

**Building an online sneaker sales system**

**Software Design Specification**

–Danang, May 2023 –

**Record of changeS**

| **Date** | **A\* M, D** | **In charge** | **Change Description** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

\*A - Added M - Modified D - Deleted

**Table of Contents**

[**I. Overview**](#_heading=h.gjdgxs) **7**

[1. Code Packages](#_heading=h.30j0zll) 7

[2. Database Design](#_heading=h.1fob9te) 7

[a. Database Schema](#_heading=h.3znysh7) 7

[b. Table Description](#_heading=h.2et92p0) 8

[**II. Code Designs**](#_heading=h.tyjcwt) **9**

[1. Account Registration](#_heading=h.uygbexacuhoc) 9

[a. Class Diagram](#_heading=h.4ze28fy5joav) 9

[b. Class Specifications](#_heading=h.dsb6nliwm3zd) 9

[c. Sequence Diagram(s)](#_heading=h.aor9lijppyii) 10

[d. Database Queries](#_heading=h.e6h8t5b3ew6q) 10

[2. Log in](#_heading=h.3rdcrjn) 11

[a. Class Diagram](#_heading=h.rv6i6wj20mwp) 11

[b. Class Specifications](#_heading=h.duywmg30rck6) 11

[c. Sequence Diagram(s)](#_heading=h.9khyplxp9g95) 12

[d. Database Queries](#_heading=h.gxfjwa5ngumt) 12

[3. Log out](#_heading=h.82ibjcpnhx1v) 13

[a. Class Diagram](#_heading=h.b7je3sdko7ay) 13

[b. Class Specifications](#_heading=h.6vohhjmzsinb) 13

[c. Sequence Diagram(s)](#_heading=h.8k0d33htag9y) 14

[4. Change password](#_heading=h.8soci4laffh2) 15

[a. Class Diagram](#_heading=h.nbak6ue5mfj0) 15

[b. Class Specifications](#_heading=h.in80r32hk8dp) 15

[c. Sequence Diagram(s)](#_heading=h.qhlehm4km8s1) 16

[d. Database Queries](#_heading=h.9lba70v3imz6) 16

[5. Forgot password](#_heading=h.n02nf7ev0bwq) 17

[a. Class Diagram](#_heading=h.pdh2jwqjk786) 17

[b. Class Specifications](#_heading=h.rp2jwprshajx) 17

[c. Sequence Diagram(s)](#_heading=h.utnfeqgmfmfh) 17

[d. Database Queries](#_heading=h.wtkrlug7mmes) 18

[6. View products](#_heading=h.5qt75japq13h) 19

[a. Class Diagram](#_heading=h.qkxptx2rmtwy) 19

[b. Class Specifications](#_heading=h.rf0hkeio8yb2) 19

[c. Sequence Diagram(s)](#_heading=h.hgdq6qn2fm4w) 20

[d. Database Queries](#_heading=h.204ciad6nhs8) 20

[7. Search products](#_heading=h.2r6psw7883l6) 21

[a. Class Diagram](#_heading=h.h1ksf5jjgcrk) 21

[b. Class Specifications](#_heading=h.woluhifrqw4b) 22

[c. Sequence Diagram(s)](#_heading=h.3xh932nt7p34) 23

[d. Database Queries](#_heading=h.w5b1srfgmr74) 23

[8. Read contents](#_heading=h.5w5baliedc1h) 24

[a. Class Diagram](#_heading=h.2fvd54xr4jfy) 24

[b. Class Specifications](#_heading=h.kuap6m5xok81) 24

[c. Sequence Diagram(s)](#_heading=h.erfltl61ol28) 25

[d. Database Queries](#_heading=h.oiqbgbdbgtzp) 25

[9. View personal information](#_heading=h.o2tvfaape0tw) 26

[a. Class Diagram](#_heading=h.32hvu07neo0v) 26

[b. Class Specifications](#_heading=h.9ornj71e5nvg) 26

[c. Sequence Diagram(s)](#_heading=h.wjoluop4k3bq) 27

[d. Database Queries](#_heading=h.bax8qxvv6jwa) 27

[10. Update personal information](#_heading=h.vj4dtswtowdy) 27

[a. Class Diagram](#_heading=h.z3jdlhe6o7c1) 27

[b. Class Specifications](#_heading=h.hisyn7qw6qbe) 28

[c. Sequence Diagram(s)](#_heading=h.w1bvta8ve3fa) 29

[d. Database Queries](#_heading=h.pul5noy0reoh) 29

[11. Purchase](#_heading=h.1rzjvq2yhwlj) 30

[a. Class Diagram](#_heading=h.7yjwa45gqbkc) 30

[b. Class Specifications](#_heading=h.sayno5lmloze) 30

[c. Sequence Diagram(s)](#_heading=h.6gm98j3m38n) 31

[d. Database Queries](#_heading=h.b2gu2o804ciw) 32

[12. Choose payment method](#_heading=h.6l9le1i6cqnr) 33

[a. Class Diagram](#_heading=h.ygqkmkb08m1c) 33

[b. Class Specifications](#_heading=h.qcamx9kf5td1) 33

[c. Sequence Diagram(s)](#_heading=h.6d9qf6mirl6f) 34

[d. Database Queries](#_heading=h.snem6ei1dzsc) 34

[13.CRUD items in cart](#_heading=h.938xs52u39a8) 35

[a. Class Diagram](#_heading=h.dp2zwihi3cpr) 35

[b. Class Specifications](#_heading=h.2vgs5rqrrfzz) 36

[c. Sequence Diagram(s)](#_heading=h.knjf06222pn5) 38

[d. Database Queries](#_heading=h.5zz0hi7w0cui) 39

[14. View purchase history](#_heading=h.ra4yhv66sauy) 39

[a. Class Diagram](#_heading=h.de1x1nddbd90) 39

[b. Class Specifications](#_heading=h.2n9pjtmjj9q4) 40

[c. Sequence Diagram(s)](#_heading=h.utmbfucxk1oc) 40

[d. Database Queries](#_heading=h.ya5ds3338jlu) 41

[15. Track Order](#_heading=h.o0cmae89eht0) 41

[a. Class Diagram](#_heading=h.79tk1mqk2ctp) 41

[b. Class Specifications](#_heading=h.pfbanvw38vpk) 42

[c. Sequence Diagram(s)](#_heading=h.epl6p8dj0igf) 43

[d. Database Queries](#_heading=h.m8yn8s5tyykk) 44

[16. View contact information](#_heading=h.9dd0p7u7u5f4) 45

[a. Class Diagram](#_heading=h.jwu9mvbb12u8) 45

[b. Class Specifications](#_heading=h.25dhsmi4dbcz) 45

[c. Sequence Diagram(s)](#_heading=h.tmb31h6g06d4) 46

[d. Database Queries](#_heading=h.p5pnexisyh3) 46

[17. Chat with the shop](#_heading=h.nd4vfhawqokq) 46

[a. Class Diagram](#_heading=h.fohhmbn8x5g4) 46

[b. Class Specifications](#_heading=h.5szypkr62rb3) 47

[c. Sequence Diagram(s)](#_heading=h.da2zo979o1zn) 48

[d. Database Queries](#_heading=h.mlgr1brgy36s) 48

[18. CRUD content](#_heading=h.7333dpysx5mw) 49

[a. Class Diagram](#_heading=h.dzh2s5ty1how) 49

[b. Class Specifications](#_heading=h.bj9yzrdfklrp) 50

[c. Sequence Diagram(s)](#_heading=h.v58wtoxkqz2f) 50

[d. Database Queries](#_heading=h.9i6jf5lcuvb4) 51

[19. Confirm order](#_heading=h.oi2dl6x7iq0a) 52

[a. Class Diagram](#_heading=h.obui0cgjczgc) 52

[b. Class Specifications](#_heading=h.fjqe5elonzf) 53

[c. Sequence Diagram(s)](#_heading=h.oy9hm65vzm2w) 54

[d. Database Queries](#_heading=h.ecieesjubgks) 54

[20. View order details](#_heading=h.8c5pt4ymrqnx) 55

[a. Class Diagram](#_heading=h.vqd251dvcxci) 55

[b. Class Specifications](#_heading=h.oezhyn2bphwh) 56

[c. Sequence Diagram(s)](#_heading=h.ofkai73g5ajx) 57

[d. Database Queries](#_heading=h.fhpjk087t6cx) 57

[21. Confirm delivery](#_heading=h.8e5dv6yt5vvu) 59

[a. Class Diagram](#_heading=h.uxhx0k65jn4t) 59

[b. Class Specifications](#_heading=h.cg46chiwteht) 59

[c. Sequence Diagram(s)](#_heading=h.c481dl39l8ky) 60

[d. Database Queries](#_heading=h.dougtkxxq5el) 60

[22. Check failed delivery](#_heading=h.zhex82dr6qd0) 61

[a. Class Diagram](#_heading=h.rkdpyfrqpuud) 61

[b. Class Specifications](#_heading=h.idyw0962vklb) 62

[c. Sequence Diagram(s)](#_heading=h.sv041gjk8hyh) 63

[d. Database Queries](#_heading=h.ae9u34fudoap) 63

[23. CRUD products](#_heading=h.u8i8jyxfq5zz) 64

[a. Class Diagram](#_heading=h.wzcwl4z6na4c) 64

[b. Class Specifications](#_heading=h.tocz0iqhe5mn) 64

[c. Sequence Diagram(s)](#_heading=h.r9l9o8ihz63l) 66

[d. Database Queries](#_heading=h.61pre53kdvc0) 66

[24. View order status](#_heading=h.8lyozfk8v721) 68

[a. Class Diagram](#_heading=h.uu4sa9ga9a8w) 68

[b. Class Specifications](#_heading=h.juql5vlp26pk) 69

[c. Sequence Diagram(s)](#_heading=h.pn8kmmlhmpr6) 70

[d. Database Queries](#_heading=h.cze29kpvv2p9) 70

[25. View sale history](#_heading=h.t95qwd28h9x) 71

[a. Class Diagram](#_heading=h.qb402s2h1m8i) 71

[b. Class Specifications](#_heading=h.ok7ezeips7jp) 72

[c. Sequence Diagram(s)](#_heading=h.253suu4vp2c9) 72

[d. Database Queries](#_heading=h.gr5gpf5bl7qo) 73

[26.View the revenue](#_heading=h.a4olm67ytu3d) 74

[a. Class Diagram](#_heading=h.5ctpo4d74y0e) 74

[b. Class Specifications](#_heading=h.5f8q8p2c5p14) 75

[c. Sequence Diagram(s)](#_heading=h.erlz6bitdpr8) 76

[d. Database Queries](#_heading=h.nvweyuwka0rj) 76

[27. View detailed order history](#_heading=h.h3rrj7fqgbue) 77

[a. Class Diagram](#_heading=h.zd3a2royjscc) 77

[b. Class Specifications](#_heading=h.k7s0grpslw85) 77

[c. Sequence Diagram(s)](#_heading=h.1ibpil8ggo9t) 78

[d. Database Queries](#_heading=h.w6fnfabr9k7) 78

[28.CRUD employees](#_heading=h.ipvgevqr83f8) 79

[a. Class Diagram](#_heading=h.7eeqo2ykxtn6) 79

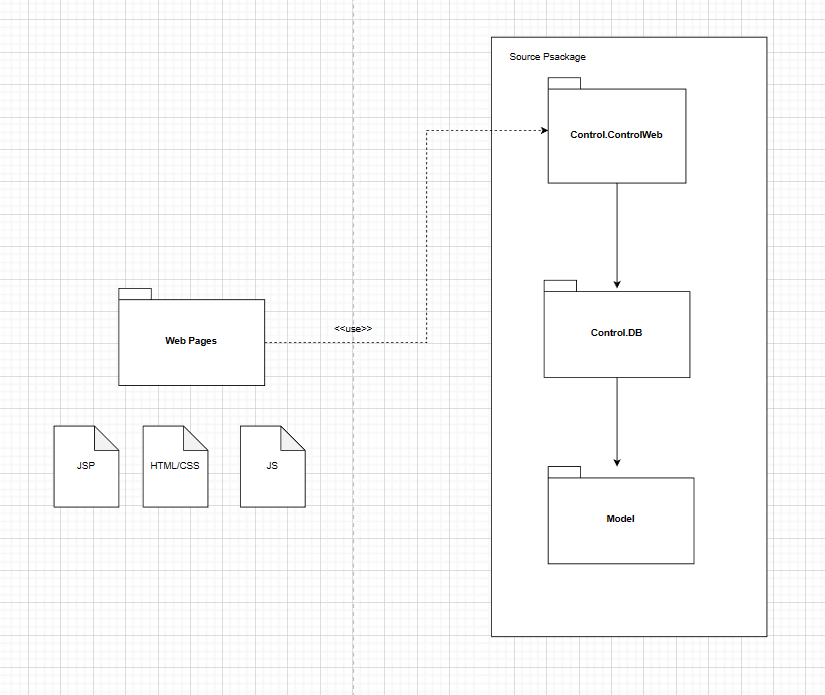
[b. Class Specifications](#_heading=h.73bwxkt16upb) 79

[c. Sequence Diagram(s)](#_heading=h.gb0i5rxl6eph) 81

[d. Database Queries](#_heading=h.o574byl5cqdq) 82

# I. Overview

## 1. Code Packages

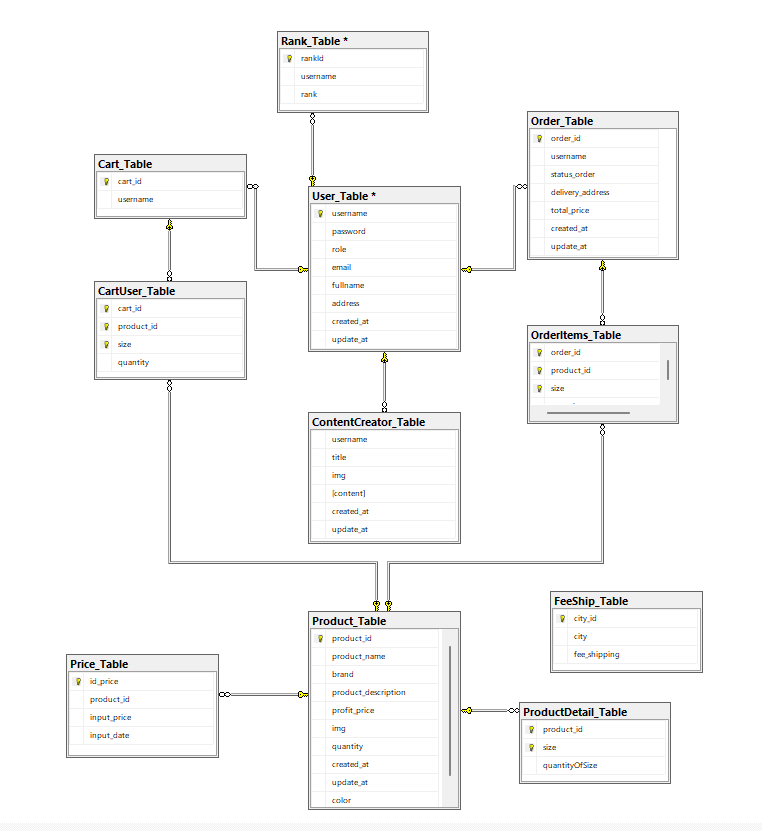


***Package descriptions***

| **No** | **Package** | **Description** |
| --- | --- | --- |
| *01* | *Web Pages* | *This package is contain the FE code* |
| *02* | *Control.ControlWeb* | *This package contains code to perform data queries and control functions with servlets* |
| *03* | *Control.DB* | *This package includes the code to connect to the database* |
| *04* | *Model* | *Thí package contains the class and object of system* |

## 2. Database Design

### a. Database Schema



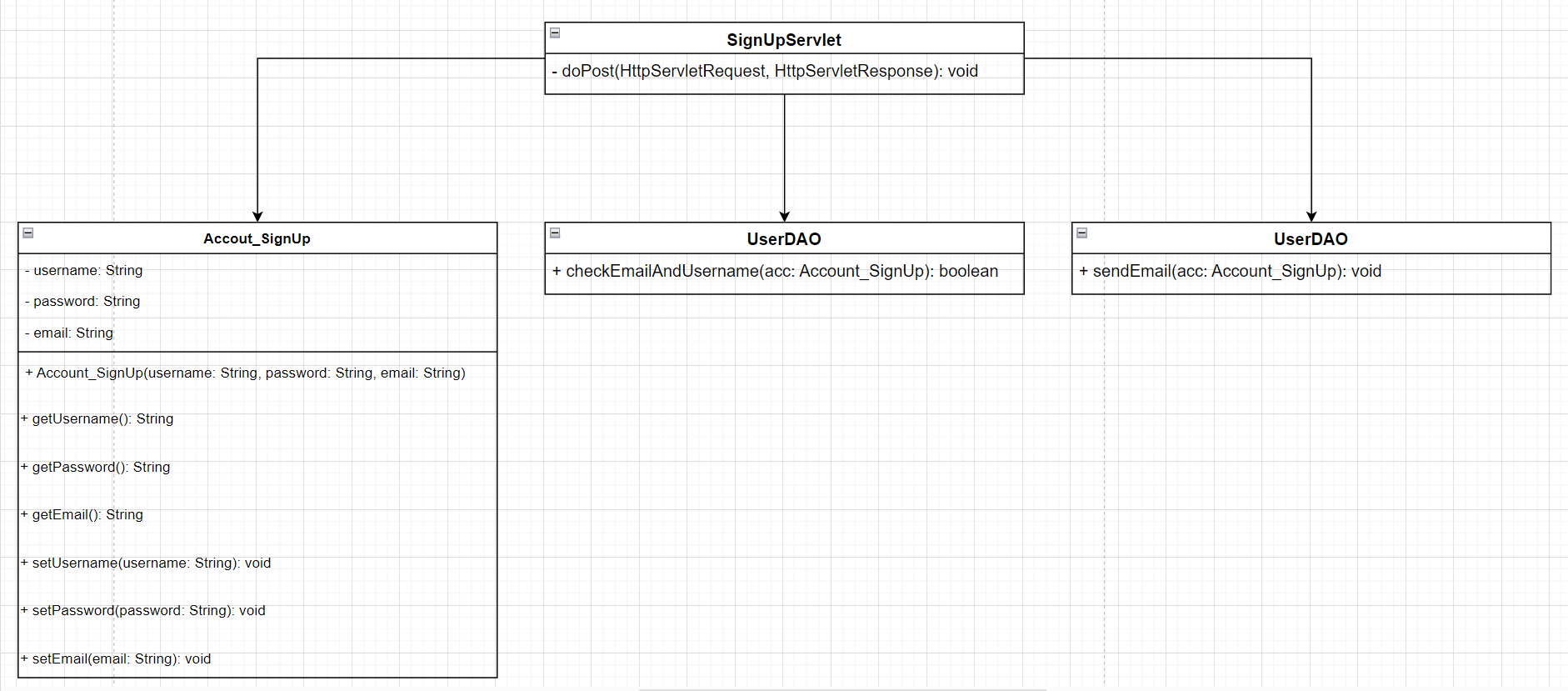
### b. Table Description

| **No** | **Table** | **Description** |
| --- | --- | --- |
| 01 | User | This table contains account’s information. |
| 02 | Content Creator | This table contains information about content. |
| 03 | Cart User | This table contains detailed information about the products in the cart. |
| 04 | Cart | This table contains the information of the shopping cart. |
| 05 | Product | This table contains information about the product. |
| 06 | Product Detail | This table contains detailed product information. |
| 07 | Price | This table contains information about the price of the product. |
| 08 | Order | This table contains the details of the order. |
| 09 | Order item | This table contains the information of the order. |
| 10 | FeeShip | This table contains list of cities that support delivery and our shipping price |
| 11 | Rank | This table contains list and data of customer rank |

