

**LEARNING HUB**

**Software Design Document**

– Hanoi, May 2023 –

# 

[**I. Overview 3**](#_f66gpq7m8pgx)

[A. Code Packages/Namespaces 3](#_5es3e2x8438c)

[1. Server-side 3](#_2ezyav6femt2)

[2. Client-side 4](#_l6dt2dcw5n18)

[B. Coding conventions 4](#_ezii3asmyiu7)

[1. File name 4](#_85pmljragdda)

[2. File organization 5](#_lw70ltem0ijp)

[3. Comment 6](#_6ir78ptag8cg)

[4. Indentation 6](#_3cil834q45k8)

[5. Declaration 7](#_xefq9qu7p833)

[6. Statement 7](#_ofsj6zn3zqw)

[7. Blank lines 8](#_z1m8sa174e91)

[8. Naming conventions 8](#_wq074w999mae)

[**II. Code Design 9**](#_34vbpbq2qrga)

[1. Common/Login 9](#_rwnb0vn8q8qb)

[2. Common/Logout 10](#_usf4e91q7elc)

[3. Common/Register 11](#_c1nvw5sidbgh)

[4. Deactivate Account 13](#_1c4mndv23jd0)

[5. Show user profile 15](#_cm83ccock0y2)

[6. Show user list 16](#_bfv54quvdvjz)

[7. Update user profile 17](#_e4jb7scami1u)

[8. Common/Feature management 19](#_3d9yrha964if)

[9. Common/Change Password 21](#_ke27k43y8gqt)

[10. Common/Forget Password 22](#_ffok7na6ghd9)

[11. View/Add/Update/Archive Flashcard Set 25](#_m9kzecm4r5wh)

[12. Flashcard/Learn Flashcard 27](#_p7nto49xigy6)

[13. View/Add/Update/Delete Flashcard(inside set) 28](#_7da3clkyrvp9)

[14. Task Management/View Kanban board 30](#_187q719wjzhh)

[15. Pomodoro/Countdown Timer by template 32](#_tz8orifiv299)

[16. Pomodoro/Screen appearance customization 32](#_tjphhtcbxnaw)

[17. Pomodoro/Session Sound 33](#_2pcunhi40rh1)

[18. Pomodoro/Input tasks for sessions 33](#_3wybzzup4knz)

[**III. Database Design 35**](#_jjgs5vfwhtwk)

[A. Database Schema 35](#_tlkx7kdhwo7a)

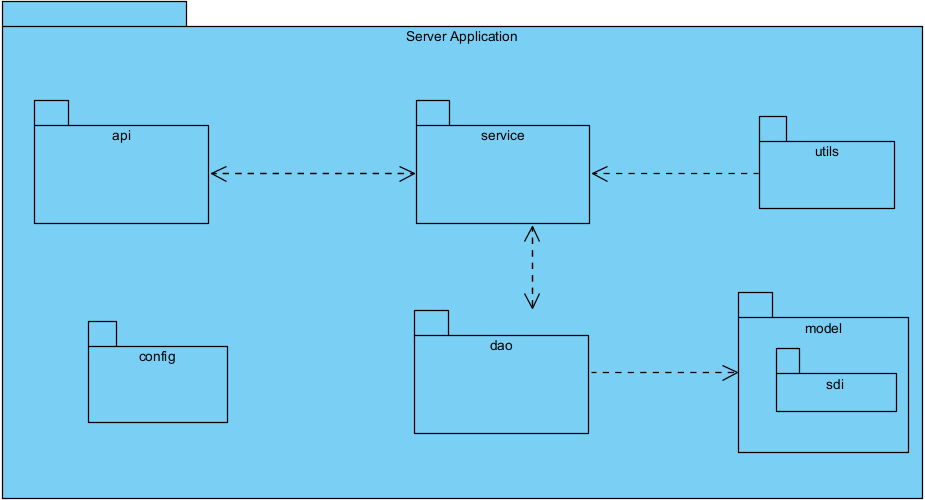
[B. Table Description 35](#_8ja07u8tayx5)

# 

# Overview

## Code Packages/Namespaces

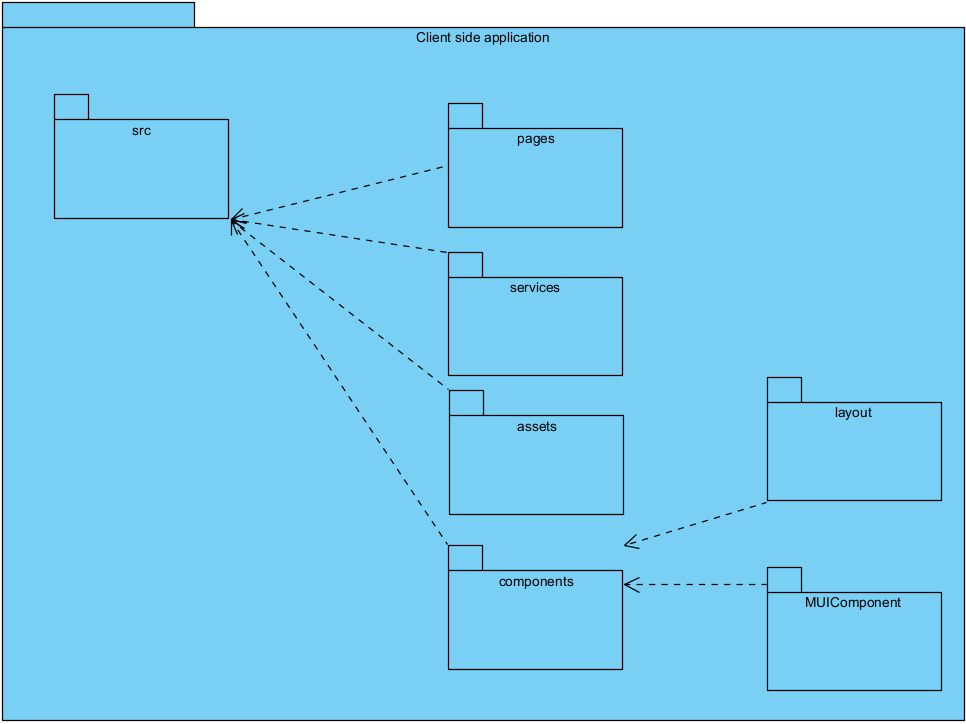
### Server-side



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

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| 01 | api | This package contains all APIs which receive a request from the client side and returns results to the client side. Files in this package are called the Controller file (name convention: object name+Controller). |
| 02 | dao | This package contains DAO files (name convention: object name + DAO). these files will handle the data from the database and send it to the service file for business processing. these files are Java interfaces. |
| 03 | model | This package contains all the object files that the system involves (name convention: object name) |
| 04 | service | This package will handle all the business logic of the system and bring the system's services to the controller file (API package). |
| 05 | config | This package contains a configuration file that customized the server behaviors and settings. |
| 06 | util | Contains minor functions, and constants for supporting service. |
| 07 | sdi | A sub-package from the model package. This package contains models helping servers connect to Gmail services |

### Client-side



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

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| 01 | src | This package contains most of the code files and components |
| 02 | pages | This package contains React components for each screen (login, logout, etc.) |
| 03 | services | This package contains all API calls from server-side |
| 04 | assets | This package contains components that are customized specifically for the application |
| 05 | components | This package contains all init components for the application such as button, link, etc. |
| 06 | layout | This package contains components that define the layout of each screen such as header, footer |
| 07 | MUIComponent | This package contains components that are customized from Material UI components |

## Coding conventions

### File name

* + - 1. File suffixes

For java source, the suffix of the file is .java and for java bytecode file, the suffix will be .class

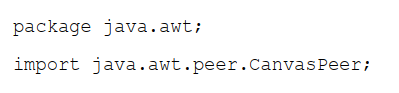
* + - 1. Common file name:
* README: the preferred name for that file that summarizes the contents of the project
  + - 1. Spring boot maven build file
* Application properties file: Application Properties support us to work in different environments. Properties files are used to keep ‘N’ number of properties in a single file to run the application in a different environment. In Spring Boot, properties are kept in the application.properties file under the classpath.
* Pom.xml: A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.
* Application.yml: As well as Java properties files, we also use YAML-based configuration files in our Spring Boot application. YAML is a convenient format for specifying hierarchical configuration data. In this application, we use yaml file to config the setting for mailing service.

### File organization

* + - 1. Java source files contain a single public class or interface.

Each Java source file contains.

* + - 1. Java source file have the following ordering
* Package and import statements



* Class and interface declarations: the following table describes the part of a class or interface declaration, in the order that they should appear

|  |  |  |
| --- | --- | --- |
| **No** | **Part of Class/interface declaration** | **Notes** |
| 1 | Class or interface statement |  |
| 2 | Class (static) variable | First the public class variables, then the protected and then the private |
| 3 | Instance variables | First public, then protected, and then private |
| 4 | Constructor |  |
| 5 | Method | These methods grouped by functionality |

### Comment

* + - 1. **For single-line comment (/\*\*/):** is used to explain the codes down below
      2. **For trailing comment (/\*\*/):** is used to explain a specific line of code
      3. **For end-of-line comment (//):** is used for commenting out the sections of code
      4. **Documentation comments (/\*\*\*/):** is used to describe Java classes, interface, constructors, methods and fields. This comment should also describe the API functions. Documentation comments should appear just before the declaration.

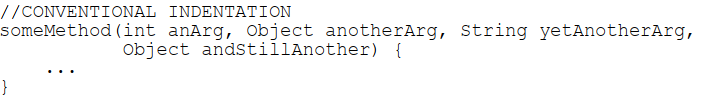
### Indentation

* + - 1. **Line length**: avoid lines longer than 100 characters
      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 as the previous line.

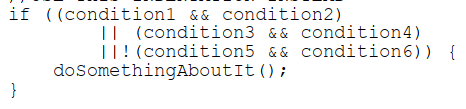
**For object/method declaration:**



**For arithmetic operators:**



**For if statements:**

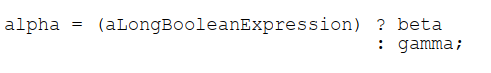


**For ternary expressions:**

* **If the expression is short:**



* **If the expression is long:**



### Declaration

* + - 1. One declaration per line. Encourage commenting
      2. Put the declaration only at the beginning of the block (except for loop statements)
      3. For 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 “{“

* Methods are separated by a blank line

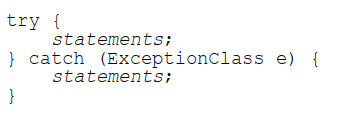
### Statement

* + - 1. Each line should contain at most one statement only.
      2. For compound statements:
* 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 an if-else or for statement.
  + - 1. Return statements with a value should not use parentheses unless they make the return value more obvious in some way.
      2. If else state always use braces {}.
      3. For switch statements: every case in the switch-case statement must include the break statement. the switch statement is encouraging to include a default case.
      4. For try-catch statements should have the following format:



### Blank lines

Two blank lines should always be used in the following circumstances:

* Between sections of a source file
* Between class and interface definitions

One blank line should always be used in the following circumstances:

* Between methods
* Between the local variables in a method and its first statement
* Before a block]
* Between logical sections inside a method to improve readability

### Naming conventions

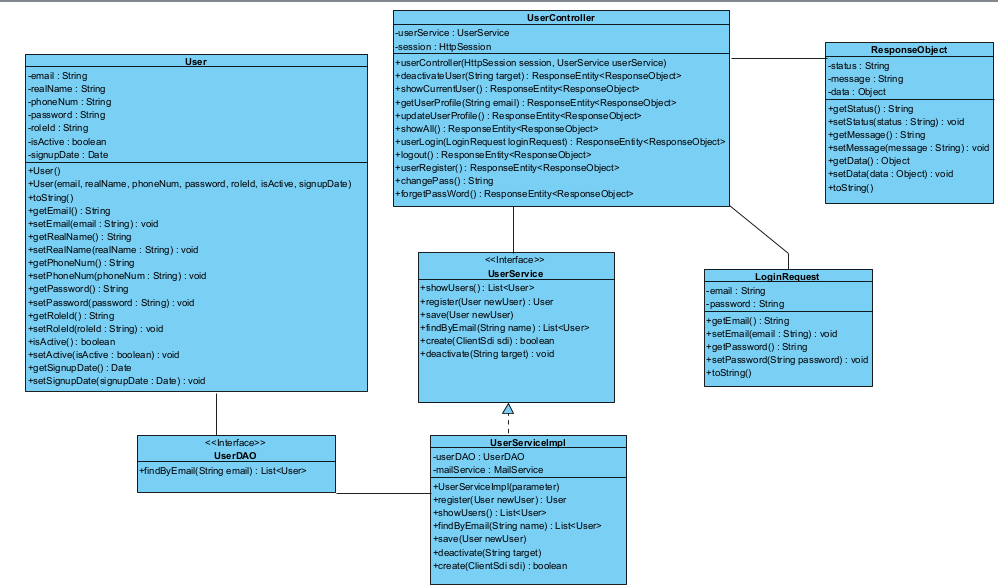
|  |  |  |
| --- | --- | --- |
| **Identifier type** | **Rule for Naming** | **Examples** |
| Classes | Should be nouns. If the class’ name is longer than one word, the first letter of each internal word is capitalized. Try to keep class names simple and descriptive. Avoid acronyms and abbreviations. Avoid naming the class too long. | class Apple  class ApplePie |
| Interfaces | Should be nouns. If the interface’s name is longer than one word, the first letter of each internal word is capitalized. Try to keep interfaces’ names simple and descriptive. Avoid acronyms and abbreviations. Avoid naming the interfaces too long. | interface Apple  interface ApplePie |
| Methods | Should be verbs. in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. | bake();  eatPie(); |
| Variables | The first letter should be in lowercase. In the mixed case with a lowercase first letter. Internal words start with a capital letter. Variable name should be short yet meaningful. One character variable names should be avoided except for temporary “throwaway” variables | int age;  String name; |
| Constants | Should be all uppercase with words separated by the underscores (“\_”) | final int MAX\_LENGTH =100; |

