

**Coffee shop management and business system "Coffee Management"**

**Software Design Specification**

– Can Tho, January 2024 –

**Table of Contents**

[I. Overview 3](#_Toc96516286)

[1. Code Packages 3](#_Toc96516287)

[2. Database Design 4](#_Toc96516288)

[a. Database Schema 4](#_Toc96516289)

[b. Table Description 4](#_Toc96516290)

[II. Code Designs 6](#_Toc96516291)

[1. Login 6](#_Toc96516292)

[a. Class Diagram 6](#_Toc96516293)

[b. Class Specifications 6](#_Toc96516294)

[c. Sequence Diagram(s) 7](#_Toc96516295)

[d. Database queries 7](#_Toc96516296)

[2. Create account 8](#_Toc96516292)

[a. Class Diagram 8](#_Toc96516293)

[b. Class Specifications 8](#_Toc96516294)

[c. Sequence Diagram(s) 9](#_Toc96516295)

[d. Database queries 9](#_Toc96516296)

[3. Update order 9](#_Toc96516292)

[a. Class Diagram 9](#_Toc96516293)

[b. Class Specifications 1](#_Toc96516294)0

[c. Sequence Diagram(s) 11](#_Toc96516295)

[d. Database queries 1](#_Toc96516296)1

[4. Delete Order 1](#_Toc96516292)1

[a. Class Diagram 1](#_Toc96516293)1

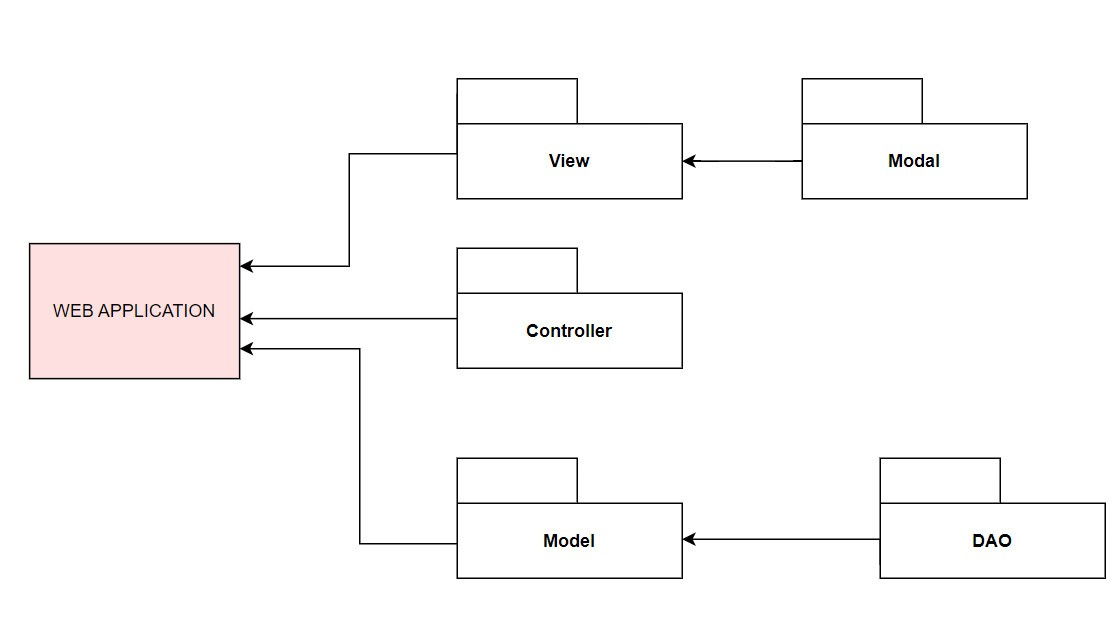
[b. Class Specifications 1](#_Toc96516294)1

