

**<<BOOKIE>>**

**Software Design Document**

– Hanoi, September 2022 –

Contents

[I. Overview 8](#_heading=h.gjdgxs)

[1. Code Packages/Namespaces 8](#_heading=h.30j0zll)

[II. Code Designs 9](#_heading=h.1fob9te)

[1. Homepage 9](#_heading=h.3znysh7)

[a. Class Diagram 9](#_heading=h.2et92p0)

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

[c. Sequence Diagram(s) 11](#_heading=h.3dy6vkm)

[d. Database queries 11](#_heading=h.1t3h5sf)

[2. Signup 12](#_heading=h.4d34og8)

[a. Class Diagram 12](#_heading=h.2s8eyo1)

[b. Class Specifications 12](#_heading=h.17dp8vu)

[c. Sequence Diagram(s) 13](#_heading=h.3rdcrjn)

[d. Database queries 14](#_heading=h.26in1rg)

[3. Login 15](#_heading=h.lnxbz9)

[a. Class Diagram 15](#_heading=h.35nkun2)

[b. Class Specifications 15](#_heading=h.1ksv4uv)

[c. Sequence Diagram(s) 16](#_heading=h.44sinio)

[d. Database queries 16](#_heading=h.2jxsxqh)

[4. Logout 16](#_heading=h.z337ya)

[a. Class Diagram 16](#_heading=h.3j2qqm3)

[b. Class Specifications 16](#_heading=h.1y810tw)

[c. Sequence Diagram(s) 17](#_heading=h.4i7ojhp)

[d. Database queries 17](#_heading=h.2xcytpi)

[5. Forget password 17](#_heading=h.1ci93xb)

[a. Class Diagram 17](#_heading=h.3whwml4)

[b. Class Specifications 17](#_heading=h.qsh70q)

[c. Sequence Diagram(s) 19](#_heading=h.35xuupr)

[d. Data queries 19](#_heading=h.1l354xk)

[6. View Admin Profile 20](#_heading=h.3as4poj)

[a. Class Diagram 20](#_heading=h.1pxezwc)

[b. Class Specifications 20](#_heading=h.49x2ik5)

[c. Sequence Diagram(s) 21](#_heading=h.2p2csry)

[d. Database queries 21](#_heading=h.147n2zr)

[7. Update Admin Profile 22](#_heading=h.3o7alnk)

[a. Class Diagram 22](#_heading=h.23ckvvd)

[b. Class Specifications 22](#_heading=h.ihv636)

[c. Sequence Diagram(s) 23](#_heading=h.32hioqz)

[d. Database queries 23](#_heading=h.1hmsyys)

[8. View list book 23](#_heading=h.41mghml)

[a. Class Diagram 23](#_heading=h.2grqrue)

[b. Class Specifications 24](#_heading=h.vx1227)

[c. Sequence Diagram(s) 24](#_heading=h.3fwokq0)

[d. Data queries 24](#_heading=h.1v1yuxt)

[9. View book details 25](#_heading=h.4f1mdlm)

[a. Class Diagram 25](#_heading=h.2u6wntf)

[b. Class Specifications 25](#_heading=h.19c6y18)

[c. Sequence Diagram(s) 26](#_heading=h.3tbugp1)

[d. Data queries 26](#_heading=h.28h4qwu)

[10. Profile 26](#_heading=h.nmf14n)

[a. Class Diagram 26](#_heading=h.37m2jsg)

[b. Class Specifications 26](#_heading=h.1mrcu09)

[c. Sequence Diagram(s) 27](#_heading=h.46r0co2)

[d. Database queries 27](#_heading=h.2lwamvv)

[11. Edit profile 27](#_heading=h.111kx3o)

[a. Class Diagram 27](#_heading=h.3l18frh)

[b. Class Specifications 27](#_heading=h.206ipza)

[c. Sequence Diagram(s) 28](#_heading=h.4k668n3)

[d. Database queries 28](#_heading=h.2zbgiuw)

[12. View security 29](#_heading=h.1egqt2p)

[a. Class Diagram 29](#_heading=h.3ygebqi)

[b. Class Specifications 29](#_heading=h.2dlolyb)