# II. Code Designs

## 1. Account Registration

### a. Class Diagram



### b. Class Specifications

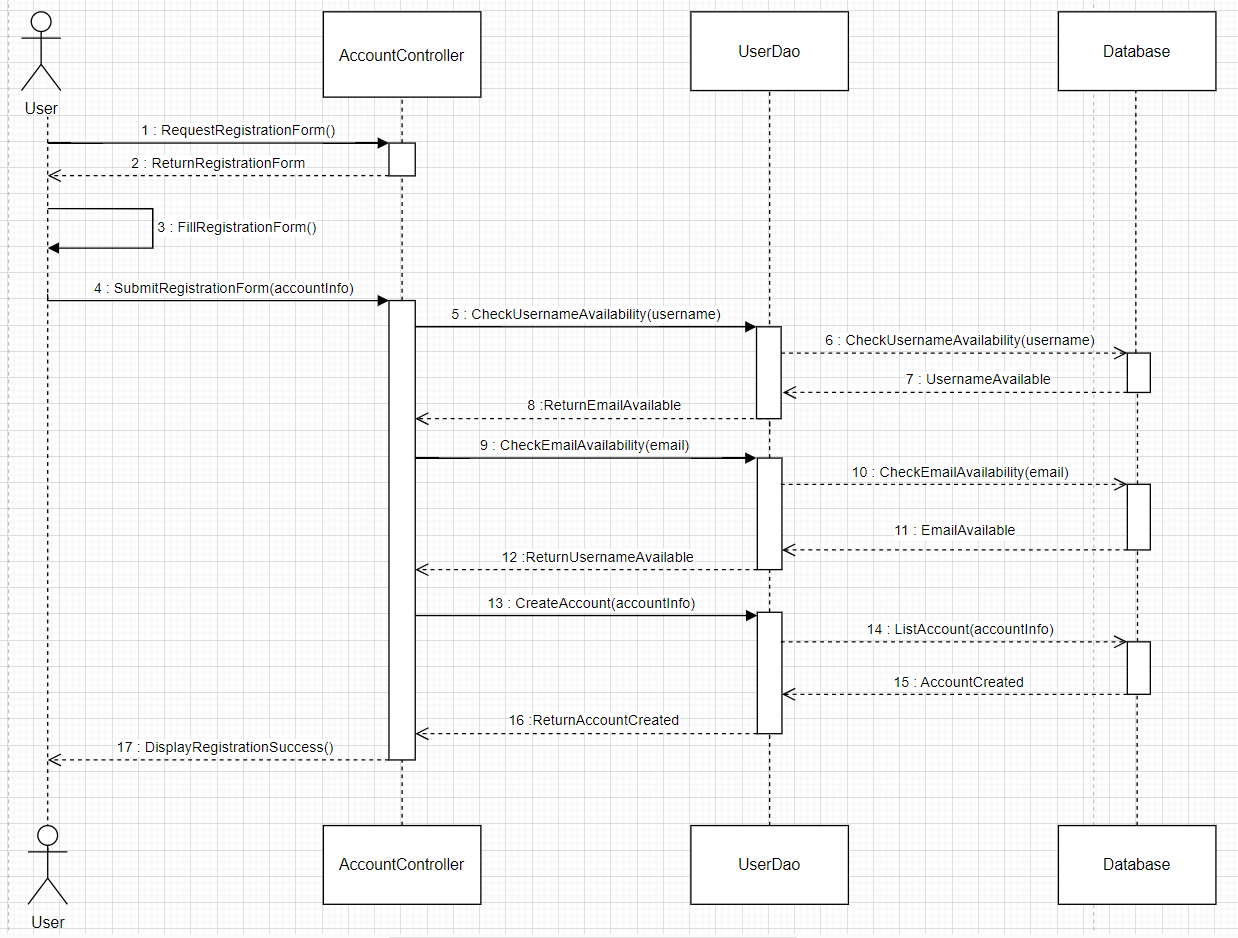
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *username* | *Used to represent the username during account registration.* |
| *2* | *password* | *Used to represent the user's password during account registration.* |
| *3* | *email* | *Used to represent the user's email address during account registration.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Account\_SignUp* | *A class that represents registered account information. It is used to create an object containing account information from data retrieved from the user interface.* |
| *2* | *getUsername()* | *Used to return the value of the username attribute. This is a getter method that provides public access to the value of the username property from outside the Account\_SignUp class.* |
| *3* | *getPassword()* | *Used to return the value of the password attribute. This is a getter method that provides public access to the value of the password property from outside the Account\_SignUp class.* |
| *4* | *getEmail()* | *Used to return the value of the email attribute. This is a getter method that provides public access to the value of the email property from outside the Account\_SignUp class.* |
| *5* | *setUsername()* | *Used to set the value for the username attribute. This is a setter method that allows changing the value of username from outside the Account\_SignUp class.* |
| *6* | *setPassword()* | *Used to set the value for the password attribute. This is a setter method that allows changing the value of the password from outside the Account\_SignUp class.* |
| *7* | *setEmail()* | *Used to set the value for the email attribute. This is a setter method that allows the email value to be changed from outside the Account\_SignUp class.* |
| *8* | *doPost* | *Called when a POST request is sent to the servlet from the front end. This is where the account registration request is handled and related operations are performed.* |
| *9* | *checkEmailAndUsername* | *Used to check if an email address and username already exist in the database.* |
| *10* | *sendEmail* | *Used to send an authentication email to the email address provided in the Account\_SignUp object. The main function of this method is to send a confirmation email to verify and authenticate the user's account.* |

### c. Sequence Diagram(s)

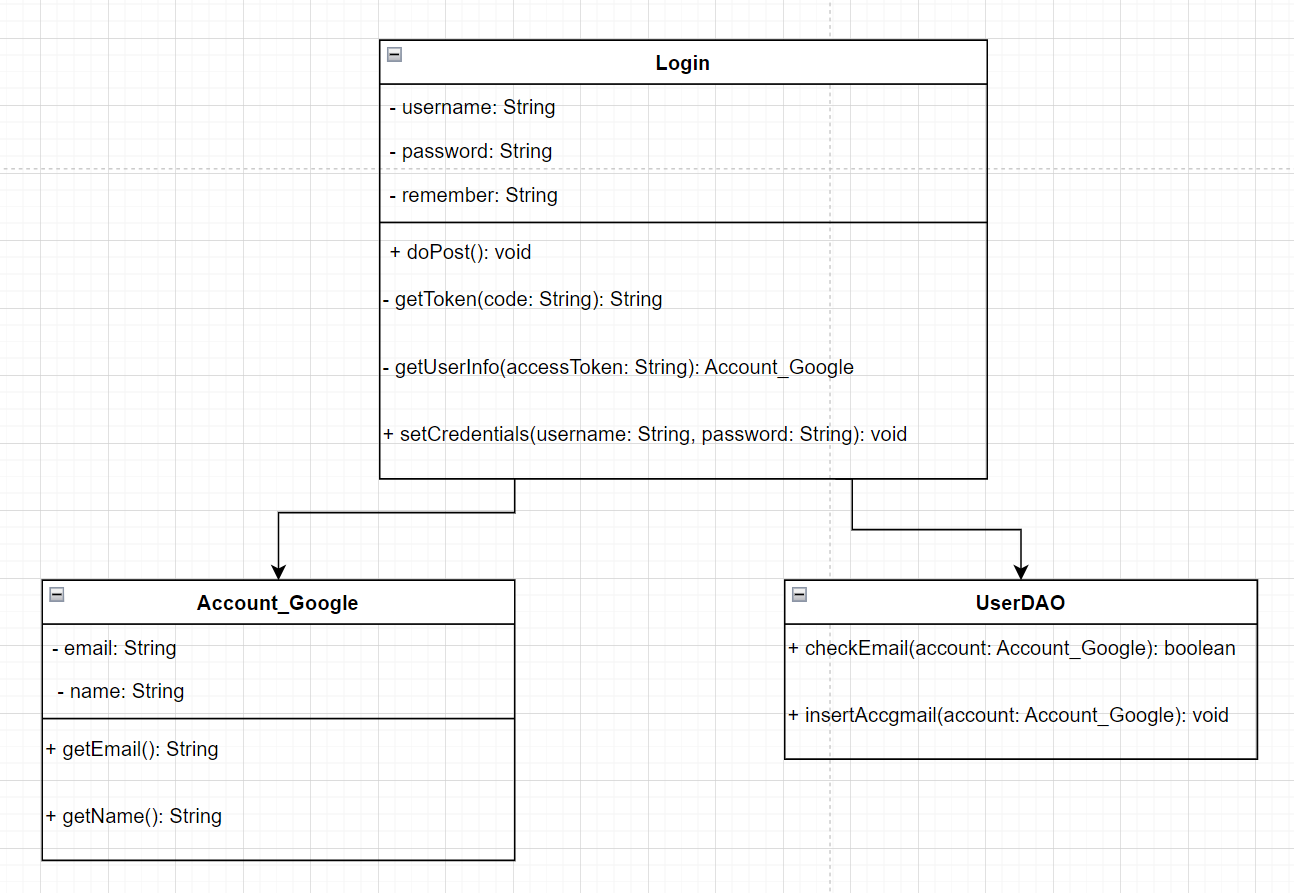


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT COUNT(\*) FROM Users WHERE username = ''; | Check if the username already exists in the database |
| 2 | SELECT COUNT(\*) FROM User WHERE email = ''; | Check if the email address already exists in the database |
| 3 | INSERT INTO Users (username, password, email) VALUES ('', '', ''); | Add new account to database |

## 2. Log in

### a. Class Diagram



### b. Class Specifications

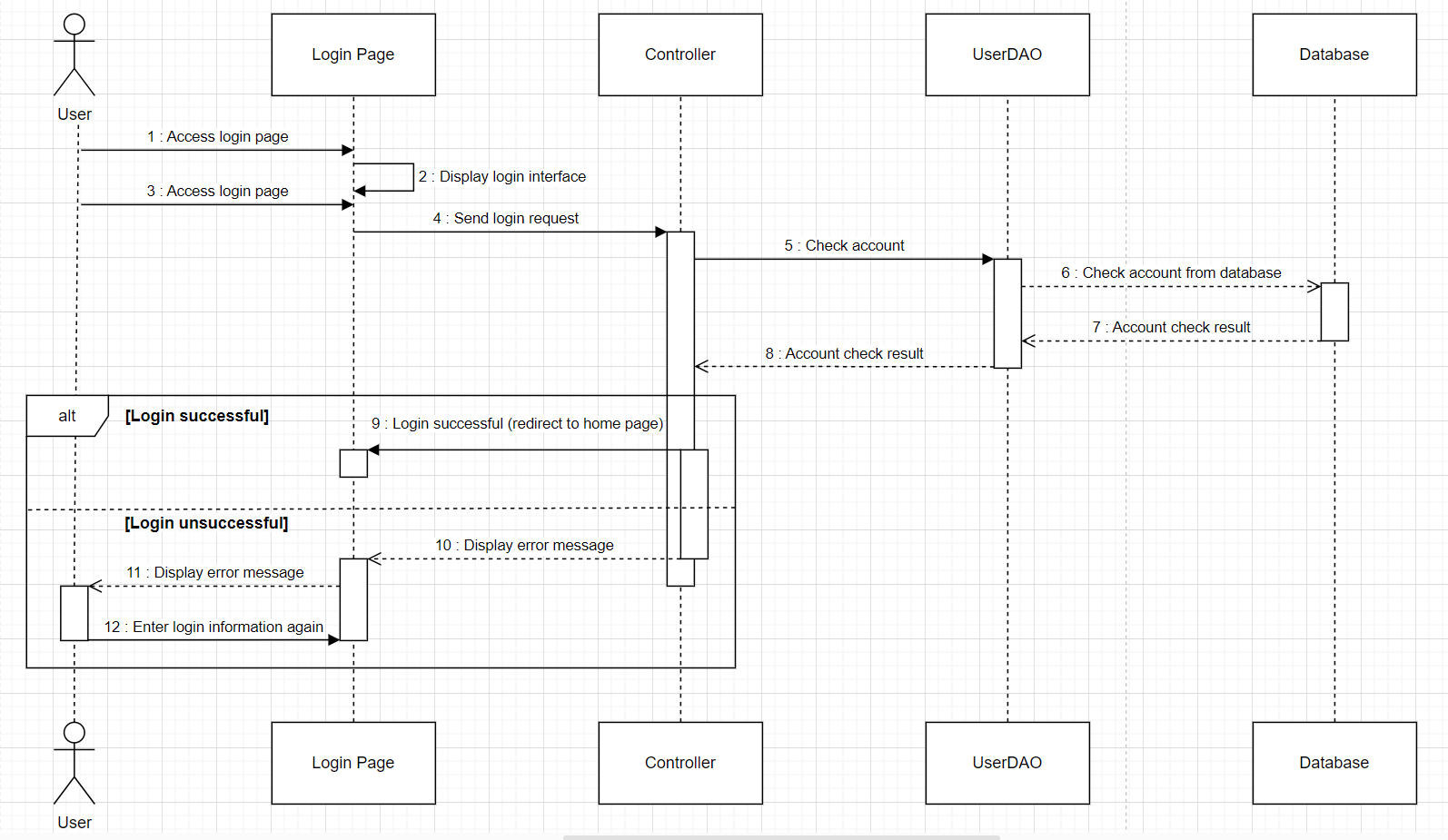
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *username* | *Used to represent the username during account login.* |
| *2* | *password* | *Used to represent the user's password during account login.* |
| *3* | *email* | *Used to represent the user's email address during account login.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *checkEmail()* | *Used to check if an email address from a Google account already exists in the database.* |
| *2* | *insertAccgmail()* | *Used to insert user information from Google accounts into the database.* |
| *3* | *doPost()* | *Used to handle POST requests from UI during login.* |
| *4* | *getToken()* | *Used to get access tokens from authorization code received from Google.* |
| *5* | *getUserInfo* | *Used to get user information from Google accounts after having access to the token.* |
| *6* | *setCredentials()* | *Used to establish credentials such as username and password, access token, or other information related to the authentication process.* |
| *7* | *getEmail()* | *A getter method used to get the user's email address from the Google account.* |
| *8* | *getName()* | *A getter method used to get the user's email address from the Google account.* |

### c. Sequence Diagram(s)

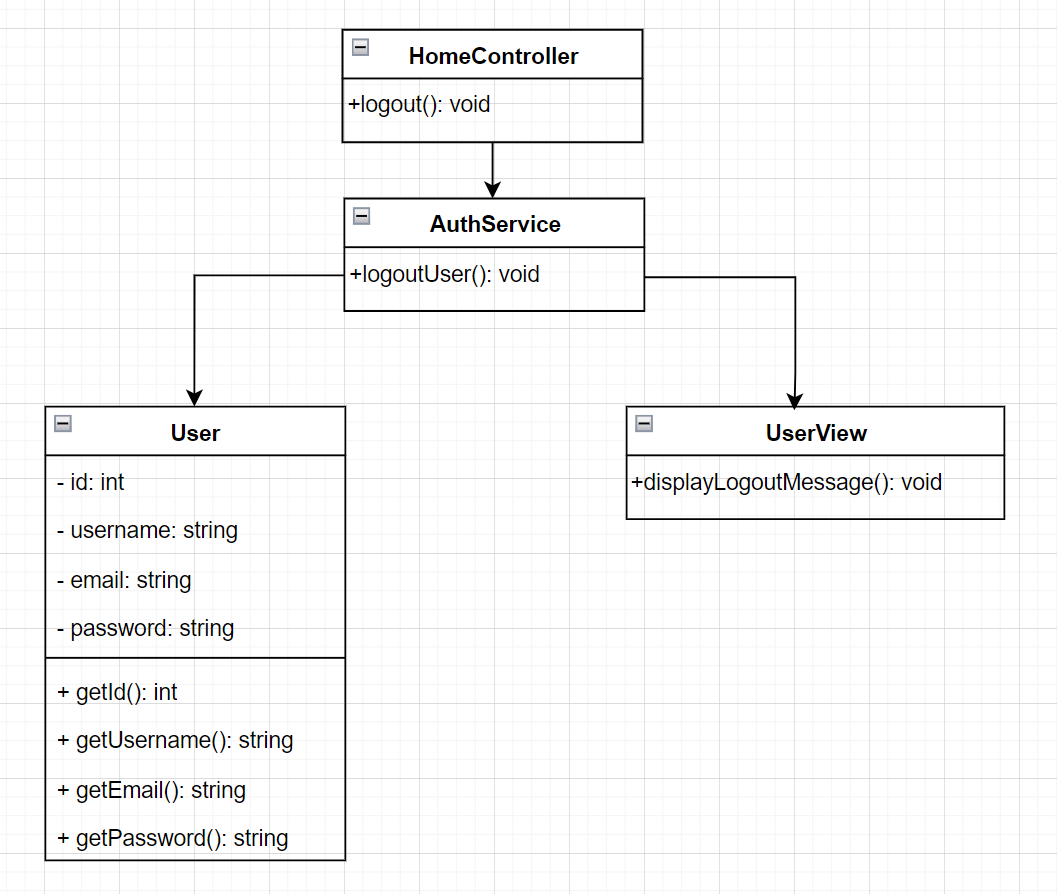


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | String query = "SELECT COUNT(\*) FROM users WHERE email = ? AND username = ?"; | Query to check if email and username exist in user table |
| 2 | String query = "UPDATE users SET username = ?, password = ? WHERE email = ?"; | Query to update user information in the users table |
| 3 | String query = "DELETE FROM users WHERE email = ?"; | Query delete user from users table |

## 3. Log out

### a. Class Diagram

**

### b. Class Specifications

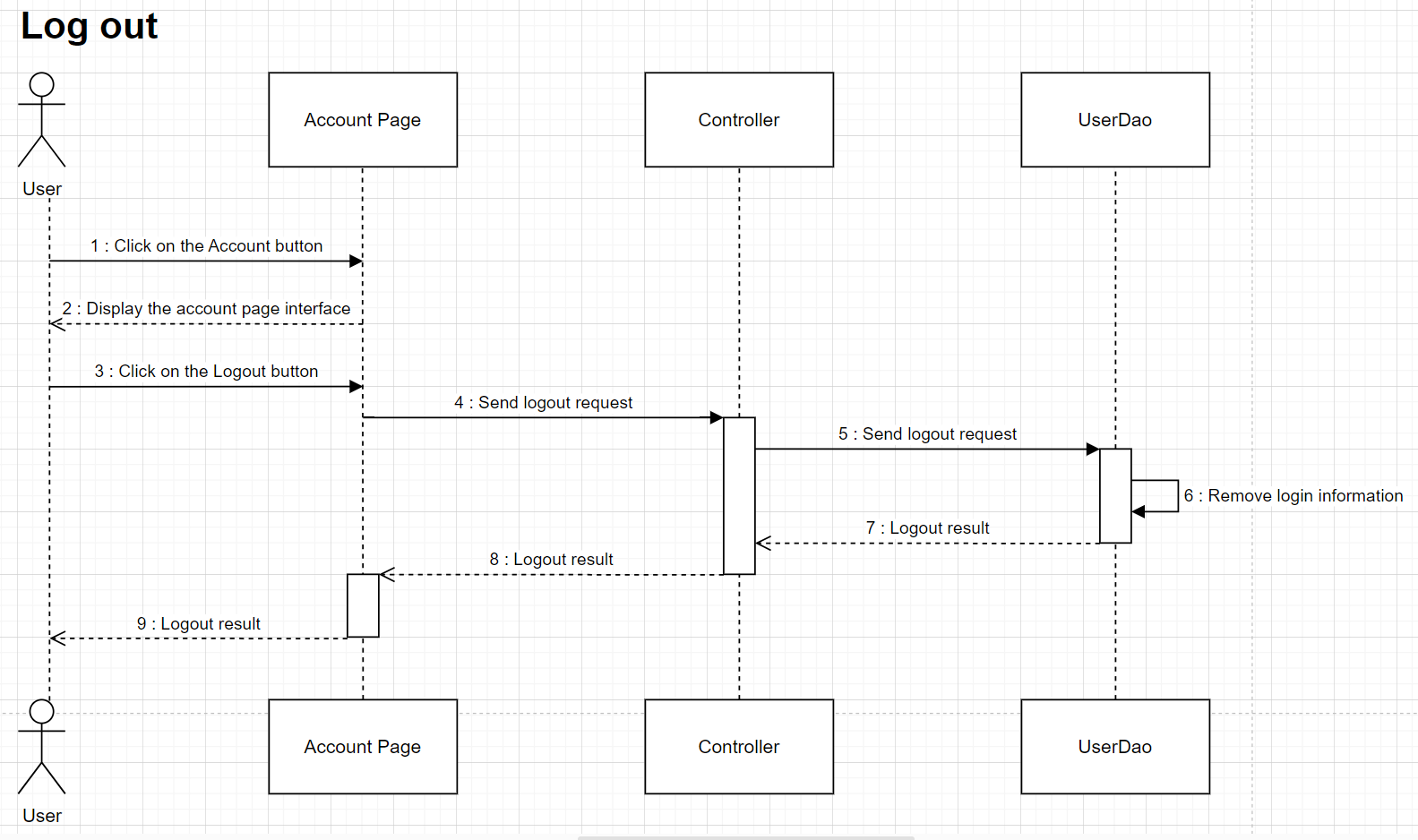
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *id* | *Used to uniquely identify an object in the system.* |
| *2* | *username* | *Username information is used to identify the currently logged out user.* |
| *3* | *email* | *Used to identify the user during login or authentication.* |
| *4* | *password* | *Used to authenticate the user and check the validity of the password* |

#### 

***Class Methods***

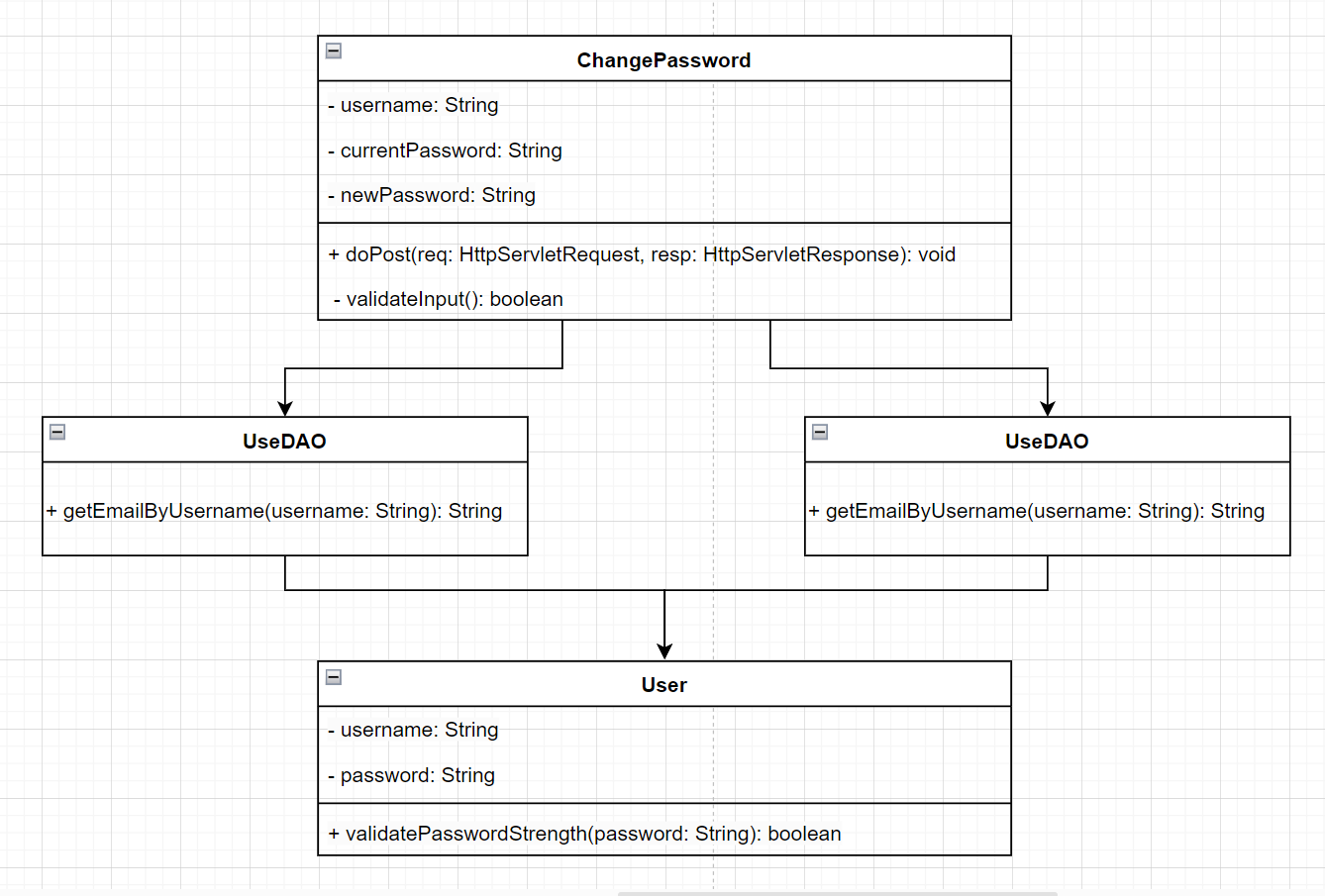
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *logoutUser()* | *Used to perform user logout operations.* |
| *2* | *getId()* | *Used to return a unique value representing an object or an element in the system* |
| *3* | *getUsername()* | *Used to retrieve and return the value of the "username" property of an object or another component in the application* |
| *4* | *getEmail()* | *Used to retrieve and return the value of the "email" property of an object or another application element* |
| *5* | *getPassword()* | *Used to retrieve and return the value of the "password" property of an object or another application element* |
| *6* | *displayLogoutMessage()* | *Used to display a notification or message to notify the user that they are logged out of the application* |