[c. Sequence Diagram(s) 1](#_Toc96516295)3

[d. Database queries 1](#_Toc96516296)3

# Overview

1. **Code Packages**

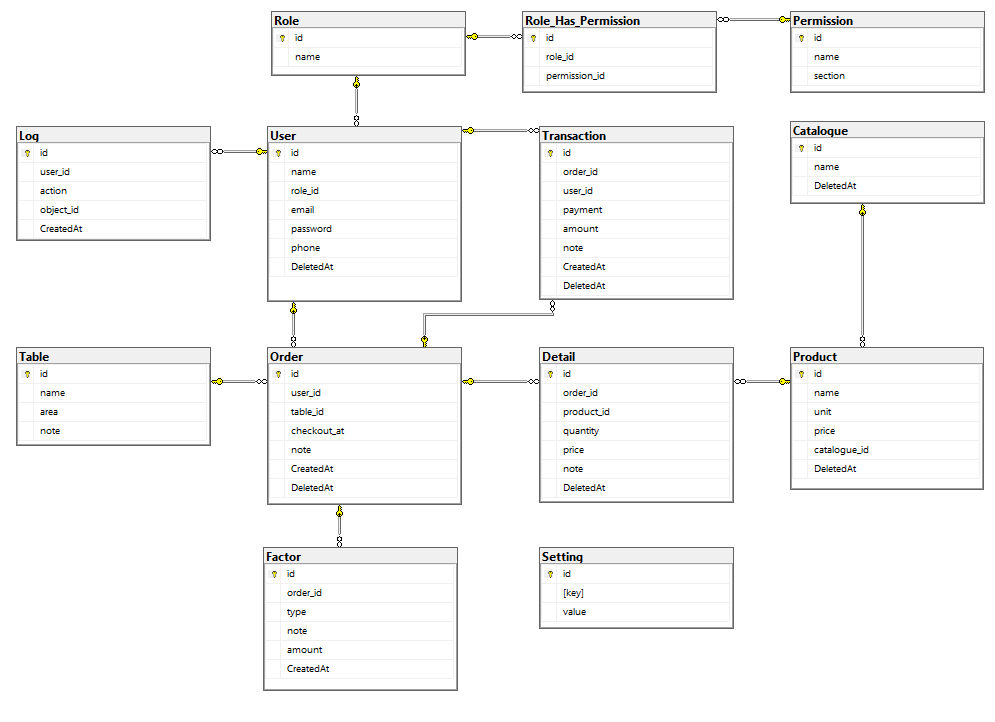


***Package descriptions***

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| *01* | *View* | *The VIEW package contains files responsible for displaying information to the user.* |
| *02* | *Modal* | *The MODAL package contains classes and components related to displaying and managing modals.* |
| *03* | *Controller* | *The CONTROLER package contains classes and components related to handling control logic (controller) in the MVC (Model-View-Controller) architecture.* |
| *04* | *Model* | *The package MODEL contains classes and components involved in data processing and state management of the application.* |
| *05* | *DAO* | *The DAO package contains classes and components related to accessing data from the database. Classes in this package often perform CRUD (Create, Read, Update, and Delete) operations to interact with the database.* |

## 2. Database Design

### a. Database Schema



### 

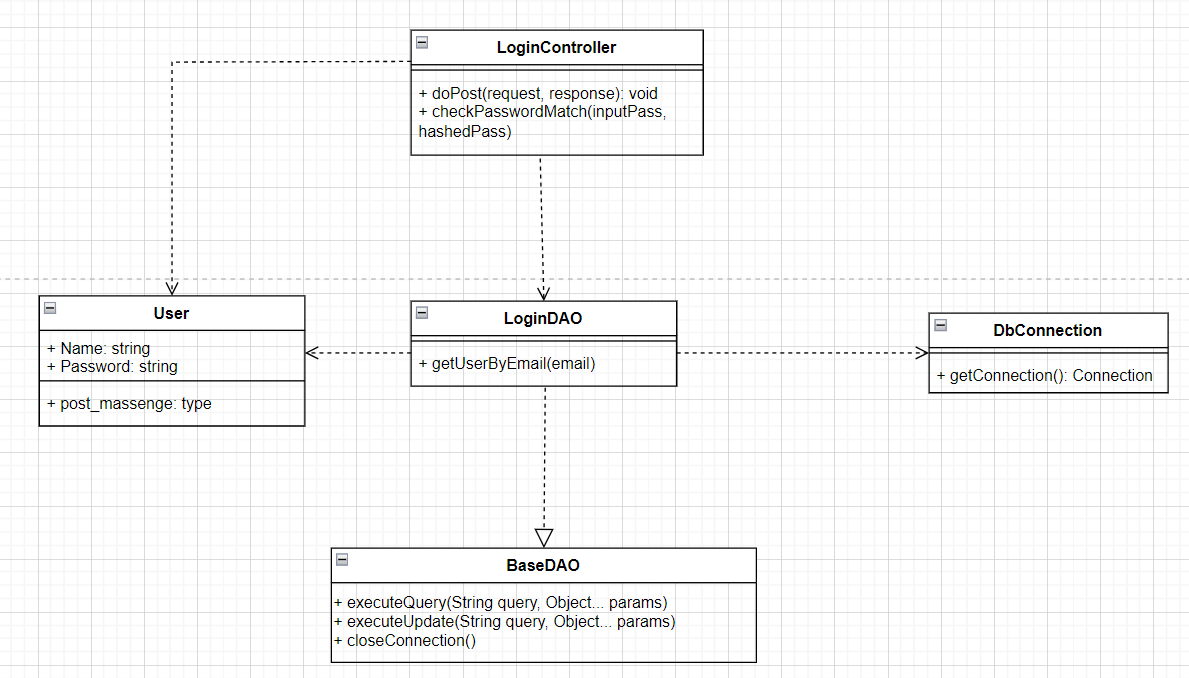
### b. Table Description

|  |  |  |
| --- | --- | --- |
| **No** | **Table** | **Description** |
| 1 | User | Save information of account  - Primary keys: id  - Foreign key: role\_id |
| 2 | Log | Save the system log  - Primary key: id  - Foreign key: user\_id |
| 3 | Order | Save information of order  - Primary key: id  - Foreign keys: user\_id, table\_id |
| 4 | Transaction | Save information of user  - Primary key: id  - Foreign keys: order\_id, user\_id |
| 5 | Detail | Save information of detail  - Primary key: id  - Foreign keys: order\_id, product\_id |
| 6 | Product | Save information of product  - Primary key: id  - Primary key: catalogue\_id |
| 7 | Catalogue | Save information of catalogue  - Primary key: id |
| 8 | Table | Save information of table  - Primary key: id |
| 9 | Role | Stores the names of user positions  - Primary key: id |
| 10 | Permission | Stores the permissions that the user can exercise  - Primary key: id |
| 11 | Role\_Has\_Permission | Stores the permissions associated with the role  - Primary key: id  - Primary key: role\_id, permission\_id |
| 12 | Setting | Stores system settings  -Primary key:id |

