

Use shortcuts for boolean tests — use x and !x, not x === true and x === false.

9. Control statements:

There should be no space between a control statement keyword and its opening parenthesis.

There should be a space between the parentheses and the opening curly brace.

```
if(iceCream) {
  alert('Woo hoo!');
}
```

10. Strings:

Use template literals

```
let myName = 'Chris';
console.log(`Hi! I'm ${myName}!`);
```

Use textContent, not innerHTML

When inserting strings into DOM nodes, use Node.textContent:

```
let text = 'Hello to all you good people';
const para = document.createElement('p');
para.textContent = text;
```

11. Conditionals:

When loops are required, feel free to choose an appropriate loop out of the available ones (for, for...of, while, etc.) Just make sure to keep the code as understandable as possible.

When using for/for...of loops, make sure to define the initializer properly, with a let keyword:

```
let cats = ['Athena', 'Luna'];

for(let i of cats) {
    console.log(i);
}
```

12. Functions and objects:

1. Function naming

For function names use lowerCamelCasing and use concise, human-readable, semantic names where appropriate.

```
function sayHello() {
    alert('Hello!');
};
```

2. Defining functions

Where possible, use the function declaration to define functions over function expressions:

```
function sum(a, b) {
    return a + b;
}
```

When using anonymous functions inside a method that requires a function as a parameter, it is acceptable (although not required) to use an arrow function to make the code shorter and cleaner.

```
let sum = (a, b) => {
    return a + b;
}
```

3. Creating objects

Use literals — not constructors — for creating general objects (i.e., when classes are not involved):

```
let myObject = {};
```

4. Object classes

```
class Person {
    constructor(name, age, gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
     greeting() {
        console.log(`Hi! I'm ${this.name}`);
     };
}
```

Use extends for inheritance:

```
class Teacher extends Person {
    ...
}
```

5. Object naming

When defining an object class (as seen above), use UpperCamelCasing (also known as PascalCasing) for the class name, and lowerCamelCasing for the object property and method names.

When defining an object instance, either a literal or via a constructor, use lowerCamelCase for the instance name:

```
let hanSolo = new Person('Han Solo', 25, 'male');


let hanSolo = {
    name: 'Han Solo',
    age: 25,
    gender: 'male'
}
```

13. Arrays

Creating arrays

Use literals — not constructors — for creating arrays:

```
let myArray = [ ];
```

Adding to an array

When adding items to an array, use push(), not direct assignment. Given the following array:

```
const pets = [];
```

14. Error handling

If certain states of your program throw uncaught errors, they will halt execution and potentially reduce the usefulness of the example. You should therefore catch errors using a try...catch block:

```
try {
    console.log(results);
}
catch(e) {
    console.error(e);
}
```

15. REFERENCES

Freecodecamp. (2020, December 14). *JavaScript ES6 — write less, do more*. Retrieved from https://www.freecodecamp.org/news/write-less-do-more-with-javascript-es6-5fd4a8e50ee2/

MDN. (2020, December 14). *Javascript_guidelines*. Retrieved from https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript