# ICES4HU
# Coding Standards

## 1 Introduction

This document outlines the coding standards for REACTion's Java and JavaScript source code. It covers things like how the code should look, naming variables, organizing code, and handling errors. Following these standards helps us create clean, readable, and maintainable code. It also promotes teamwork and allows us to review each other's code easily. By sticking to these guidelines, we ensure that our codebase is professional, high-quality, and consistent.

## 2 Description

This document is a guide for a software development team to adopt a common coding standard and best practices in their projects. It is used to ensure consistency, readability and maintainability. It includes topics such as coding standards, code formatting, naming conventions, interpretation guidelines, error handling, and documentation. It facilitates collaboration among team members on software projects and improves the quality of the software while facilitating the understanding and maintenance of the code.

## 3 Coding Standards Specifications

## 3.1 Naming Standards

Following standards listed below are the rules for source code when naming variables, with sections of programming languages used in the project.

### 3.1.1 Java

#### 3.1.1.1 Package Names

Each package is named with lowercase letters. If there are consecutive words, they shall be written with lowercase letters. Uppercase letters or underscores will not be used.

Example: java.com.reaction.ices4hu.core

#### 3.1.1.2 Class Names

Each class should be named in PascalCase. If the class is a controller, it should end with Controller. Same rule applies to request classes ending with Request and data access object classes ending with Dao.

example: **LoginRequest.java**

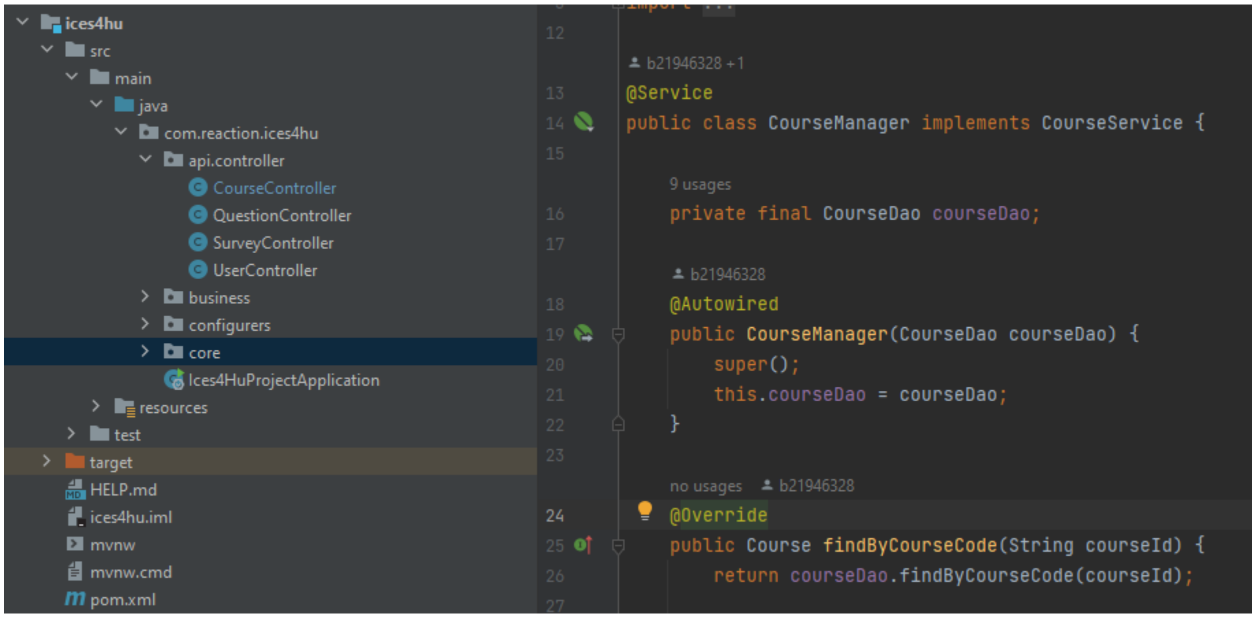
#### 3.1.1.3 Method Names

Each should be named in camelCase.