# II. Code Designs

## 1. Login

### a. Class Diagram



### b. Class Specifications

#### Login Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getUserByEmail(email) | The getUserByEmail method is a crucial component of the LoginDAO class, responsible for retrieving user information from a data source (such as a database) based on their email address |

#### BaseDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | executeQuery(String query,Object...  params) | The executeQuery method allows you to execute SELECT queries with or without parameters and  returns a ResultSet |
| 02 | executeUpdate(String query,Object... params) | The executeUpdate method allows you to execute INSERT, UPDATE, or DELETE queries with  or without parameters and returns the number of affected rows. |
| 03 | closeConnection() | The closeConnection method is used to close the database connection when it's no longer needed. |

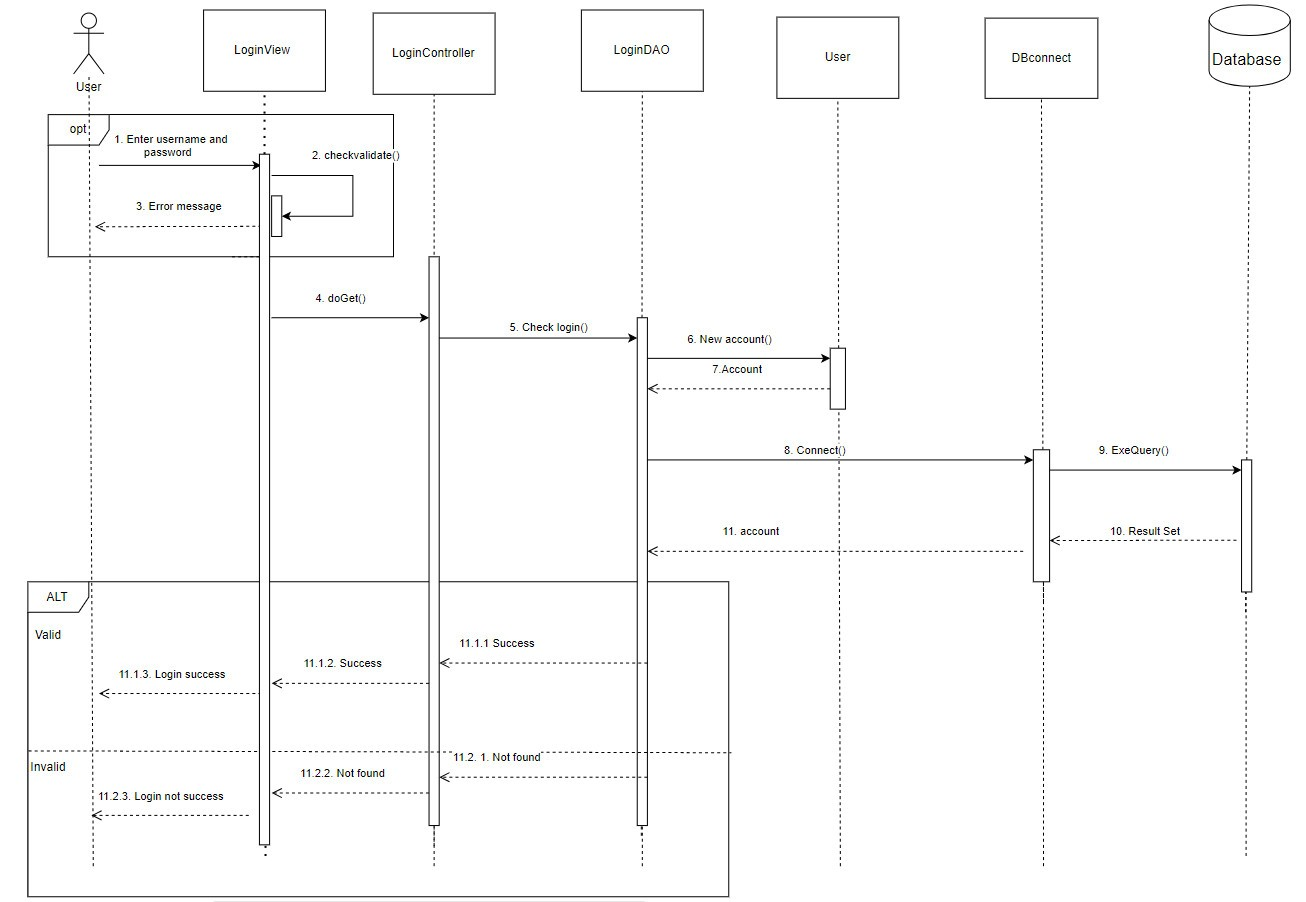
#### DBConnection Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection(): Connection | Connect to MySQL database and return a connection object. |