### 

# Code Design

### Common/Login

* + - 1. Class Diagram



* + - 1. Class Specification
         1. UserController

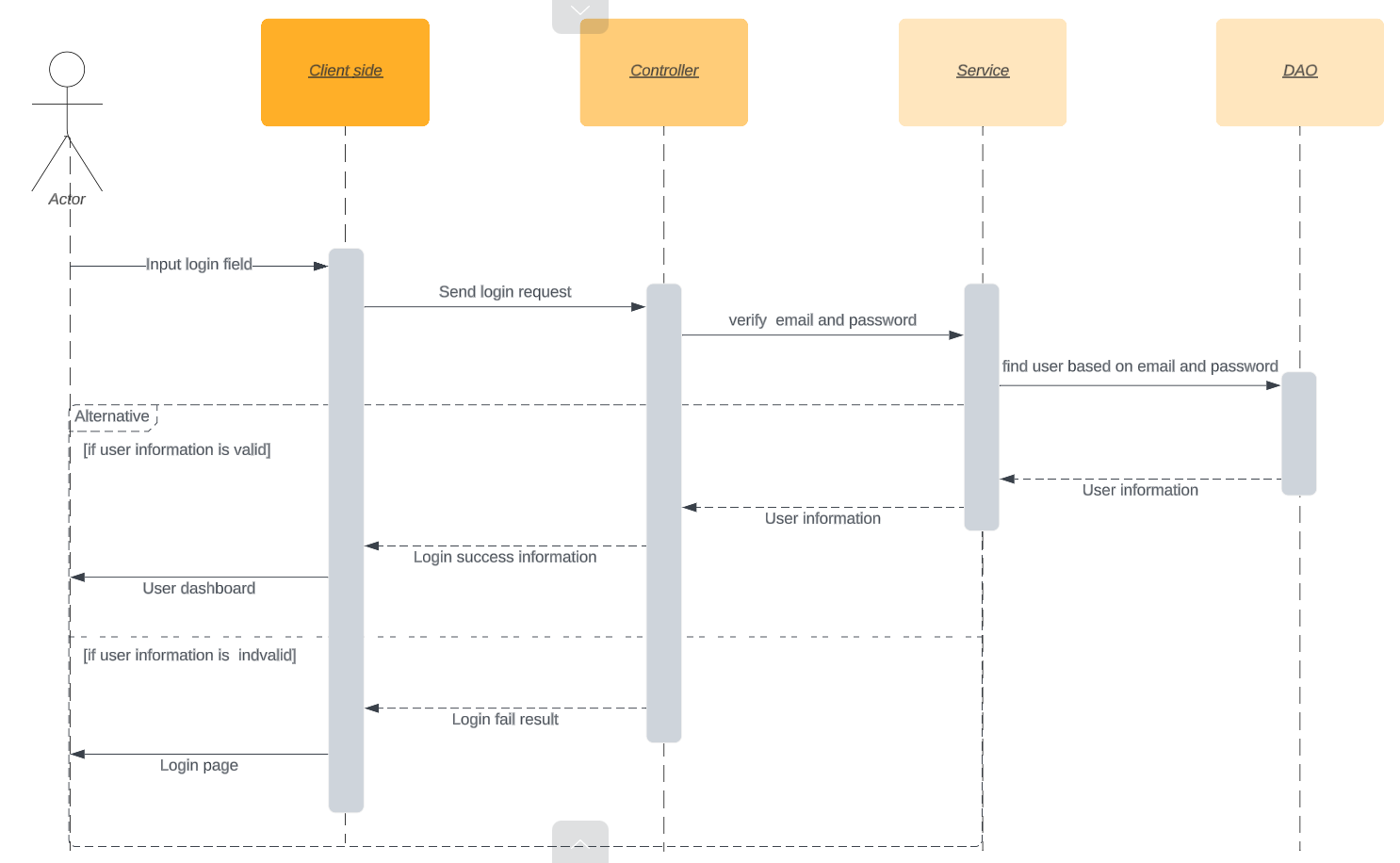
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | userLogin | Check input email and password. Save user information to session |

* + - * 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findByEmail | Find user by email.Return object user |

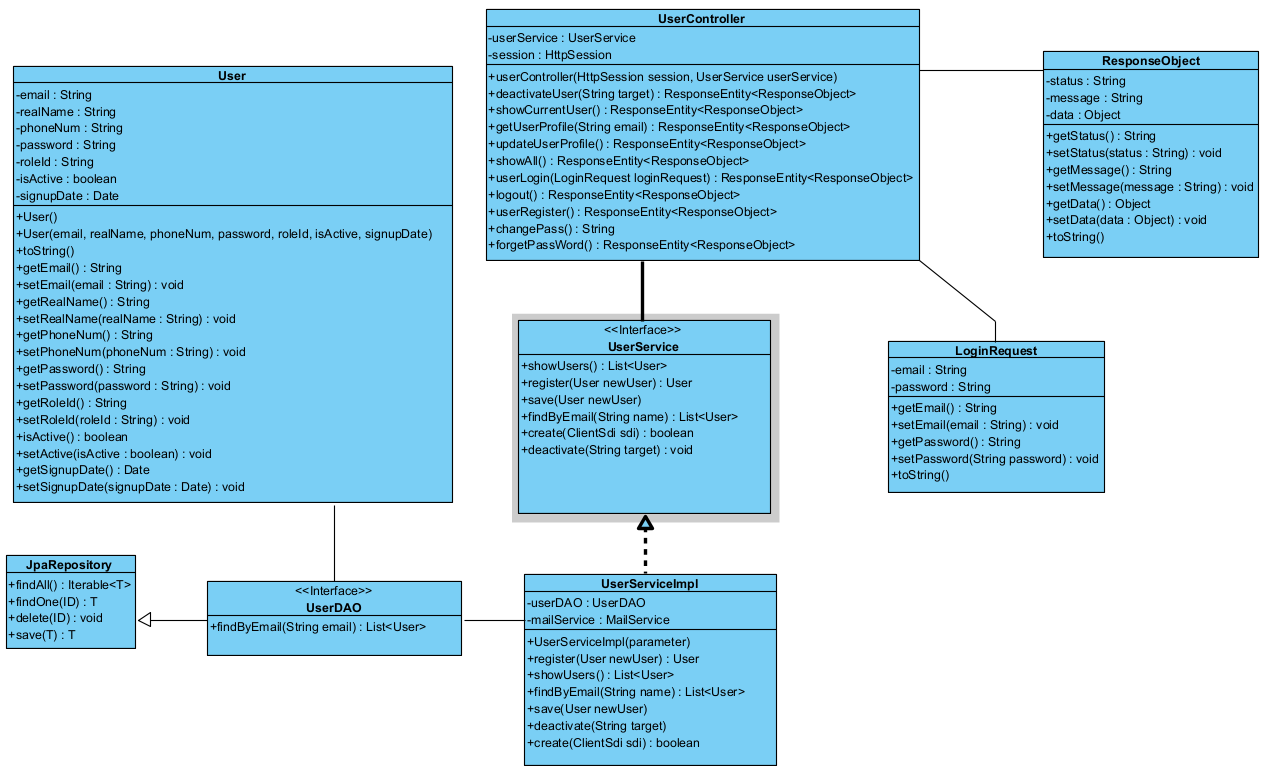
* + - * 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findByEmail | Find user by email in the database |

* + - 1. Sequence Diagram
      2. Components

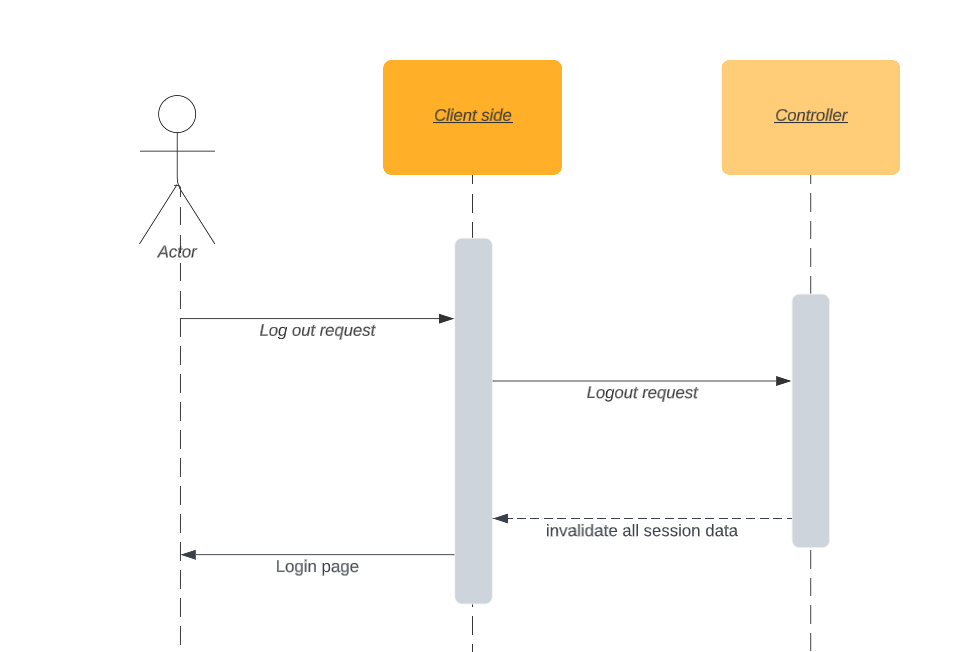
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | userLogin | Check input email and password. Save user information to session |

### Common/Logout

* + - 1. Class Diagram
      2. Class Specification
         1. UserController

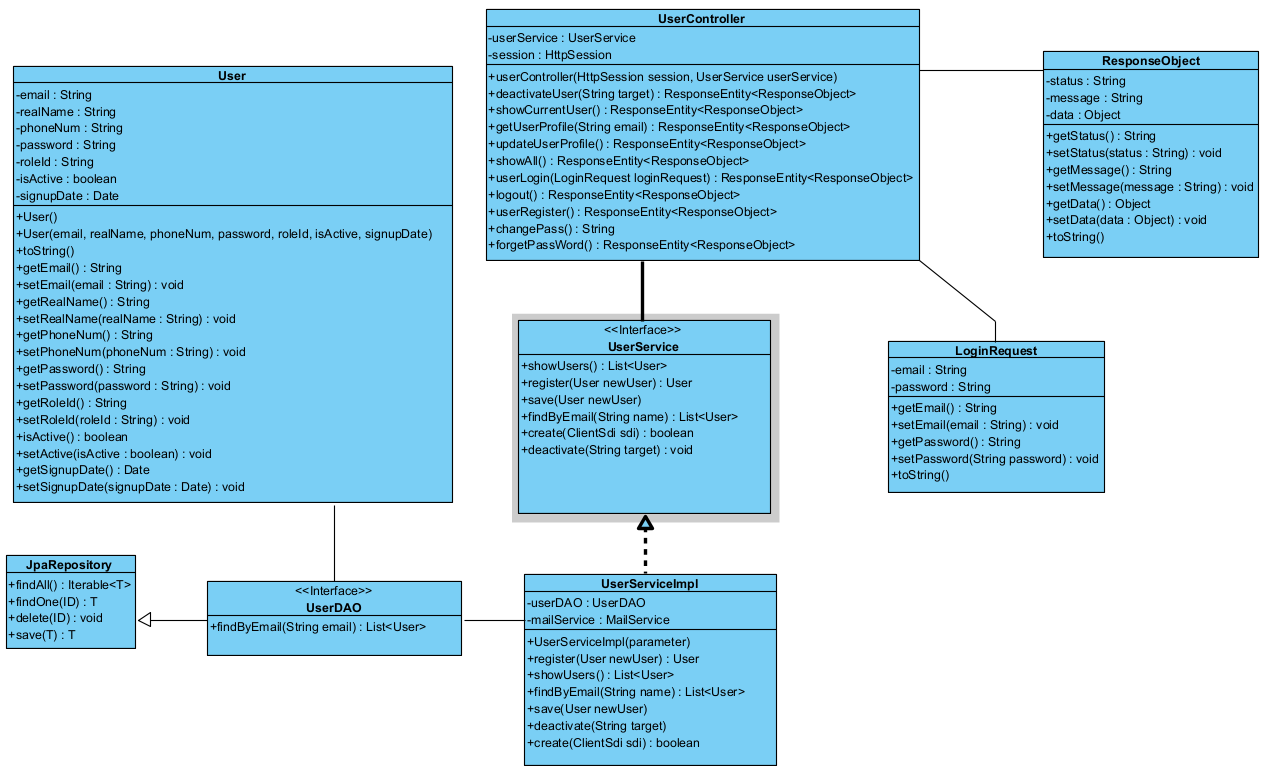
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | userLogout | Logout user from the system. Delete all saved information in the session. |