### c. Sequence Diagram(s)

**

## 4. Change password

### a. Class Diagram



### b. Class Specifications

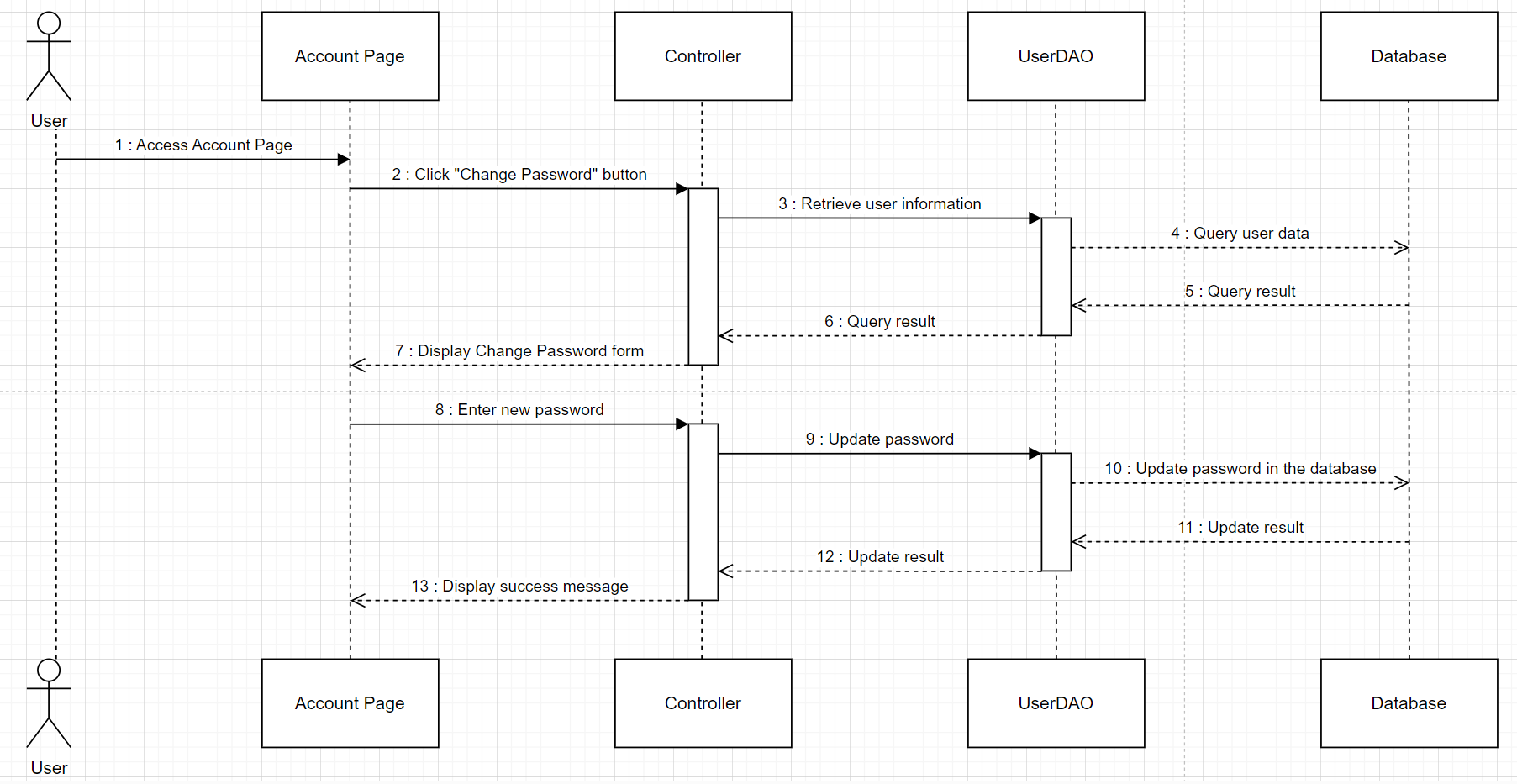
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *username* | *Used to identify the current user changing the password* |
| *2* | *currentPassword* | *Represents the current password that the user is using in the system* |
| *3* | *newPassword* | *Represents the new password that the user wants to set in the system* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *doPost()* | *Handle POST request when user submits password change information* |
| *2* | *validateInput()* | *Used to check the validity of input data during password change* |
| *3* | *getEmailByUsername()* | *Get the user's email address from the database based on the provided username* |
| *4* | *validatePassword()* | *Used to apply requirements and rules to ensure passwords meet safety and security criteria* |

### c. Sequence Diagram(s)

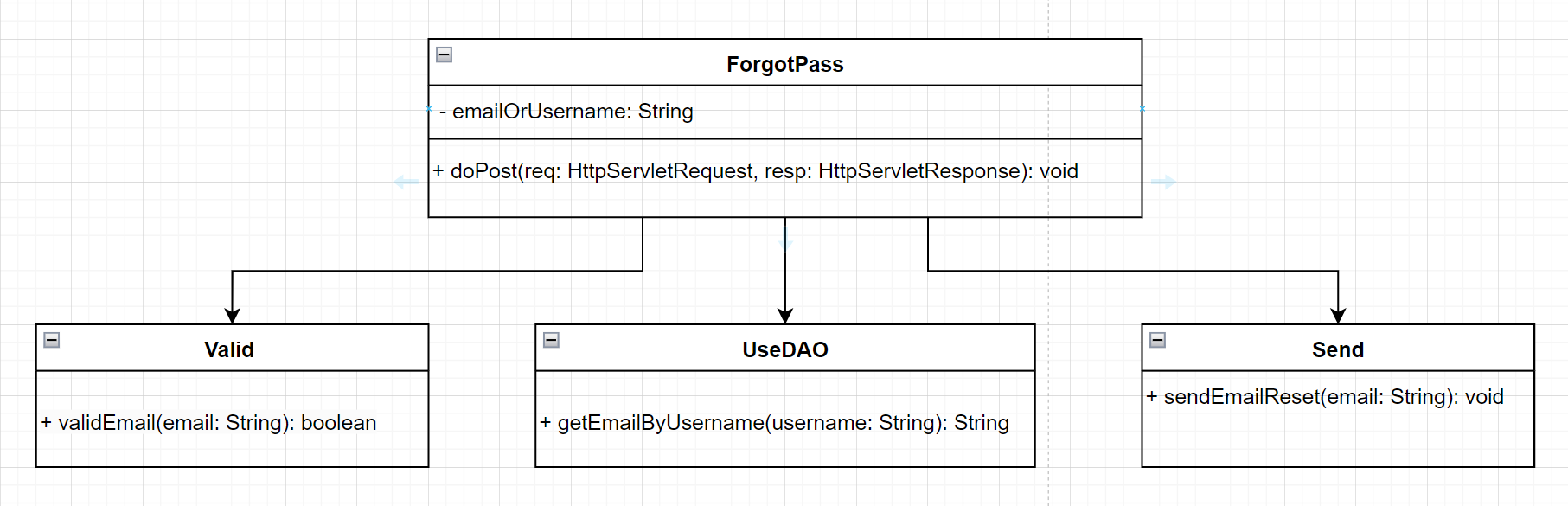


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT password FROM users WHERE username = ? | Query to get user's current password from database |
| 2 | UPDATE users SET password = ? WHERE username = ? | User's new password update query |
| 3 | INSERT INTO notifications (username, title, content) VALUES (?, ?, ?) | Query more notifications or record password change activity |

## 5. Forgot password

### a. Class Diagram

**

### b. Class Specifications

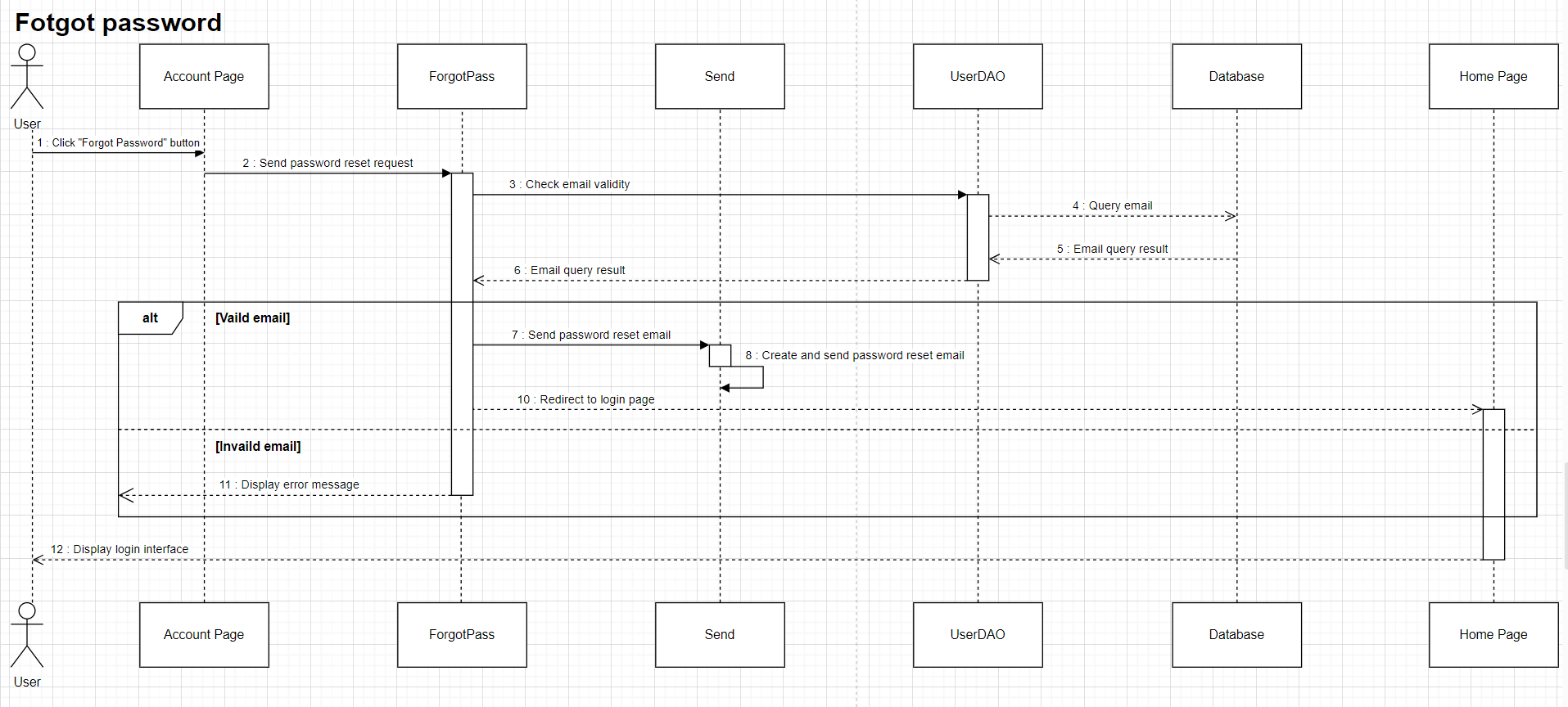
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *emailOrUsername* | *Represents the value a user provides to identify their account when they can't remember the password* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *doPost()* | *Validates the user's current password by comparing it with the entered currentPass value.* |
| *2* | *vailEmail()* | *Used to ensure that email addresses are entered in the correct format* |
| *3* | *getEmailByUsername()* | *Implemented to get a user's email address based on username (username)* |
| *4* | *sendEmailReset()* | *Implemented to send password recovery emails to users* |

### c. Sequence Diagram(s)

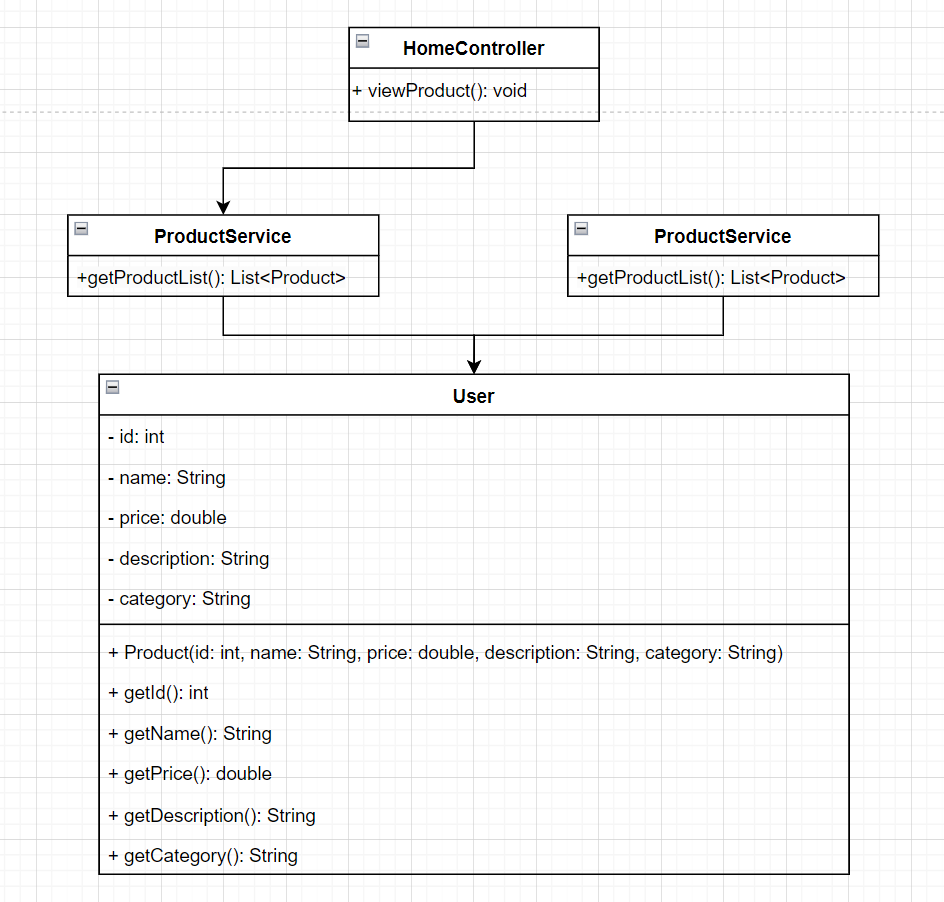


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT COUNT(\*) FROM users WHERE email = ? OR username = ? | Query to check if email address or username already exists in database |
| 2 | SELECT \* FROM users WHERE email = ? OR username = ? | Query to get user information based on email address or username |
| 3 | UPDATE users SET reset\_token = ? WHERE email = ? OR username = ? | Query to update password recovery token for user |
| 4 | SELECT COUNT(\*) FROM users WHERE reset\_token = ? AND (email = ? OR username = ?) | Query to check password recovery token validity |
| 5 | UPDATE users SET password = ? WHERE email = ? OR username = ? | Query to update new password for user |

## 6. View products

### a. Class Diagram



### b. Class Specifications

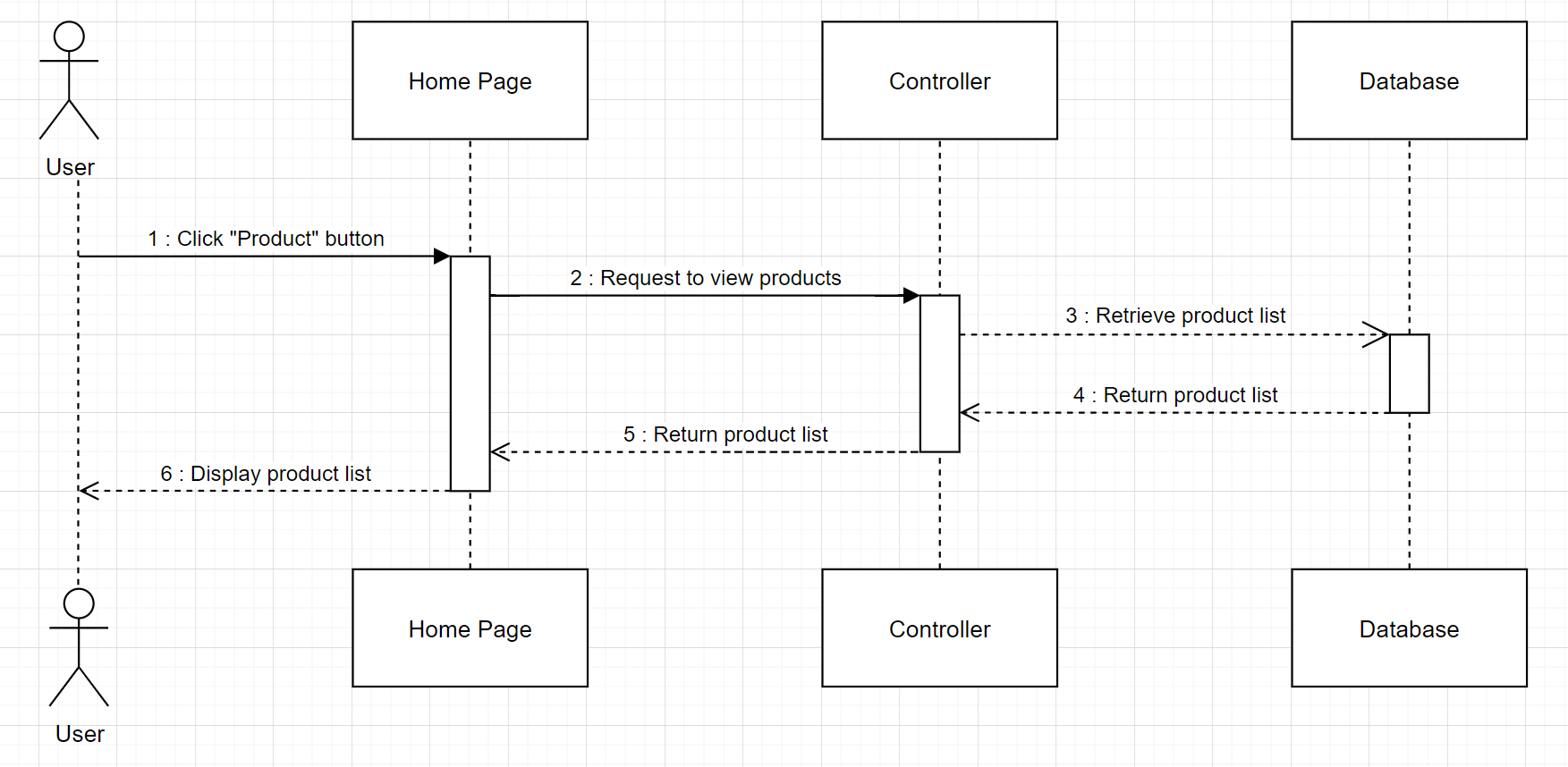
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *id* | *Unique value used to identify a particular product in a database or system* |
| *2* | *name* | *Product name is a basic attribute to identify and display the name of a particular product* |
| *3* | *price* | *Used to return the price of the product* |
| *4* | *description* | *Provide detailed information and product descriptions to users* |
| *5* | *category* | *Categories help organize and categorize products into corresponding groups, making it easy for users to search and view products in the same group.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Product()* | *Used to retrieve detailed product information and perform related tasks* |
| *2* | *getId()* | *Used to identify products in a database or system* |
| *3* | *getName()* | *Used to set the name value for the product* |
| *4* | *getPrice()* | *Used to set the price value for the product* |
| *5* | *getDescription* | *Used to set a description value for the product* |
| *6* | *getCategory()* | *Used to set the category value for the product.* |
| *7* | *viewProduct()* | *Used to display detailed information about a particular product.* |
| *8* | *getProductList()* | *Used to get product list* |

### c. Sequence Diagram(s)

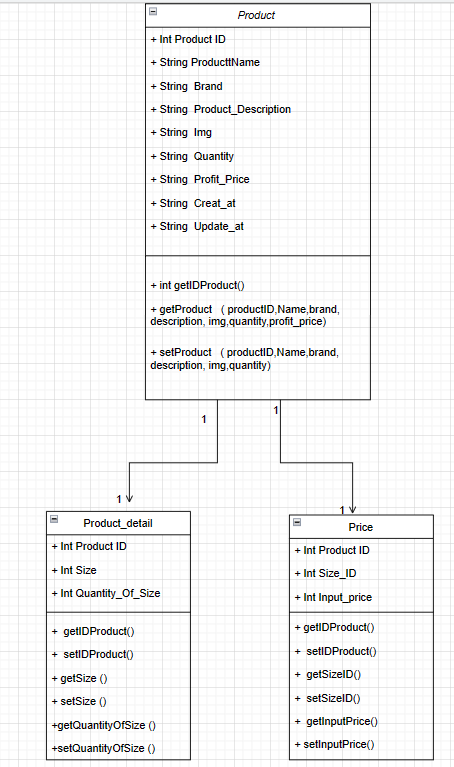
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT \* FROM products; | Query get all products |
| 2 | SELECT \* FROM products WHERE id = ?; | Query to get products by ID |
| 3 | SELECT \* FROM products WHERE category = ?; | Query to get products by category |
| 4 | SELECT \* FROM products WHERE name LIKE '%?%'; | Query to get products by name |