#### LoginController Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doPost(request, response): void | The doPost method is an integral part of the LoginController class, designed to handle incoming HTTP POST requests initiated by users attempting to log in to a web application. |
| 02 | checkPasswordMatch(inputPass, hashedPass) | The checkPasswordMatch method is an essential component of the LoginController class, designed to verify whether a given password matches the stored or expected password for a specific user. This method is crucial for securely authenticating users during the login process and ensuring that only authorised individuals gain access to the application. |

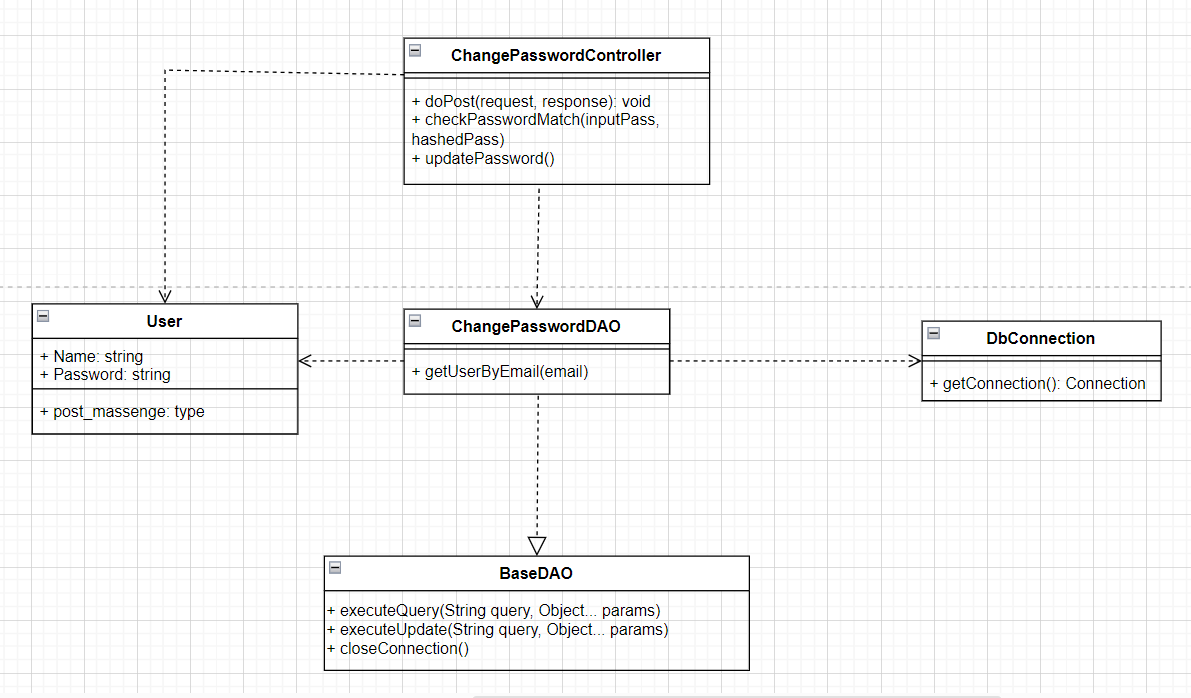
### c. Sequence Diagram(s)

d. Database Queries

SELECT \* FROM users WHERE email =?

**2. Change password**

### a. Class Diagram



### b. Class Specifications

#### ChangePasswordController Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doPost(request, response): void | The doPost method is an integral part of the LoginController class, designed to handle incoming HTTP POST requests initiated by users attempting to log in to a web application. |
| 02 | checkPasswordMatch(inputPass, hashedPass) | The checkPasswordMatch method is an essential component of the LoginController class, designed to verify whether a given password matches the stored or expected password for a specific user. This method is crucial for securely authenticating users during the login process and ensuring that only authorised individuals gain access to the application. |
| 03 | UpdatePassword() | + This function helps us change the old password to a new password |

#### ChangePasswordDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getUserByEmail(email) | The getUserByEmail method is a crucial component of the LoginDAO class, responsible for retrieving user information from a data source (such as a database) based on their email address |