[c. Sequence Diagram(s) 30](#_heading=h.sqyw64)

[d. Database queries 30](#_heading=h.3cqmetx)

[13. Change password 30](#_heading=h.1rvwp1q)

[a. Class Diagram 30](#_heading=h.4bvk7pj)

[b. Class Specifications 30](#_heading=h.2r0uhxc)

[c. Sequence Diagram(s) 31](#_heading=h.1664s55)

[d. Database queries 32](#_heading=h.3q5sasy)

[14. Comment 32](#_heading=h.25b2l0r)

[a. Class diagram 32](#_heading=h.kgcv8k)

[b. Class Specifications 32](#_heading=h.34g0dwd)

[c. Sequence Diagram 33](#_heading=h.1jlao46)

[d. Database queries 33](#_heading=h.43ky6rz)

[15. Rating Book 34](#_heading=h.2iq8gzs)

[a. Class diagram 34](#_heading=h.xvir7l)

[b. Class Specifications 34](#_heading=h.3hv69ve)

[c. Sequence Diagram 35](#_heading=h.1x0gk37)

[d. Database queries 35](#_heading=h.4h042r0)

[16. Add to Favorite 35](#_heading=h.2w5ecyt)

[a. Class diagram 35](#_heading=h.1baon6m)

[b. Class Specifications 35](#_heading=h.3vac5uf)

[c. Sequence Diagram 36](#_heading=h.2afmg28)

[d. Database queries 36](#_heading=h.pkwqa1)

[17. Report Book 37](#_heading=h.39kk8xu)

[a. Class diagram 37](#_heading=h.1opuj5n)

[b. Class Specifications 37](#_heading=h.48pi1tg)

[c. Sequence Diagram 38](#_heading=h.2nusc19)

[d. Database queries 38](#_heading=h.1302m92)

[18.1 View List Books 38](#_heading=h.3mzq4wv)

[a. Class Diagram 38](#_heading=h.2250f4o)

[b. Class Specifications 38](#_heading=h.haapch)

[c. Sequence Diagram(s) 39](#_heading=h.upglbi)

[d. Data queries 39](#_heading=h.3ep43zb)

[18.2 Add Book 40](#_heading=h.1tuee74)

[a. Class Diagram 40](#_heading=h.4du1wux)

[b. Class Specifications 40](#_heading=h.184mhaj)

[c. Sequence Diagram(s) 41](#_heading=h.1ljsd9k)

[d. Data queries 41](#_heading=h.45jfvxd)

[18.3 Update Book 42](#_heading=h.2koq656)

[a. Class Diagram 42](#_heading=h.zu0gcz)

[b. Class Specifications 42](#_heading=h.3jtnz0s)

[c. Sequence Diagram(s) 43](#_heading=h.3x8tuzt)

[d. Data queries 43](#_heading=h.rjefff)

[18.4. Disable/Enable Book 44](#_heading=h.3bj1y38)

[a. Class Diagram 44](#_heading=h.1qoc8b1)

[b. Class Specifications 44](#_heading=h.4anzqyu)

[c. Sequence Diagram(s) 45](#_heading=h.2pta16n)

[d. Data queries 45](#_heading=h.14ykbeg)

[19.1 View Table of Contents 46](#_heading=h.3oy7u29)

[a. Class Diagram 46](#_heading=h.243i4a2)

[b. Class Specifications 46](#_heading=h.j8sehv)

[c. Sequence Diagram(s) 47](#_heading=h.338fx5o)

[d. Data queries 47](#_heading=h.1idq7dh)

[19.2 Add Volume 49](#_heading=h.42ddq1a)

[a. Class Diagram 49](#_heading=h.2hio093)

[b. Class Specifications 49](#_heading=h.wnyagw)

[c. Sequence Diagram(s) 50](#_heading=h.3gnlt4p)

[d. Data queries 50](#_heading=h.1vsw3ci)

[19.3. Edit Volume 51](#_heading=h.4fsjm0b)

[a. Class Diagram 51](#_heading=h.2uxtw84)

[b. Class Specifications 51](#_heading=h.1a346fx)

[c. Sequence Diagram(s) 52](#_heading=h.3u2rp3q)

[d. Data queries 52](#_heading=h.2981zbj)