## 7. Search products

### a. Class Diagram

**

### b. Class Specifications

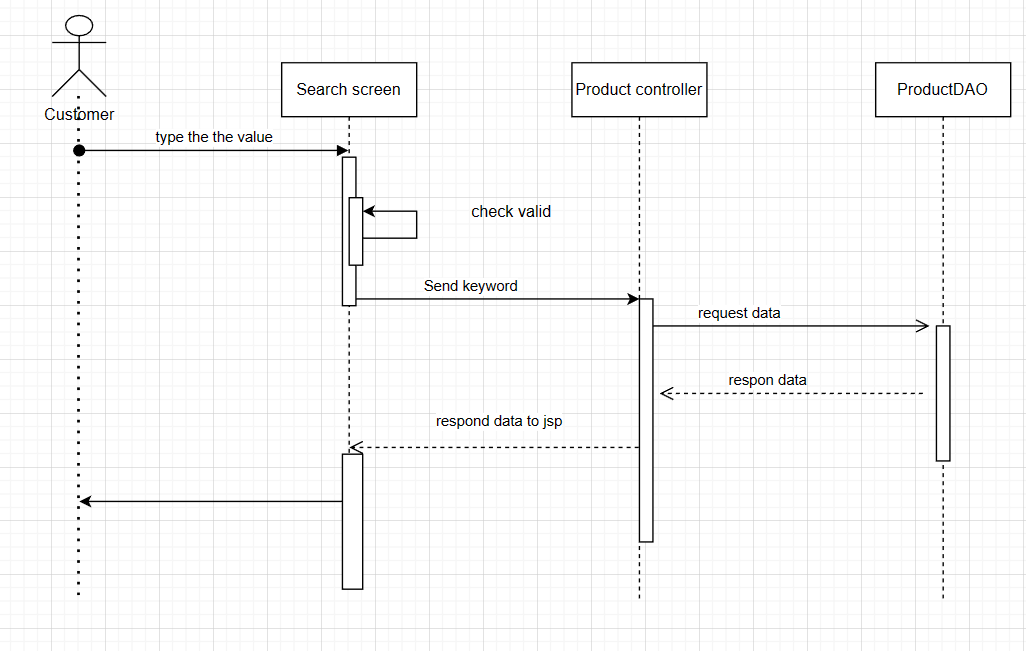
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Product ID* | *To identify the product id* |
| *2* | *ProductName* | *To identify the product name* |
| *3* | *Brand* | *To identify the brand name of that product* |
| *4* | *Product\_Description* | *To contain the description for the product* |
| *5* | *img* | *to contain the product image link* |
| *6* | *quantity* | *to contain the quantity of the product* |
| *7* | *Profit\_price* | *To contain the actual selling price of the product* |
| *8* | *Create-at* | *to contain the product creation date* |
| *9* | *Update-at* | *to keep the product update date fixed* |
| *10* | *Size* | *Contains the size of the product* |
| *11* | *Quantity\_Of\_Size* | *contains the number of products of that size* |
| *12* | *Size ID* | *Contains the ID of that size* |
| *13* | *input price* | *Contains the entry price of the product* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *getIDProduct()* | *Method to get product id* |
| *2* | *getProduct()* | *method to get product information* |
| *3* | *setProduct()* | *method to edit product information* |
| *4* | *getSize()* | *method to get the size of the product* |
| *5* | *setSize()* | *method to fix the size of the product* |
| *6* | *get& setquatityofSize()* | *method to get and fix the number of products by size* |
| *7* | *get&setInputPrice()* | *method to get and edit the input price* |

### c. Sequence Diagram(s)

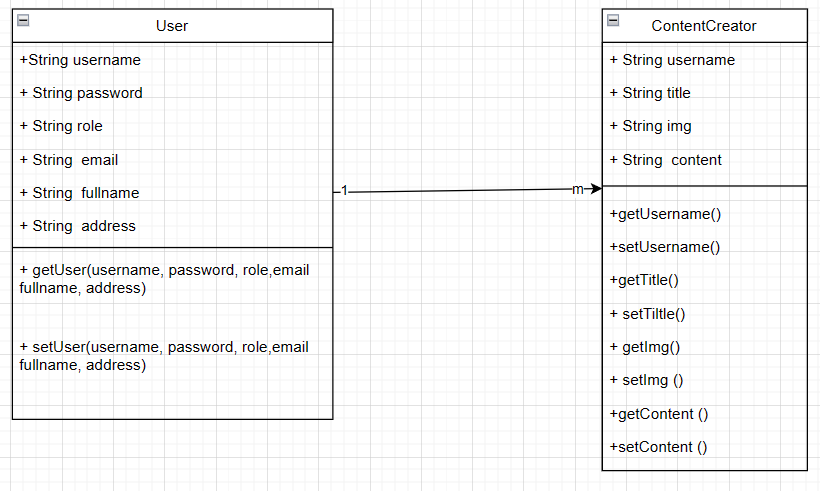
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Select \* from Product\_Table | list all product |
| 2 | while(rs.next()){  if(rs.getString(2).toUpperCase().contains(search.toUpperCase())){  Product p = new Product(rs.getInt(1), rs.getString(2), rs.getString(3),rs.getString(4), rs.getFloat(5), rs.getString(6), rs.getInt(7), rs.getDate(8), rs.getDate(9), rs.getString(10));  list.add(p);  }  } | list prroduct same name in the search box |

## 8. Read contents

### a. Class Diagram



### b. Class Specifications

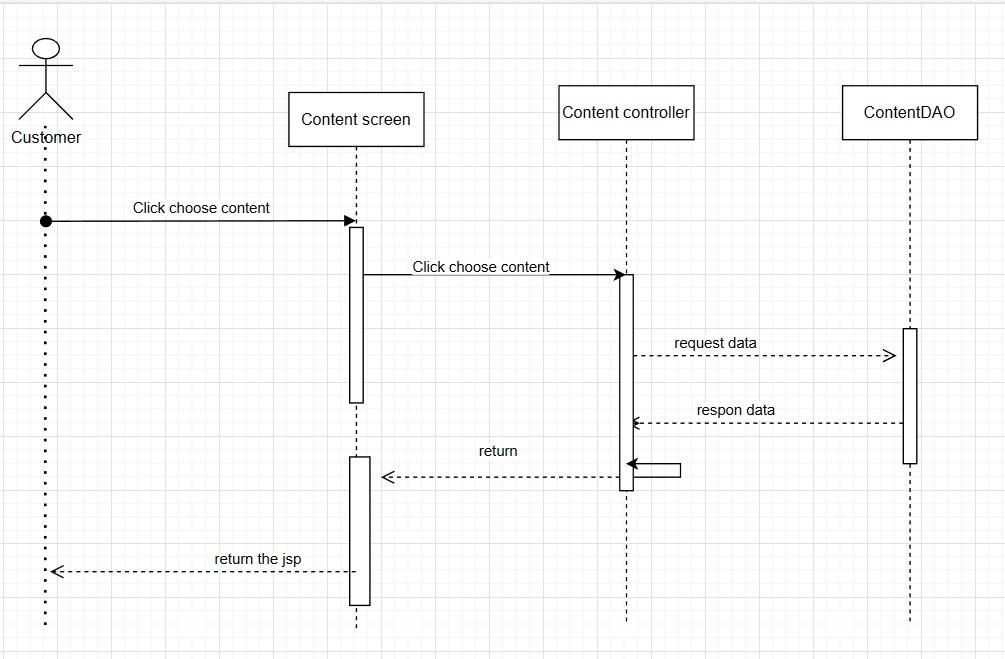
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Username* | *To store the username of the account* |
| *2* | *password* | *Leave the password of the account* |
| *3* | *role* | *to save the account's role on the system* |
| *4* | *email* | *To save the account's email* |
| *5* | *fullname* | *To contain the full name of the account holder* |
| *6* | *address* | *To store customer address* |
| *7* | *tittle* | *To contain the name of the post* |
| *8* | *img* | *to include the link of the image for the post* |
| *9* | *content* | *To contain the content of the post* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *GET & SET user* | *method to get or edit user details* |
| *2* | *GET & SET TItle* | *Method to get or edit post title* |
| *3* | *GET & SET img* | *method to get or edit a post's image* |
| *4* | *GET & SET Content* | *Method to get or edit the content of a post* |

### c. Sequence Diagram(s)

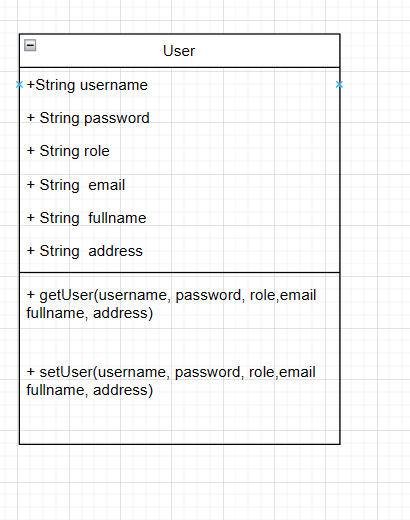


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT \* FROM ContentCreator\_Table WHERE content\_id = ? | Get content by post ID |
| 2 | Select \* from ContentCreator\_Table | list of all posts |

## 9. View personal information

### a. Class Diagram

**

### b. Class Specifications

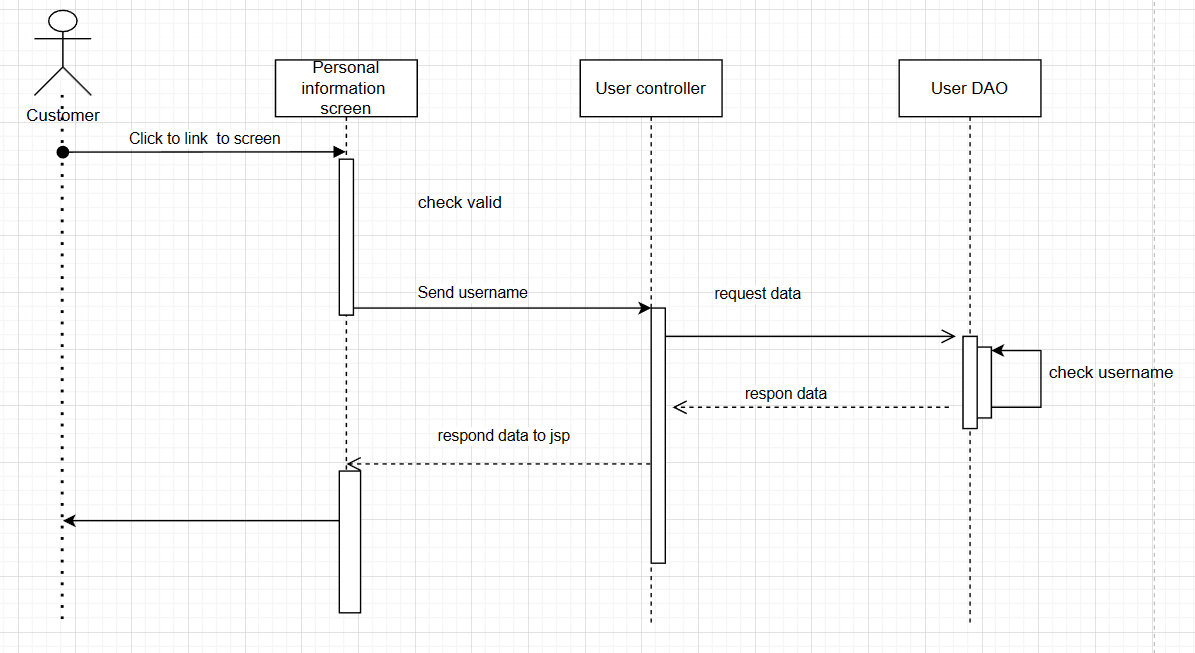
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Username* | *To store the username of the account* |
| *2* | *password* | *Leave the password of the account* |
| *3* | *role* | *to save the account's role on the system* |
| *4* | *email* | *To save the account's email* |
| *5* | *full name* | *From containing the full name of the account holder* |
| *6* | *address* | *To store customer address* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *GET & SET user* | *method to get or edit user details* |

### c. Sequence Diagram(s)

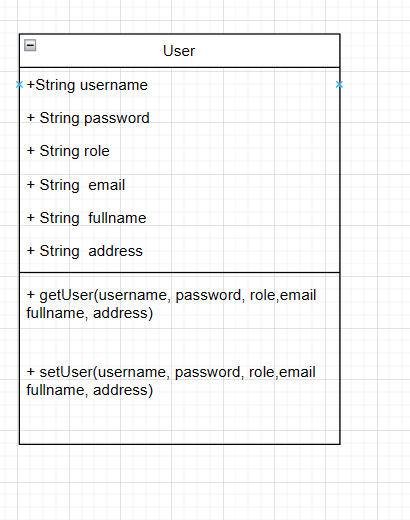
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | "Select username , password , email From User\_Table” | list user in database |
| 2 | Select username, role, email, fullname, address, phone , img From User\_Table where username = ? | list of user information by username |

## 10. Update personal information

### a. Class Diagram

**

### b. Class Specifications

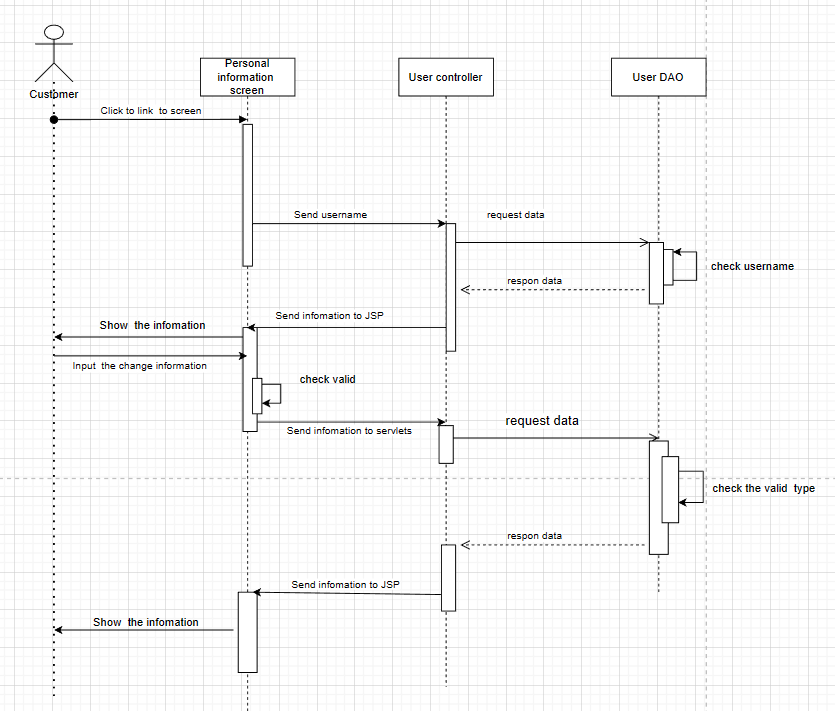
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Username* | *To store the username of the account* |
| *2* | *password* | *Leave the password of the account* |
| *3* | *role* | *to save the account's role on the system* |
| *4* | *email* | *To save the account's email* |
| *5* | *fullname* | *To contain the full name of the account holder* |
| *6* | *address* | *To store customer address* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *GET & SET user* | *method to get or modify the details of user* |

### c. Sequence Diagram(s)



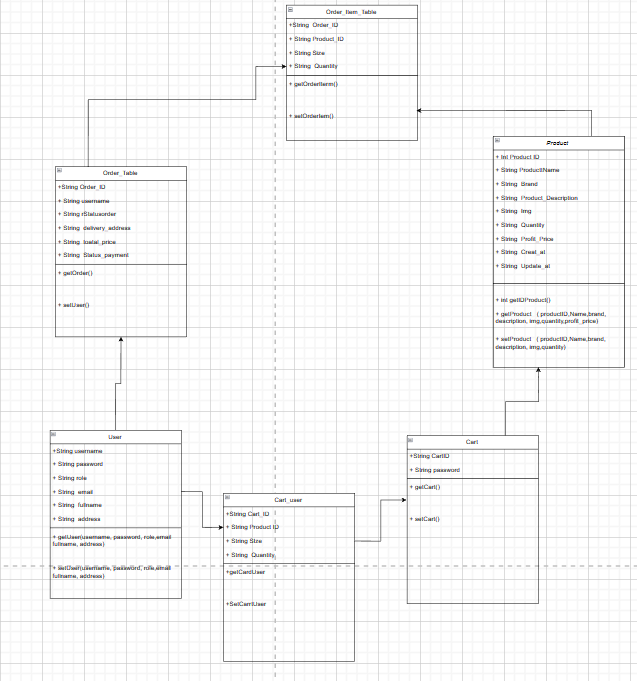
### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| *1* | *Update User\_Table Set fullname = ? , address = ? , phone = ? , img = ? where username = ?* | *edit the information of the previously registered account* |

## 

## 11. Purchase

### a. Class Diagram

**

### b. Class Specifications

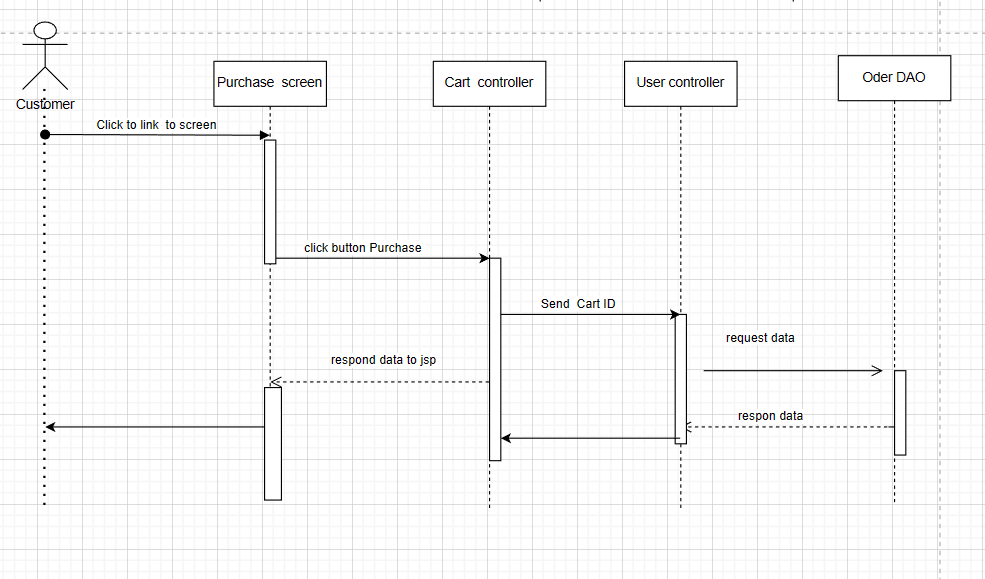
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1.* | *Order\_ID* | *Contains the id of the order* |
| *2* | *Product\_ID* | *contains the id of the product* |
| *3* | *Size* | *contains the id of the product* |
| *4* | *Quantity* | *Contains product quantity* |
| *5* | *username* | *contains the customer's user name* |
| *6* | *Status order* | *Contains a status of order* |
| *7* | *delivery\_address* | *contains the desired delivery address of the product* |
| *8* | *total\_price* | *contains the total price of the order* |
| *9* | *Status\_payment* | *contains the status of the order* |
| *10* | *password* | *contains the password of the login key* |
| *11* | *role* | *contains the role of that account* |
| *12* | *email* | *contains the email of that account* |
| *13* | *fullname* | *contains the full name of the account* |
| *14* | *address* | *contains the account's address* |
| *15* | *product name* | *contains the name of the product* |
| *16* | *Cart\_ID* | *contains the ID of the cart* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Getter &setter USER* | *Method to get and edit user's content* |
| *2* | *Getter&setter Cart* | *Method to get and edit content of Cart* |
| *3* | *Getter&setter Product* | *Method to get and edit the contents of the Product* |

### c. Sequence Diagram(s)

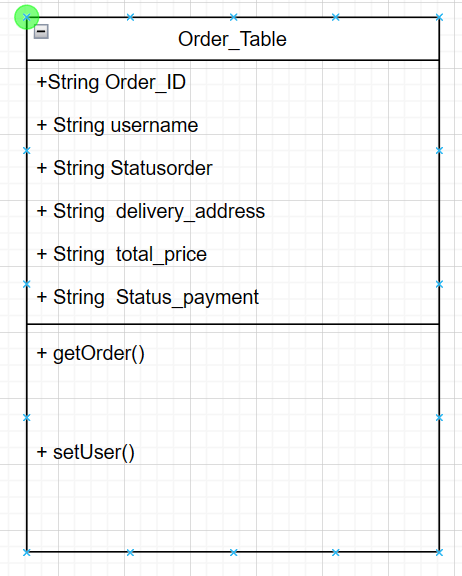


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Update CartUser\_Table Set quantity = quantity + ? where cart\_id = ? and product\_id = ? and size = ? | to update information in cart\_user |
| 2 | Select \* from CartUser\_Table where cart\_id = ? | get cartUser information by cart ID |
| 3 | Select username, role, email, fullname, address, phone , img From User\_Table where username = ? | get information of customer in table user\_table |

## 12. Choose payment method

### a. Class Diagram

**

### b. Class Specifications

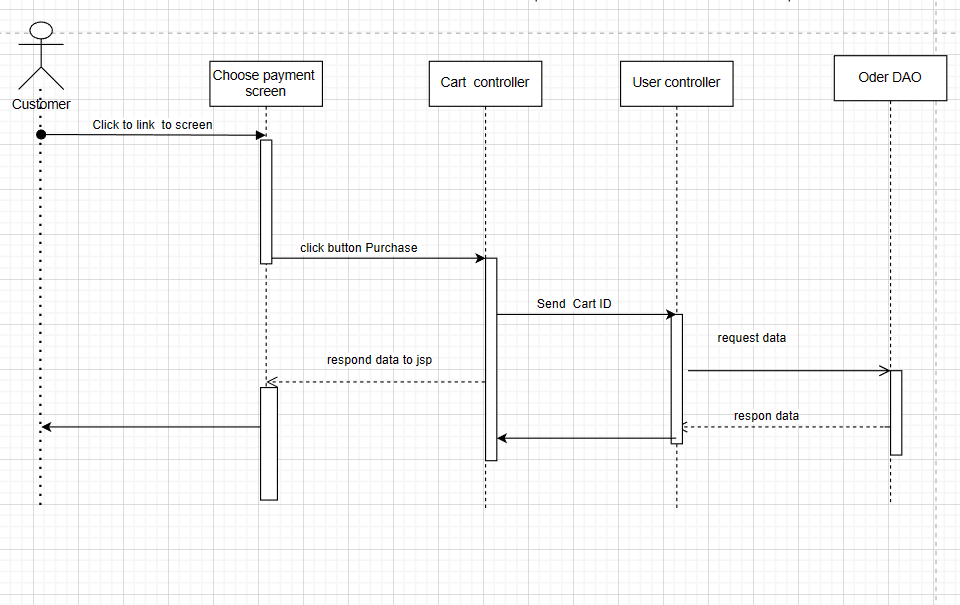
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Order\_ID* | *contains the ID of the order* |
| *2* | *username* | *contains the user name of the account* |
| *3* | *Status\_order* | *contains the status of the order* |
| *4* | *delivery\_address* | *contains the order address* |
| *5* | *total\_price* | *contains the total price of the order* |
| *6* | *status\_payment* | *contains payment status* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *getOrder* | *get order data* |
| *2* | *setOrder* | *edit the data of the order table* |