#### BaseDAO Class

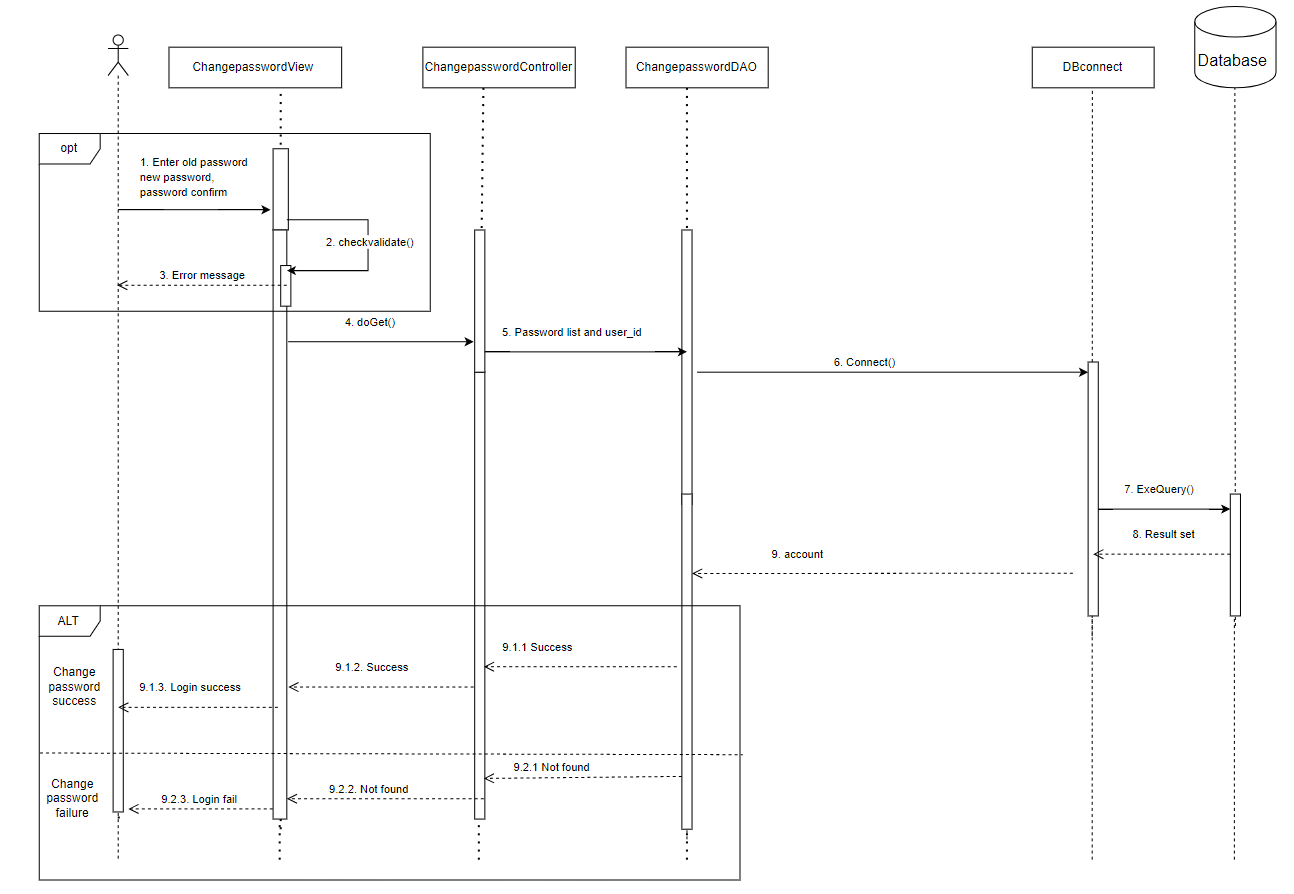
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | executeQuery(String query,Object...  params) | The executeQuery method allows you to execute SELECT queries with or without parameters and  returns a ResultSet |
| 02 | executeUpdate(String query,Object... params) | The executeUpdate method allows you to execute INSERT, UPDATE, or DELETE queries with  or without parameters and returns the number of affected rows. |
| 03 | closeConnection() | The closeConnection method is used to close the database connection when it's no longer needed. |

#### DBConnection Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection(): Connecttion | Connect to MySQL database and return a connection object. |

### c. Sequence Diagram(s)

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



### d. Database Queries

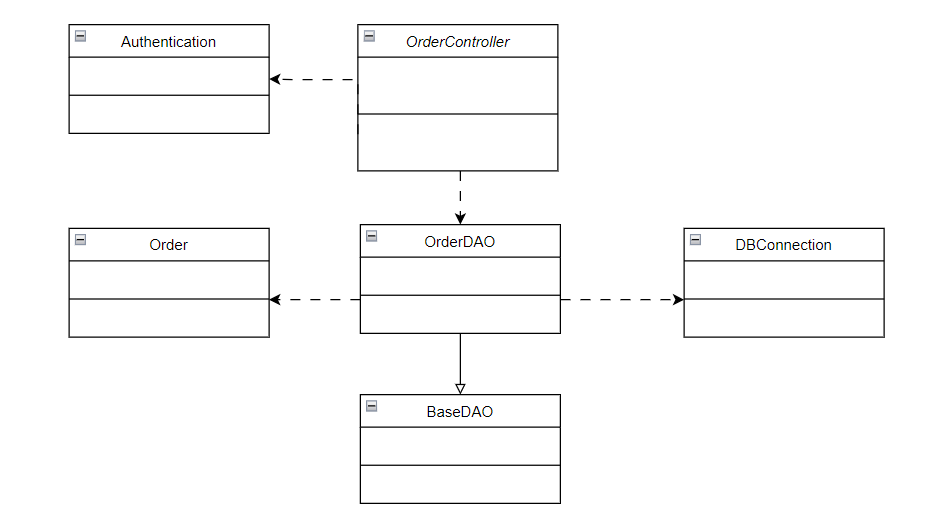
UPDATE user

SET password = 'mat\_khau\_moi'

WHERE id = user\_id ;

