

**Online learning Application**

**Software Design Document**

– Hanoi, May 2022 –

Contents

[I. Overview 4](#_Toc103733649)

[1. Code Packages 4](#_Toc103733650)

[2. Coding Conventions 4](#_Toc103733651)

[1. Java Coding Conventions 4](#_Toc103733652)

[2. JavaScript Coding Conventions 4](#_Toc103733653)

[II. Code Designs 4](#_Toc103733654)

[1. <Feature/Function Name1> 4](#_Toc103733655)

[3. a. Class Diagram 5](#_Toc103733656)

[4. b. Class Specifications 5](#_Toc103733657)

[5. DashboardFilter Class 5](#_Toc103733658)

[6. MyCourseFilter Class 5](#_Toc103733659)

[7. PostManagerFilter Class 5](#_Toc103733660)

[8. SliderManagerFilter Class 5](#_Toc103733661)

[9. LessonContentFilter Class 5](#_Toc103733662)

[10. AccountManagerFilter Class 5](#_Toc103733663)

[11. AccountSettingFilter Class 5](#_Toc103733664)

[12. CourseManagerFilter Class 5](#_Toc103733665)

[13. c. Sequence Diagram(s) 5](#_Toc103733666)

[14. d. Database queries 5](#_Toc103733667)

[2. <Feature/Function Name2> 5](#_Toc103733668)

[III. Database Design 5](#_Toc103733669)

[1. Database Schema 5](#_Toc103733670)

[2. Table Description 5](#_Toc103733671)

# I. Overview

## 1. Code Packages

*[Provide the package diagram for each sub-system. The content of this section including the overall package diagram, the explanation, package and class naming conventions in each package. Please see the sample and description table format below – following Java project naming convention]*



***Package descriptions & package class naming conventions***

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| *1* | model | *<Description of the package>*  *<Class naming convention>* |
| *2* | view |  |
| *3* | *controller* |  |
| *4* | *dao* |  |
| *5* | *service* |  |
| *6* | *util* |  |
| *7* | *filter* | *Includes classes that implement the application's Filter interface. These classes will receive and process requests before this request reaches the classes in the controller. The classes in the package implement authorization functions to restrict access to restricted resources and perform unauthorized functions.* |

## 2. Coding Conventions

### Java Coding Conventions

##### Indentation

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).

##### Line Length

Avoid lines longer than 80 characters, since they’re not handled well by many terminals and tools.

##### Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

• Break after a comma.

• Break before an operator.

• Prefer higher-level breaks to lower-level breaks.

• Align the new line with the beginning of the expression at the same level on the previous line.

• If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

//DON’T USE THIS INDENTATION

**if** **((**condition1 **&&** condition2**)**

**||** **(**condition3 **&&** condition4**)**

**||!(**condition5 **&&** condition6**))** **{** //BAD WRAPS

doSomethingAboutIt**();** //MAKE THIS LINE EASY TO MISS

**}**

//USE THIS INDENTATION INSTEAD

**if** **((**condition1 **&&** condition2**)**

**||** **(**condition3 **&&** condition4**)**

**||** **!(**condition5 **&&** condition6**))** **{**

doSomethingAboutIt**();**

**}**

//OR USE THIS

**if** **((**condition1 **&&** condition2**)** **||** **(**condition3 **&&** condition4**)**

**||** **!(**condition5 **&&** condition6**))** **{**

doSomethingAboutIt**();**

**}**

Here are three acceptable ways to format ternary expressions:

// First way

alpha **=** **(**aLongBooleanExpression**)** **?** beta **:** gamma**;**

// Second way

alpha **=** **(**aLongBooleanExpression**)** **?** beta

**:** gamma**;**

// Third way

alpha **=** **(**aLongBooleanExpression**)**

**?** beta

**:** gamma**;**

#### Comments

##### Block Comments

Block Comments Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments should be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe. A block comment should be preceded by a blank line to set it apart from the rest of the code. Block comments have an asterisk “\*” at the beginning of each line except the first.

/\*

\* Here is a block comment.