### c. Sequence Diagram(s)

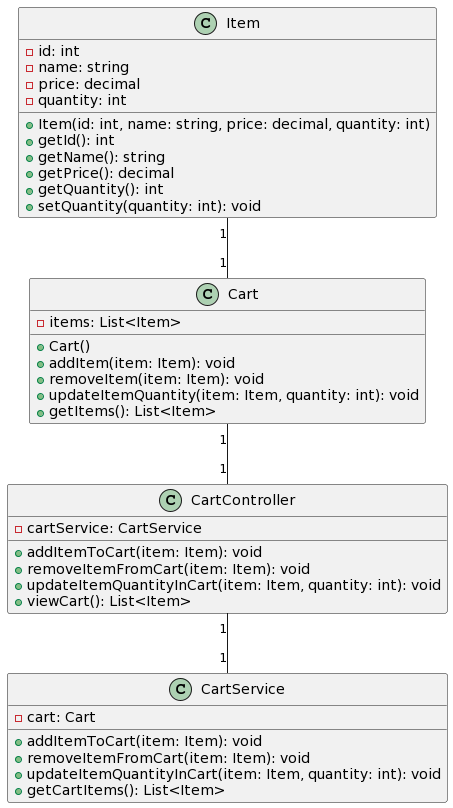


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Update CartUser\_Table Set quantity = quantity + ? where cart\_id = ? and product\_id = ? and size = ? | to update information in cart\_user |
| 2 | Select \* from CartUser\_Table where cart\_id = ? | get information of cartUser by cart ID |
| 3 | Select username, role, email, fullname, address, phone , img From User\_Table where username = ? | get information of customer in table user\_table |

## 13.CRUD items in cart

### a. Class Diagram



### b. Class Specifications

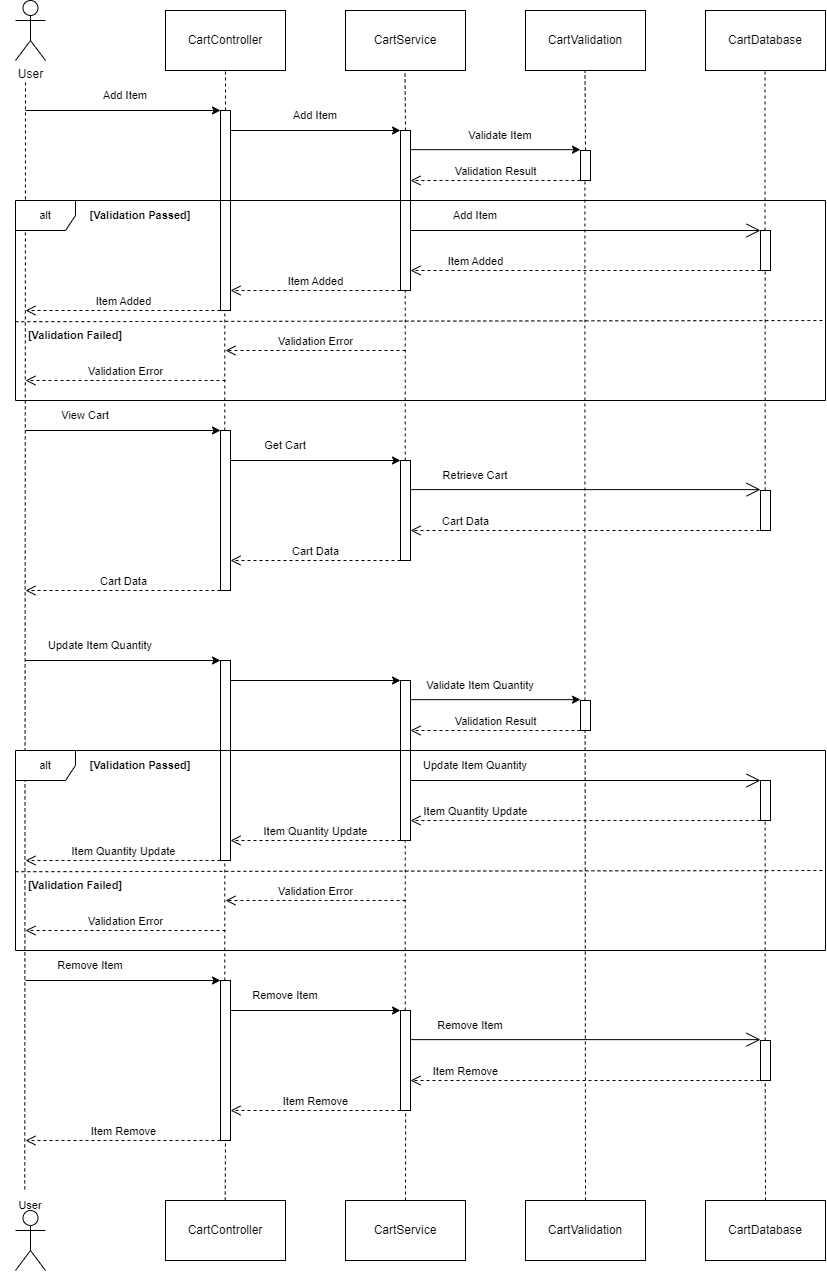
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | id (int) | The ID of the item. |
| 2 | name (string) | The name of the item. |
| 3 | price (decimal) | The price of the item. |
| 4 | quantity (int) | The quantity of the item in the cart. |
| 5 | items (List<Item>) | The list of items in the cart. |
| 6 | cartService (CartService) | An instance of the CartService class. |
| 7 | cart (Cart) | An instance of the Cart class. |

#### 

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | Item(id: int, name: string, price: decimal, quantity: int) | Constructor method to create a new Item object with the provided attributes. |
| 2 | getId() | Retrieves the ID of the item. |
| 3 | getName() | Retrieves the name of the item. |
| 4 | getPrice() | Retrieves the price of the item. |
| 5 | getQuantity() | Retrieves the quantity of the item in the cart. |
| 6 | setQuantity(quantity: int) | Sets/updates the quantity of the item in the cart. |
| 7 | Cart() | Constructor method to create a new Cart object. |
| 8 | addItem(item: Item) | Adds an item to the cart. |
| 9 | removeItem(item: Item) | Removes an item from the cart. |
| 10 | updateItemQuantity(item: Item, quantity: int) | Updates the quantity of a specific item in the cart. |
| 11 | getItems() | Retrieves the list of items in the cart. |
| 12 | addItemToCart(item: Item) | Adds an item to the cart using the cartService. |
| 13 | removeItemFromCart(item: Item) | Removes an item from the cart using the cartService. |
| 14 | updateItemQuantityInCart(item: Item, quantity: int) | Updates the quantity of a specific item in the cart using the cartService. |
| 15 | viewCart() | Retrieves the list of items in the cart using the cartService. |
| 16 | addItemToCart(item: Item) | Adds an item to the cart. |
| 17 | removeItemFromCart(item: Item) | Removes an item from the cart. |
| 18 | updateItemQuantityInCart(item: Item, quantity: int) | Updates the quantity of a specific item in the cart. |
| 19 | getCartItems() | Retrieves the list of items in the cart. |

### c. Sequence Diagram(s)

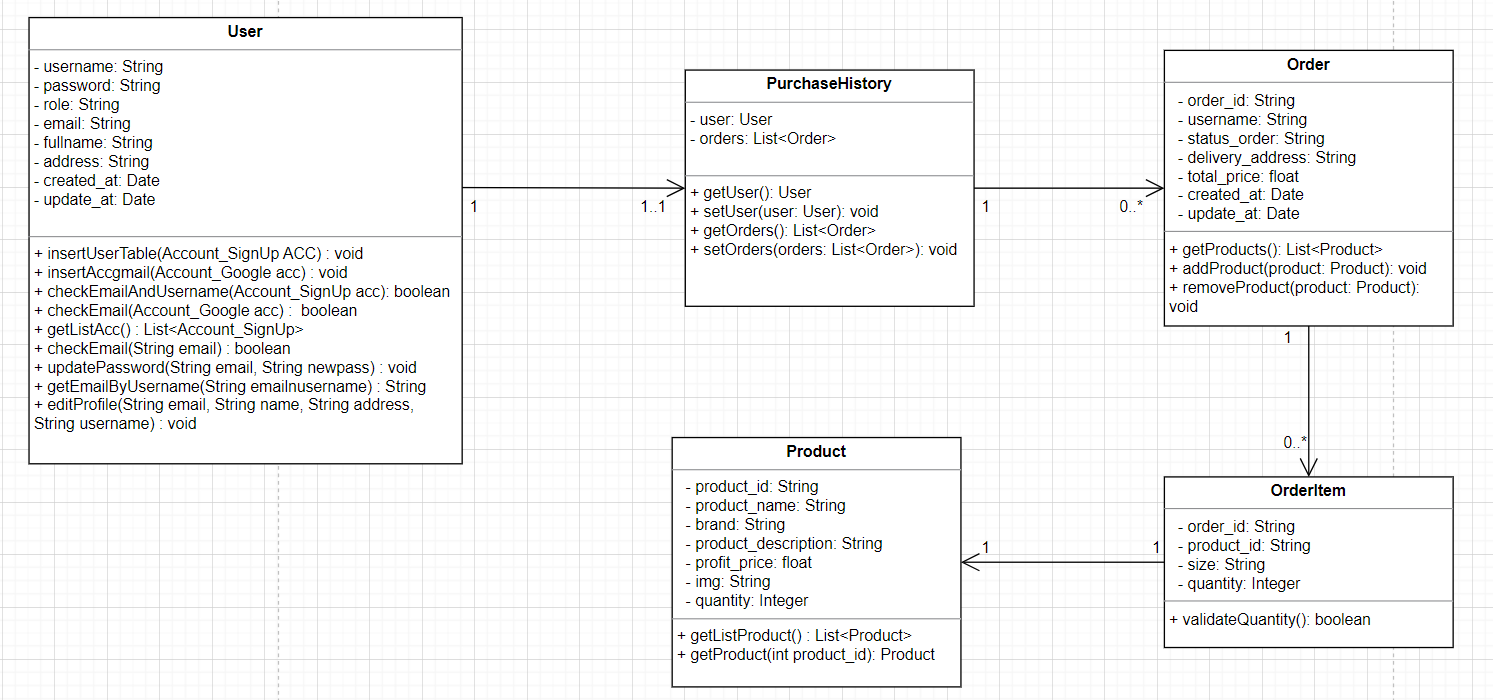
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | INSERT INTO Items (id, name, price, quantity) VALUES (?, ?, ?, ?) | This query is used to insert a new item into the "Items" table in the database. The question marks represent the placeholders for the actual values to be inserted. |
| 2 | SELECT \* FROM Items WHERE id = ? | This query is used to retrieve an item from the "Items" table based on the provided item ID. The question mark represents the placeholder for the actual item ID value. |
| 3 | UPDATE Items SET name = ?, price = ?, quantity = ? WHERE id = ? | This query is used to update the details of an existing item in the "Items" table based on the provided item ID. The question marks represent the placeholders for the actual values to be updated. |
| 4 | DELETE FROM Items WHERE id = ? | This query is used to delete an item from the "Items" table based on the provided item ID. The question mark represents the placeholder for the actual item ID value. |

## 14. View purchase history

### a. Class Diagram



### b. Class Specifications

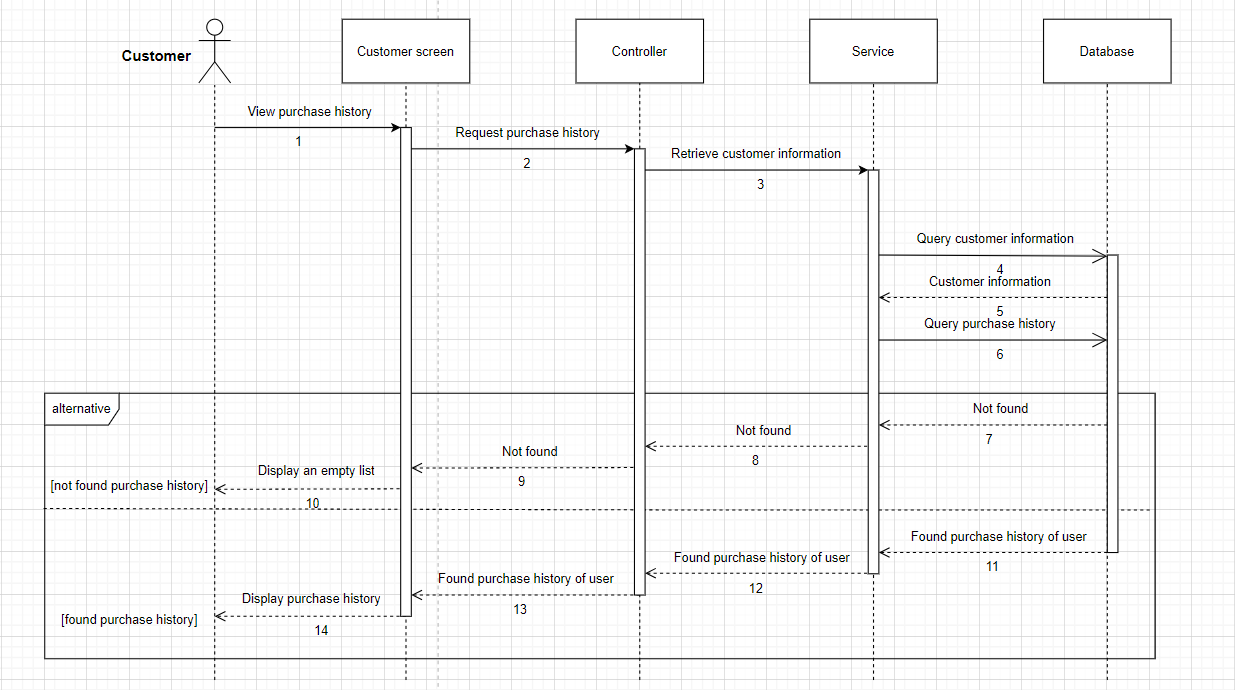
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *username* | *The username of the user.* |
| *2* | *password* | *The password of the user.* |
| *3* | *role* | *The role of the user.* |
| *4* | *email* | *The email address of the user.* |
| *5* | *fullname* | *The full name of the user.* |
| *6* | *address* | *The address of the user.* |
| *7* | *created\_at* | *The date and time when the user was created.* |
| *8* | *updated\_at* | *The date and time when the user was last updated.* |
| *9* | *user* | *Represents the user associated with the purchase history. It can be an instance of the User class.* |
| *10* | *orders* | *A list of orders made by the user. Each order is an instance of the Order class.* |
| *11* | *order\_id* | *The ID of the order.* |
| *12* | *username* | *The username of the user who placed the order.* |
| *13* | *status\_order* | *The status of the order.* |
| *14* | *delivery\_address* | *The delivery address for the order.* |
| *15* | *total\_price* | *The total price of the order.* |
| *16* | *created\_at* | *The date and time when the order was created.* |
| *17* | *updated\_at* | *The date and time when the order was last updated.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | getAllOrder() | This method retrieves a list of orders for a specific user from the database. |
| 2 | getListOrderForUser() | This method retrieves a list of orders for a specific user from the database. |

### c. Sequence Diagram(s)

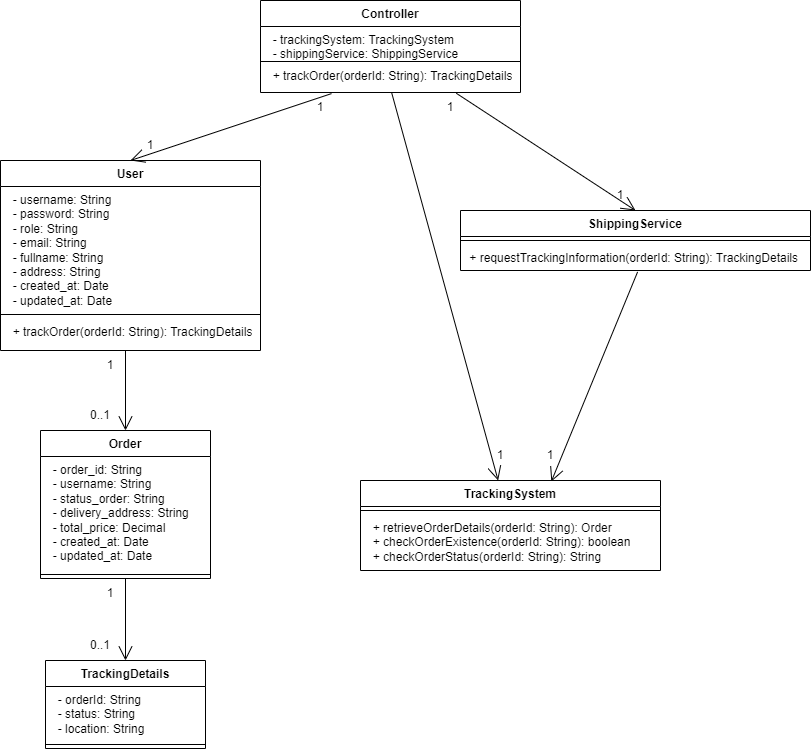


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Select \* From Order\_Table where username = ? order by created\_at desc | This query is used to retrieve orders from the Order\_Table table for a specific user, filtered by their username. The result set is ordered by the created\_at column in descending order. |
| 2 | Select \* From Order\_Table order by created\_at desc | This query is used to retrieve all orders from the Order\_Table table. The result set is ordered by the created\_at column in descending order. |

## 15. Track Order

### a. Class Diagram

**

### b. Class Specifications

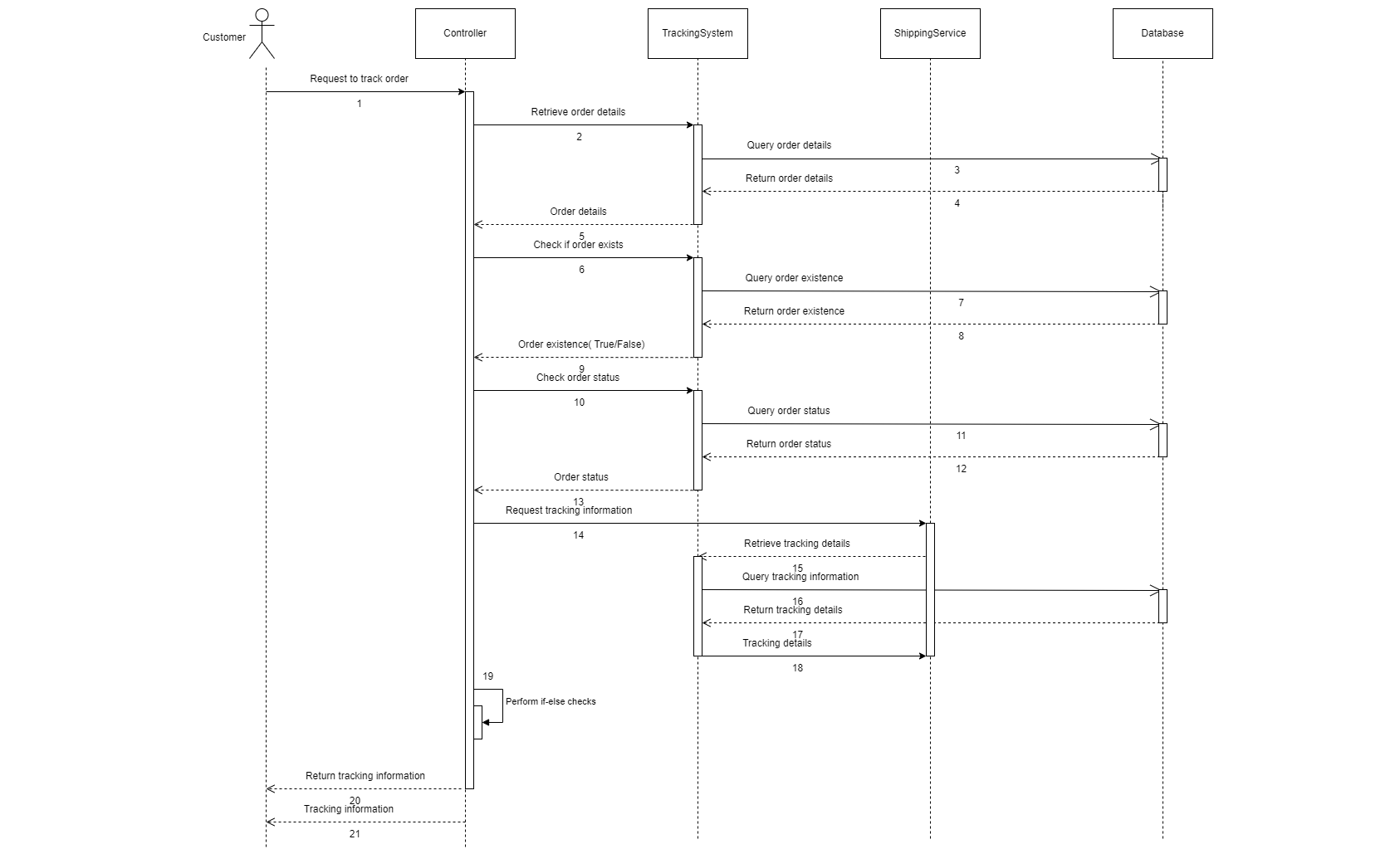
| **No** | **Attributes** | **Description** |
| --- | --- | --- |
| *1* | *username* | *The username of the user.* |
| *2* | *password* | *The password of the user.* |
| *3* | *role* | *The role of the user.* |
| *4* | *email* | *The email address of the user.* |
| *5* | *fullname* | *The full name of the user.* |
| *6* | *address* | *The address of the user.* |
| *7* | *created\_at* | *The date and time when the user was created.* |
| *8* | *updated\_at* | *The date and time when the user was last updated.* |
| *9* | *order\_id* | *The ID of the order.* |
| *10* | *username* | *The username of the user who placed the order.* |
| *11* | *status\_order* | *The status of the order.* |
| *12* | *delivery\_address* | *The delivery address for the order.* |
| *13* | *total\_price* | *The total price of the order.* |
| *14* | *created\_at* | *The date and time when the order was created.* |
| *15* | *updated\_at* | *The date and time when the order was last updated.* |
| *16* | *orderId* | *The ID of the order.* |
| *17* | *status* | *The status of the order.* |
| *18* | *location* | *The current location of the order.* |
| *19* | *trackingSystem* | *The tracking system instance.* |
| *20* | *shippingService* | *The shipping service instance.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *trackOrder(orderId: String)* | *Retrieves the tracking details for the given orderId.* |
| *2* | *retrieveOrderDetails(orderId: String)* | *Retrieves the order details for the given orderId.* |
| *3* | *checkOrderExistence(orderId: String)* | *Checks if an order with the given orderId exists.* |
| *4* | *checkOrderStatus(orderId: String)* | *Retrieves the status of the order with the given orderId.* |
| *5* | *requestTrackingInformation(orderId: String)* | *Retrieves the tracking details for the given orderId.* |
| *6* | *trackOrder(orderId: String)* | *Orchestrates the tracking process for the given orderId.* |

