Fpt University

TRƯỜNG ĐẠI HỌC FPT

ONLINE LEARNING SYSTEM

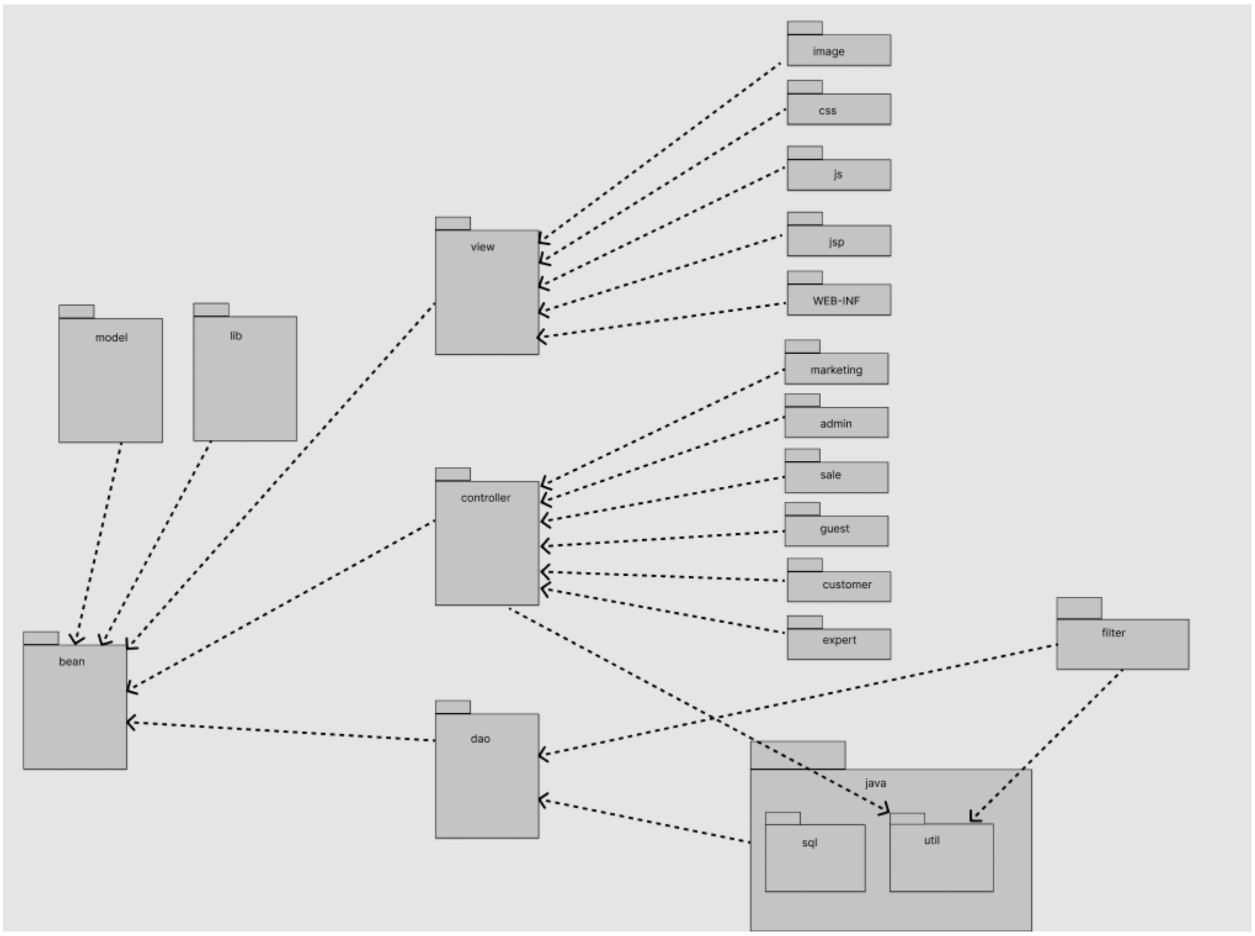
Software Design Document

– Hanoi, May 2022 –

Table of Contents

I. Overview

1. Code Packages/Namespaces



Package descriptions & package class naming conventions

No	Package	Description
01	bean	bean is a package containing the main modules of a program.
02	model	This is the data layer that contains the business logic of the system, and also represents the state of the application. It's independent of the presentation layer, the controller fetches the data from the model layer and sends it to the view layer.
03	view	The view layer represents the output of the application, usually some form of UI. The presentation layer is used to display the Model data fetched by the Controller.

04	controller	The controller layer acts as an interface between View and Model. It receives requests from the View layer and processes them, including the necessary validations.
05	util	java.util Package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes.
06	lib	contains the file sqljdbc is an API standard (Application Program Interface) that allows to connect programs written in Java with database management systems.
07	sql	java.sql provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java programming language.
08	filter	Package filter is a firewall technique used to control network access by monitoring outgoing and incoming packets and allowing them to pass or halt based on the source and destination Internet Protocol (IP) addresses, protocols and ports.
09	dao	The dao layer will communicate with the storage system, the database management system, such as performing jobs related to data storage and query (search, add, delete, edit, ...).
10	image	Storing images
11	css	Storing css files or Bootstrap files
12	js	Storing js files
13	jsp	Storing jsp files
14	WEB-INF	Storing web.xml files

2. Coding Conventions

Java Code Conventions

1. Why Have Code Conventions

1.1. Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well ` packaged and clean as any other product you create.

1.2 Acknowledgments

- This document reflects the Java language coding standards presented in the *Java Language Specification*, from Sun Microsystems. Major contributions are from Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, and Scott Hommel.
- For adaptation, modification, or redistribution of this document: http://java.sun.com/docs/codeconv/html/Copyright.doc.html.

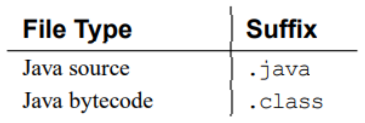
  Comments on this document: http://java.sun.com/docs/forms/sendusmail.html.

2. File Names

This section lists commonly used file suffixes and names.

2.1 File Suffixes

JavaSoft uses the following file suffixes:



File Type	Suffix
Java source	.java
Java bytecode	.class

2.2 Common File Names

3. File Organization

- A file consists of sections that should be separated by blank lines and an optional comment identifying each section.
- Files longer than 2000 lines are cumbersome and should be avoided.

3.1 Java Source Files

- Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file:
    - ❖ Beginning comments
    - ❖ Package and Import statements
    - ❖ Class and interface declarations

3.1.1 Beginning Comments

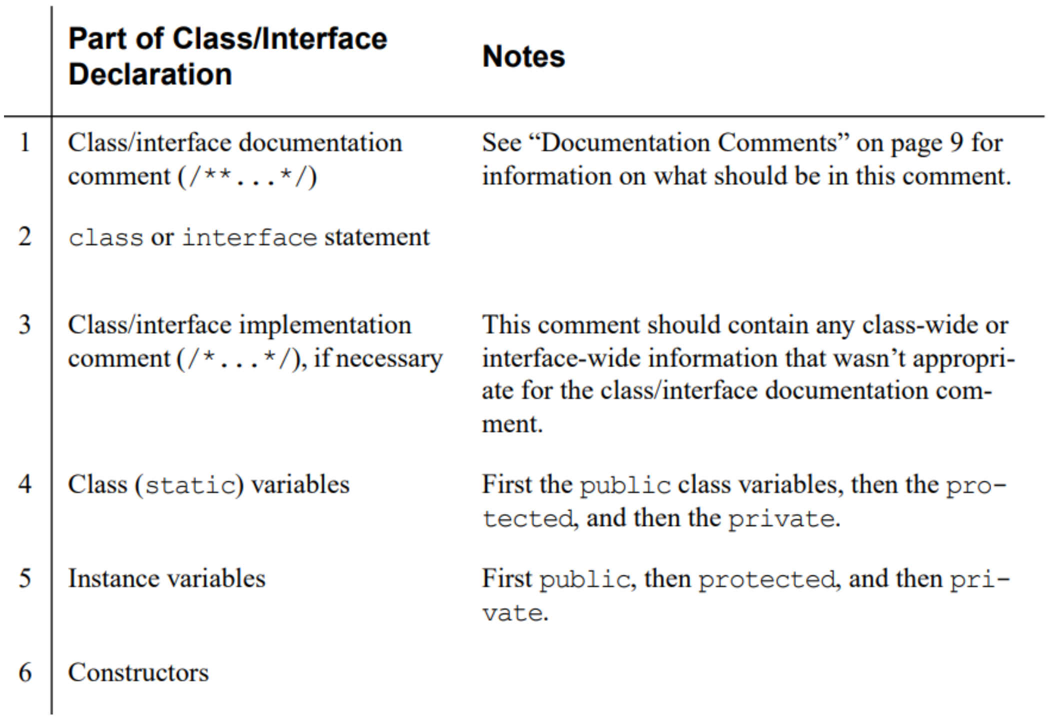
❖ All source files should begin with a c-style comment that lists the programmer(s), the date, a copyright notice, and also a brief description of the purpose of the program.

3.1.2 Package and Import Statements

❖ The first non-comment line of most Java source files is a package statement. After that, import statements can follow.

3.1.3 Class and Interface Declarations

❖ The following table describes the parts of a class or interface declaration, in the order that they should appear. See "Java Source File Example".



	Part of Class/Interface Declaration	**Notes**
1	Class/interface documentation comment (`/**...*/`)	See "Documentation Comments" on page 9 for information on what should be in this comment.
2	`class` or `interface` statement	
3	Class/interface implementation comment (`/*...*/`), if necessary	This comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment.
4	Class (`static`) variables	First the `public` class variables, then the `protected`, and then the `private`.
5	Instance variables	First `public`, then `protected`, and then `private`.
6	Constructors	

	Part of Class/Interface Declaration	Notes
7	Methods	These methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier.

4 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).

4.1 Line Length

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

Note: Examples for use in documentation should have a shorter line length—generally no more than 70 characters

4.2 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.