\*/

##### Single-Line Comments

Single-Line Comments Short comments can appear on a single line indented to the level of the code that follows. If a comment can’t be written in a single line, it should follow the block comment format (see section 5.1.1). A single-line comment should be preceded by a blank line. Here’s an example of a single-line comment in Java code.

**if** **(**condition**)** **{**

/\* Handle the condition. \*/

**}**

Note: Don’t try comment like this

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// | |

// | Beautiful Comment |

// |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

//

##### End-Of-Line Comments

The // comment delimiter begins a comment that continues to the newline. It can comment out a complete line or only a partial line. It shouldn’t be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code. Examples of all three styles follow:

**if** **(**number **>** 1**)** **{**

**return** **true;** // Explain why here.

**}** **else** **{**

**return** **false;** // Explain why here.

**}**

#### Declarations

##### Number Per Line

One declaration per line is recommended since it encourages commenting.

int level**;** // indentation level

int size**;** // size of table

In other words, is preferred over

int level**,** size**;**

##### Placement

Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces “{” and “}”.) Don’t wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

public void doSomething**(**int number**)** **{**

boolean condition **=** **(**number **<** 0**);**

**if** **(**condition**)** **{**

System**.**out**.**println**(**"Negative"**);**

**}**

**}**

##### Initialization

Try to initialize local variables where they’re declared. The only reason not to initialize a variable where it’s declared is if the initial value depends on some computation occurring first.

##### Class and Interface Declarations

When coding Java classes and interfaces, the following formatting rules should be followed:

* No space between a method name and the parenthesis “(“ starting its parameter list.
* Open brace “{” appears at the end of the same line as the declaration statement.
* Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{“.
* Methods are separated by a blank line.

#### Statements

##### Simple Statements

Each line should contain at most one statement.

Example:

count**++;** count**--;** // AVOID!

##### if, if-else, if-else-if-else Statements

The if-else class of statements should have the following form:

**if** **(**condition**)** **{**

statements**;**

**}**

// or

**if** **(**condition**)** **{**

statements**;**

**}** **else** **{**

statements**;**

**}**

// or

**if** **(**condition**)** **{**

statements**;**

**}** **else** **if** **(**condition**)** **{**

statements**;**

**}** **else** **if** **(**condition**)** **{**

statements**;**

**}**

Note: if statements always use braces {}. Avoid the following error-prone form:

**if** **(**condition**)**

printSomething**();**

Note: restrict use nested if/else over three-level nested:

**if** **(**condition1**)** **{**

**if** **(**condition2**)** **{**

**if** **(**condition3**)** **{**

**if** **(**condition4**)** **{**

**...**

**}**

**}**

**}**

**}**

##### while Statements

A while statement should have the following form:

**while** **(**condition**)** **{**

statements**;**

**}**

##### switch Statements

A switch statement should have the following form:

**switch** **(**condition**)** **{**

**case** ABC**:**

statements**;**

/\* falls through \*/

**case** DEF**:**

statements**;**

**break;**

**case** XYZ**:**

statements**;**

**break;**

**default:**

statements**;**

**break;**

**}**

##### try-catch Statements

A try-catch statement should have the following format:

**try** **{**

statements**;**

**}** **catch** **(**ExceptionClass e**)** **{**

statements**;**

**}**

### JavaScript Coding Conventions

#### Declarations And Variable Names

Use **camelCase** for identifier names.

Note: Don’t use the var keyword when declaring a variable.

**function** doSomething**()** **{**

**let** age **=** 10**;** // This is preferred

**var** size **=** 0**;** // Don't do this

**}**

#### Spaces Around Operators

Always put spaces around operators ( = + - \* / ), and after commas:

**let** x **=** y **+** z**;**

**const** myArray **=** **[**"Volvo"**,** "Saab"**,** "Fiat"**];**

#### Statement Rules

General rules for simple statements:

* Always end a simple statement with a semicolon.

**const** cars **=** **[**"Volvo"**,** "Saab"**,** "Fiat"**];**

**const** person **=** **{**

firstName**:** "John"**,**

lastName**:** "Doe"**,**

age**:** 50**,**

eyeColor**:** "blue"

**};**

General rules for complex (compound) statements:

* Put the opening bracket at the end of the first line.
* Use one space before the opening bracket.
* Put the closing bracket on a new line, without leading spaces.
* Do not end a complex statement with a semicolon.