### c. Sequence Diagram(s)

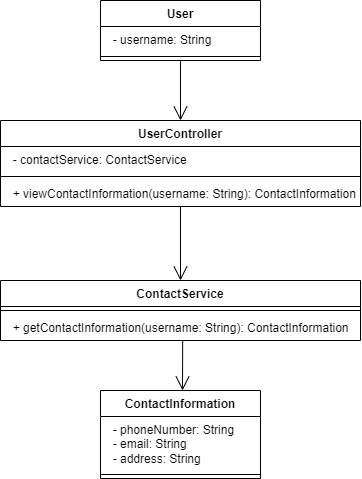


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT \* FROM orders WHERE order\_id = {orderId}; | Allows the tracking system to retrieve the necessary order details to provide accurate tracking information to the user. |
| 2 | SELECT COUNT(\*) FROM orders WHERE order\_id = {orderId}; | Helps the tracking system verify if an order with the provided orderId exists, ensuring that valid and existing orders are tracked. |
| 3 | SELECT status\_order FROM orders WHERE order\_id = {orderId}; | Allows the tracking system to determine the current status of an order, providing accurate tracking information to the user. |
| 4 | SELECT \* FROM tracking\_details WHERE order\_id = {orderId}; | Enables the shipping service to obtain the latest tracking information for an order, which can be presented to the user for tracking purposes. |
| 5 | INSERT INTO users (username, password, role, email, fullname, address, created\_at, updated\_at)  VALUES ({username}, {password}, {role}, {email}, {fullname}, {address}, {created\_at}, {updated\_at}); | Allows the system to store and manage user accounts, facilitating user registration and authentication processes. |

## 16. View contact information

### a. Class Diagram



### b. Class Specifications

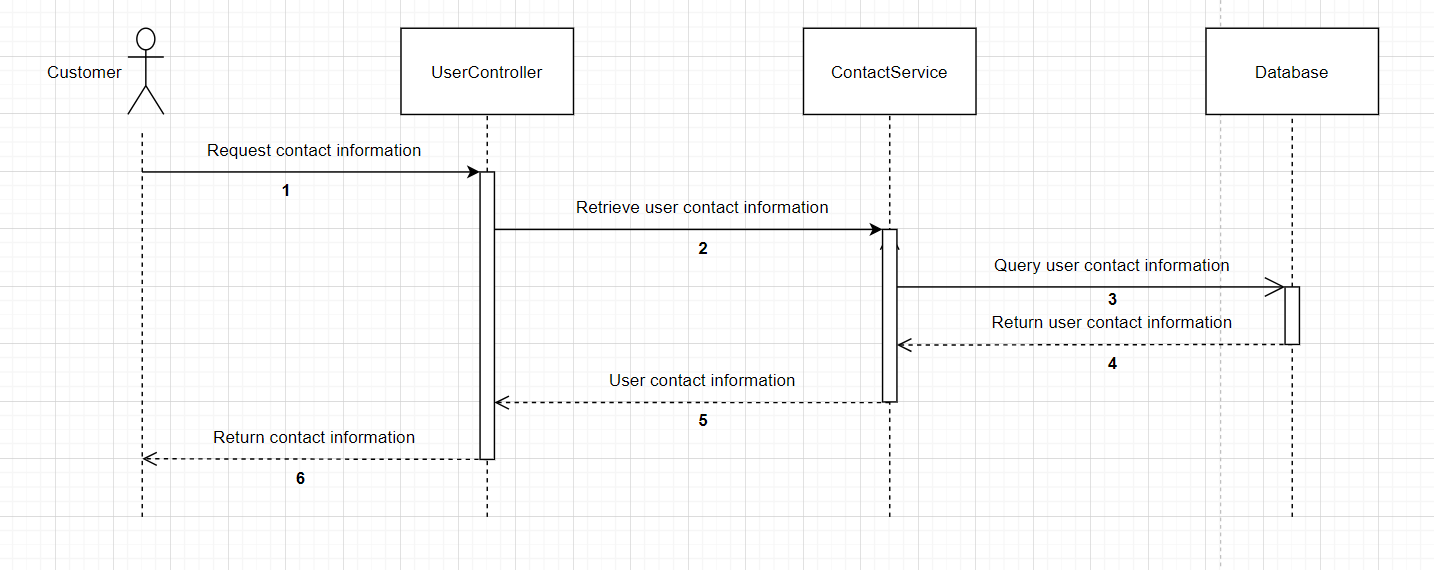
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *username* | *A string representing the username of the user.* |
| *2* | *contactService* | *An instance of the ContactService class.* |
| *3* | *phoneNumber* | *A string representing the phone number of the user.* |
| *4* | *email* | *A string representing the email address of the user.* |
| *5* | *address* | *A string representing the address of the user.* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *viewContactInformation(username: String)* | *Retrieves the contact information for the user with the specified username.* |
| *2* | *getContactInformation(username: String)* | *Retrieves the contact information for the user with the specified username.* |

### c. Sequence Diagram(s)

**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT phoneNumber, email, address FROM User WHERE username = 'example\_username' | Retrieves the phone number, email, and address of a user with the specified username from the User table in the database. |

## 17. Chat with the shop

### a. Class Diagram

**

### b. Class Specifications

| **No** | **Attributes** | **Description** |
| --- | --- | --- |
| 1 | username | The username of the user. |
| 2 | email | The email address of the user. |
| 3 | shop | Reference to the Shop object involved in the chat. |
| 4 | user | Reference to the User object initiating the chat. |
| 5 | shopId | The unique identifier of the shop. |
| 6 | shopName | The name of the shop. |
| 7 | chatId | The unique identifier of the chat. |

#### 

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | initiateChatWithShop() | Initiates a chat with the shop. |
| 2 | sendMessage(message: String) | Send a message from the user to the shop. |
| 3 | receiveResponse(response: String) | Receives a response from the shop. |
| 4 | validateMessage(message: String) | Validates the message received from the user and returns a response. |
| 5 | saveMessage(message: String) | Save the chat message for future reference. |

### c. Sequence Diagram(s)

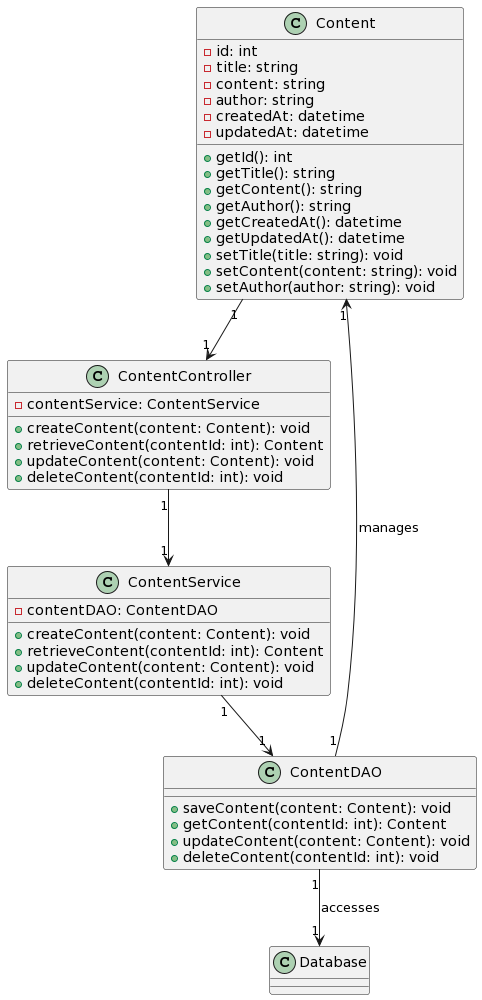


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | INSERT INTO chat\_messages (sender\_id, receiver\_id, message\_content, timestamp) VALUES (:senderId, :receiverId, :messageContent, :timestamp) | Used to store a new chat message in the database. |
| 2 | SELECT \* FROM chat\_messages WHERE (sender\_id = :user1Id AND receiver\_id = :user2Id) OR (sender\_id = :user2Id AND receiver\_id = :user1Id) ORDER BY timestamp ASC | Used to retrieve chat history between two users. |
| 3 | UPDATE chat\_messages SET is\_read = true WHERE message\_id = :messageId | Used to mark a chat message as read or update other message attributes. |

## 18. CRUD content

### a. Class Diagram



### b. Class Specifications

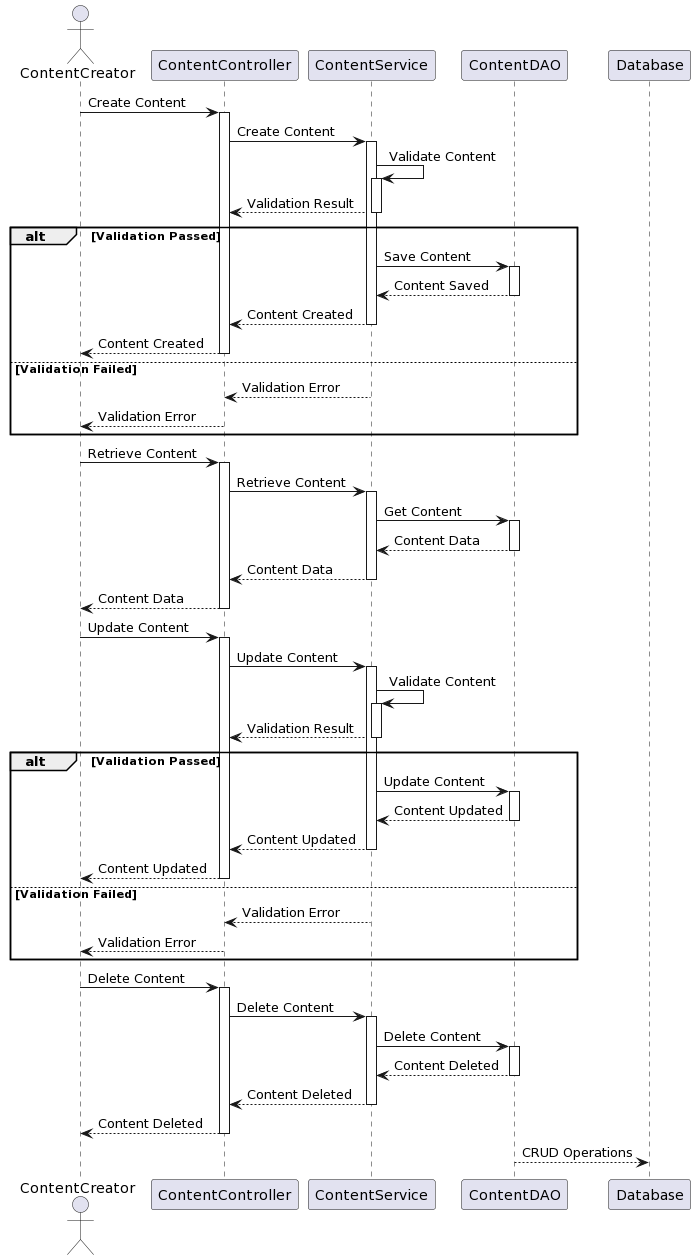
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | id | Represents the unique identifier of the content. |
| 2 | title | Represents the title of the content. |
| 3 | content | Represents the actual content or body of the content. |
| 4 | author | Represents the author of the content. |
| 5 | createdAt | Represents the date and time when the content was created. |
| 6 | updatedAt | Represents the date and time when the content was last updated. |
| 7 | contentService | An instance of the ContentService class to handle content-related operations. |
| 8 | contentDAO | An instance of the ContentDAO class to handle data access operations for content. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | getId() | Returns the unique identifier of the content. |
| 2 | getTitle() | Returns the title of the content. |
| 3 | getContent() | Returns the content body. |
| 4 | getAuthor() | Returns the author of the content. |
| 5 | getCreatedAt() | Returns the creation date and time of the content. |
| 6 | getUpdatedAt() | Returns the last update date and time of the content. |
| 7 | setTitle(title: string) | Sets the title of the content. |
| 8 | setContent(content: string) | Sets the content body. |
| 9 | setAuthor(author: string) | Sets the author of the content. |
| 10 | createContent(content: Content) | This function creates a new content item and saves it to the database. |
| 11 | retrieveContent(contentId: int) | This function retrieves an existing content item from the database based on its ID. |
| 12 | updateContent(content: Content) | This function updates an existing content item with the provided data. |
| 14 | deleteContent(contentId: int) | This function deletes an existing content item from the database based on its ID. |

### c. Sequence Diagram(s)

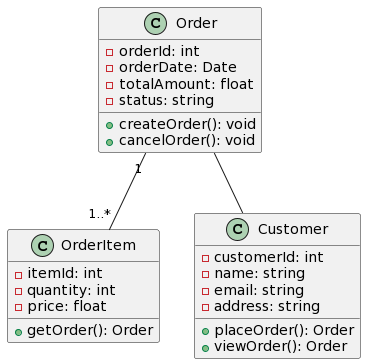


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | INSERT INTO content (title, content, author) VALUES (?, ?, ?) | This query is used to create a new content item in the database by inserting the provided title, content, and author values into the content table. |
| 2 | SELECT \* FROM content WHERE id = ? | This query is used to retrieve an existing content item from the database based on its unique identifier (ID). |
| 3 | UPDATE content SET title = ?, content = ?, author = ? WHERE id = ? | This query is used to update an existing content item in the database with new or modified values for its title, content, and author. |
| 4 | DELETE FROM content WHERE id = ? | This query is used to delete an existing content item from the database based on its unique identifier (ID). |

## 19. Confirm order

### a. Class Diagram

**

### b. Class Specifications

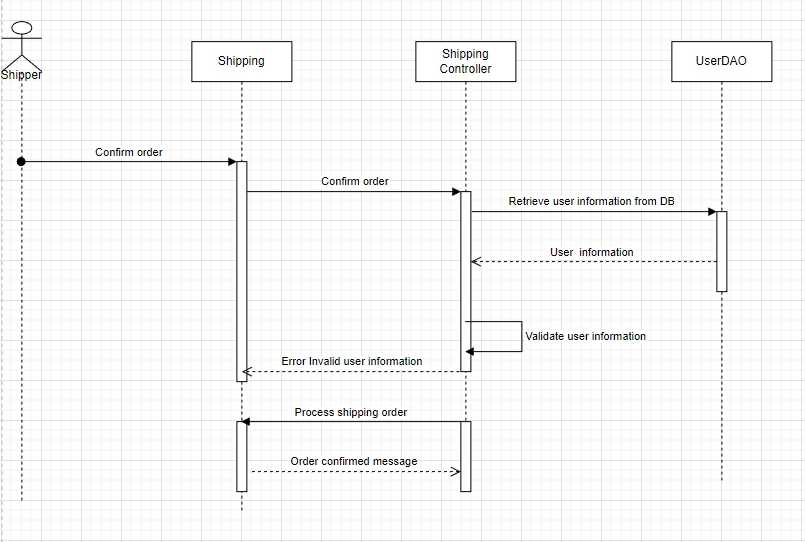
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | orderId | Private attribute representing the order ID. |
| 02 | orderDate | Private attribute representing the order date. |
| 03 | totalAmount | Private attribute representing the total amount of the order. |
| 04 | status | Private attribute representing the status of the order. |
| 05 | itemId | Private attribute representing the item ID in the order. |
| 06 | quantity | Private attribute representing the quantity of the item. |
| 07 | price | Private attribute representing the price of the item. |
| 08 | customerId | Private attribute representing the customer ID. |
| 09 | name | Private attribute representing the name of the customer. |
| 10 | email | Private attribute representing the email address of the customer. |
| 11 | address | Private attribute representing the address of the customer. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | createOrder() | Public method used to create a new order. |
| 02 | cancelOrder() | Public method used to cancel the order. |
| 03 | getOrder() | Public method used to retrieve the order information related to the item. |
| 04 | placeOrder() | Public method used to place a new order. |
| 05 | viewOrder() | Public method used to view order information. |

### c. Sequence Diagram(s)

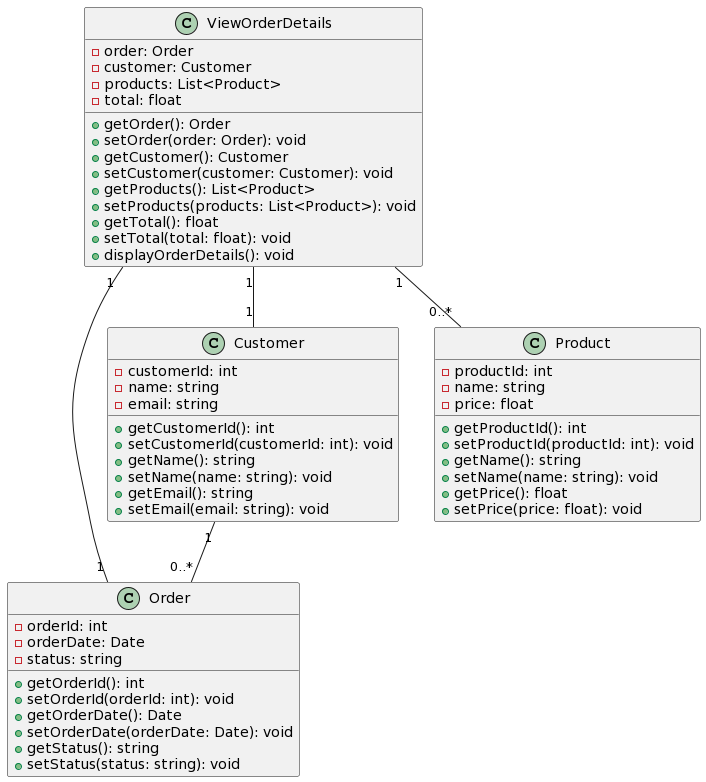
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 01 | Confirm order from Shipper to Servlet | "Shipper" confirms the order |
| 02 | Confirm order from Servlet to DB | Information is passed from "Ship" to "Controller" for checking |
| 03 | Retrieve user information | Get user information from "DAO". |
| 04 | User information | Return user information |
| 05 | Validate user information | Check the validity of user information |
| 06 | Error: Invalid user information | If the information is not valid, the error message "Invalid user information" |
| 07 | Process shipping order | If the information is valid, "Controller" proceeds to process the order |
| 08 | Order confirmed message | Send a confirmation message back to "Shipping". |

## 20. View order details

### a. Class Diagram

**

### b. Class Specifications

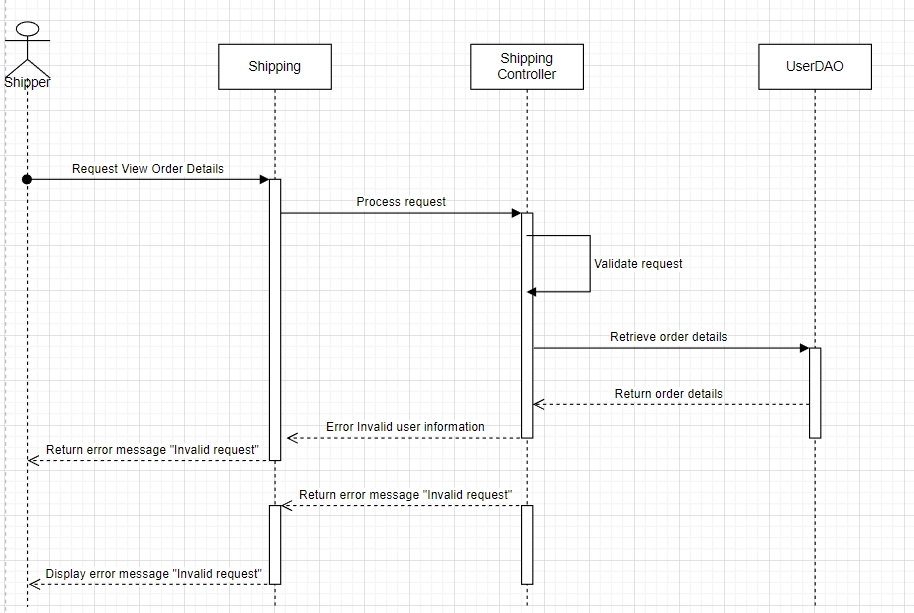
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | order | Represents the order associated with the view. It is of type Order. |
| 02 | customer | Represents the customer associated with the view. It is of type Customer. |
| 03 | products | Represents the list of products associated with the view. It is of type List<Product>. |
| 04 | total | Represents the total amount of the order. It is of type float. |
| 05 | orderId | Represents the unique identifier of the order. It is of type int. |
| 06 | orderDate | Represents the date when the order was placed. It is of type Date. |
| 07 | status | Represents the status of the order. It is of type string. |
| 08 | customerId | Represents the unique identifier of the customer. It is of type int. |
| 09 | name | Represents the name of the customer. It is of type string. |
| 10 | email | Represents the email address of the customer. It is of type string. |
| 11 | productId | Represents the unique identifier of the product. It is of type int. |
| 12 | name | Represents the name of the product. It is of type string. |
| 13 | price | Represents the price of the product. It is of type float. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getOrder() | Returns the associated order. |
| 02 | setOrder(order: Order) | Set the associated order. |
| 03 | getCustomer() | Returns the associated customer. |
| 04 | setCustomer(customer: Customer) | Set the associated customer. |
| 05 | getProducts() | Return the list of associated products. |
| 06 | setProducts(products: List<Product>) | Set the list of associated products. |
| 07 | getTotal() | Return the total amount of the order. |
| 08 | setTotal(total: float) | Set the total amount of the order. |
| 09 | displayOrderDetails() | Displays the details of the order. |
| 10 | getOrderId() | Returns the order ID. |
| 11 | setOrderId(orderId: int) | Set the order ID. |
| 12 | getOrderDate() | Return the order date. |
| 13 | setOrderDate(orderDate: Date) | Set the order date. |
| 14 | getStatus() | Return the status of the order. |
| 15 | setStatus(status: string) | Set the status of the order. |
| 16 | getCustomerId() | Returns the customer ID. |
| 17 | setCustomerId(customerId: int) | Set the customer ID. |
| 18 | getName() | Returns the name of the customer. |
| 19 | setName(name: string) | Set the name of the customer. |
| 20 | getEmail() | Returns the email address of the customer. |
| 21 | setEmail(email: string) | Set the email address of the customer. |
| 22 | getProductId() | Returns the product ID. |
| 23 | setProductId(productId: int) | Set the product ID. |
| 24 | getName() | Returns the name of the product. |
| 25 | setName(name: string) | Set the name of the product. |
| 26 | getPrice() | Returns the price of the product. |
| 27 | setPrice(price: float) | Set the price of the product |