example: **deleteCourse**

### 3.1.1.4 Variable Names

Each variable, local or class, should be named in camelCase. This rule applies to parameters as well.



```java
@Service
public class CourseManager implements CourseService {

    private final CourseDao courseDao;

    @Autowired
    public CourseManager(CourseDao courseDao) {
        super();
        this.courseDao = courseDao;
    }

    @Override
    public Course findByCourseCode(String courseId) {
        return courseDao.findByCourseCode(courseId);
```

### 3.1.2 Javascript

### 3.1.2.1 Class Names

Each class should be named in PascalCase.

example: **InstructorHomepage.js**

### 3.1.2.2 Class Method Names

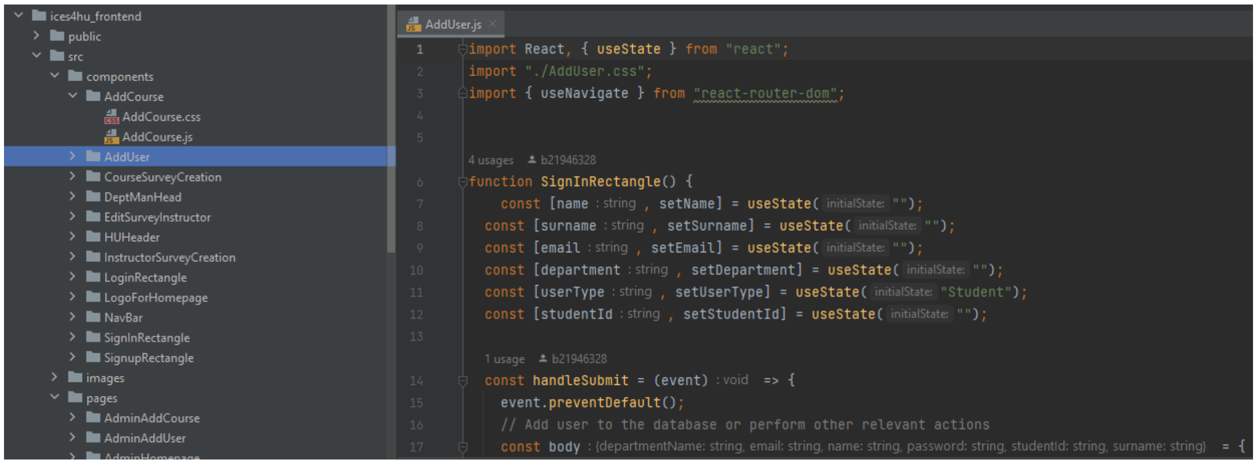
Each class method should be named in camelCase.

### 3.1.1.3 Function Names

Each function should be named in camelCase.

### 3.1.1.4 Variable Names

Each variable, local or class, should be named in camelCase. This rule applies to parameters as well.



```
ices4hu_frontend
  public
  src
    components
      AddCourse
        AddCourse.css
        AddCourse.js
      AddUser
      CourseSurveyCreation
      DeptManHead
      EditSurveyInstructor
      HUHeader
      InstructorSurveyCreation
      LoginRectangle
      LogoForHomepage
      NavBar
      SignInRectangle
      SignupRectangle
    images
    pages
      AdminAddCourse
      AdminAddUser
```

AddUser.js

```
import React, { useState } from "react";
import "./AddUser.css";
import { useNavigate } from "react-router-dom";


4 usages  b21946328
function SignInRectangle() {
    const [name : string , setName] = useState( initialState: "");
  const [surname : string , setSurname] = useState( initialState: "");
  const [email : string , setEmail] = useState( initialState: "");
  const [department : string , setDepartment] = useState( initialState: "");
  const [userType : string , setUserType] = useState( initialState: "Student");
  const [studentId : string , setStudentId] = useState( initialState: "");

  1 usage  b21946328
  const handleSubmit = (event) : void  => {
    event.preventDefault();
    // Add user to the database or perform other relevant actions
    const body : {departmentName: string, email: string, name: string, password: string, studentId: string, surname: string}  = {
```

### 3.2 File encoding

The guidelines to be followed when altering the source files' content and the file structure of the source code repository are listed below, sorted by the programming languages that are being utilized.