[19.4. Delete Volume 53](#_heading=h.odc9jc)

[a. Class Diagram 53](#_heading=h.38czs75)

[b. Class Specifications 53](#_heading=h.1nia2ey)

[c. Sequence Diagram(s) 53](#_heading=h.47hxl2r)

[d. Data queries 54](#_heading=h.2mn7vak)

[19.5 Add Chapter 54](#_heading=h.11si5id)

[a. Class Diagram 54](#_heading=h.3ls5o66)

[b. Class Specifications 54](#_heading=h.20xfydz)

[c. Sequence Diagram(s) 55](#_heading=h.4kx3h1s)

[d. Data queries 56](#_heading=h.302dr9l)

[19.6 Edit Chapter 56](#_heading=h.1f7o1he)

[a. Class Diagram 56](#_heading=h.3z7bk57)

[b. Class Specifications 57](#_heading=h.2eclud0)

[c. Sequence Diagram(s) 58](#_heading=h.thw4kt)

[d. Data queries 58](#_heading=h.3dhjn8m)

[19.7 Delete Chapter 58](#_heading=h.1smtxgf)

[a. Class Diagram 58](#_heading=h.4cmhg48)

[b. Class Specifications 58](#_heading=h.2rrrqc1)

[c. Sequence Diagram(s) 59](#_heading=h.16x20ju)

[d. Data queries 59](#_heading=h.3qwpj7n)

[20. View List of User Account 60](#_heading=h.261ztfg)

[a. Class Diagram 60](#_heading=h.l7a3n9)

[b. Class Specifications 60](#_heading=h.1kc7wiv)

[c. Sequence Diagram(s) 61](#_heading=h.ymfzma)

[d. Database queries 61](#_heading=h.1xrdshw)

[21. View List of User Account 61](#_heading=h.2wwbldi)

[a. Class Diagram 61](#_heading=h.1c1lvlb)

[b. Class Specifications 62](#_heading=h.2b6jogx)

[c. Sequence Diagram(s) 64](#_heading=h.qbtyoq)

[d. Database queries 65](#_heading=h.1pgrrkc)

[22. Reading interface 66](#_heading=h.49gfa85)

[a. Class Diagram 66](#_heading=h.2olpkfy)

[b. Class Specifications 66](#_heading=h.13qzunr)

[c. Sequence Diagram(s) 68](#_heading=h.22vxnjd)

[d. Database queries 68](#_heading=h.i17xr6)

[23. Pre-read 69](#_heading=h.320vgez)

[a. Class Diagram 69](#_heading=h.1h65qms)

[b. Class Specifications 69](#_heading=h.415t9al)

[c. Sequence Diagram(s) 71](#_heading=h.1ulbmlt)

[d. Database queries 71](#_heading=h.4ekz59m)

[24. Table of Contents 72](#_heading=h.2tq9fhf)

[a. Class Diagram 72](#_heading=h.18vjpp8)

[b. Class Specifications 72](#_heading=h.3sv78d1)

[c. Sequence Diagram(s) 74](#_heading=h.280hiku)

[d. Database queries 74](#_heading=h.n5rssn)

[25. Wallet 75](#_heading=h.375fbgg)

[a. Class Diagram 75](#_heading=h.1maplo9)

[b. Class Specifications 75](#_heading=h.46ad4c2)

[c. Sequence Diagram(s) 78](#_heading=h.2lfnejv)

[d. Database queries 79](#_heading=h.10kxoro)

[26. Payment 80](#_heading=h.3kkl7fh)

[a. Class Diagram 80](#_heading=h.1zpvhna)

[c. Sequence Diagram(s) 83](#_heading=h.4jpj0b3)

[d. Database queries 83](#_heading=h.2yutaiw)

[27. View purchase history 84](#_heading=h.1e03kqp)

[a. Class Diagram 84](#_heading=h.3xzr3ei)

[b. Class Specifications 84](#_heading=h.2d51dmb)

[c. Sequence Diagram(s) 86](#_heading=h.sabnu4)

[d. Database queries 86](#_heading=h.3c9z6hx)

[28. Purchase a book 87](#_heading=h.1rf9gpq)

[a. Class Diagram 87](#_heading=h.4bewzdj)