## 3. Delete Order

### a. Class Diagram



### b. Class Specifications

#### OrderController Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deleteOrder(HttpServletRequest  request, HttpServletResponse  response) | The OrderController class is a controller component within a web application that manages HTTP requests related to interactions with orders. This class is part of the overall control layer, handling user interactions and directing them to appropriate services or data access objects. The deleteOrder method specifically deals with handling requests to delete a specific order in the system. |

#### Authentication Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getRole() | The getRole method is used to retrieve information about the role of a user. This method can be employed for display or processing purposes related to role management within the system. |

#### OrderDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deleteOrder() | The OrderDAO class is a Data Access Object responsible for managing interactions with the underlying data storage related to order within a web application. It encapsulates methods to perform database operations, and the deleteOrder method specifically handles the deletion of a order. |

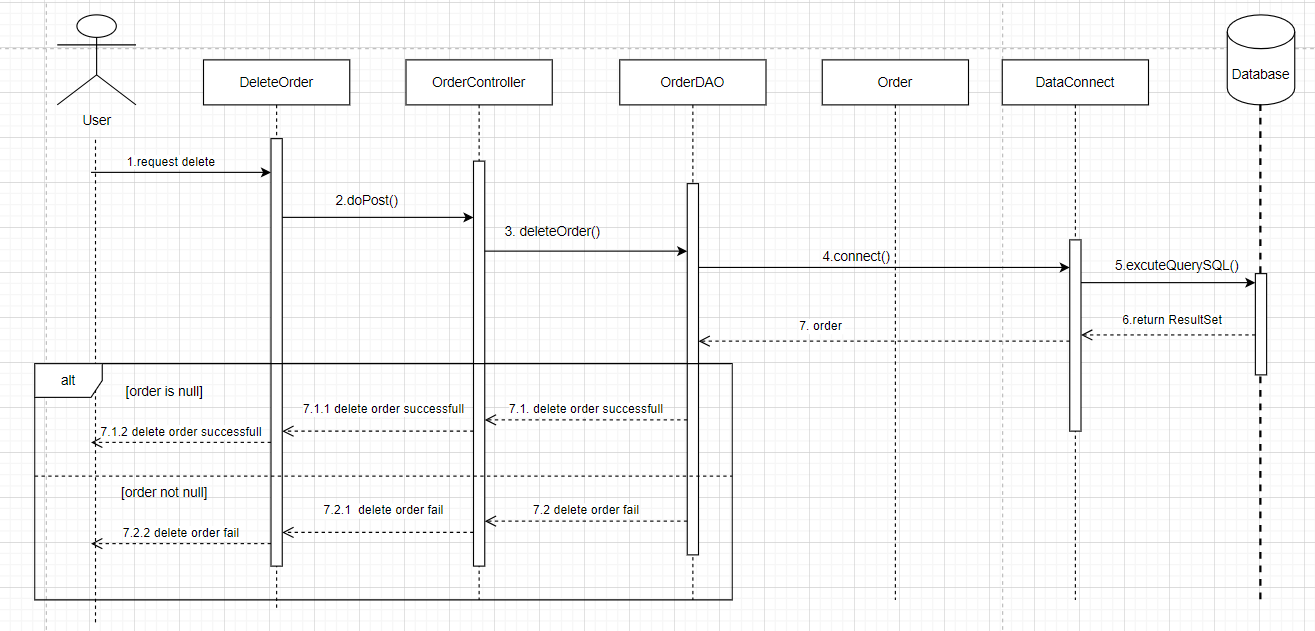
#### DBConnection Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection(): Connecttion | Connect to MySQL database and return a connection object |

#### BaseDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | executeQuery(String query,Object...  params) | The executeQuery method allows you to execute SELECT queries with or without parameters and  returns a ResultSet |
| 02 | executeUpdate(String query,Object... params) | The executeUpdate method allows you to execute INSERT, UPDATE, or DELETE queries with  or without parameters and returns the number of affected rows. |
| 03 | closeConnection() | The closeConnection method is used to close the database connection when it's no longer needed. |

### c. Sequence Diagram(s)

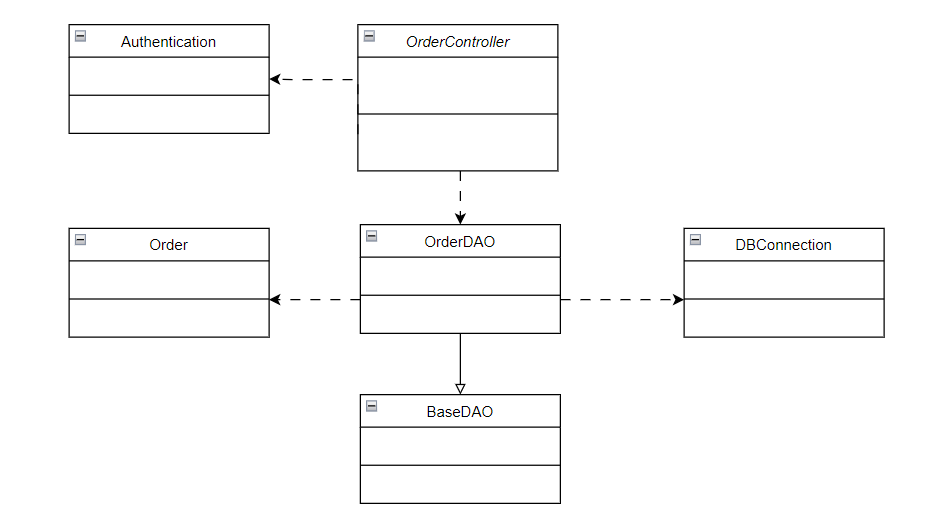


### d. Database Queries

DELETE FROM [dbo].[Order] WHERE id = ?;