Source files are encoded in UTF-8.

### 3.2.1 Java

#### 3.2.1.1 File Names

Every source file in the filesystem must be named in PascalCase.

#### 3.2.1.2 Folder Structure

All folders shold be in camelCase. The source file tree's folders should follow the Java language definition for package names. Layer-by-layer organization of the packages means that logically, source code from the same layer should be placed inside the same package.

### 3.2.2 JavaScript

#### 3.2.2.1 File Names

Every source file in the filesystem must be named in PascalCase.

#### 3.2.2.2 Folder Structure

All folders shold be in PascalCase. Layer-by-layer organization of the packages means that logically, source code from the same layer should be placed inside the same package.

## 3.3 Comment Standards

### 3.3.1 Java

#### 3.3.1.1 Line Comment

The first character of inline comments in source code should be lowercase, and there should be one space between "/*" and "*/" before and after them.

#### 3.3.1.2 Inline Comment

The first letter of each line comments in the source code should be capitalized, and there should be 1 whitespace after "//". There should be two whitespaces before "//" if the line comment comes after a line of source code.

### 3.3.2 JavaScript

#### 3.3.2.1 Line Comment

The first character of inline comments in source code should be lowercase, and there should be one space between "/*" and "*/" before and after them.

#### 3.3.2.2 Inline Comment

The first letter of each line comments in the source code should be capitalized, and there should be 1 whitespace after "//". There should be two whitespaces before "//" if the line comment comes after a line of source code.

## 3.4 Programming Practice Conventions

This part is about file structure of source code repository rules to apply in organization. They are mostly programming language independent.

### 3.4.1 Nestedness

#### 3.4.1.1 If Statements

If there are more than two nested if clauses, it is advisable to question the design. Having excessive nested if clauses significantly raises the cognitive complexity of the source code and places a heavy burden on the reader.

#### 3.4.1.2 Loops

If there are more than three nested loops, it is recommended to evaluate the design. Having excessive nested loops can raise concerns about the code's structure and readability.

### 3.4.2 Notation

#### 3.4.2.1 Overridden Methods

In Java source code, it is considered good practice to include the @Override annotation whenever a method is overridden.

#### 3.4.2.2 NotNull

In Java source code, @NotNull annotation should be used if the field can not be null.

## 3.5 Formatting

Below are the guidelines that should be adhered to when organizing the file structure within the source code repository. These guidelines are organized based on the programming languages employed, ensuring a systematic and coherent arrangement of files within the repository. By following these rules, developers can maintain a well-structured and easily navigable codebase, enhancing collaboration and facilitating efficient development processes

### 3.5.1 Java

#### 3.5.1.1 Indentation

Indentation in the code should be achieved using spaces, specifically by using a length of four whitespace characters (four spaces).

#### 3.5.1.2 If clauses

There should be no line breaks before the left bracket.

Example: **if (statement) { ... }**

### 3.5.2 JavaScript

#### 3.5.2.1 Indentation

Indentation in the code should be achieved using spaces, specifically by using a length of four whitespace characters (four spaces).

#### 3.5.2.2 Use of semicolon

While the JavaScript language specification permits statements to be written without semicolons, it is recommended to conclude every statement with a semicolon.Thus, every statement should be ended with a semicolon.

#### 3.5.2.3 If clauses

There should be no line breaks before the left bracket.

Example: **if (statement) { ... }**