Here are some examples of breaking method calls:

```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```

Following are two examples of breaking an arithmetic expression. The first is preferred, since the break occurs outside the parenthesized expression, which is at a higher level.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
            + 4 * longname6; // PREFER

longName1 = longName2 * (longName3 + longName4
                         - longName5) + 4 * longname6; // AVOID
```

Following are two examples of indenting method declarations. The first is the conventional case. The second would shift the second and third lines to the far right if it used conventional indentation, so instead it indents only 8 spaces.

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ...
}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

Line wrapping for if statements should generally use the 8-space rule, since conventional (4 space) indentation makes seeing the body difficult. For example:

```
//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:

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

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

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

5 Comments

- Java programs can have two kinds of comments: implementation comments and documentation comments.

- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. For example, information about how the corresponding package is built or in what directory it resides should not be included as a comment.
- **Note: The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.**
- Comments should not be enclosed in large boxes drawn with asterisks or other characters.
- Comments should never include special characters such as form-feed and backspace.

5.1 Block Comments

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.

*/
```

5.2 Single-Line Comments

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. */ ...

}
```

5.3 Trailing Comments

Avoid the assembly language style of commenting every line of executable code with a trailing comment. Here's an example of a trailing comment in Java code:

```
if (a == 2) {

return TRUE;   /* special case */

} else {

return isprime(a);        /* works only for odd a */

}
```

5.4 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.

```
// Do a double-flip.
```

6. Declarations

6.1 Number Per Line

- One declaration per line is recommended since it encourages commenting.
- In absolutely no case should variables and functions be declared on the same line.
- Do not put different types on the same line.

6.2 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. The one exception to the rule is indexes of for loops, which in Java can be declared in the for statement
- Avoid local declarations that hide declarations at higher levels.

6.3 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.

6.4 Class and Interface Declarations

- 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 "{"

7 Statements

7.1 Simple Statements

Each line should contain at most one statement. Example:

```
argv++; argc--;          // AVOID!
```

Do not use the comma operator to group multiple statements unless it is for an obvious reason. Example:

```
if (err) {
    Format.print(System.out, "error"), exit(1); //VERY WRONG!
}
```

7.2 Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces “{ statements }”. See the following sections for examples. • The enclosed statements should be indented one more level than the compound statement. • The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement. • Braces are used around all statements, even singletons, when they are part of a control structure, such as a if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

7.3 return Statements

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

7.4 if, if-else, if-else-if-else Statements

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

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

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) //AVOID! THIS OMITS THE BRACES {}!
    statement;
```

7.5 for Statements

A for statement should have the following form:

```
for (initialization; condition; update) {
    statements;
}
```

An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```
for (initialization; condition; update);
```

When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables. If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause).

7.6 while Statements

A while statement should have the following form:

```
while (condition) {
    statements;
}
```

An empty while statement should have the following form:

```
while (condition);
```

7.7 do-while Statements

A do-while statement should have the following form:

```
do {
    statements;
} while (condition);
```

7.8 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;
}
```

Every time a case falls through (doesn't include a break statement), add a comment where the break statement would normally be. This is shown in the preceding code example with the /* falls through */ comment.

Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

7.9 try-catch Statements

A try-catch statement should have the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

8 White Space

8.1 Blank Lines

- Blank lines improve readability by setting off sections of code that are logically related.
- Two blank lines should always be used:
  - ❖ Between sections of a source file
  - ❖ Between class and interface definitions
- One blank line should always be used:
  - ❖ 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

8.2 Blank Spaces

- Blank spaces should be used:
  - ❖ A keyword followed by a parenthesis should be separated by a space (Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.)

```
while (true) {
    ...
}
```

  - ❖ A blank space should appear after commas in argument lists
  - ❖ All binary operators except should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
    n++;
}
prints("size is " + foo + "\n");
```

  - ❖ The expressions in a for statement should be separated by blank spaces