* + - 1. Sequence Diagram



### Common/Register

* + - 1. Class diagram



* + - 1. Class Specification
         1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | userRegister | Register a new account to the system |

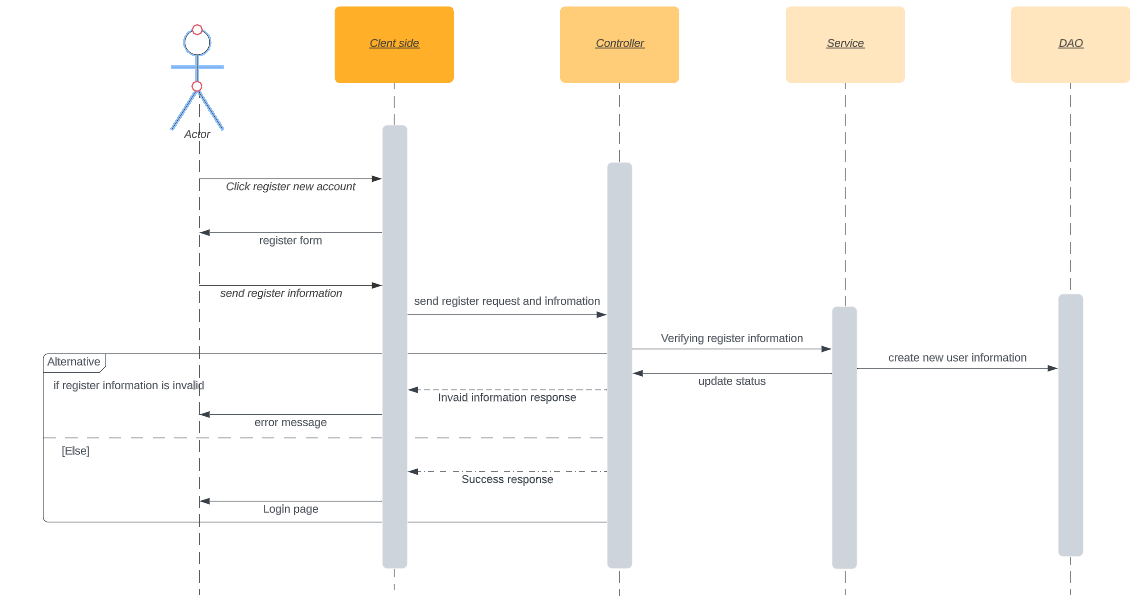
* + - * 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | register | Check input information of account |

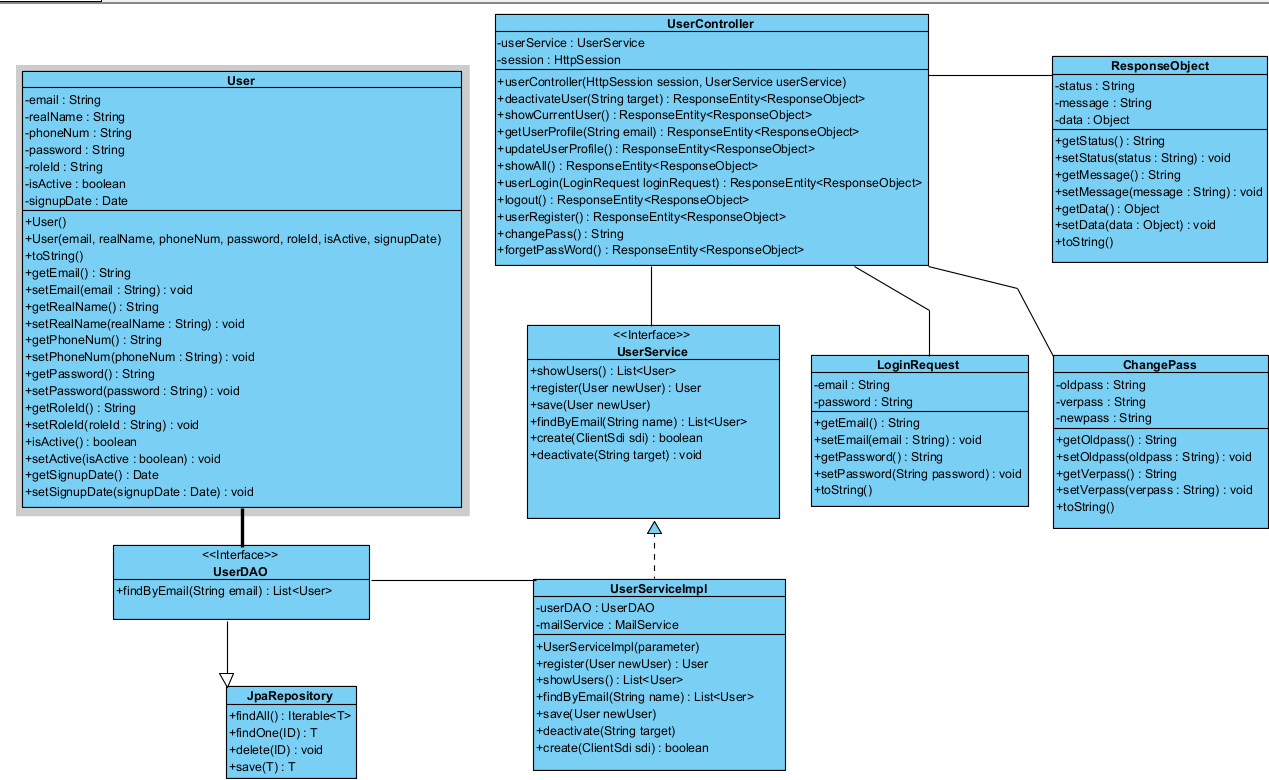
* + - * 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Save a new account in the database |

* + - 1. Sequence Diagram



### Deactivate Account

1. Class diagram
   * + 1. Class Specification
          1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | deactiveUser | Check if the current user has user information in the session.Check if the current user has the authority to perform this action.Deactivate the user with the email |

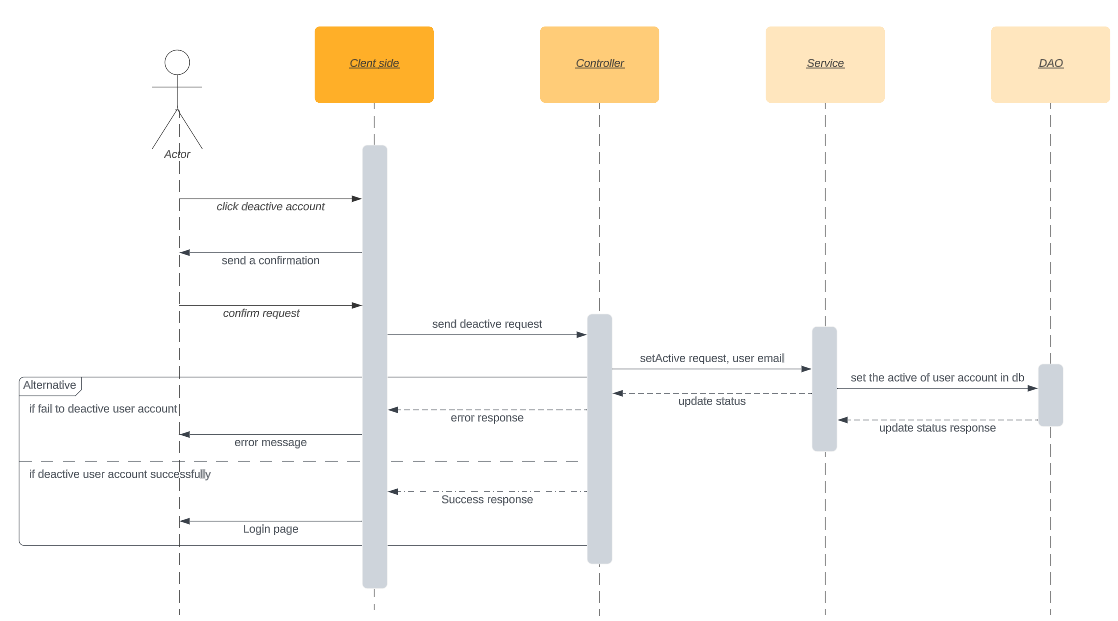
* + - * 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | deactivate | Deactivate the user with the email |

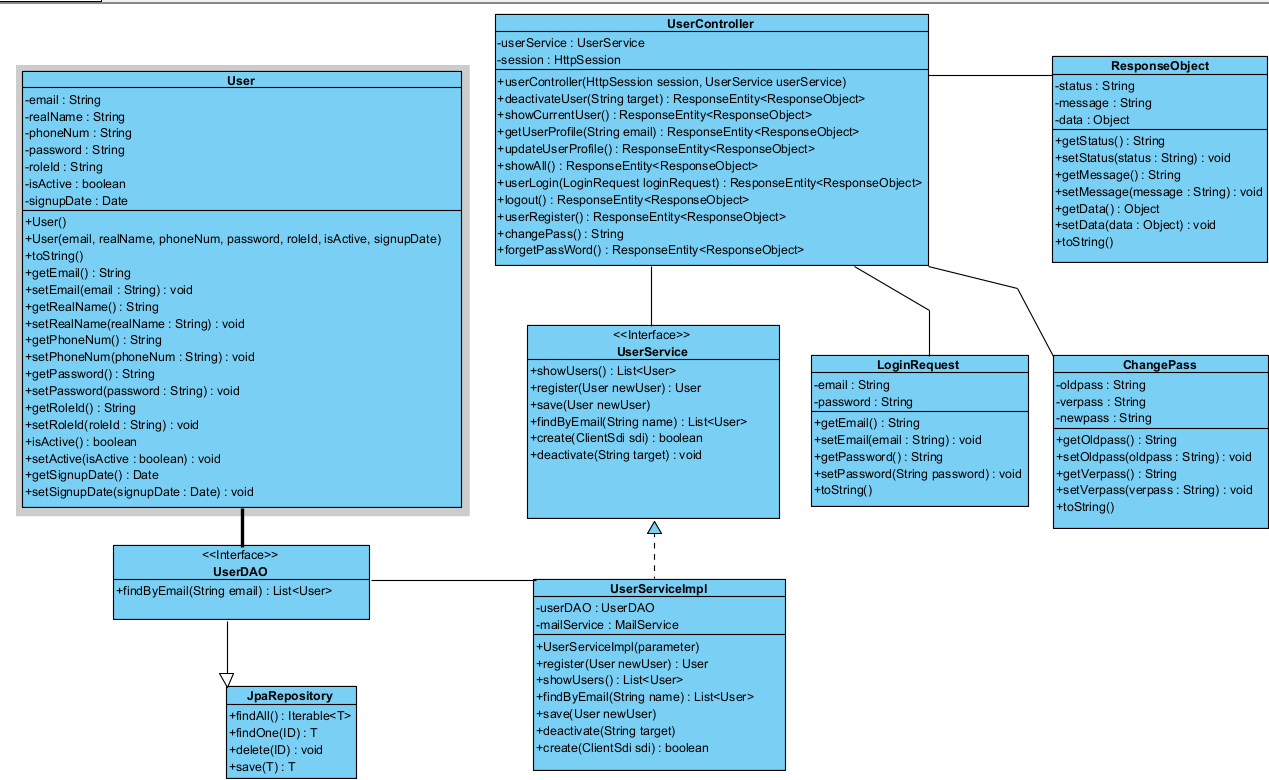
* + - * 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findById | Find user by email in the database |