### c. Sequence Diagram(s)



### 

### 

### 

### 

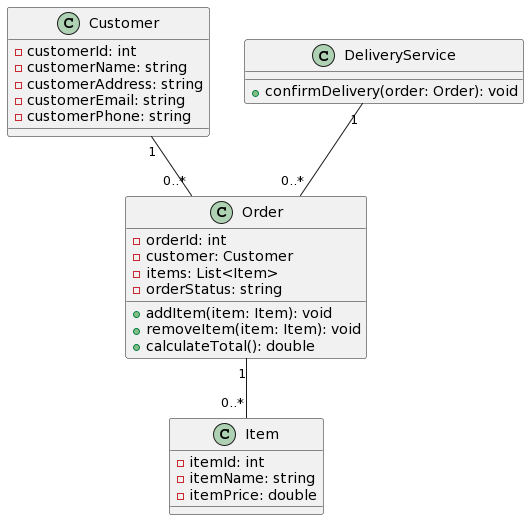
### 

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 01 | Request View Order Details | The "User" initiates the process by sending a "Request View Order Details" to the "ViewOrder" component. The "User" represents the person or entity interacting with the system. |
| 02 | Request View Order Details | The "ViewOrder" component, which acts as the boundary between the user interface and the control logic, receives the request and passes it to the "OrderController" for further processing. |
| 03 | Validate request | The "OrderController" is responsible for controlling the flow of actions. It validates the request by performing any necessary checks or validations. |
| 04 | Retrieve order details | If the request is valid, the "OrderController" proceeds to retrieve the order details from the "Database". It sends a request to the "Database" entity to fetch the relevant information. |
| 05 | Return order details | The "Database" processes the request and retrieves the order details from its storage. It then returns the order details back to the "OrderController". |
| 06 | Return order details | Upon receiving the order details, the "OrderController" forwards them to the "ViewOrder" component. |
| 07 | Display order details | Finally, the "ViewOrder" component displays the order details to the "User" through the user interface, completing the process. |
| 08 | Return error message "Invalid request" | If the "OrderController" determines that the request is invalid during the validation step, it returns an error message, such as "Invalid request", to the "ViewOrder" component. |
| 09 | Display error message "Invalid request" | The "ViewOrder" component then displays the error message to the "User" through the user interface, indicating that the request was not valid. |

## 21. Confirm delivery

### a. Class Diagram

**

### b. Class Specifications

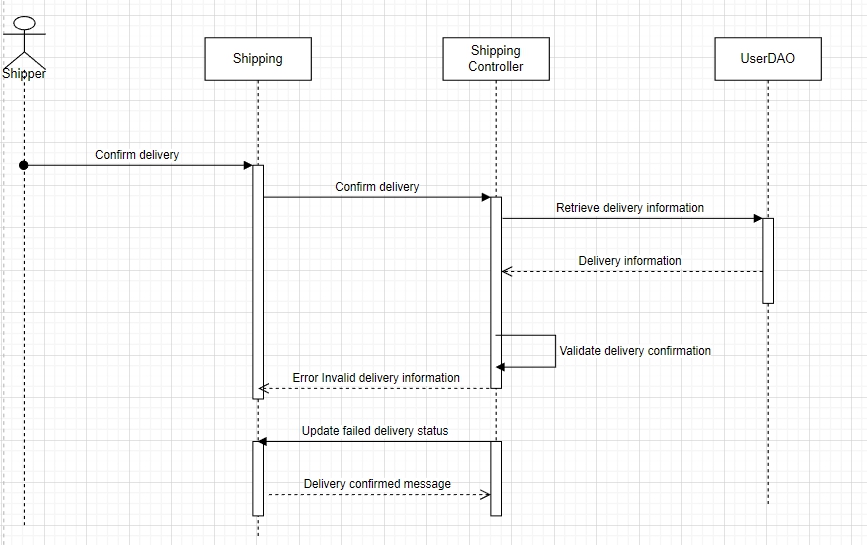
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | customerId | An integer representing the unique ID of the customer. |
| 02 | customerName | A string representing the name of the customer. |
| 03 | customerAddress | A string representing the address of the customer. |
| 04 | customerEmail | A string representing the email of the customer. |
| 05 | customerPhone | A string representing the phone number of the customer. |
| 06 | orderId | An integer representing the unique ID of the order. |
| 07 | customer | An instance of the Customer class representing the customer who placed the order. |
| 08 | items | A list of Item instances representing the items included in the order. |
| 09 | orderStatus | A string representing the current status of the order |
| 10 | itemId | An integer representing the unique ID of the item. |
| 11 | itemName | A string representing the name of the item. |
| 12 | itemPrice | A double representing the price of the item. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addItem(item: Item) | Adds an item to the order. |
| 02 | removeItem(item: Item) | Removes an item from the order. |
| 03 | calculateTotal() | Calculates the total cost of the order. |
| 04 | confirmDelivery(order: Order) | Confirms the delivery of a specific order. |

### c. Sequence Diagram(s)

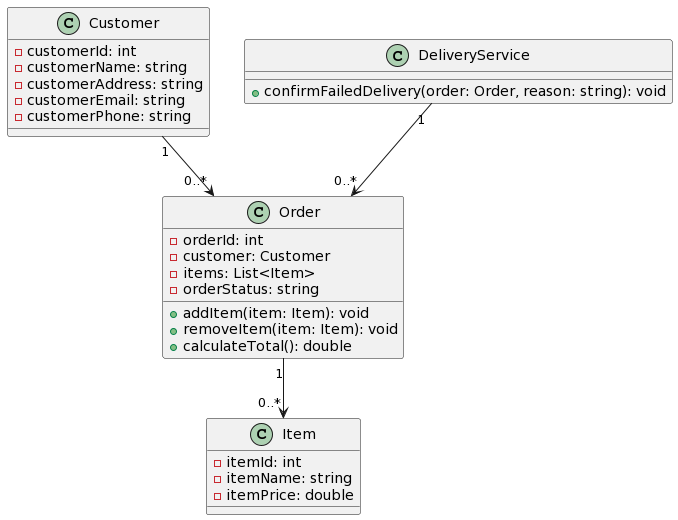
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 01 | Confirm delivery from Shipper to Servlet | "Shipper" confirms delivery |
| 02 | Confirm delivery from Servlet to DB | Information is passed from "Ship" to "Controller" |
| 03 | Retrieve delivery information | Get delivery information from "DAO" |
| 04 | Delivery information | Return delivery information |
| 05 | Validate delivery information | "Controller" checks the validity of the delivery information |
| 06 | Error: Invalid delivery information | If the information is invalid, the error message "Invalid delivery information" is sent from "Controller" to "Ship" |
| 07 | Update delivery status | If the information is valid, "Controller" updates the delivery status |
| 08 | Delivery confirmed message | Send a confirmation message back to "Ship". |

## 22. Check failed delivery

### a. Class Diagram

**

### b. Class Specifications

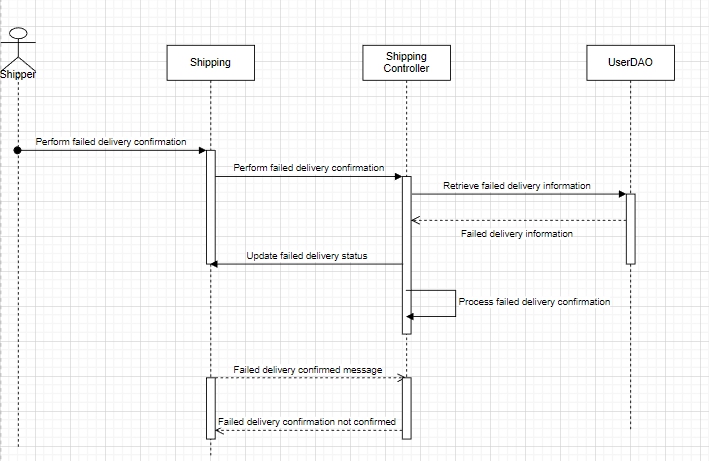
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | customerId | An integer representing the unique ID of the customer. |
| 02 | customerName | A string representing the name of the customer. |
| 03 | customerAddress | A string representing the address of the customer. |
| 04 | customerEmail | A string representing the email of the customer. |
| 05 | customerPhone | A string representing the phone number of the customer. |
| 06 | orderId | An integer representing the unique ID of the order. |
| 07 | customer | An instance of the Customer class representing the customer who placed the order. |
| 08 | items | A list of Item instances representing the items included in the order. |
| 09 | orderStatus | A string representing the current status of the order |
| 10 | itemId | An integer representing the unique ID of the item. |
| 11 | itemName | A string representing the name of the item. |
| 12 | itemPrice | A double representing the price of the item. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addItem(item: Item) | Adds an item to the order. |
| 02 | removeItem(item: Item) | Removes an item from the order. |
| 03 | calculateTotal() | Calculates the total cost of the order. |
| 04 | confirmFailedDelivery(order: Order, reason: string) | Confirms the failed delivery of a specific order and accepts a reason parameter to describe the reason for the failed delivery. |

### c. Sequence Diagram(s)

**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 01 | Perform failed delivery confirmation | "Shipper" begins with a failed delivery confirmation by sending a request to "Shipping". |
| 02 | Perform failed delivery confirmation | "Shipping" receives the request and forwards it to the "Shipping Controller". |
| 03 | Retrieve failed delivery information | "Shipping Controller" failed to retrieve shipping information from "DAO" |
| 04 | Failed delivery information | "DAO" provides information about failed delivery for "Shipping Controller". |
| 05 | Process failed delivery confirmation | The "Shipping Controller" handles the failed shipping information to perform the confirmation. |
| 06 | Update failed delivery status | If the failed delivery confirmation is done successfully, "Shipping Controller" updates the failed delivery status in "Ship". |
| 07 | Failed delivery confirmed message | "Ship" sends "Failed delivery confirmed" message back to "Shipping Controller". |
| 08 | Failed delivery confirmation not confirmed  end | If failed delivery confirmation is not done, "Shipping Controller" sends message "Failed delivery confirmation not confirmed" |

## 23. CRUD products

### a. Class Diagram

**

### b. Class Specifications

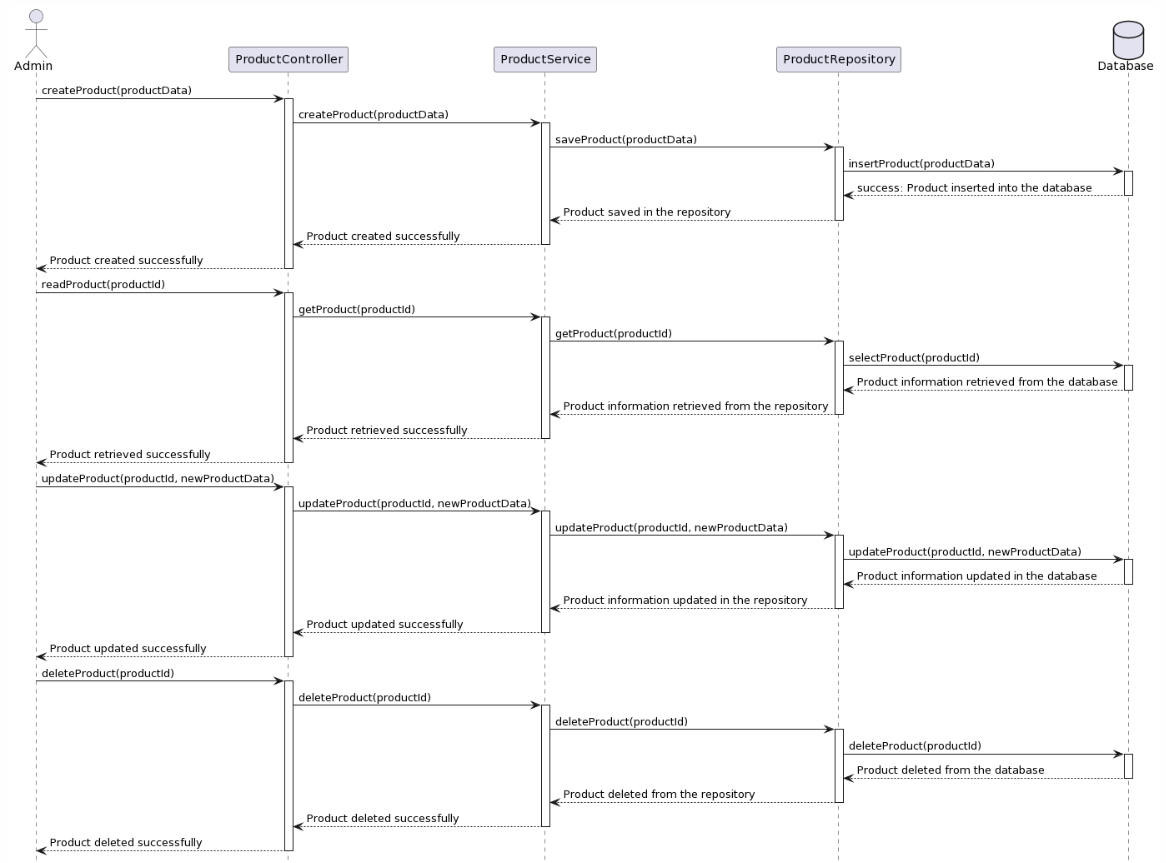
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | productId | Represents the unique identifier of the product. |
| 2 | productName | Represents the name of the product. |
| 3 | price | Represents the price of the product. |
| 4 | quantity | Represents the available quantity of the product. |
| 5 | adminId | Represents the unique identifier of the administrator. |
| 6 | name | Represents the name of the administrator. |
| 7 | email | Represents the email address of the administrator. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | Product(productId: int, productName: String, price: float, quantity: int) | Constructor to initialize the attributes of the product. |
| 2 | getProductId(): int | Retrieves the product's ID. |
| 3 | getName(): String | Retrieves the product's name. |
| 4 | getPrice(): float | Retrieves the product's price. |
| 5 | getQuantity(): int | Retrieves the product's quantity. |
| 6 | updateName(newName: String) | Updates the product's name with the provided value. |
| 7 | updatePrice(newPrice: float) | Updates the product's price with the provided value. |
| 8 | updateQuantity(newQuantity: int) | Updates the product's quantity with the provided value. |
| 9 | createProduct() | Creates a new product in the inventory. |
| 10 | retrieveProduct(productId: int): Product | Retrieves a specific product from the inventory based on the provided product ID. |
| 11 | updateProduct(productId: int) | Updates the details of a specific product in the inventory based on the provided product ID. |
| 12 | deleteProduct(productId: int) | Deletes a specific product from the inventory based on the provided product ID. |
| 13 | Admin(adminId: int, name: String, email: String) | Constructor to initialize the attributes of the administrator. |
| 14 | getAdminId(): int | Retrieves the administrator's ID. |
| 15 | getName(): String | Retrieves the administrator's name. |
| 16 | getEmail(): String | Retrieves the administrator's email. |
| 17 | createProduct() | Allows the administrator to create a new product. |
| 18 | updateProduct(productId: int) | Allows the administrator to update the details of a specific product. |
| 19 | deleteProduct(productId: int) | Allows the administrator to delete a specific product. |

### c. Sequence Diagram(s)

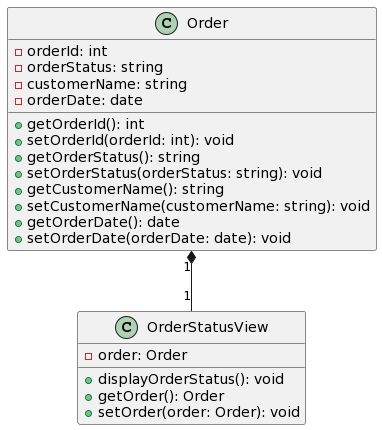


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | createProduct(productData) | The User sends a createProduct request to the Controller. |
| 2 | createProduct(productData) | The Controller activates and forwards the request to the Service. |
| 3 | saveProduct(productData) | The Service activates and calls the Repository to save the product data. |
| 4 | insertProduct(productData) | The Repository activates and inserts the product data into the Database. |
| 5 | Product inserted into the database | Once the product is successfully inserted, the Database responds with a success message. |
| 6 | Product saved in the repository | The Repository deactivates and passes the success message to the Service. |
| 7 | Product created successfully | The Service deactivates and passes the success message to the Controller. |
| 8 | Product created successfully | The Controller deactivates and responds with a success message to the User. |
| 9 | readProduct(productId) | The User sends a readProduct request with a productId to the Controller. |
| 10 | getProduct(productId) | The Controller activates and forwards the request to the Service. |
| 11 | getProduct(productId) | The Service activates and calls the Repository to retrieve the product data. |
| 12 | selectProduct(productId) | The Repository activates and selects the product data from the Database based on the productId. |
| 13 | Product information is retrieved from the database | The Database responds with the product data. |
| 14 | Product information retrieved from the repository | The Repository deactivates and passes the product data to the Service. |
| 15 | Product were retrieved successfully | The Service deactivates and passes the product data to the Controller. |
| 16 | Product retrieved successfully | The Controller deactivates and responds with the product data to the User. |
| 17 | updateProduct(productId, newProductData) | The User sends an updateProduct request with a productId and newProductData to the Controller. |
| 18 | updateProduct(productId, newProductData) | The Controller activates and forwards the request to the Service. |
| 19 | updateProduct(productId, newProductData) | The Service activates and calls the Repository to update the product data. |
| 20 | updateProduct(productId, newProductData) | The Repository activates and updates the product data in the Database based on the productId. |
| 21 | Product information is updated in the database | Once the update is successful, the Database responds with a success message. |
| 22 | Product information is updated in the repository | The Repository deactivates and passes the success message to the Service. |
| 23 | Product updated successfully | The Service deactivates and passes the success message to the Controller. |
| 24 | Product updated successfully | The Controller deactivates and responds with a success message to the User. |
| 25 | deleteProduct(productId) | The User sends a deleteProduct request with a productId to the Controller. |
| 26 | deleteProduct(productId) | The Controller activates and forwards the request to the Service. |
| 27 | deleteProduct(productId) | The Service activates and calls the Repository to delete the product data. |
| 28 | deleteProduct(productId) | The Repository activates and deletes the product data from the Database based on the productId. |
| 29 | Product deleted from the database | Once the deletion is successful, the Database responds with a success message. |
| 30 | Product deleted from the repository | The Repository deactivates and passes the success message to the Service. |
| 31 | Product deleted successfully | The Service deactivates and passes the success message to the Controller. |
| 32 | Product deleted successfully | The Controller deactivates and responds with a success message to the User. |

## 24. View order status

### a. Class Diagram

**

### b. Class Specifications

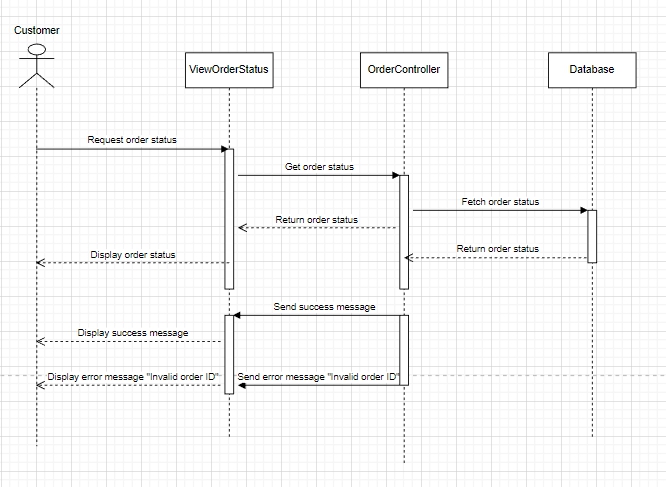
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | orderId | Represents the unique identifier of the order. |
| 02 | orderStatus | Stores the current status of the order. |
| 03 | customerName | Holds the name of the customer who placed the order. |
| 04 | orderDate | Represents the date when the order was placed. |
| 05 | order | Represents the associated Order object in the OrderStatusView class. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getOrderId() | Retrieves the orderId attribute of the Order class. |
| 02 | setOrderId(orderId: int) | Set the orderId attribute of the Order class to the specified value. |
| 03 | getOrderStatus() | Retrieves the orderStatus attribute of the Order class. |
| 04 | setOrderStatus(orderStatus: string) | Set the orderStatus attribute of the Order class to the specified value. |
| 05 | getCustomerName() | Retrieves the customerName attribute of the Order class. |
| 06 | setCustomerName(customerName: string) | Set the customerName attribute of the Order class to the specified value. |
| 07 | getOrderDate() | Retrieves the orderDate attribute of the Order class. |
| 08 | setOrderDate(orderDate: date) | Set the orderDate attribute of the Order class to the specified value. |
| 09 | displayOrderStatus() | Displays the order status in the OrderStatusView class. |
| 10 | getOrder() | Retrieves the associated Order object in the OrderStatusView class. |
| 11 | setOrder(order: Order) | Set the associated Order object in the OrderStatusView class to the specified value. |