```
for (expr1; expr2; expr3)
```

  - ❖ Casts should be followed by a blank

```
myMethod((byte) aNum, (Object) x);
myFunc((int) (cp + 5), ((int) (i + 3))
                          + 1);
```

9 Naming Conventions

Naming convention make programs more understandable by making them easier to read. They can also give information about the function of the identifier—for example, whether it’s a constant, package, or class—which can be helpful in understanding the code.

Identifier Type	Rules for Naming
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).
Interfaces	Interface names should be capitalized like class names.
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic— that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary “throwaway” variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores (“_”). (ANSI constants should be avoided, for ease of debugging.)

10 Programming Practices

10.1 Providing Access to Instance and Class Variables

Don’t make any instance or class variable public without good reason. Often, instance variables don’t need to be explicitly set or gotten—often that happens as a side effect of method calls.

10.2 Referring to Class Variables and Methods

Avoid using an object to access a class (static) variable or method. Use a class name instead.

10.3 Constants

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

10.4 Variable Assignments

Avoid assigning several variables to the same value in a single statement. It is hard to read. Do not use the assignment operator in a place where it can be easily confused with the equality operator. Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler, and besides, it rarely actually helps.

10.5 Miscellaneous Practices

10.5.1 Parentheses

It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems.

10.5.2 Returning Values

Try to make the structure of your program match the intent.

10.5.3 Expressions before ‘?’ in the Conditional Operator

If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized.

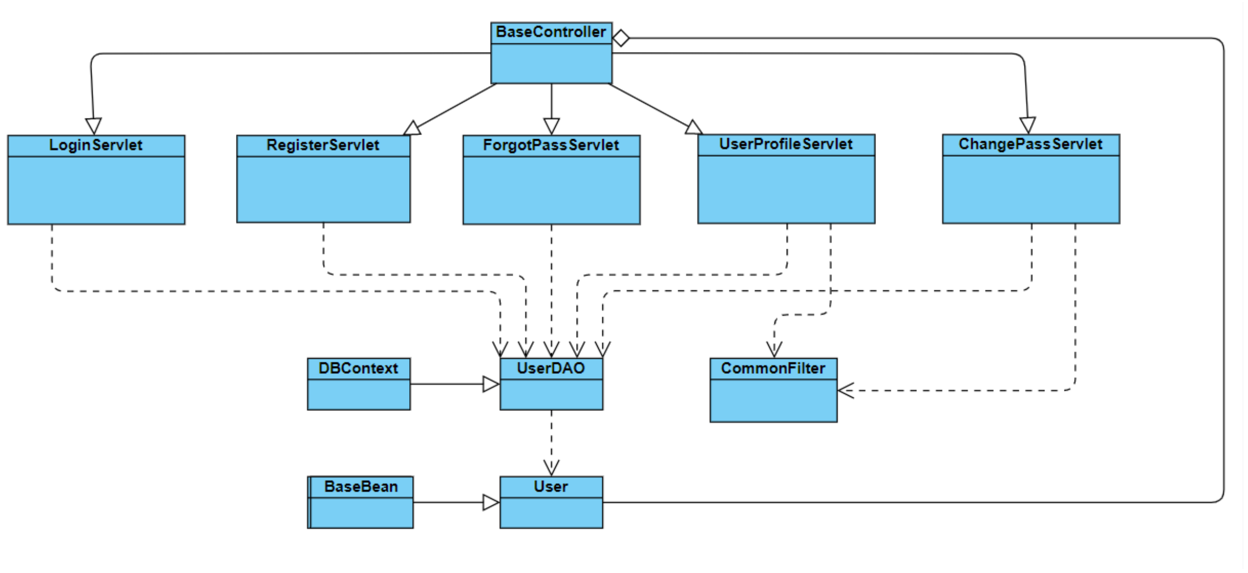
10.5.4 Special Comments

Use XXX in a comment to flag something that is bogus but works. Use FIXME to flag something that is bogus and broken.

II. Code Designs

1. <Common feature>

a. Class Diagram



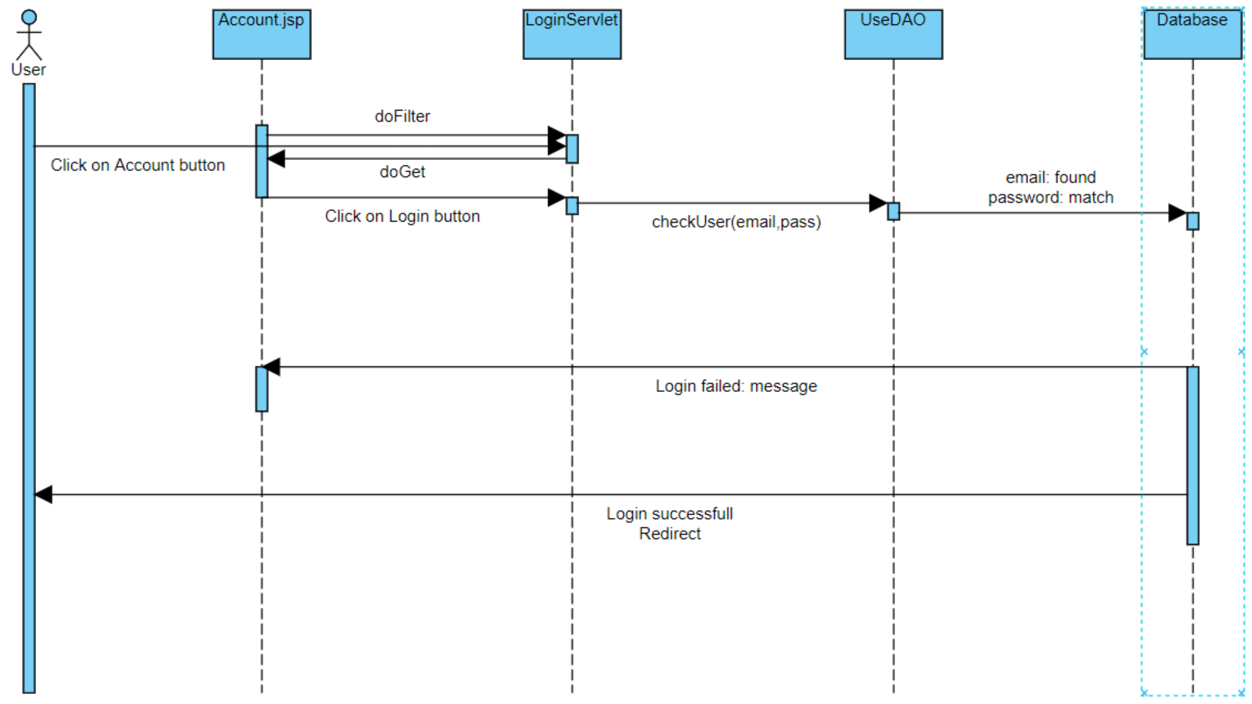
b. Class Specifications

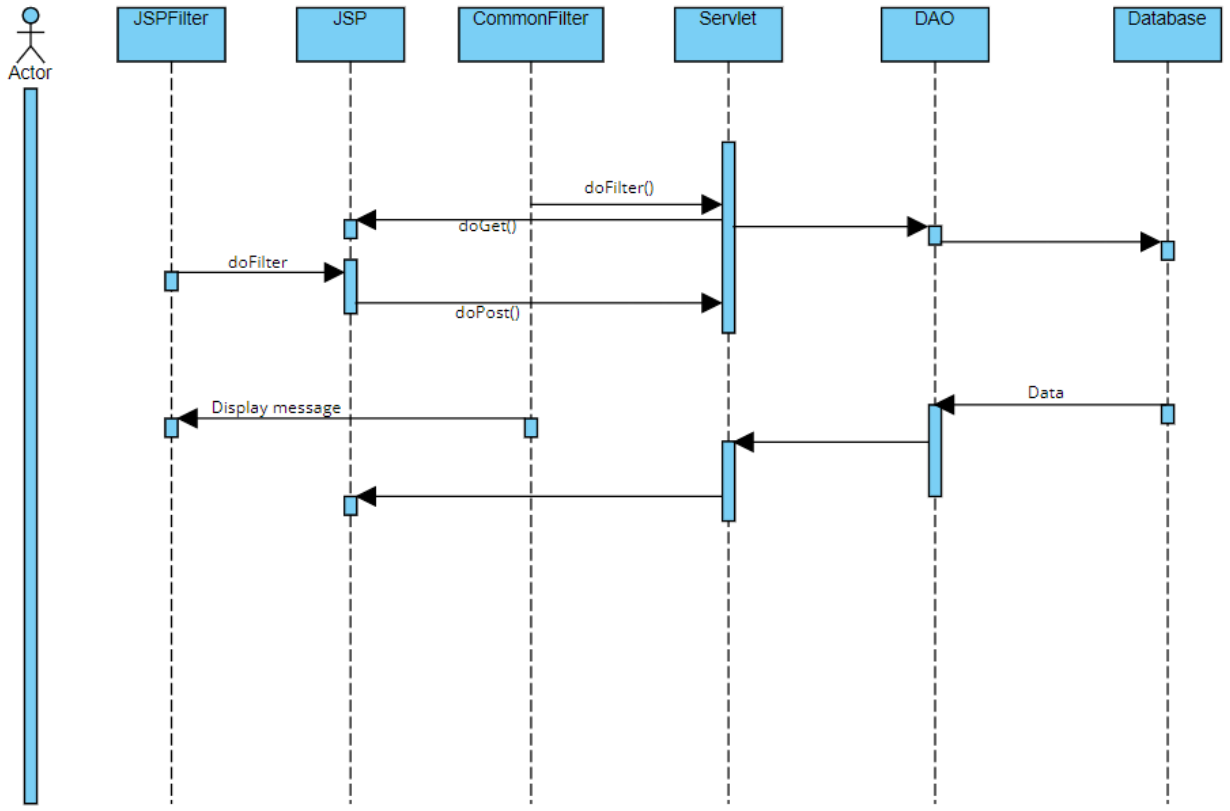
Class UserDAO

No	Method	Description
1	getAll()	Get all user’s information
2	checkUser(String email)	Check user (exists in the database or not) by email
3	checkUser(String email, String pass)	Check user (exists in the database or not) by email and password
4	getRole(User u)	Get the user's role
5	getUserByUid(int uId)	Get user’s information by user id
6	updateUser(User u, int uid)	Update user’s information by user id
7	updateNewPass(String email, String newpass)	Update new password
8	updateUserPassword(User u)	Update new password
9	Encryption(String password)	Encrypting password to save in database

10	insertUser(User ca)	Register new account

c. Sequence Diagram(s)



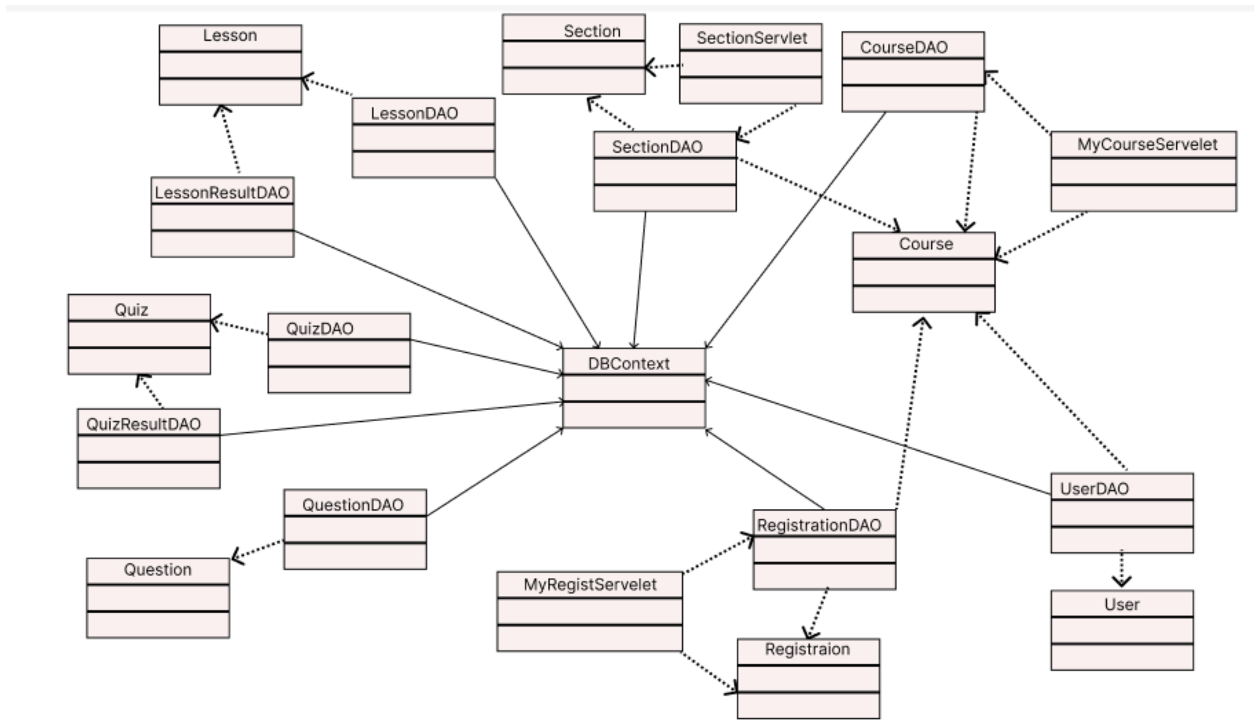


d. Database Queries

- select * from [User] where user_email=?
- select * from [User] where user_email=? and password=?
- select r.role_name from [User] u join [Role] r on u.role_id = r.role_id\n" where u.user_id = ?
- select * from [User] where user_id=?
- update [User] set user_email = ?, full_name = ?, user_img = ?, gender_id = ? , user_dob = ?, user_phone = ?, user_address = ?, user_wallet= ? where user_id = ?