1. Sequence diagram



### Show user profile

1. Class diagram
2. Class Specification
   1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | getUserProfile | Check if the current user has user information in the session.Check if the current user has the authority to perform this action.Retrieve a list of users with the specified email. |

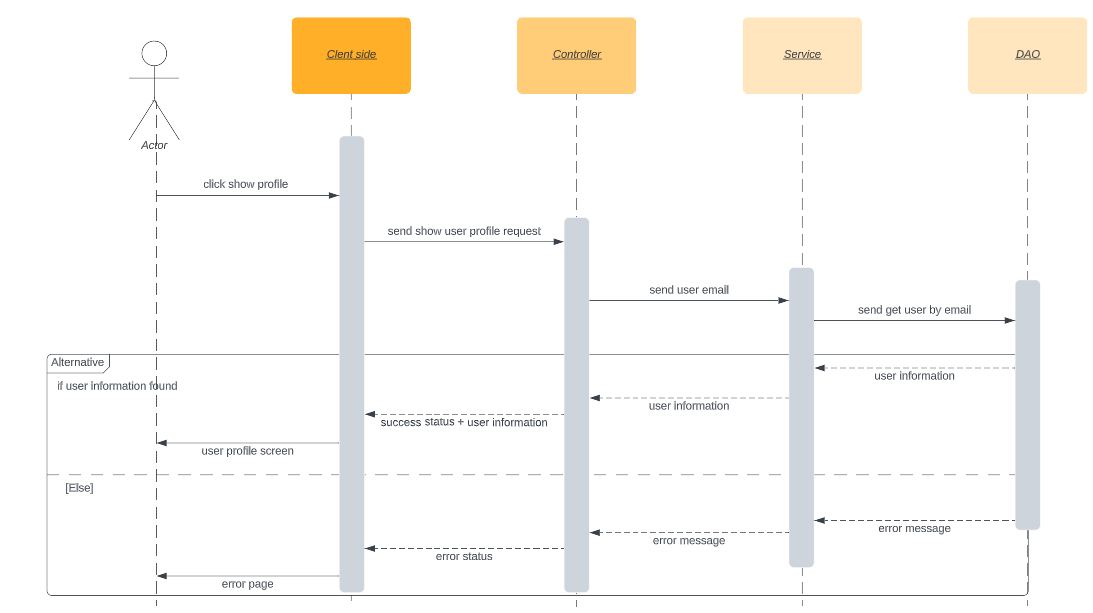
* 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findByEmail | Find user by email.Return object user |

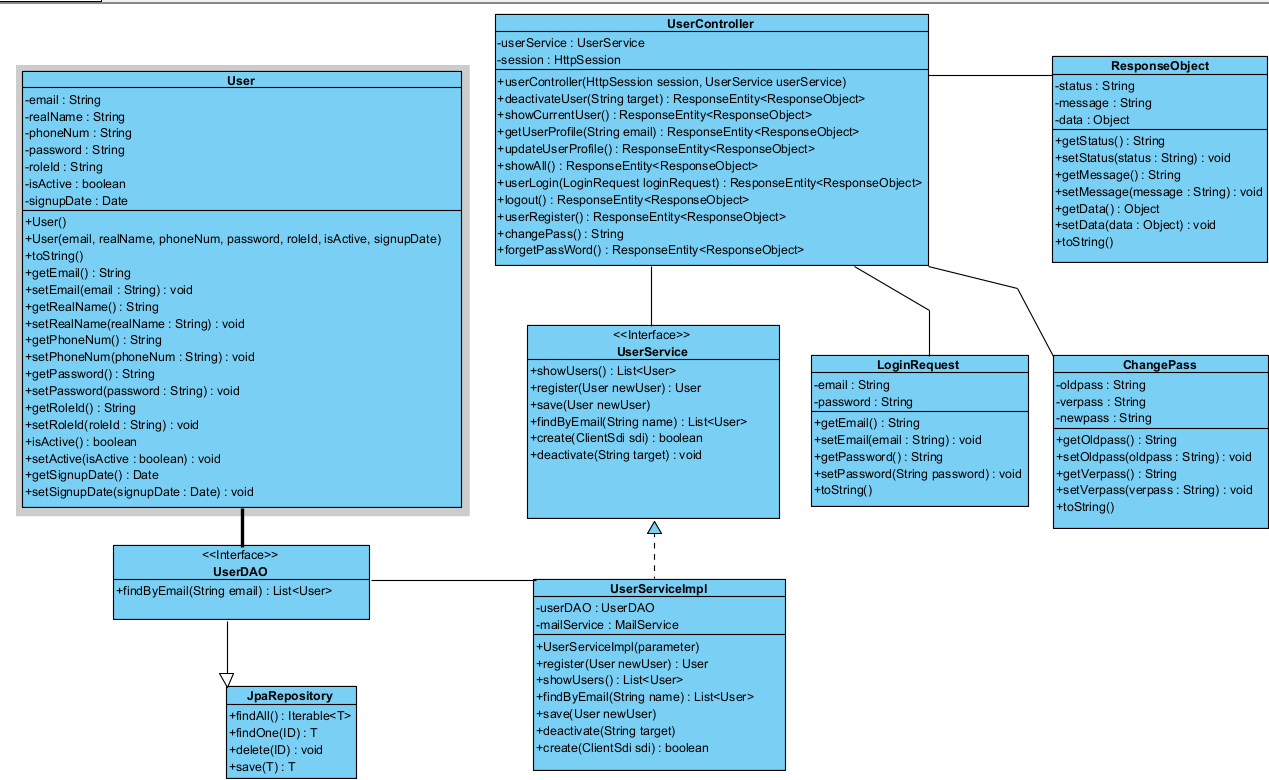
* 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findByEmail | Find user by email in the database |

1. Sequence diagram



### Show user list

1. Class diagram
2. Class Specification
   1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showAll | Check if the current user has user information in the session.Check if the current user has the authority to perform this action.Show list user |

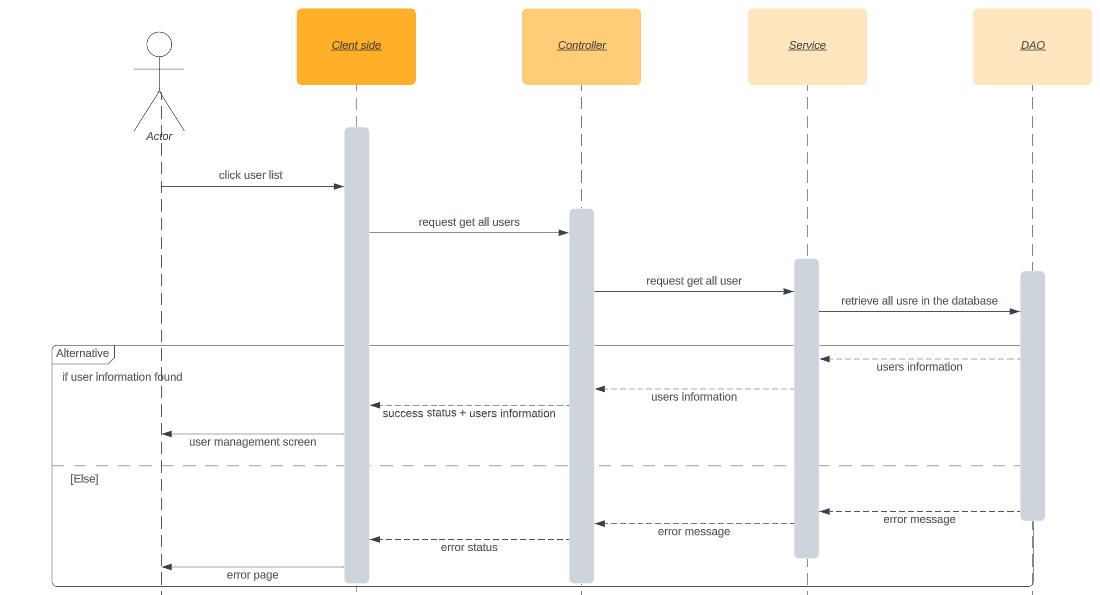
* 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showUsers | Show list user.Return list user |

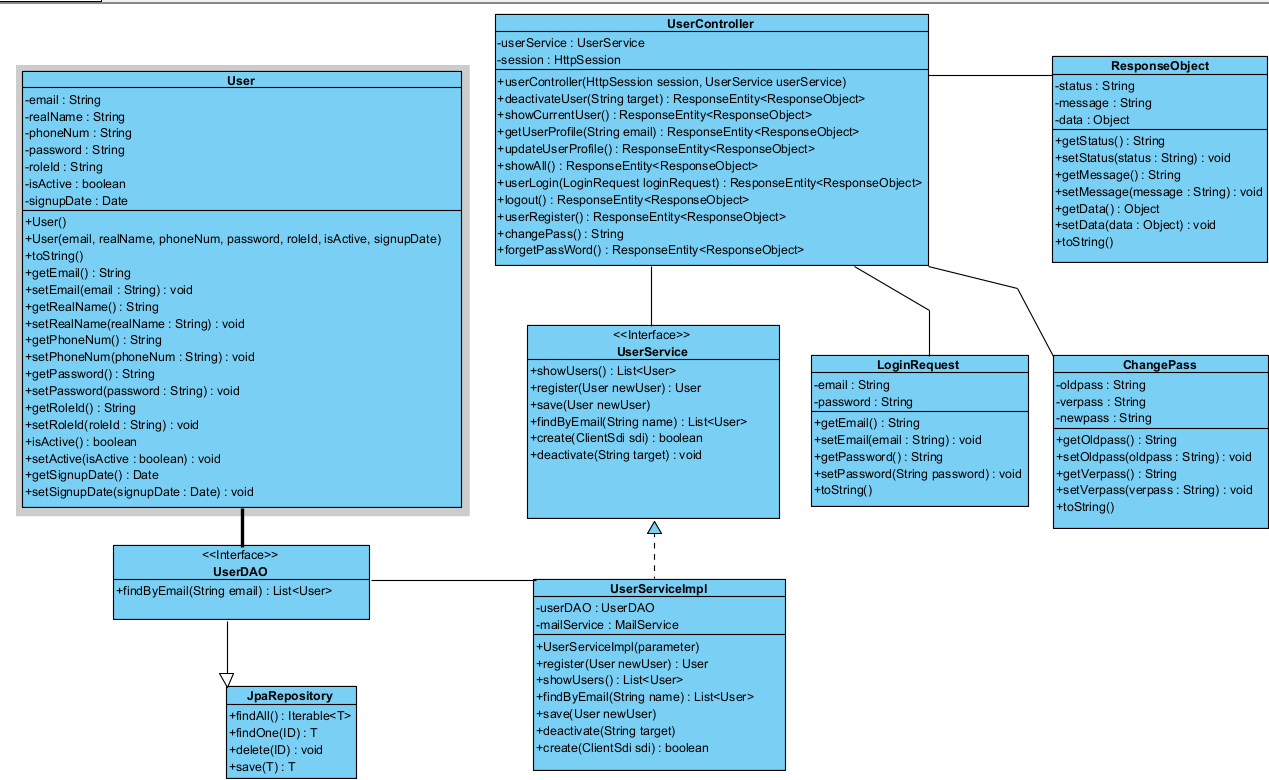
* 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findAll | List all user in the database |

1. Sequece diagram



### Update user profile

1. Class diagram
2. Class Specification
   1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | updateUserProfile | Check if the current user has user information in the session.Check if the current user has the authority to perform this action..Update user |

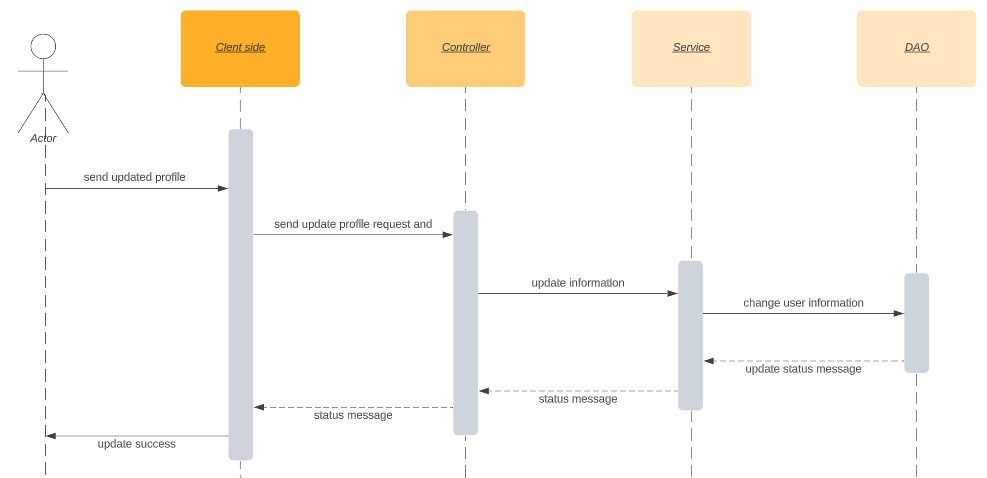
* 1. UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Update user |