### c. Sequence Diagram(s)

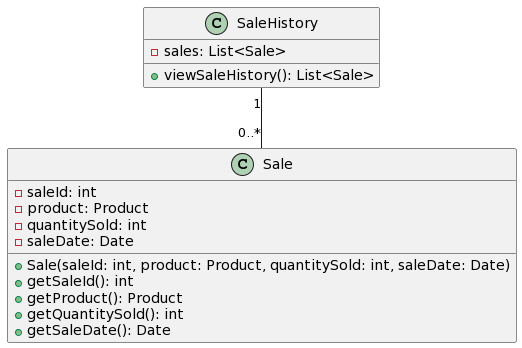
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 01 | Request order status | The user initiates the process by requesting the order status. |
| 02 | Get order status | The web application receives the request and communicates with the order controller to get the order status. |
| 03 | Fetch order status | The order controller interacts with the order database to fetch the order status. |
| 04 | Return order status | The order database returns the order status to the order controller. |
| 05 | Return order status | The order controller sends the order status back to the web application. |
| 06 | Display order status | The web application displays the order status to the user. |
| 07 | Send success message | The order controller sends a success message to the web application. |
| 08 | Display success message | The web application displays a success message to the user. |
| 09 | Send error message "Invalid order ID" | The order controller sends an error message ("Invalid order ID") to the web application. |
| 10 | Display error message "Invalid order ID" | The web application displays an error message ("Invalid order ID") to the user. |

## 25. View sale history

### a. Class Diagram

**

### b. Class Specifications

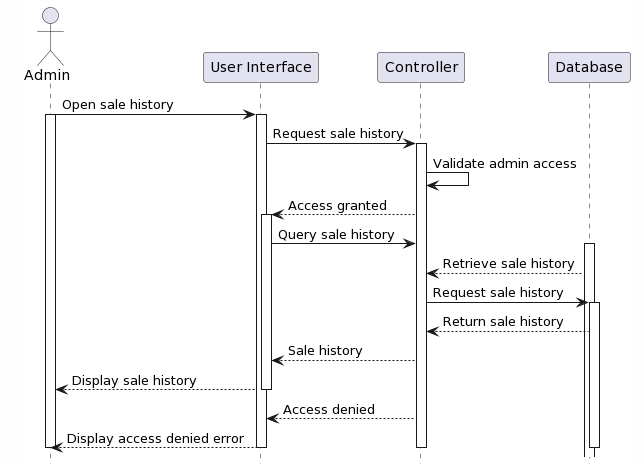
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | sale | List of Sale objects |
| 2 | saleId | Represents the unique identifier of the sale. |
| 3 | product | Represents the product being sold (associated with the Product class). |
| 4 | quantitySold | Represents the quantity of the product sold. |
| 5 | saleDate | Represents the date of the sale. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | viewSaleHistory() | Retrieves the list of sales in the sale history. |
| 2 | Sale(saleId: int, product: Product, quantitySold: int, saleDate: Date) | Constructor to initialize the attributes of the sale. |
| 3 | getSaleId() | Retrieves the sale's ID. |
| 4 | getProduct() | Retrieves the product associated with the sale. |
| 5 | getQuantitySold() | Retrieves the quantity of the product sold. |
| 6 | getSaleDate() | Retrieves the date of the sale. |

### c. Sequence Diagram(s)

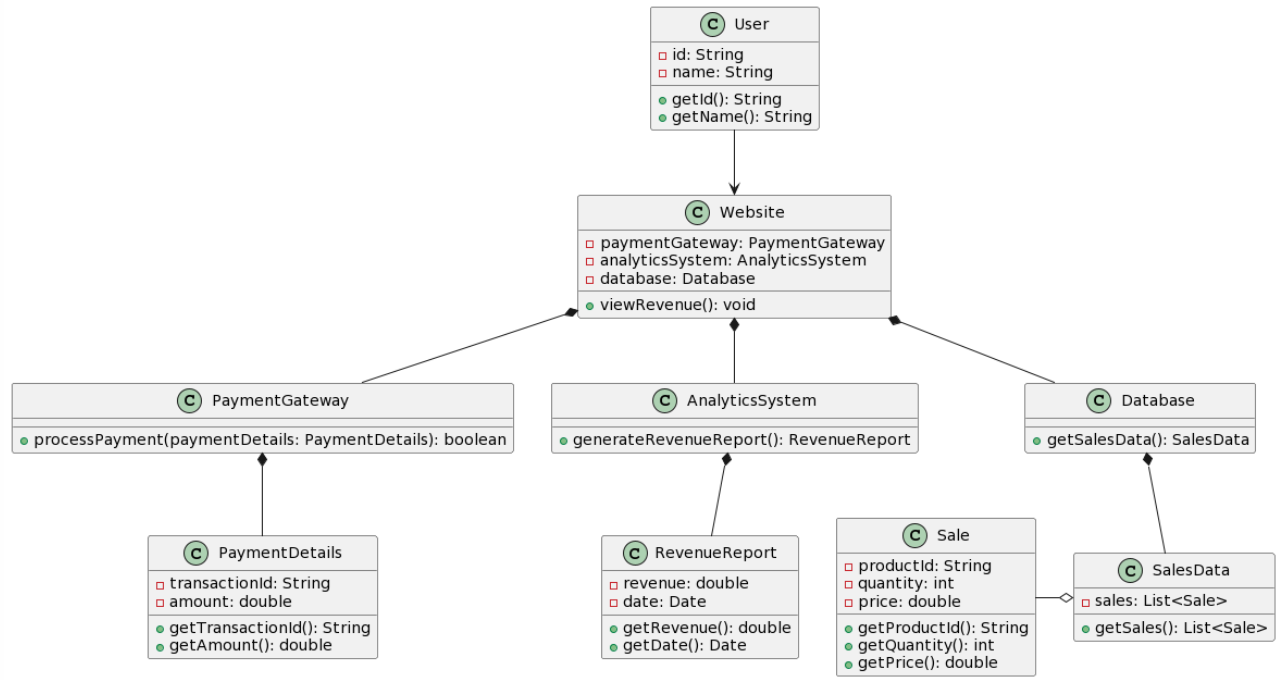


### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Open sale history | The admin initiates the process by expressing the intention to view the sale history. |
| 2 | Request sale history | The user interface (UI) receives the admin's request and sends a corresponding request to the controller component. |
| 3 | Validate admin access | The controller validates the admin's access rights to ensure they have the necessary permissions to view the sale history. |
| 4 | Access granted | If the admin's access is validated successfully, the controller grants access to proceed. |
| 5 | Query sale history | The UI, upon receiving access, sends a query to the controller to fetch the sale history. |
| 6 | Retrieve sale history | The controller, having received the query, contacts the database (DB) to retrieve the sale history. |
| 7 | Request sale history | The controller sends a request to the DB, specifying the need for the sale history. |
| 8 | Return sale history | The DB retrieves the sale history and returns it to the controller. |
| 9 | Sale history | The controller receives the sale history from the DB. |
| 10 | Display sale history | If the access is granted and the sale history is obtained, the controller sends it back to the UI, which displays it for the admin to view. |
| 11 | Access denied | If the admin's access is not validated successfully, the controller denies access to the sale history. |
| 12 | Display access denied error | The UI receives the access denial from the controller and displays an error message indicating that the admin does not have permission to view the sale history. |

## 26.View the revenue

### a. Class Diagram

**

### b. Class Specifications

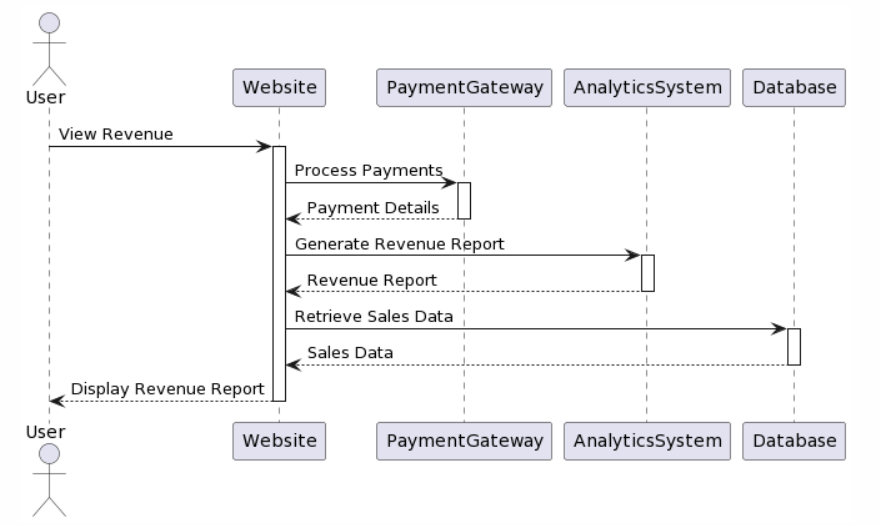
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *id* | *Used to represent an attribute that uniquely identifies the object of a class* |
| *2* | *name* | *Used to represent the name of an object* |
| *3* | *paymentGateway* | *Used to represent a payment gateway or payment system used within a system* |
| *4* | *analyticsSystem* | *Used to represent data analysis or statistical systems within a system* |
| *5* | *database* | *Used to represent the database used in a system* |
| *6* | *transactionId* | *Used to represent the transaction code of a transaction or payment transaction in the system* |
| *7* | *amount* | *Used to represent the amount involved in a transaction or a payment in the system* |
| *8* | *revenue* | *Used to represent revenue or total income generated by a system or an object* |
| *9* | *date* | *Used to represent the date or time associated with an event, a transaction, or an object in the system.* |
| *10* | *productId* | *Used to represent a unique identifier of a product in the system* |
| *11* | *quantity* | *Used to represent the quantity of an object or an item in the system* |
| *12* | *price* | *Used to represent the price of an object or an item in the system* |
| *13* | *sale* | *Used to represent information about a discount or promotion applied to a product or object in the system* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *getId()* | *Used to retrieve and return the unique identifier of an object* |
| *2* | *getName()* | *Used to retrieve and return an object's name value* |
| *3* | *viewRevenue()* | *Used to display and retrieve information about revenue or total income in the system* |
| *4* | *processPayment()* | *Used to perform payment processing in the system* |
|  | *generateRevenueReport()* | *Used to generate reports of revenue or total income in the system* |
| *5* | *getSalesData()* | *Used to get sales data from a database or other data source in the system* |
| *6* | *getTransactionId()* | *Used to retrieve and return the identifier of a transaction or item in the system* |
| *7* | *getAmount()* | *Used to retrieve and return the amount value of a transaction or item in the system* |
| *8* | *getRevenue()* | *Used to retrieve and return the revenue or total income value in the system* |
| *9* | *getDate()* | *Used to retrieve and return the date or time value of an event, transaction, or item in the system.* |
| *10* | *getProductId()* | *Used to retrieve and return the identifier (ID) value of a product in the system* |
| *11* | *getQuantity()* | *Used to retrieve and return the quantity value of an item or product in the system* |
| *12* | *getPrice()* | *Used to retrieve and return the value of an item or product in the system* |
| *13* | *getSales()* | *Used to retrieve and return sales information in the system* |

### c. Sequence Diagram(s)

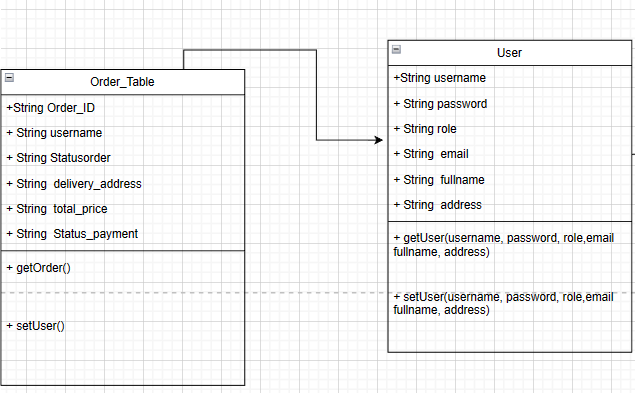
**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | SELECT SUM(revenue) AS total\_revenue FROM revenue\_table WHERE date >= 'start\_date' AND date <= 'end\_date'; | Query total revenue for a specific period |
| 2 | SELECT date, SUM(revenue) AS daily\_revenue FROM revenue\_table WHERE date >= 'start\_date' AND date <= 'end\_date' GROUP BY date; | Query revenue by day for a specific time period |
| 3 | SELECT category, SUM(revenue) AS category\_revenue FROM revenue\_table GROUP BY category; | Revenue query by product category |
| 4 | SELECT product\_name, SUM(revenue) AS product\_revenue FROM revenue\_table WHERE product\_id = 'product\_id' GROUP BY product\_name; | Revenue query by specific product |
| 5 | SELECT user\_id, SUM(revenue) AS user\_revenue FROM revenue\_table WHERE user\_id = 'user\_id' GROUP BY user\_id; | Revenue query by user or customer |

## 27. View detailed order history

### a. Class Diagram

**

### b. Class Specifications

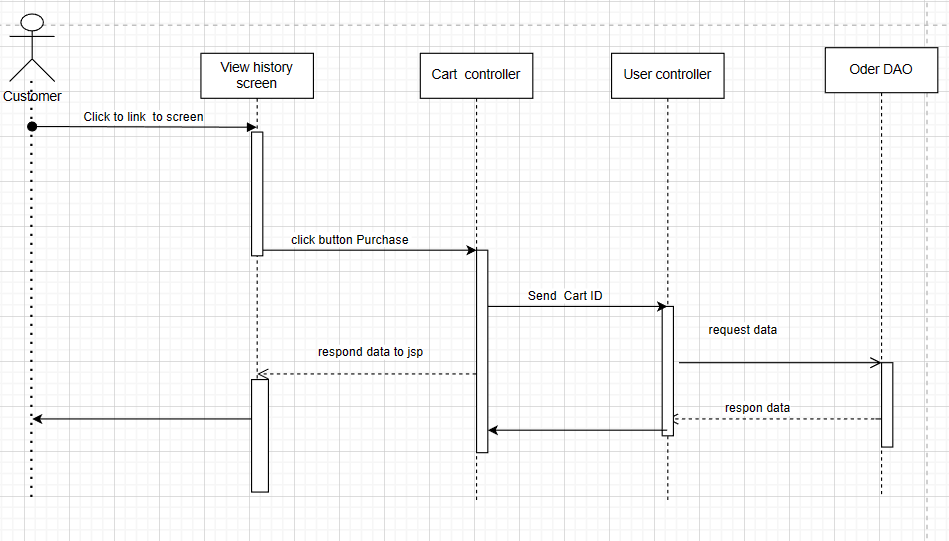
| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *Order\_ID* | *to contain the id of the order* |
| *2* | *Username* | *to contain the name of that account* |
| *3* | *Status order* | *to contain the status of the order* |
| *4* | *delivery\_address* | *to contain the address address* |
| *5* | *total\_price* | *to contain the total price of the order* |
| *6* | *status\_payment* | *to contain the status of job payment* |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *1* | *getter & setter Order* | *get and edit the information of Order* |
| *2* | *getter & setter user* | *get and edit the information of User* |

### c. Sequence Diagram(s)

**

### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | Select username, role, email, fullname, address, phone , img From User\_Table | To get the list of users |
| 2 | Select \* from CartUser\_Table where cart\_id = ? | Get cart data by cart id |

## 28.CRUD employees

### a. Class Diagram

**

### b. Class Specifications

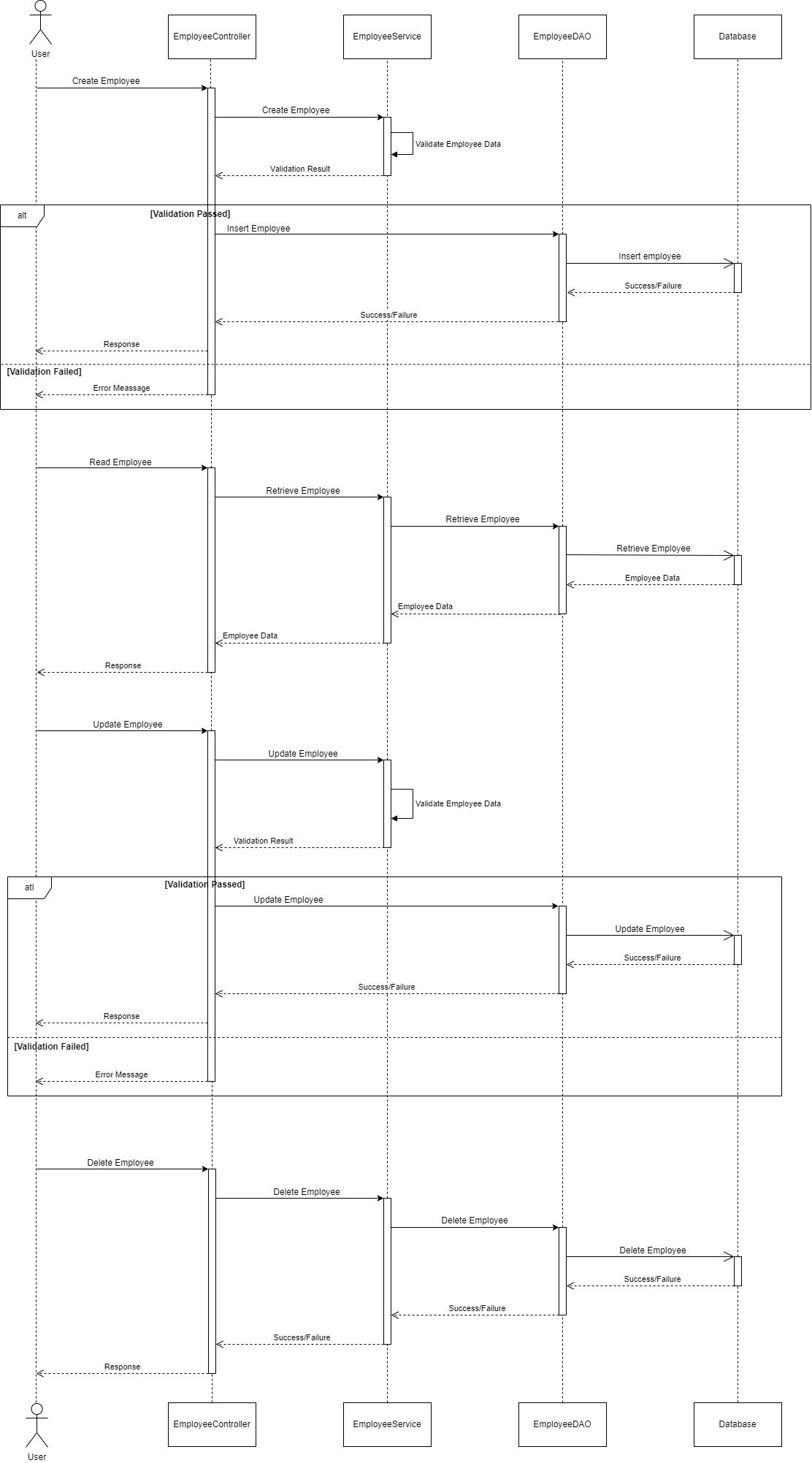
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | id | The ID of the employee. |
| 2 | name | The name of the employee. |
| 3 | position | The position/title of the employee. |
| 4 | department | The department of the employee. |
| 5 | email | The email address of the employee. |
| 6 | phoneNumber | The phone number of the employee. |
| 7 | employeeService  (EmployeeService) | An instance of the EmployeeService class. |
| 8 | employeeDAO (EmployeeDAO) | An instance of the EmployeeDAO class. |

#### 

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | Employee(id: int, name: string, position: string, department: string, email: string, phoneNumber: string) | Constructor method to create a new Employee object with the provided attributes. |
| 2 | getId() | Retrieves the ID of the employee. |
| 3 | getName() | Retrieves the name of the employee. |
| 4 | getPosition() | Retrieves the position/title of the employee. |
| 5 | getDepartment() | Retrieves the department of the employee. |
| 6 | getEmail() | Retrieves the email address of the employee. |
| 7 | getPhoneNumber() | Retrieves the phone number of the employee. |
| 8 | setName(name: string) | Sets/updates the name of the employee. |
| 9 | setPosition(position: string) | Sets/updates the position/title of the employee. |
| 10 | setDepartment(department: string) | Sets/updates the department of the employee. |
| 11 | setEmail(email: string) | Sets/updates the email address of the employee. |
| 12 | setPhoneNumber(phoneNumber: string) | Sets/updates the phone number of the employee. |
| 13 | createEmployee(name: string, position: string, department: string, email: string, phoneNumber: string) | Creates a new employee with the provided details and returns the ID of the created employee. |
| 14 | retrieveEmployee(employeeId: int) | Retrieves the employee object with the specified ID. |
| 15 | updateEmployee(employeeId: int, name: string, position: string, department: string, email: string, phoneNumber: string) | Updates the details of the employee with the specified ID and returns true if the update is successful. |
| 16 | deleteEmployee(employeeId: int) | Deletes the employee with the specified ID and returns true if the deletion is successful. |
| 17 | insertEmployee(employee: Employee) | Inserts the employee object into the database and returns true if the insertion is successful. |
| 18 | retrieveEmployee(employeeId: int) | Retrieves the employee object with the specified ID from the database. |
| 19 | updateEmployee(employeeId: int, employee: Employee) | Updates the details of the employee with the specified ID in the database and returns true if the update is successful. |
| 20 | deleteEmployee(employeeId: int) | Deletes the employee with the specified ID from the database and returns true if the deletion is successful. |

### c. Sequence Diagram(s)



### d. Database Queries

| **No** | **Query** | **Use for** |
| --- | --- | --- |
| 1 | INSERT INTO Employees (id, name, position, department, email, phoneNumber) VALUES (?, ?, ?, ?, ?, ?) | This query is used to insert a new employee record into the "Employees" table in the database. The question marks represent the placeholders for the actual values to be inserted. |
| 2 | SELECT \* FROM Employees WHERE id = ? | This query is used to retrieve an employee record from the "Employees" table based on the provided employee ID. The question mark represents the placeholder for the actual employee ID value. |
| 3 | UPDATE Employees SET name = ?, position = ?, department = ?, email = ?, phoneNumber = ? WHERE id = ? | This query is used to update the details of an existing employee record in the "Employees" table based on the provided employee ID. The question marks represent the placeholders for the actual values to be updated. |
| 4 | DELETE FROM Employees WHERE id = ? | This query is used to delete an employee record from the "Employees" table based on the provided employee ID. The question mark represents the placeholder for the actual employee ID value. |