- update [User] set password = ? where user_email = ?
- Update [User] set [password] = ? where user_email like ?
- insert into [User](user_email, password, full_name, gender_id, role_id) values(?,?,?,?,?)

2. <Public Feature>

a. Class Diagram.



b. Class Specifications

- CourseDAO class

No	Method	Description
01	getAll	get all courses when user clicks to button in home page
02	getCourseById	get course by courseId
03	getCourseByFilter	get course by filters selection
04	getCourseBySid	get course by subject Id

- PostDAO class

No	Method	Description
01	getPost	get all posts

02	getTop3Post	specify top 3 posts
03	getPostByBlogID	get all posts by blog Id

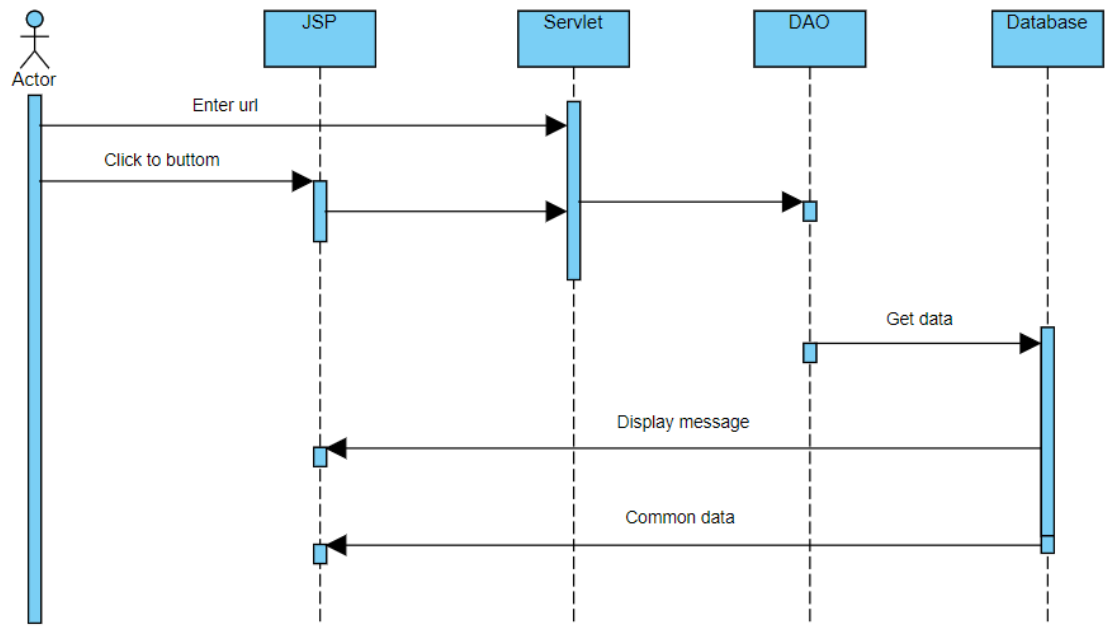
- BlogDAO class

No	Method	Description
01	getBlog	get all blogs
02	getBlogById	get blog by blog id

- SubjectDAO and LevelDAO and LecturerDAO class

No	Method	Description
01	getAll	get all subjects, levels and lecturers
02	getLectByLectID	get lecturer by lecturer id

c. Sequence Diagram(s)

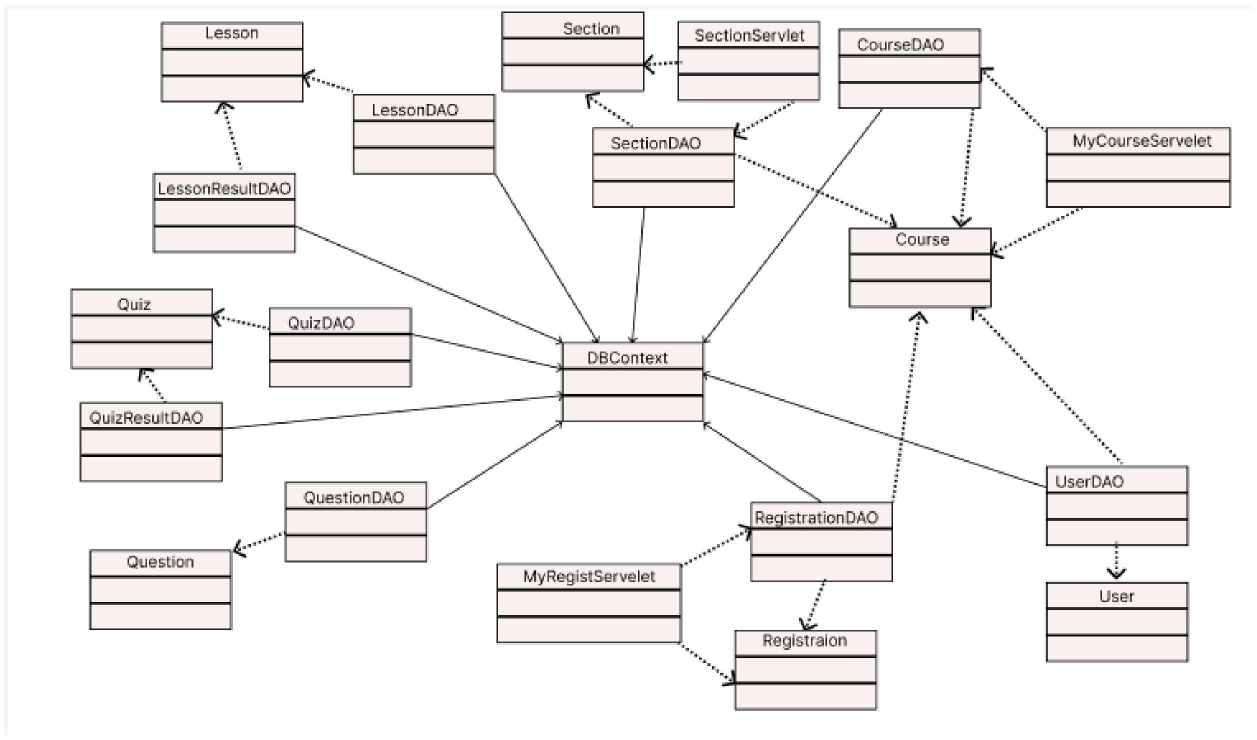


d. Database Queries

- select * from Course
- select * from Course where course_id like ?
- select * from Course where sub_id = ?
- select post_id,post_img,post_title,post_desc,post_date,post_status,p.blog_id from Post p inner join Blog b on p.blog_id=b.blog_id order by post_date desc
- select top 3 post_id,post_img,post_title,post_desc,post_date,post_status,p.blog_id from Post p inner join Blog b on p.blog_id=b.blog_id order by post_date desc
- select * from Post where post_title like ?
- select * from Blog where blog_id = ?
- select * from Blog
- select * from Lecturer
- select * from Lecturer where lecturer_id = ?
- Select * from Level
- select * from Subject

3. <Customer Feature>

a. Class Diagram



b. Class Specifications

- UserDAO

No	Method	Description
1	getAll()	Get all user's information
2	checkUser(String email)	Check user (exists in the database or not) by email
3	checkUser(String email, String pass)	Check user (exists in the database or not) by email and password
4	getRole(User u)	Get the user's role
5	getUserByUid(int uId)	Get user's information by user id
6	updateUser(User u, int uid)	Update user's information by user id
7	updateNewPass(String email, String newpass)	Update new password

8	updateUserPassword(User u)	Update new password
9	Encryption(String password)	Encrypting password to save in database
10	insertUser(User ca)	Register new account

- CourseDAO class

No	Method	Description
1	getAll	list all courses
2	getCourseById	get course by id
3	getCourseByFilter	get course by filters selection

- SectionDAO class

No	Method	Description
1	getListSectionByCourseID	list all section by courseid

- LessonDAO class

No	Method	Description
1	getAll	list all lessons
2	getLessonByCId	get lesson by course id
3	getLessonByFilter	get Lesson by filters selection

- LessonResultDAO class

No	Method	Description
1	getListLessonByUid	list all lessons by userid
2	updateLessonResult	update lesson by lesson id & user id
3	insertLessonResult	insert lesson by lesson id & user id & lessonstatus
4	deleteLessonResult	delete lesson by lesson id & user id

- QuizDAO class

No	Method	Description
1	getAll	list all quizzes
2	getQuizByCId	get Quiz by course id
3	getQuizByQid	get Quiz by Quiz id

- QuizResultDAO class

No	Method	Description
1	getListStatusQuizResultByUid	list all status of quizzes
2	getQuizResByQuizResId	get Quiz by quiz results
3	getQuizname	get quizname by quizresultid