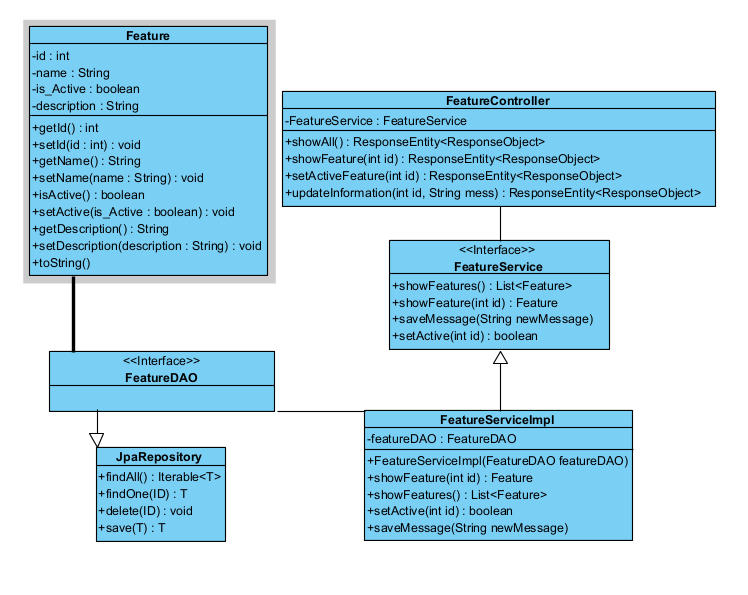
* 1. UserDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Update user in the database |

1. Sequence diagram



### Common/Feature management

1. Class diagram
2. Class specification
   1. FeatureController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showAll | Show all features into the screen |
| 02 | setActive | Changing the "active" status of a feature based on the role of the current user and returns the result of the change |

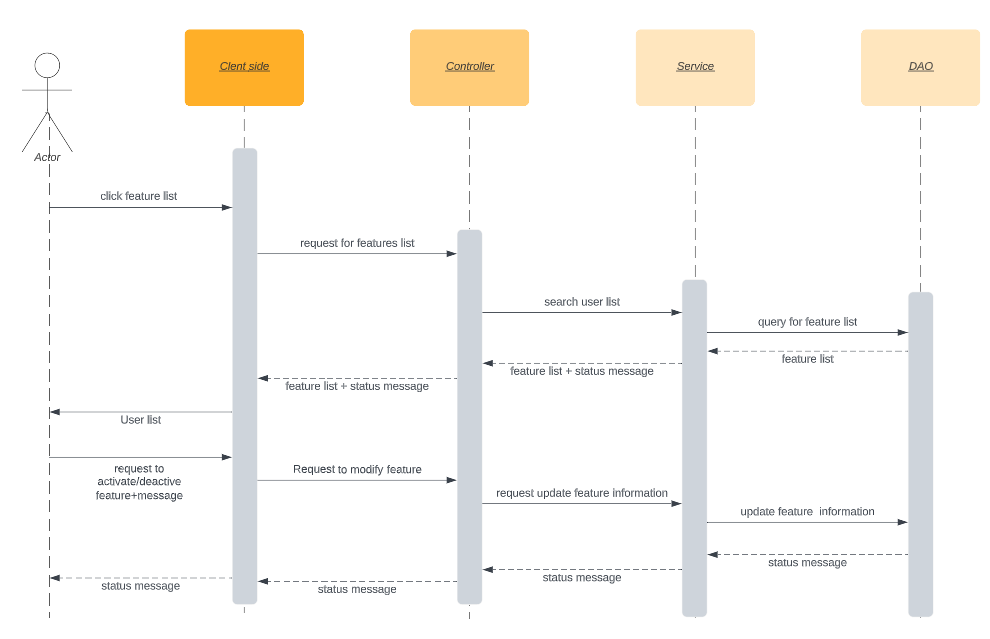
* 1. FeatureService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showAll | Show all features.Return list features |
| 02 | setActive | Changing the "active" status of a feature based on its ID and updating the associated message |

III)FeatureDAO

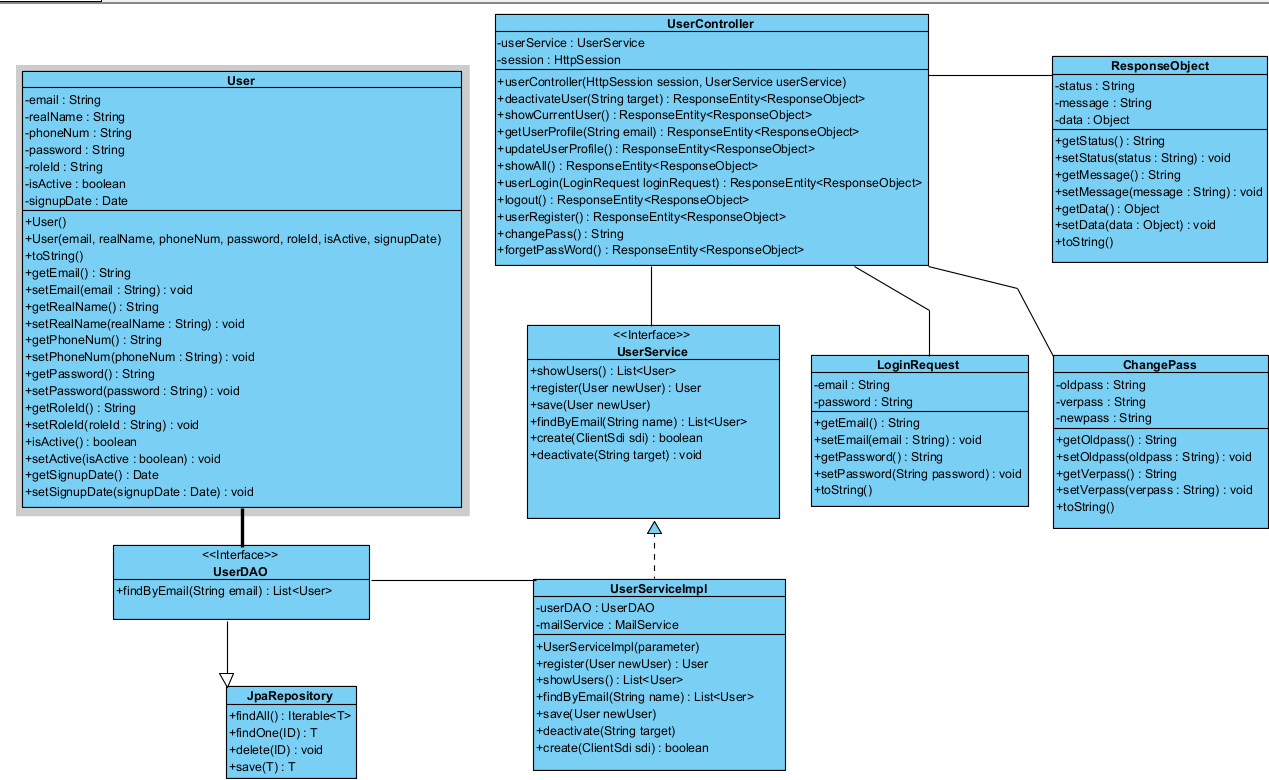
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | findById | Find feature by ID in the database |
| 02 | Save | Update feature in the database |

1. Sequence Diagram



### Common/Change Password

1. Class diagram



1. Class Specification

I) UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | changePass | Input old password to check and new password that user wants to change. The user is also required to confirm new password so this function will need to compare new password to confirm password |

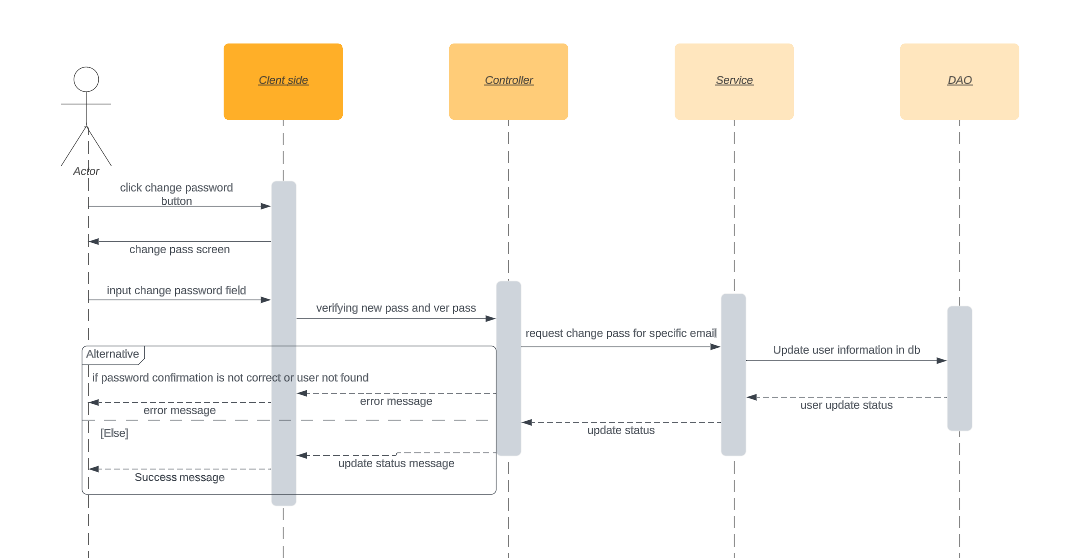
ii)UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Update password user |

Iii)UserDAO

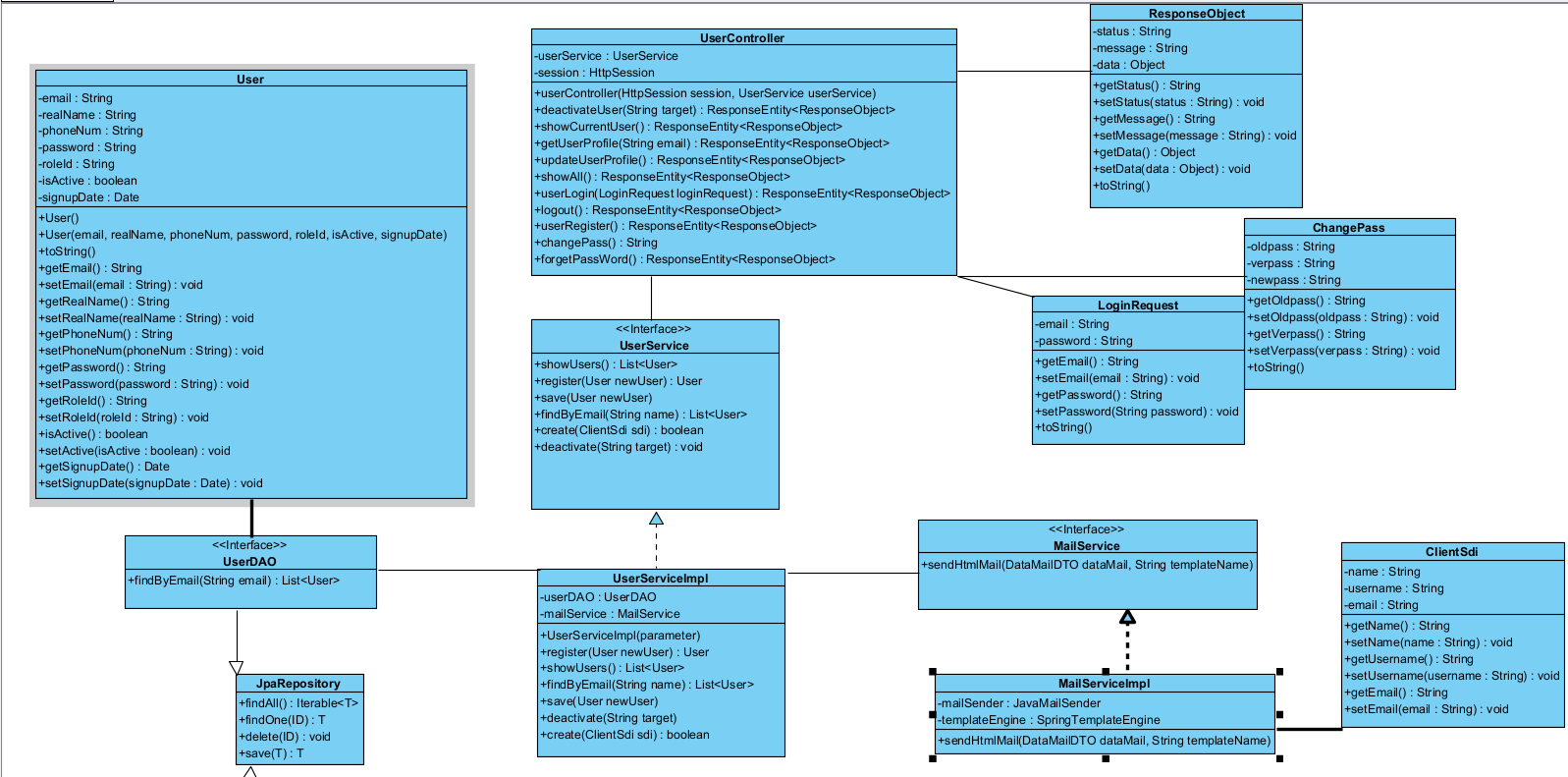
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Update password user in the database |