## 4. Update Order

### a. Class Diagram



### b. Class Specifications

#### OrderController Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | updateOrder(HttpServletRequest  request, HttpServletResponse  response) | The OrderController class is a controller component within a web application that manages HTTP requests related to interactions with orders. This class is part of the overall control layer, handling user interactions and directing them to appropriate services or data access objects. The updateOrder method specifically deals with handling requests to update a specific order in the system. |

#### Authentication Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getRole() | The getRole method is used to retrieve information about the role of a user. This method can be employed for display or processing purposes related to role management within the system. |

#### OrderDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | updateOrder() | The OrderDAO class is a Data Access Object responsible for managing interactions with the underlying data storage related to orders within a web application. It encapsulates methods to perform database operations, and the updateOrder method specifically handles the update for a order. |

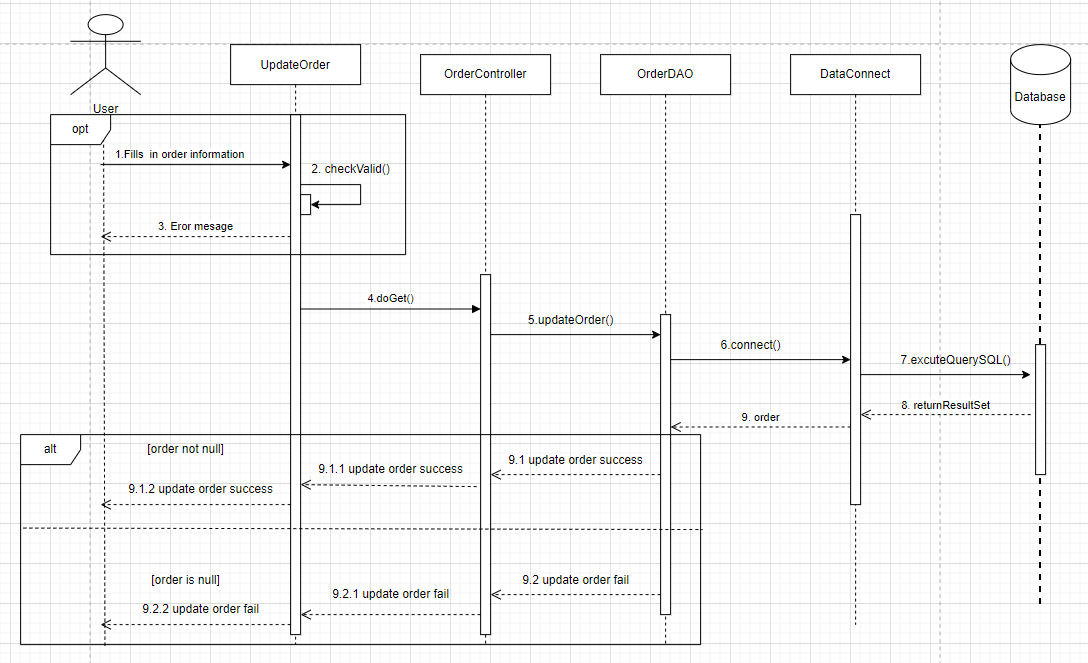
#### DBConnection Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection(): Connecttion | Connect to MySQL database and return a connection object. |

#### BaseDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | executeQuery(String query,Object...  params) | The executeQuery method allows you to execute SELECT queries with or without parameters and  returns a ResultSet |
| 02 | executeUpdate(String query,Object... params) | The executeUpdate method allows you to execute INSERT, UPDATE, or DELETE queries with  or without parameters and returns the number of affected rows. |
| 03 | closeConnection() | The closeConnection method is used to close the database connection when it's no longer needed. |

### c. Sequence Diagram(s)



### d. Database Queries

UPDATE [dbo].[Order]