4	getHighestGradeByQid	get highgrade by quizid

	getListQuizResByQuizResId	get list quiz result by quizresultid

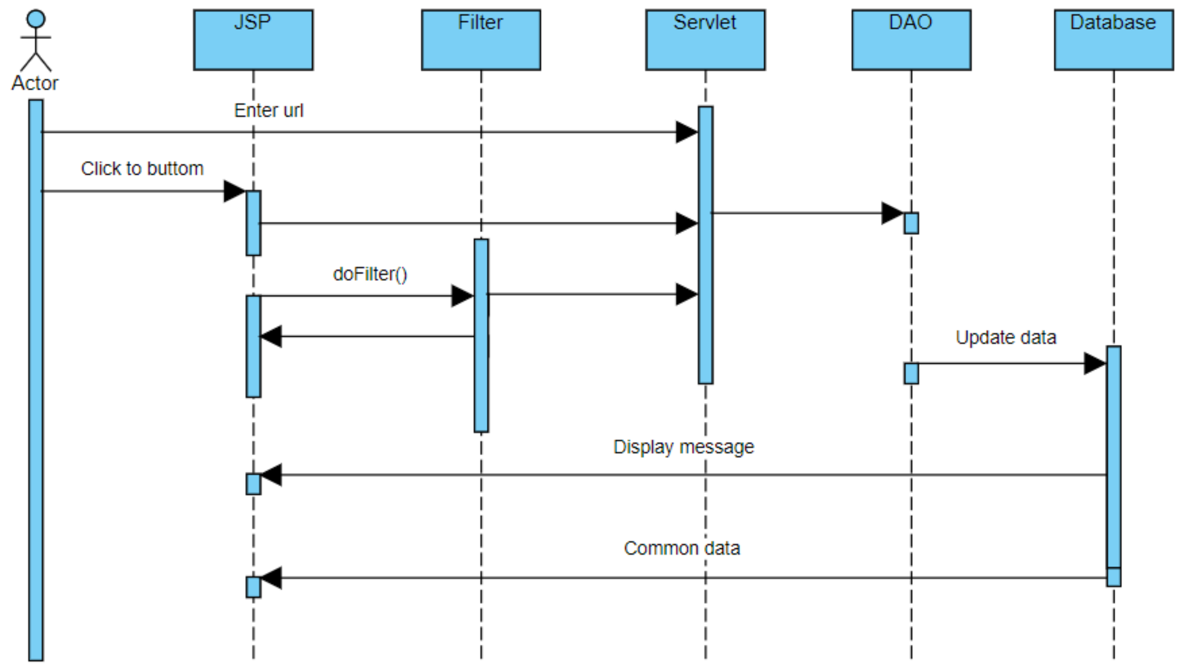
- QuestionDAO class

No	Method	Description
1	getAll	get all questions

- RegistrationDAOclass

No	Method	Description
1	getAll	get all Registration
2	checkEnoll	check if user registered the course by course id & user id
3	getRegByRegId	get registration by regid

c. Sequence Diagram(s)

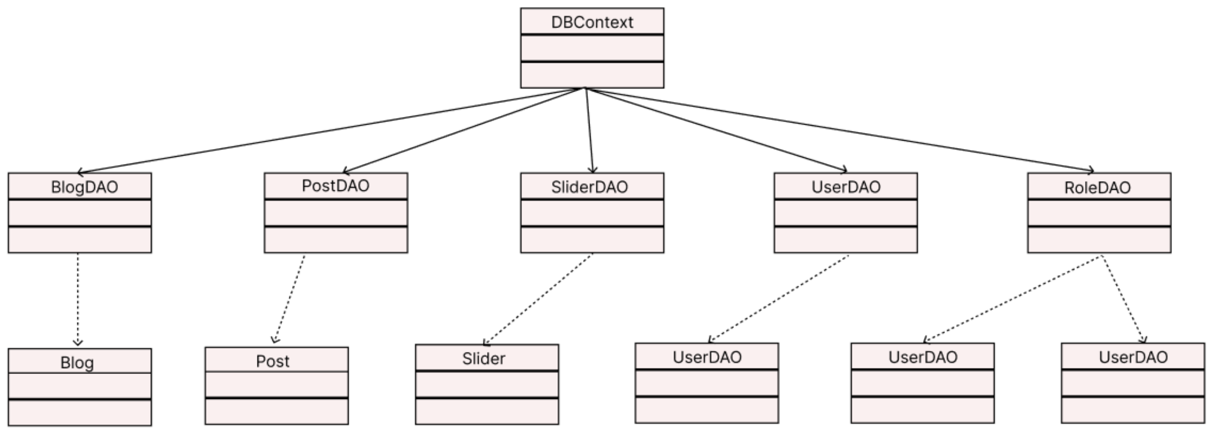


d. Database Queries

- select * from Course
- select * from Course where course_id like ?
- select * from Course where sub_id = ?
- select * from [User] where user_email=? and password=?
- select * from [User] where user_id = ?
- select * from Role
- select * from Lesson where course_id like ?
- select * from Lesson where course_id like ? and section_id = ?
- select * from Lesson where lesson_id = ?
- select * from Lesson_Result where user_id = ?
- update Lesson_Result set lesson_status = 1 where lesson_id = ? and user_id = ?
- insert into Lesson_Result(lesson_id,[user_id],lesson_status) values (?,?,0)
- delete Lesson_Result where lesson_id= ? and user_id = ?
- select * from Question where ques_id = ?
- select * from Quiz where course_id like ?
- select * from Quiz
- select * from Question
- Select * from Registration
- select * from Registration where user_id = ? and course_id = ?
- select * from Registration where reg_id = ?

4. <Marketing Feature>

a. Class Diagram



b. Class Specifications

- BlogDAO

No	Method	Description
1	getBlogByID	list Blog by id
2	getBlog	list all Blog

- PostDAO

No	Method	Description
1	getPost	list all Post
2	getPostByBlogID	get Post by blog_id

3	getPostByID	get Post by id
4	searchPost	search Post by name

- SliderDAO

No	Method	Description
1	getSlider	list all Slider
2	searchSlider	search Slider by title
3	getSliderByID	get Slider by id

- UserDAO

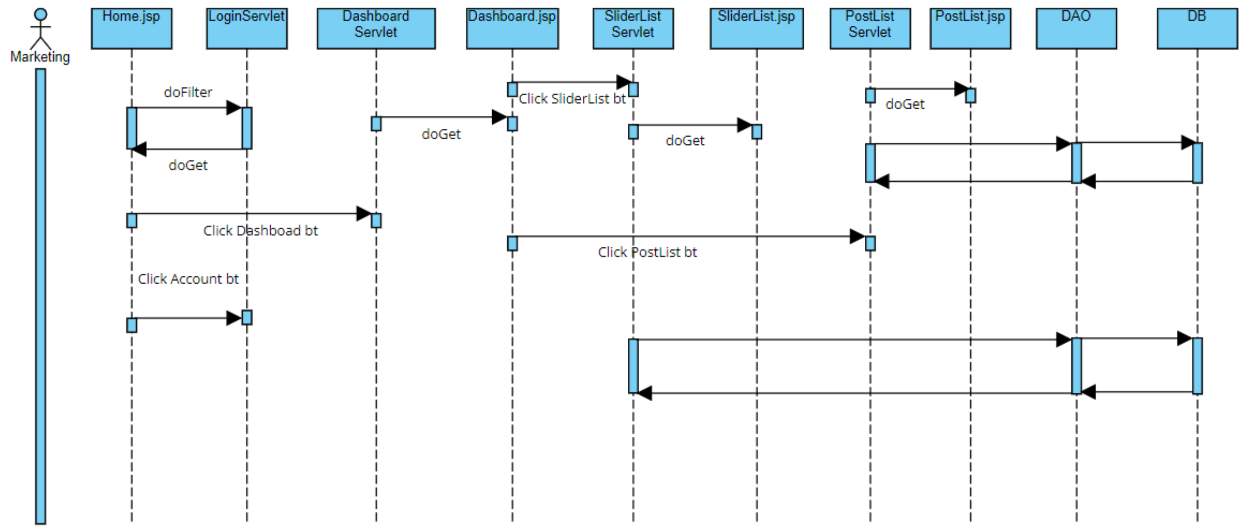
No	Method	Description
1	checkUser	check email and password
2	getUserByID	get User by id

- RoleDAO

No	Method	Description

1	getRole	get all Role

c. Sequence Diagram(s)



d. Database Queries

select * from Blog

select * from Blog where blog_id = ?

select * from Post

select * from Post where post_id = ?

select * from Post where blog_id = ?

select * from Post where post_title like ?

select * from Slider

select * from Slider where slider_id = ?

select * from Slider where slider_title like ?

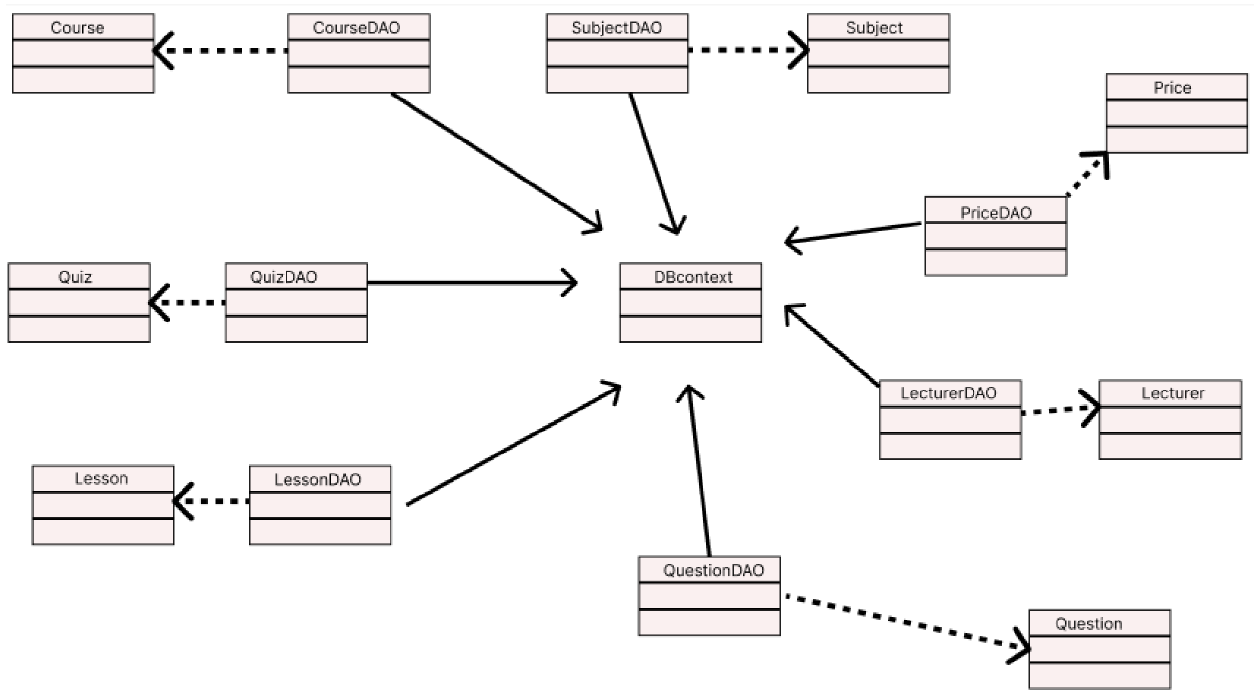
select * from [User] where user_email=? and password=?

select * from [User] where user_id = ?

select * from Role

5. <Course Content>

a. Class Diagram



b. Class Specifications

- CourseDAO class

No	Method	Description
1	getAll	list all courses
2	getCourseById	get course by id
3	getCourseByFilter	get course by filters selection

- LessonDAO class

No	Method	Description

1	getAll	list all lessons
2	getLessonByCId	get lesson by course id
3	getLessonByFilter	get Lesson by filters selection

- SubjectDAO class

No	Method	Description
1	getAll	list all subjects

- QuizDAO class

No	Method	Description
1	getAll	list all quizzes
2	getQuizByCId	get Quiz by course id
3	getQuizByQid	get Quiz by Quiz id

- LecturerDAO class

No	Method	Description
1	getAll	list all lecturer

- PriceDAO class

No	Method	Description
1	getPricePackageByCid	get PricePackageby course cid

- QuestionDAO class

No	Method	Description
1	getAll	get all questions

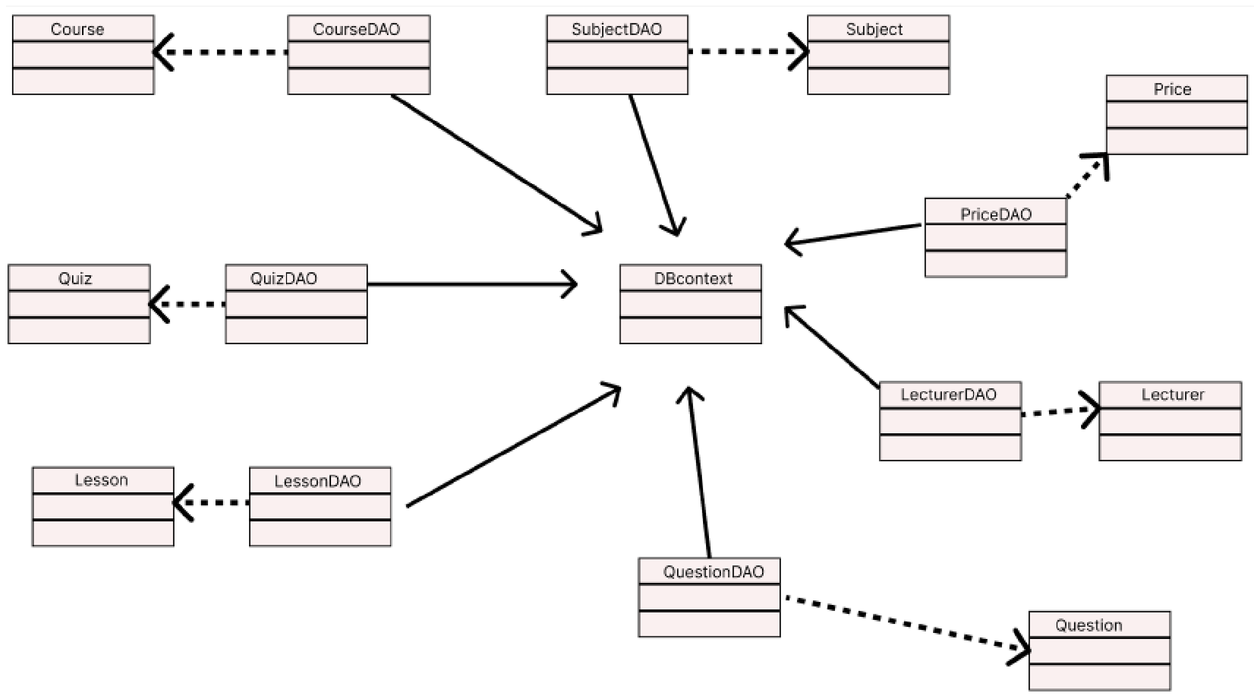