1. Sequence diagram



### Common/Forget Password

1. Class diagram



1. Class specification
   1. UserController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | forgetPassword | Send an email and receive a new password via email |

ii)UserService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | create | Create a new user password and send an email to the user |

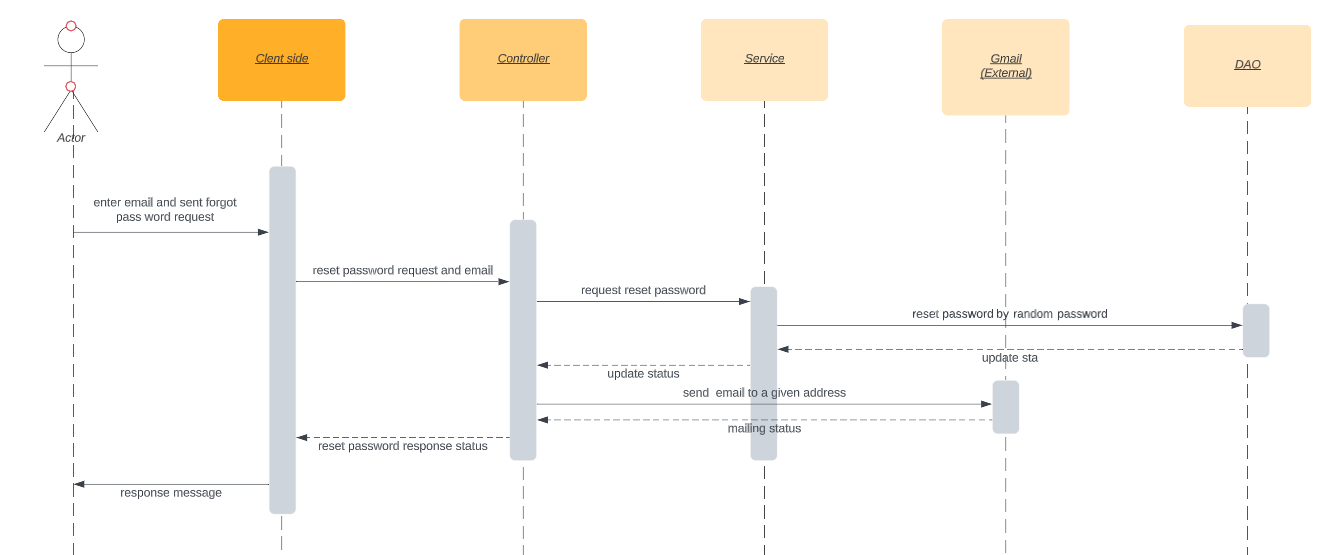
iii)MaiService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | sendHtmlMail | Send an HTML email using the JavaMail API and a template engine |

IV)UserDAO

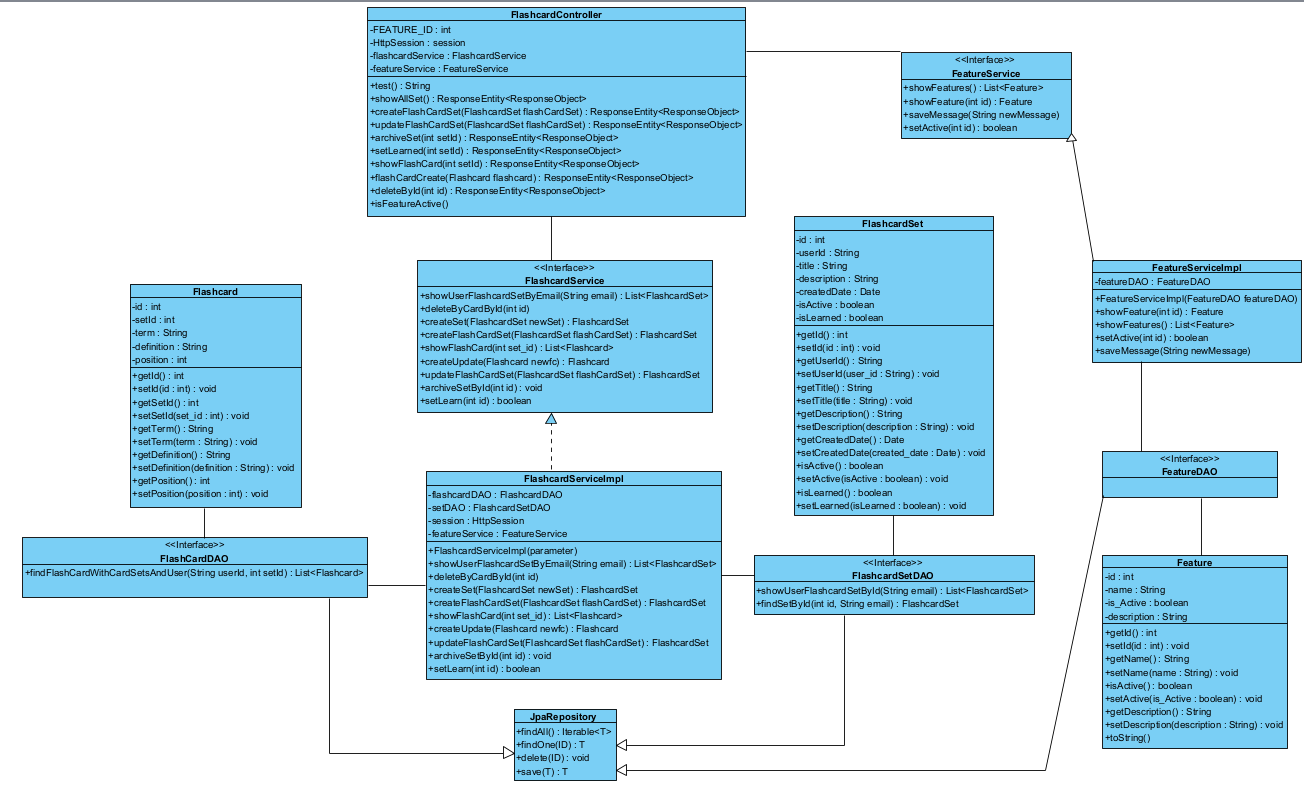
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Update password user in the database |

1. Sequence diagram



### View/Add/Update/Archive Flashcard Set

1. Class diagram



1. Class specification
   1. FlashcardController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | createFlashCardSet | Create a new set |
| 02 | updateFlashCardSet | Update current set |
| 03 | archiveSet | archive flashcard set |
| 04 | showAllSet | List all sets that one user is having |

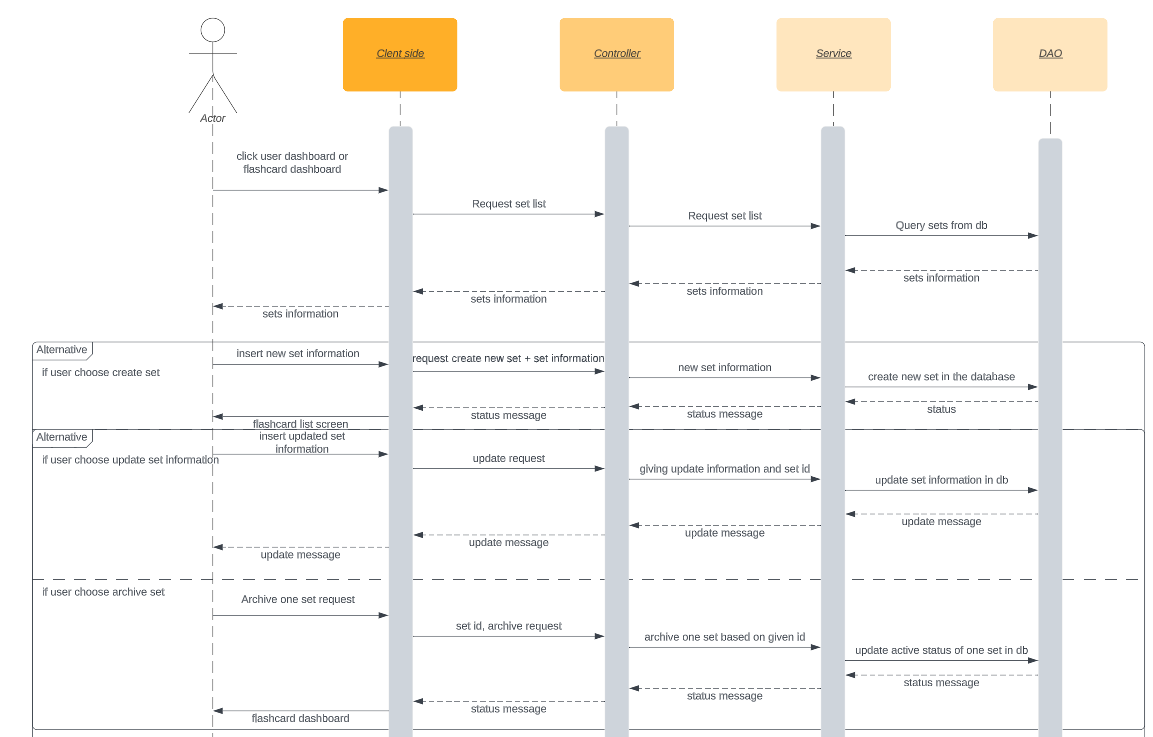
II)FlashcardService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | createFlashCardSet | Create a new set |
| 02 | updateFlashCardSet | Update current set |
| 03 | archiveSetById | archive flashcard set |
| 04 | showUserFlashcardSetByEmail | List all sets that one user is having. Return list flashcard set |

II|) FlashcardDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Create a new set in the database |
| 02 | save | Update current set in the database |
| 03 | save | update set information in the database |
| 04 | showUserFlashcardSetById | List all sets that one user is having in the database |