SET

[id] = ?,

[user\_id] = ?,

[customer\_id] = ?,

[table\_id] = 10,

[checkout\_at] = ‘?’,

[note] = '?',

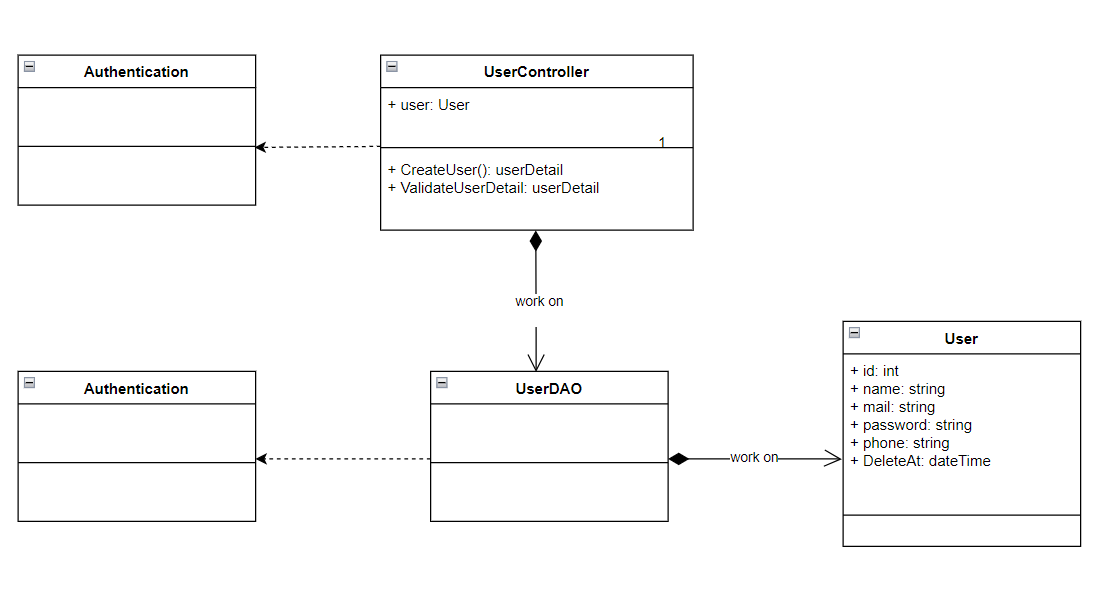
[CreatedAt] = '?',

[DeletedAt] = NULL

WHERE [id] = ?;

## 5. Create new user

### a. Class Diagram



### b. Class Specifications

#### UserController Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | CreateUser | Get the Method and handle user requests |

#### Authentication Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getRole() | The getRole method is used to retrieve information about the role of a user. This method can be employed for display or processing purposes related to role management within the system. |

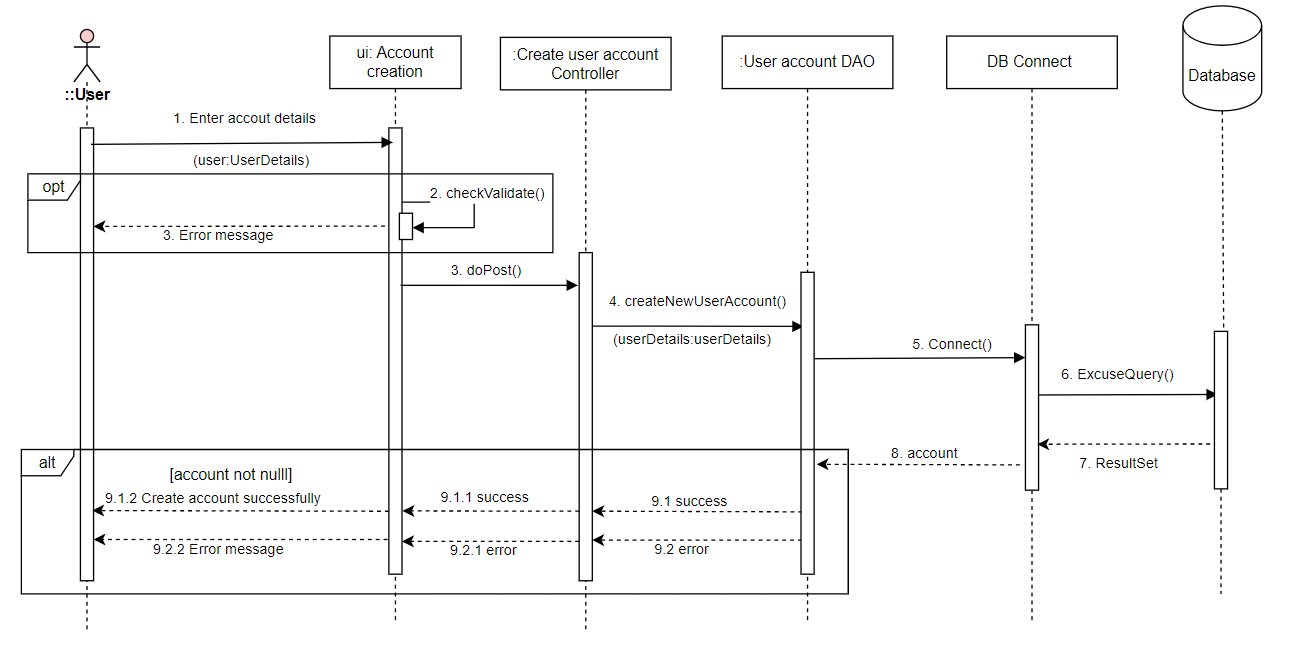
#### UserDAO Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | *UserDAO()* | The UserDAO class is a Data Access Object responsible for managing interactions with the underlying data storage related to orders within a web application. It encapsulates methods to perform database operations, and the CreateUser method specifically handles the create new user |

#### DBConnection Class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection(): Connecttion | Connect to MySQL database and return a connection object. |

### c. Sequence Diagram(s)



### d. Database Queries

INSERT INTO users (id, name, email, phone, password, DeletedAt)

VALUES

('user\_id', ‘user\_name', 'user\_email@example.com', 'user\_phone', 'hashed\_password', NULL);