#### Object Rules

General rules for object definitions:

* Place the opening bracket on the same line as the object name.
* Use colon plus one space between each property and its value.
* Use quotes around string values, not around numeric values.
* Do not add a comma after the last property-value pair.
* Place the closing bracket on a new line, without leading spaces.
* Always end an object definition with a semicolon.

**const** person **=** **{**

firstName**:** "John"**,**

lastName**:** "Doe"**,**

age**:** 50**,**

eyeColor**:** "blue"

**};**

#### Line Length Smaller Than 80 Characters

For readability, avoid lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.

# II. Code Designs

## 1. <Feature/Function Name1>

*[Provide the detailed design for the function <Feature/Function Name1>. It include Class Diagram, Class Specifications, and Sequence Diagram(s)]*

### a. Class Diagram

*[This part presents the class diagram for the relevant feature]*



### b. Class Specifications

### DashboardFilter Class

Block requests to access the Dashboard screen based on account permissions. Only accounts with Sales, Marketing and Admin permissions have access to the Dashboard when making a request.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Block requests to access the Dashboard, requests when insufficient permissions will be redirected to a message page with content that cannot be accessed because of insufficient permissions. |

### MyCourseFilter Class

When request to query data about registered courses if not logged in or already logged in but this request with intent to get course data of another account then request will fail.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Making a request to get data about registered courses when not logged in will redirect to the login screen.  Make a request to get data about registered courses when logged in, but this request includes parameters that are geared towards retrieving data with the intention of retrieving information other than those of the currently logged in account. request will not be fulfilled and redirect to error page. |

### PostManagerFilter Class

Requests to list posts, edit posts, delete posts based on accounts with insufficient permissions will fail. Only accounts with Marketing, Admin rights can access.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Requests to enter the post edit screen, list posts, delete posts without permission will be redirected to the 'Error Page' |

### SliderManagerFilter Class

Requests to list slider, edit slider, delete slider based on accounts with insufficient permissions will fail. Only accounts with Marketing, Admin rights can access.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Requests to enter the slider edit screen, list sliders, delete sliders without permission will be redirected to the 'Error page' |

### LessonContentFilter Class

Handle requests for access to any course content.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | When not logged in if a request is made, the user is redirected to the 'Introduce Course' page.  When logged in but accessing course content that the user is not registered with, the action will be redirected to the 'Introduce Course' page.  With an account that has registered for the course and has Admin rights, access will be successful. |
|  |  |  |

### AccountManagerFilter Class

All accounts with non-Admin permissions when performing access to the account management screen will not be allowed.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Requests to enter the account edit screen, list accounts, edit accounts that do not have permission will be redirected to the ‘Error Page'. |

### AccountSettingFilter Class

Performing access to the account's configuration page when the account's login session does not exist is not performed.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | Redirects to the login screen when trying to access the account's configuration page when no account login session exists. |

### CourseManagerFilter Class

Block and redirect requests to edit, list, and manage courses. Only accounts with Expert and Admin permissions can access the above screens.

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter() | When not logged in will redirect to the login screen.  Using an account with unauthorized permissions will redirect to an error page.  When using an account with Expert privileges, feedback and changes will be limited depending on the resource that belongs to that Expert only. With Admin rights, resources will not be limited. |

### c. Sequence Diagram(s)

*[Provide the sequence diagram(s) for the feature, see the sample below]*



### d. Database queries

*[Provide the detailed SQL (select, insert, update...) which are used in implementing the function/screen]*

## 2. <Feature/Function Name2>

…

# III. Database Design

## 1. Database Schema

*[Provide the tables relationship like example below – following MySQL database naming convention]*



## 2. Table Description

|  |  |  |
| --- | --- | --- |
| **No** | **Table** | **Description** |
| *01* | *<Table name>* | *<Description of the table>*  *- Primary keys: <<list of primary key fields>>*  *- Foreign keys: <<list of foreign key fields>>* |
| *02* |  |  |