1. Sequence diagram

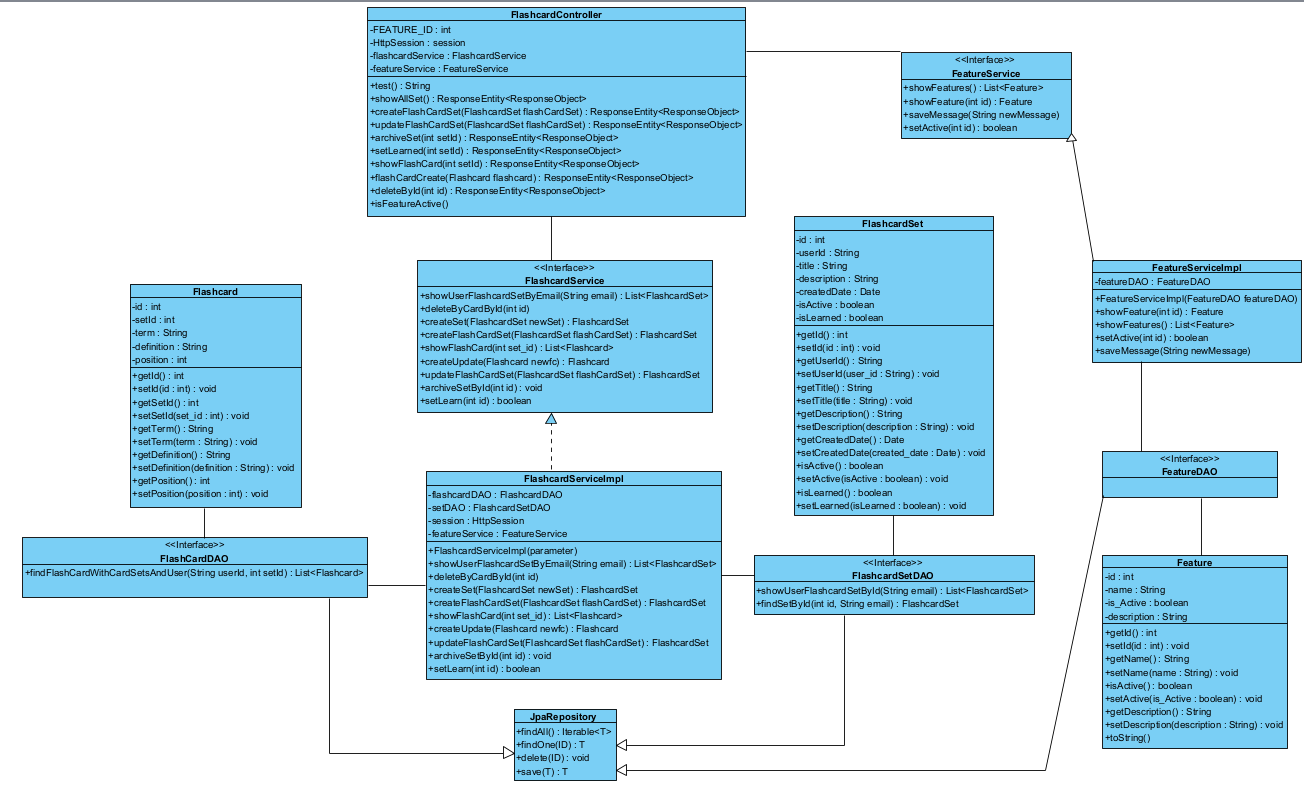


1. SQL sequence

select s from FlashcardSet s where s.userId= :email AND s.isActive= true

### Flashcard/Learn Flashcard

1. Class diagram



1. Class Specification

I) FlashcardController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | setLearned | Change the status of a flashcard set to "learned" |

ii)FlashcardService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | setLearn | Checks the user's email, finds the flashcard set with the specified ID and user's email, toggles the "learned" status of the set, and returns the updated |

Iii)FlashcardDAO

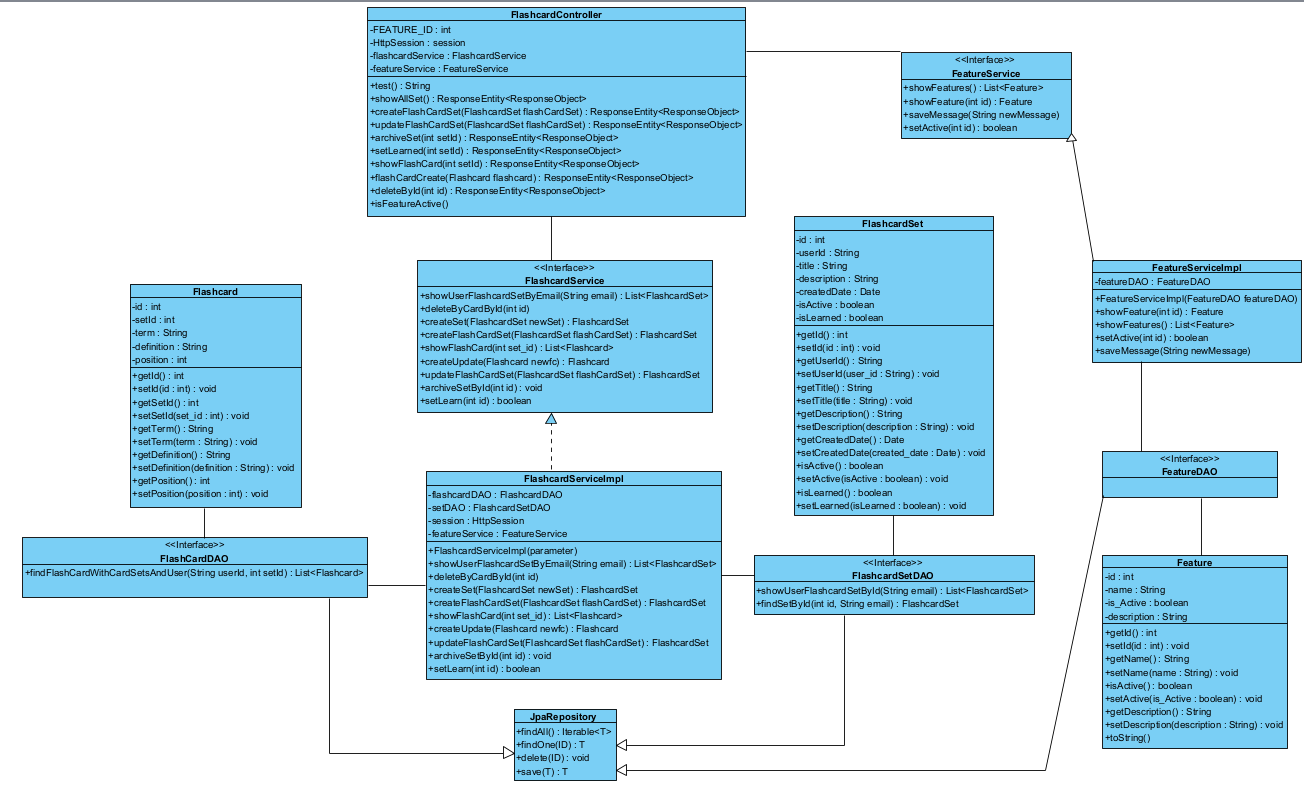
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | update set information in the database |
| 02 | findSetById | Find flash card set by email and setID in the database |

c) SQL sequence

select s from FlashcardSet s where s.userId=:email AND s.isActive= true AND s.id = :setId

### View/Add/Update/Delete Flashcard(inside set)

1. Class diagram



1. Class specification
   1. FlashcardController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | flashCardCreate | Create new flashcard inside a set |
| 02 | flashCardCreate | Update flashcard content inside a set |
| 03 | deleteById | Delete flashcard inside a set |
| 04 | showFlashcard | List all flashcards in one set |

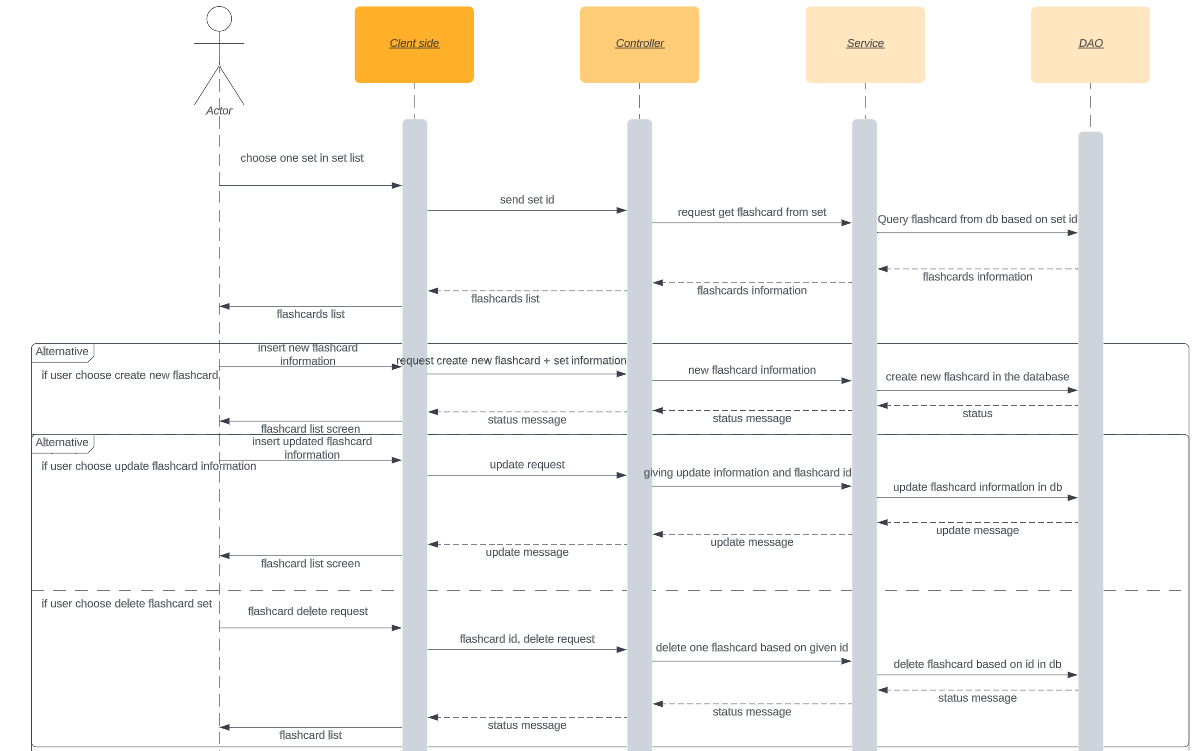
II)FlashcardService

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | createUpdate | Create new flashcard inside a set |
| 02 | createUpdate | Update flashcard content inside a set |
| 03 | deleteByCardById | Delete flashcard inside a set |
| 04 | showFlashcard | List all flashcards in one set |

II|)FlashcardDAO

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | save | Create new flashcard inside a set in the database |
| 02 | save | Update flashcard content inside a set in the database |
| 03 | deleteById | Delete flashcard inside a set in the database |
| 04 | findFlashCardWithCardSetsAndUser | List all flashcards in one set in the database |

c) Sequence diagram



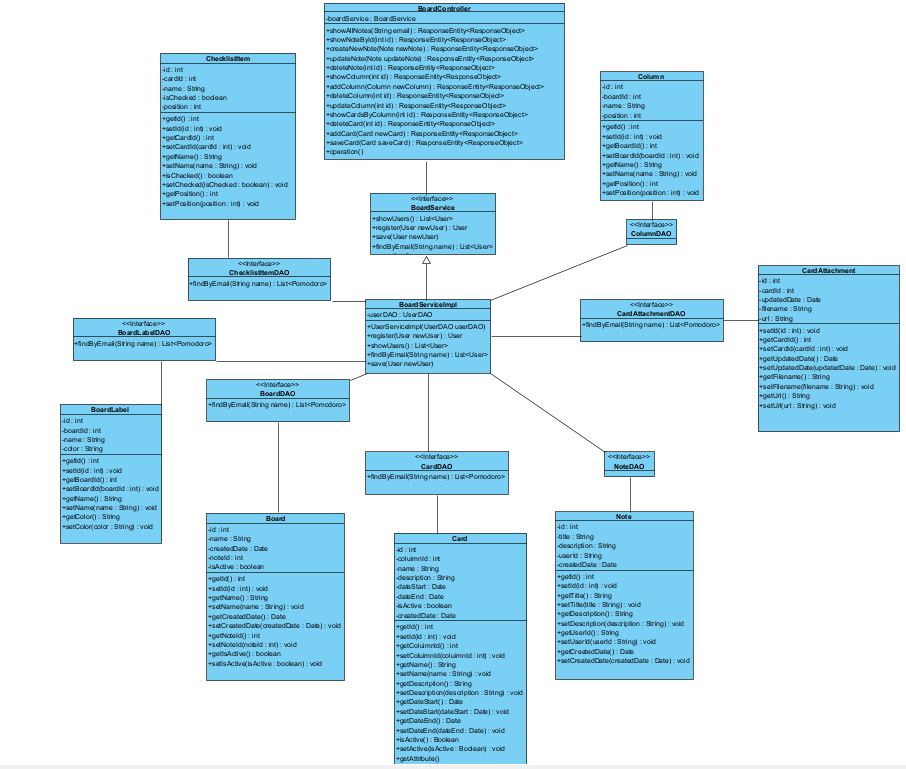
1. Database Query

SQL sequence

SELECT \* FROM flashcard fl WHERE fl.set\_id = :setId AND fl.set\_id IN (SELECT id FROM flashcard\_set f WHERE f.user\_id = :userId