c. Sequence Diagram(s)

d. Database Queries

- select * from Course
- select * from Course where course_id like ?
- select * from Course where sub_id = ?
- select * from Lesson where course_id like ?
- select * from Lesson where lesson_id = ?
- Select * from Price_Package
- select * from Question where ques_id = ?
- select * from Quiz where course_id like ?
- select * from Quiz
- select * from Question
- select * from Subject

6. <Testing Content>

a. Class Diagram



b. Class Specifications

- CourseDAO class

No	Method	Description
1	getAll	list all courses
2	getCourseById	get course by id
3	getCourseByFilter	get course by filters selection

- LessonDAO class

No	Method	Description
1	getAll	list all lessons
2	getLessonByCId	get lesson by course id

3	getLessonByFilter	get Lesson by filters selection

- SubjectDAO class

No	Method	Description
1	getAll	list all subjects

- QuizDAO class

No	Method	Description
1	getAll	list all quizzes
2	getQuizByCId	get Quiz by course id
3	getQuizByQid	get Quiz by Quiz id

- LecturerDAO class

No	Method	Description
1	getAll	list all lecturer

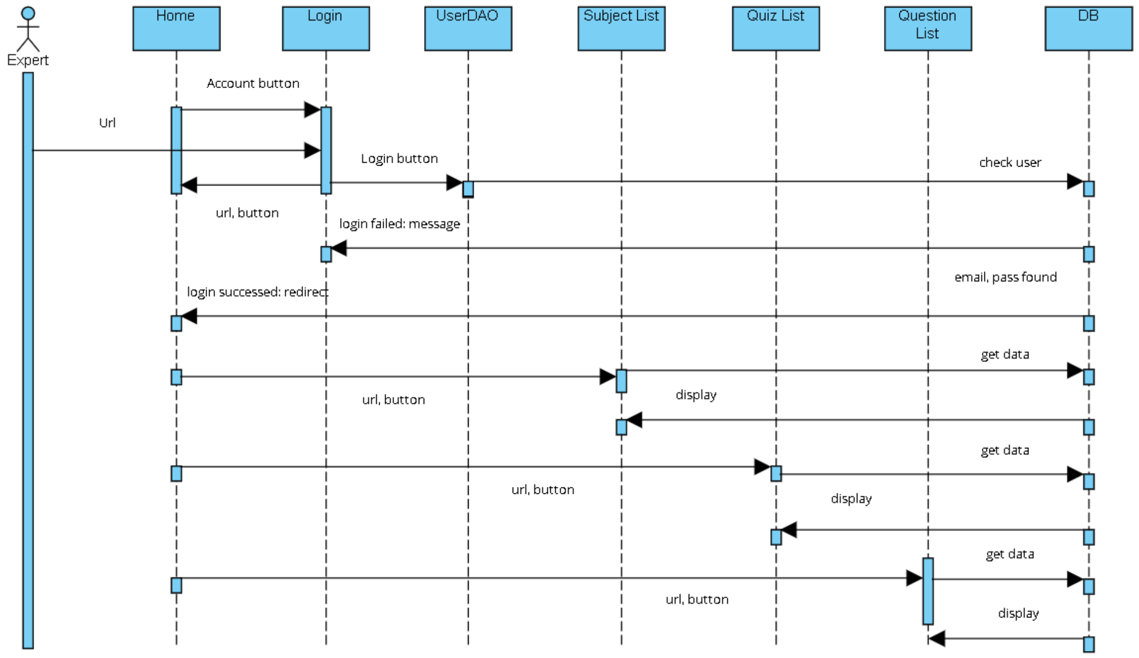
- PriceDAO class

No	Method	Description
1	getPricePackageByCid	get PricePackageby course cid

- QuestionDAO class

No	Method	Description
1	getAll	get all questions

c. Sequence Diagram(s)



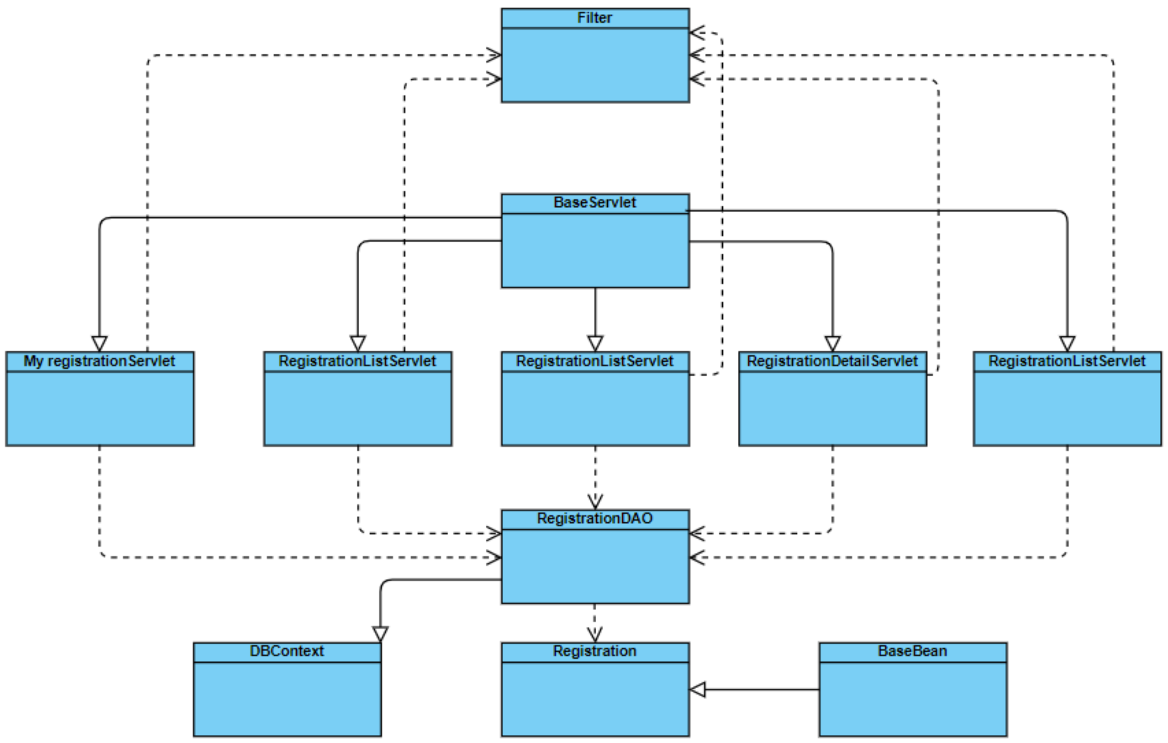
d. Database Queries

- select * from Course
- select * from Course where course_id like ?
- select * from Course where sub_id = ?
- select * from Lesson where course_id like ?
- select * from Lesson where lesson_id = ?
- Select * from Price_Package

- select * from Question where ques_id = ?
- select * from Quiz where course_id like ?
- select * from Quiz
- select * from Question
- select * from Subject

7 <Sales Feature>

a. Class Diagram



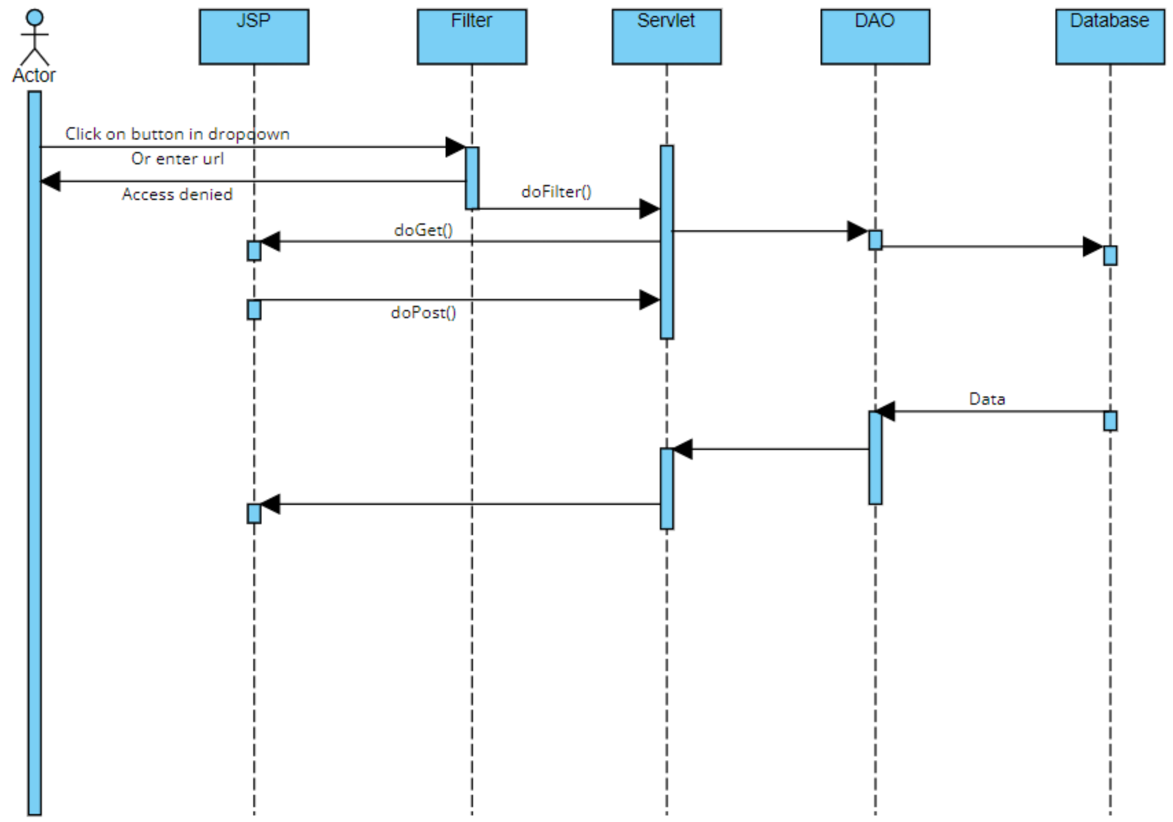
b. Class Specifications

- Registration

No	Method	Description
1	countTotalOfRes	count number of registration

2	getAllAndFilterListReg	get all and filter registration
3	bothSearchAndFilter	get all and filter registration by user_id

c. Sequence Diagram(s)



d. Database Queries

- select r.reg_id, r.reg_time, r. course_id, r.user_id, r.reg_status_id, r.last_updated_by

  from Registration r

  join Course c on r.course_id = c.course_id

  join Subject s on c.sub_id = s.sub_id

  join Subject_Category sc on s.subject_cate_id = sc.subject_cate_id

  join Price_Package pp on r.package_id = pp.package_id join [User] u on u.user_id = r.user_id

where 1=1 and (c.course_name like '%k%' or s.sub_name like '%k%' or sc.subject_cate_name like '%k%' or u.user_email like '%k%')

- select count(*)

  from Registration r

  join Course c on r.course_id = c.course_id

  join Subject s on c.sub_id = s.sub_id

  join Subject_Category sc on s.subject_cate_id = sc.subject_cate_id

  join Price_Package pp on r.package_id = pp.package_id

  where 1=1

  order by c.course_price*pp.multiple desc

  offset 0 rows

  fetch next 10 rows only

- select r.reg_id, r.reg_time, r.reg_note, r. course_id, r.user_id, r.reg_status_id, r.package_id,r.last_updated_by

  from Registration r

  join Course c on r.course_id = c.course_id

  join Subject s on c.sub_id = s.sub_id

  join Subject_Category sc on s.subject_cate_id = sc.subject_cate_id

  join Price_Package pp on r.package_id = pp.package_id

  join [User] u on u.user_id = r.user_id

  where r.user_id = 8 and (c.course_name like '%programming%' or s.sub_name like '%programming%' or u.user_email like '%programming%')

