PATATES
Coding Standards

1 Introduction

This document includes descriptions and methods to provide some rules about coding. These rules are basically our coding standards. These coding standards will be related the source code of Patates Online Book Store application. All team member must obey to these coding standards. That is important because every team member has developer role also. This means every member will write code and they will need to read and understand others codes. To handle that, we need some coding standards.

2 Description

Coding standards are basically a guide for developers that describes how to write code. Important concerns of coding standards are naming, comments, clean and understandable coding. The main purpose of that is not writing efficient or working code. The main purpose of that writing understandable and so that maintainable code. This is very important because in the team there are 5 developer. Everybody writes code. So everybody should understand and may change the written last code. At first, determining some rules might look slowing down the coding. But for all coding and developing progress, that will provide change to write code faster and easier.

On the other hand, of course coding standards may change depending on language we used. But standards which describes in this documents are general and high level standards.

3 Coding Standards Specifications

a- Naming Standards

- Classes:

  We named classes with generally nouns and using pascal case. Pascal case means first letter of every word will be capitalized (included first one). Class names must be short enough and understandable.

```
public class OrderController

public class OrderService
```

- Packages:

  Package names must be some group of word. These words and standards may be different according to related organization's naming standards.

```
package com.patates.webapi.Services.UserServices;

package com.patates.webapi.Controllers;
```

- Interfaces:

  Naming of interfaces are similar with classes. Pascal case should be used. Main difference is that should be generally adjectives. Interface names must be short enough and understandable.

```
public interface SearchServices

public interface UserServices
```

- Methods:

  Method names are generally verb or verb noun combination. Camel case model is used for methods. First words start with lowecase and if there were more words, they starts with uppercases. Method names must be short enough and understandable.

```
public String useCouponCode(@RequestParam String code)

public PasswordEncoder passwordEncoder()
```

- Variables:

  All variables, instances, parameters should have a name in camel case notation. They generally starts with nouns. Variable names must be short enough and understandable. Temporary variable (like loop variables) can be created with one letter.

```
private String imageUrl;

private double price;
```

b- File Organization Standards

- File Naming: All file names must be short enough and understandable. Every file name must reflect the what includes in it and what works for.

- Backend and frontend source codes are in different folders. Also their subfolders arranged by components. All files grouped by their functionality. Controllers are located in controllers folder, models are located in models folder. This provide good reflection and make files easily accessible. Because we know where is what.

- In a code file, at first, there may be comment about the file. After that, package declaration comes for specify the related package. Following the package declaration, import statements are located. After all of these, classes or interfaces can be defined in any order. There must be no duplicate or useless import statements. Also there must be no hard-coded values.

```java
package com.patates.webapi.Controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class SwaggerController {
    @RequestMapping("/")
    public String greeting() {
        return "redirect:/swagger-ui.html";
    }
}
```

c- Comment Standards

Every file, function, class and variable names are good-named so we can easily understand what it does and how it works. But of course there may be some confusing or complex operations. If there is, we should write comments about that. Also these comments will increase the readibility of the code. Further, too much comments shows that code has low quality so.

The important part of comments is that we should keep comments updated. If we change a code, we should change its comment of course. If old comment has forgotten, that may create problem at that point and can cause misunderstanding. These comments may have different types.

- One Line Comments:

  At some points we may need little comments like TODO etc. Here developers will use single line comments.

```
// This is single line comment.
```

- Block Comments:

  At some points we may need to make longer comments. Here developers will use block comments. Also this block comments may be used for description of file.

```
/*
    This
    is
    block
    comment
*/
```

d- White Spaces

- Empty Lines:

  Empty lines make the code more readible. So developers need to leave new line to different sections of code. For example at first declare the package of file and leave new line. Make import operations without new line and at the end of the imports, leave new line. Declare a class and declare some attributes in it. After the attribute declarations, break new line and start to write functions. Leave new line of course between two functions.

- Whitespace:

  Like empty lines, whitespaces will increase the readibility of code also.

  - Use whitespace between parameters of function after the commas.

```
functionName(parameter1, parameter2, parameter3)
```

  - Use whitespace when some variable assigned or applied mathematical operations.

```
var = newValue
var = (first + second) / third
```

  - Use whitespaces when comparison operations used.

```
if(first == second)
```

e- Coding Conventions

- Every expression should locate on a line. One line cannot hold two expression.
- Some long expression may be divided into two lines.

```
functionName (longParameterName1, longParameterName2,
                                longParameterName3, longParameterName4)
```

- Each tabs corresponds to 5 whitespaces.
- Intellij IDE should be used while coding. This will provide many convenience and also it helps to follow some coding standards of us automatically.
- Ternary statements may be used instead of writing multi line short statements.