### Kanban

1. Class diagram



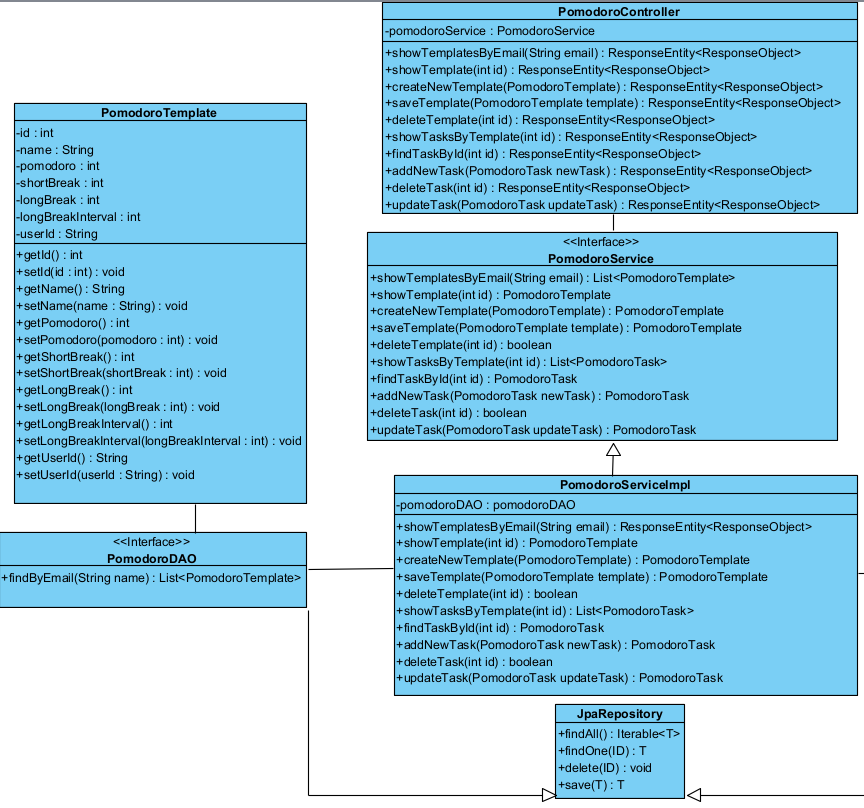
1. Class specification
   1. BoardController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 |  |  |
| 02 |  |  |
| 03 |  |  |
| 04 |  |  |
| 05 |  |  |
| 06 |  |  |
| 07 |  |  |
| 08 |  |  |
| 09 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |

### 

### Archive column

1. Class diagram



1. Class specification

### 

### Pomodoro/Screen appearance customization

1. Class diagram
2. Class specification
   1. PomodoroController

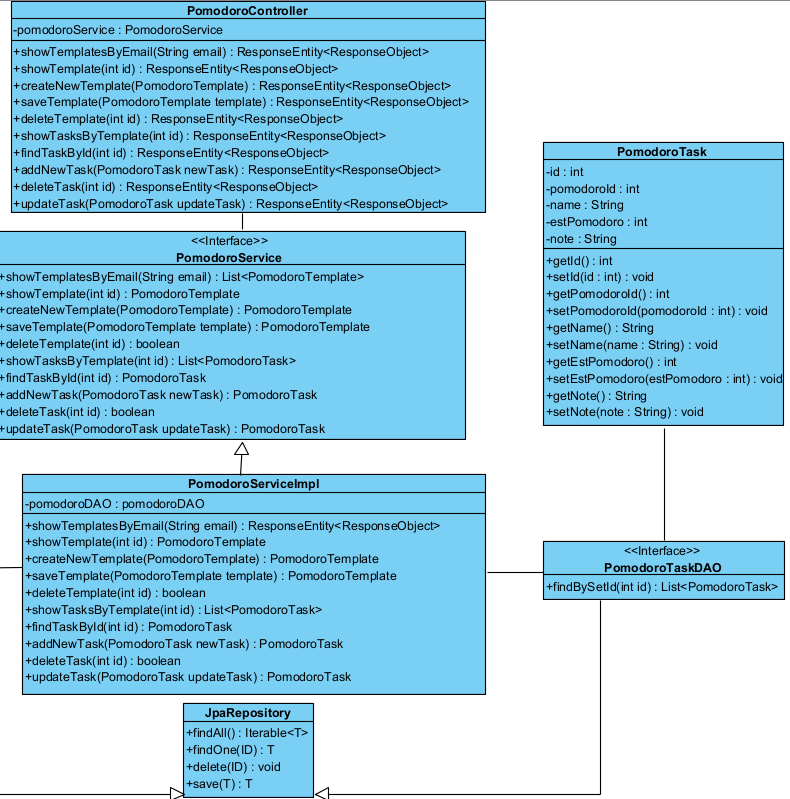
|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showTemplatesByEmail | Show all templates that users have based on their email |
| 02 | showTemplate | Show default template |
| 03 | createNewTemplate | Create new template |
| 04 | saveTemplate | Save new template |
| 05 | deleteTemplate | Delete template |

### Pomodoro/Session Sound

1. Class diagram
2. Class specification

### Pomodoro/Input tasks for sessions

1. Class diagram



1. Class specification
   1. PomodoroController

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | showTasksByTemplate | Show all tasks corresponding to the template |
| 02 | findTaskById | Show task details |
| 03 | addNewTask | Add new task to the list |
| 04 | deleteTask | Delete task from the list |
| 05 | updateTask | Update task inside the list |

# 

### Labels

a). Class diagram

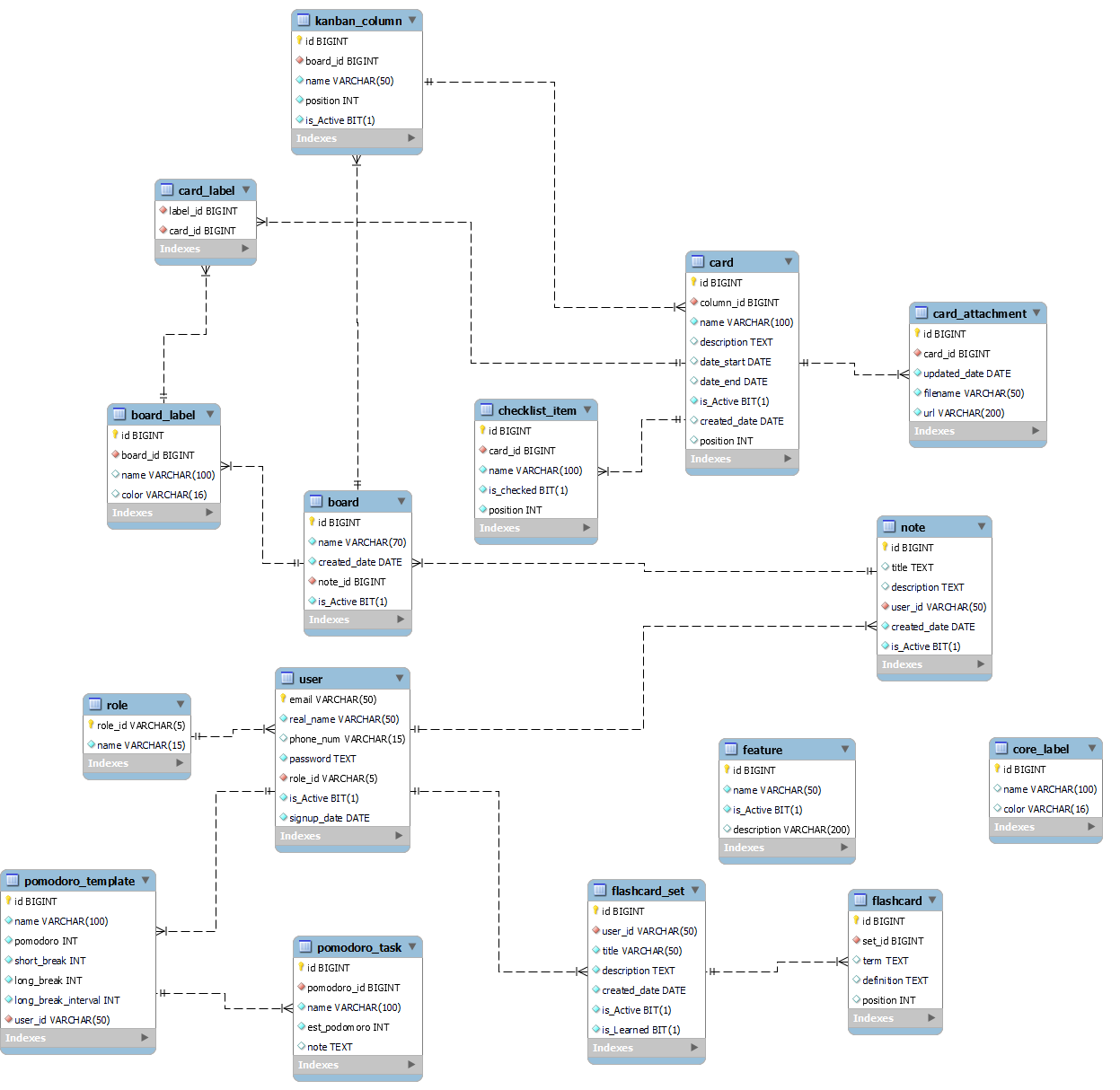
b).Class specification

c).Labels control

|  |  |  |
| --- | --- | --- |
| **no** | **Method** | **Description** |
| 01 | **getAllLabels** | Retrieves all core labels. |
| 02 | **getLabelById** | Retrieves a core label by its ID. |
| 03 | **createLabel** | Creates a new core label. |
| 04 | **getAllLabelsByBoardId** | Retrieves all board labels by board ID. |
| 05 | **getLabelById** | Retrieves a board label by board ID and label ID. |
| 06 | **createLabel** | Creates a new board label |
| 07 | **updateLabel** | Updates a board label. |
| 08 | **deleteLabel** | Deletes a board label. |
| 09 | **updateLabel** | Updates a core label. |
| 10 | **deleteLabel** | Deletes a core label. |
| 11 | **addLabelToCard** | Adds a label to a card. |
| 12 | **getLabelsByCardId** | Retrieves all card labels by card ID. |
| 13 | **getCardsByLabelId** | Retrieves all card labels by label ID. |
| 14 | **removeLabelFromCard** | Removes a label from a card. |

# Database Design

## Database Schema



## Table Description

|  |  |  |
| --- | --- | --- |
| **No** | **Table** | **Description** |
| 1 | Role | This table is used to store user role: admin, user, guest   * Primary key: role\_id * Foreign key: none |
| 2 | User | This table is used to store information of users. Based on this table, the system can authorize users   * Primary key: email * Foreign key: (role\_id) references Role(role\_id) |
| 3 | Feature | This table is used to store information of features. Based on this table, the system will identify the active status of each feature   * Primary key: id * Foreign key: none |
| 4 | Note | This table is used to store information of notes, including creator, create-date, title, etc. Based on this table, the system will identify notes of each user   * Primary key: id * Foreign key: (user\_id) references User(email) |
| 5 | Board | This table is used to store information of a Kanban board   * Primary key: id * Foreign key: (note\_id) references Note(id) |
| 6 | Column | This table is used to store information of each column inside a Kanban board. Based on this table, the system will identify latest position of each column inside a board everytime it is rendered   * Primary key: id * Foreign key: (board\_id) references Board(id) |
| 7 | Card | This table is used to store information of each card inside a column of a Kanban board. Based on this table, the system will identify latest position of each card inside a column everytime it is rendered   * Primary key: id * Foreign key: id references Column(id) |
| 8 | Checklist\_item | This table is used to store check list of each card in kanban board   * Primary key: id * Foreign key: references |
| 9 | Card\_attachment | This table is used to store attachments inside a card   * Primary key: id * Foreign key: (card\_id) references Card(id) |
| 10 | Core\_label | This table is used to store default color labels of a Kanban table   * Primary key: id * Foreign key: none |
| 11 | Board\_label | This table is used to store additional label of a Kanban table   * Primary key: id * Foreign key: (board\_id) references Board(id) |
| 12 | Card\_label | This table is used to store labels of each card. This table handles the many-to-many relationship between label and card.   * Primary key: (label\_id) + (card\_id) * Foreign key: (label\_id) references Board\_label(id)   (card\_id) references Card(id) |
| 13 | Pomodoro\_template | This table is used to store a template for Pomodoro setting. Based on this table, the system will identify the appearance every time a user turns on the Pomodoro feature.   * Primary key: id * Foreign key: (user\_id) references User(email) |
| 14 | Pomodoro\_task | This table is used to store tasks to-do in focus sessions. Based on this table, the system will identify which task the user is working on during the session, and calculate the spent time on each task   * Primary key: id * Foreign key: (pomodoro\_id) references pomodoro\_template(id) |
| 15 | Flashcard\_set | This table is used to store sets of flashcards that one user is having. Based on this table, the system can identify the number of sets and the learning status of each set   * Primary key: id * Foreign key: user\_id references User(email) |
| 16 | Flashcard | This table is used to store flashcards in each set. Based on this table, the system will identify the content inside each flashcard.   * Primary key: id * Foreign key: set\_id references flashcard\_set(id) |