  and sc.subject_cate_id in (1)

  and s.sub_id in (1)

and pp.package_id in (1,2)

and ((c.course_price*pp.multiple) >= 0 and (c.course_price*pp.multiple) <= 100)

and r.reg_status_id in (1)

order by c.course_price*pp.multiple desc

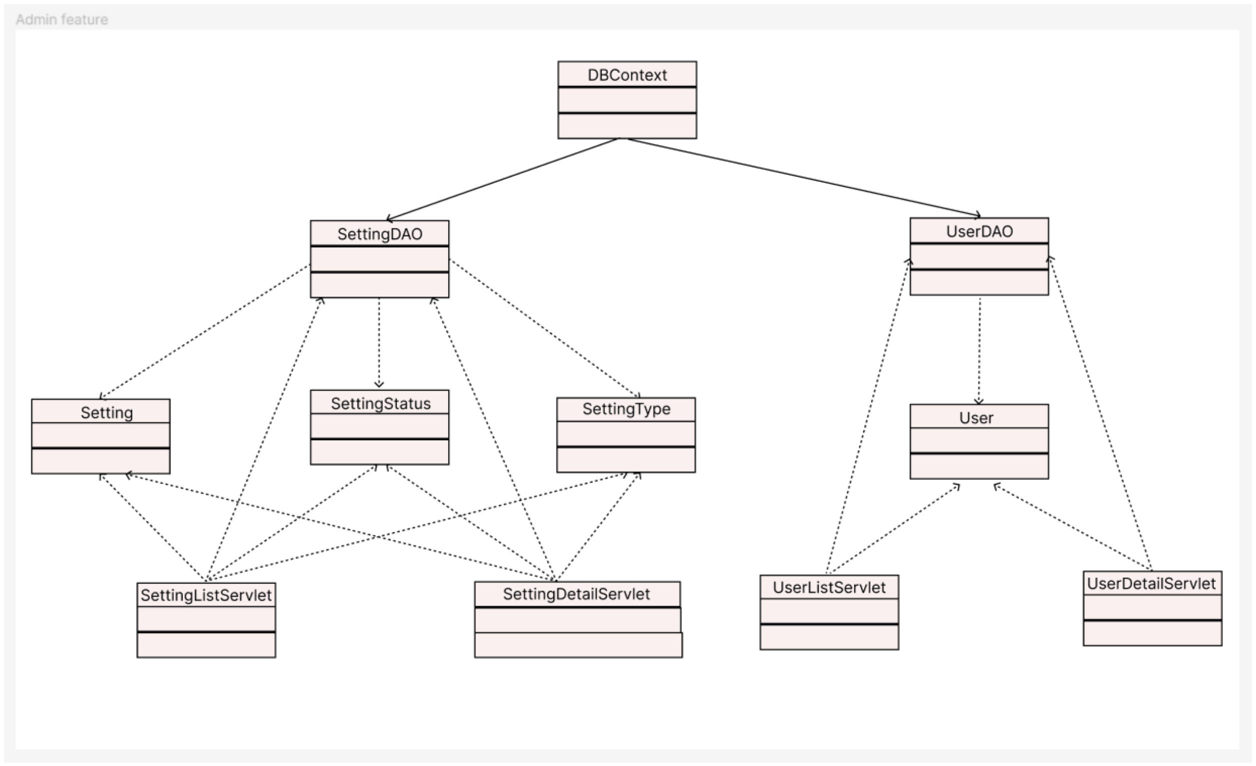
offset 0 rows

fetch next 10 rows only

- select * from Registration where user_id = ? and course_id = ?
- insert into Registration values(?,?,?, ?, ?, ?, null)
- select * from Registration where reg_id = ?
- select * from Registration where user_id = ?
- select c.course_price from Registration r join Course c on r.course_id = c.course_id where r.reg_id = ?
- select p.duration from Registration r join Price_Package p on r.package_id = p.package_id where r.reg_id = ?
- select DATEADD(MONTH,?,?) from Registration r where r.reg_id = ?
- update Registration set reg_status_id = ?, reg_note = ? where reg_id = ?

8 <Admin Feature>

a. Class Diagram



b. Class Specifications

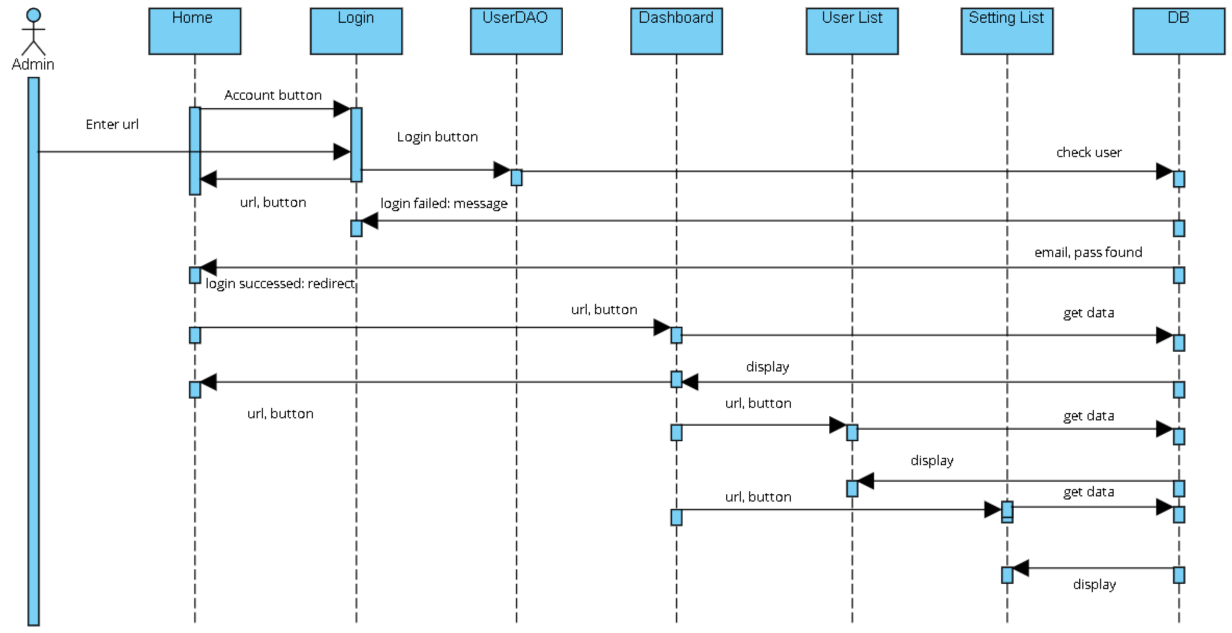
- UserDAO

No	Method	Description
1	getListUserAR(int gid, int rid, int ustatus, String ufullname, String uemail, String uphone, int sort, int page)	Get all user’s information with condition
2	getUserByUid(int uId)	Get user by user id
3	updateUserAR(User u, int uid)	Update user
4	insertUserAR(User u)	Add new user

- SettingDAO class

No	Method	Description
1	getAllSettingType()	list all setting
2	getAllSettingStatus()	get all setting status
3	countTotalOfSetting()	cout total setting
4	getSetting(String search, String sortBy, String[] typeId, String[] statusId, int page)	get list setting with condition
5	getListSettingTypeBySettingList(List<Setting> list)	get List SettingType with condition
6	getListSettingStatusBySettingList(List<Setting> list)	get list SettingStatus with condition
7	getSettingById(int id)	get setting by setting id
8	getSettingTypeBySettingId(int sid)	get SettingType by setting type id
9	getSettingStatusBySettingId(int sid)	get SettingStatus by setting status id

c. Sequence Diagram(s)

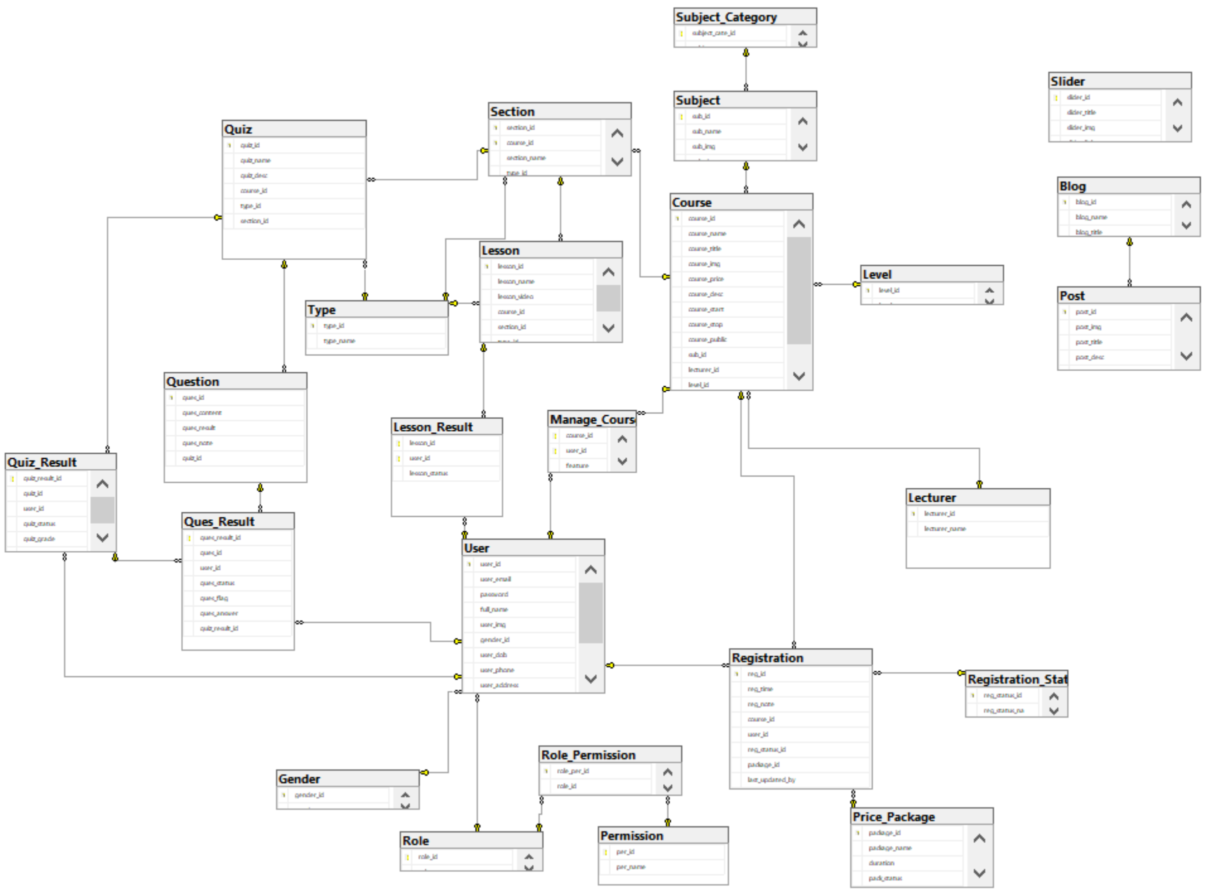


d. Database Queries

- select * from [User] where user_id != 0 and gender_id = ? and role_id = ? and user_status = ? and full_name like ? and user_email like ? and user_phone like ? order by user_id offset ? rows FETCH NEXT 5 ROWS ONLY
- select * from [User] where user_id= ?
- update [User] set user_email = ?, full_name = ?, user_img = ?, gender_id = ? , user_dob = ?, user_phone = ?, user_address = ?, user_wallet= ?, role_id = ?, user_status = ? where user_id = ?
- insert [User] (user_email, password, full_name, user_img, gender_id, user_dob, user_phone, user_address, user_wallet, role_id, user_status) values (?,?,?,?,?,?,?,?,?,?,?)
- select * from SettingType
- select * from SettingStatus select COUNT(*) from Setting where 1=1 and (SettingOrder like ? or SettingValue like ? and SettingTypeId in ? and SettingStatusId in ?
- select s.SettingId, s.SettingTypeId, s.SettingValue, s.SettingOrder, s.SettingStatusId, s.SettingDescription from Setting s join SettingType st on s.SettingTypeId = st.SettingTypeId join SettingStatus ss on s.SettingStatusId = ss.SettingStatusId where 1=1
- select ss.SettingStatusId, ss.SettingStatusName from SettingStatus ss join Setting s on ss.SettingStatusId = s.SettingStatusId where s.SettingId = ?
- select * from Setting where SettingId = ?
- select st.SettingTypeId, st.SettingTypeName from SettingType st join Setting s on st.SettingTypeId = s.SettingTypeId where s.SettingId = ?
- select ss.SettingStatusId, ss.SettingStatusName from SettingStatus ss join Setting s on ss.SettingStatusId = s.SettingStatusId s.SettingId = ?
-

III. Database Design

1. Database Schema



2. Table Description

No	Table	Description
01	*Blog*	*- Primary keys: PK_Blog : blog_id*
02	*Course*	*- Primary keys: PK_Course : course_id* *- Foreign keys:*

		FK_Course_Lecturer: lecturer_id - lecture_id (Lecturer) *FK_Course_Level: level_id - level_id (Level)* *FK_Course_Subject: sub_id - sub_id(Subject)*
03	*Gender*	*- Primary keys: PK_Gender : gender_id*
04	*Lecturer*	*- Primary keys: PK_Lecturer : lecturer_id*
05	*Lesson*	*- Primary keys: PK_Lesson : lesson_id* *- Foreign keys:* *FK_Lesson_Section: section_id - section_id (Section)* *FK_Lesson_Type: type_id - type_id (Type)*
06	*Lesson_Result*	*- Primary keys: PK_Lesson_Result: lesson_id,user_id* *- Foreign keys:* *FK_Lesson_Result_Lesson: lesson_id - lesson_id(Lesson)* *FK_Lesson_Result_User: user_id - user_id(User)*
07	*Level*	*- Primary keys: PK_Level : level_id*
08	*Permission*	*- Primary keys: PK_Permission: per_id*
09	*Post*	*- Primary keys: PK_Post: post_id* *- Foreign keys:* *FK_Post_Blog : blog_id - blog_id (Blog)*
10	*Price_Package*	*- Primary keys:PK_Price_Package*
11	*Question_Result*	*- Primary keys: PK_Question_Result*

		- Foreign keys: *FK_Ques_Result_Question: ques_id - ques_id (Question)* *FK_Ques_Result_Quiz_Result: quiz_result_id - quiz_result_id(Quiz_Result)* *FK_Ques_Result_User: user_id - user_id(User)*
12	*Question*	*- Primary keys: PK_Question: ques_id* *- Foreign keys:* *FK_Question_Quiz: quiz_id - quiz_id (Quiz)*
13	*Quiz*	*- Primary keys: PK_Quiz: quiz_id* *- Foreign keys:* *FK_Quiz_Section: section_id - section_id(Section)* *FK_Quiz_Type: type_id - typr_id(Type)*
14	*Quiz_Result*	*- Primary keys: Quiz_Result: quiz_result_id* *- Foreign keys:* *FK_Quiz_Result_Quiz: quiz_id- quiz_id (Quiz)* *FK_Quiz_Result_User: user_id- user_id(User)*
15	*Registration*	*- Primary keys:PK_Registration : registration_id* *- Foreign keys:* *FK_Registration_Course: course_id - course_id(Course)* *FK_Registration_Price_Package: package_id - package_id (Package)* *FK_Registration_Registration_Status: reg_status_id - reg_status_id(Registration_Status)* *FK_Registration_User: user_id - user_id(User)*

16	*Registration_Status*	*- Primary keys: PK_Registration: reg_status_id*
17	*Role*	*- Primary keys:PK_Role: role_id*
18	*Role_Permission*	*- Primary keys:PK_Role_Permission: role_permission_id* *- Foreign keys:* *FK_Role_Permission_Permission: per_id - per_id (Permission)* *FK_Role_Permission_Role: role_id - role_id (Role)*
19	*Section*	*- Primary keys:PK_Section: section_id* *- Foreign keys:* *FK_Section_Course: course_id - course_id(Course)* *FK_Section_Type: type_id - type_id (Type)*
20	*Slider*	*- Primary keys:PK_Slider: slider_id*
21	*Subject*	*- Primary keys:PK_Subject: sub_id* *- Foreign key:* *FK_Subject_Subject_Category: subject_cate_id - subject_cate_id(Subject_category)*
22	*Type*	*- Primary keys:PK_Type: type_id*
23	*User*	*- Primary keys:PK_User: user_id* *- Foreign keys: gender_id, role_id* *FK_User_Gender: gender_id - gender_id(Gender)*
24	*Subject_category*	*- Primary key: subject_cate_id*

25	*Manage_Course*	*- Pirmary key: course_id, user_id* *- Foreign keys:* *course_id, user_id* *FK_Course_ManageCourse: course_id - course_id(Course)* *FK_User_ManageCourse: user_id - user_id(User)*