[b. Class Specifications 87](#_heading=h.2qk79lc)

[c. Sequence Diagram(s) 89](#_heading=h.15phjt5)

[d. Database queries 90](#_heading=h.3pp52gy)

[29. Change email 90](#_heading=h.24ufcor)

[a. Class Diagram 90](#_heading=h.jzpmwk)

[b. Class Specifications 91](#_heading=h.33zd5kd)

[c. Sequence Diagram(s) 92](#_heading=h.1j4nfs6)

[d. Database queries 92](#_heading=h.434ayfz)

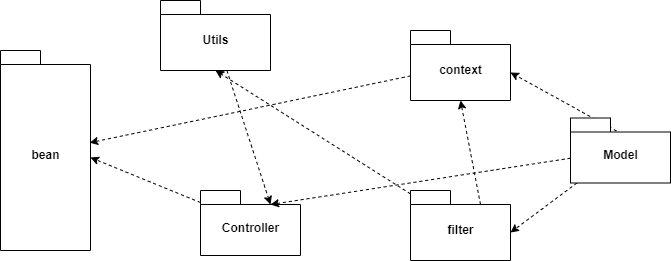
[III. Database Design 93](#_heading=h.2i9l8ns)

[1. Database Schema 93](#_heading=h.26sx1u5)

[2. Table Description 93](#_heading=h.ly7c1y)

# I. Overview

## 1. Code Packages/Namespaces



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

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| 01 | DAO | Used to contain files that arre related to Database Connected Files |
| 02 | controller | Used to contain files that are directly control the system |
| 03 | Utilities | Used to contain files that helps the controllers to manage the system |
| 04 | Filter | Used to contain files whose purpose are user authorization and encoding the web |
| 05 | Model | Used to contain files that represent all the object appear in the project |
|  |  |  |

# II. Code Designs

## 1. Homepage

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

**- BookDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getNumberBook | Return an integer which is quantity of books |
| 02 | getWeeklySaleBooks | Return a list contains 4 that has sale in week to display in homepage |
| 03 | getFeaturedBooks | Random selected 15 books in all book to display in homepage |
| 04 | getFavouriteBooks | Get top 3 books that has highest user choose to display in homepage |

**- Book class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Book(int id, String title, String author, int categoryid, float rating, int favourite, float price, boolean issale, String image, String description, int views) | Get all information of a books for details |
| 02 | Book(int id, String title, String author, float price, boolean issale, int favourite, String image) | Get general information of a books for display on homepage |

**- Home class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | processRequest(HttpServletRequest request, HttpServletResponse response) | - ArrayList to take a list from getFeaturedBooks() from Book  - ArrayList to take a list from getWeeklySaleBooks() from Book  - ArrayList to take a list from getFavouriteBooks() from Book |
| 02 | doGet() | Doing processRequest |
| 03 | doPost() | Doing processRequest |

**- DBContext class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Database queries

- getFeaturedBooks(): “select \* from [Book]”

- getWeeklySaleBooks(): "select \* from [Book] where [is\_sale] = 1"

- getFavoriteBooks(): “select top(3)\* from BOOK order by favourite desc”

## 2. Signup

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

**- UserDAO class  
*Class Method***

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getUser | Get username and password |
| 02 | generatePass | Encode users’ password |
| 03 | checkDupEmail | Checking that user’s email is exist or not |
| 04 | checkDupUsername | Checking that user’s username is exist or not |
| 05 | editProfile | Allow user to edit their information |
| 06 | changePassword | Allow user to change their password |
| 07 | getNumberUser | Get Total Number |
| 08 | getAllUser | Taking a list contains all user includes their information |
| 09 | getUserName | Taking user’s ursername |
| 10 | updateUser | Update information of user |
| 11 | checkOldPass | Checking user’s old password when they want to change their one. |

**- User class  
*Class Method***

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | User(int id, String name, String gender, String dob, String email, String phone, String address, String username, String password, boolean is\_super) | Get all information of user for detail |
| 02 | User(String name, String gender, String dob, String email, String phone, String address, String username, String password) | Get general information of user |

**- DBContext class  
Class Method**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

### c. Sequence Diagram(s)

Diagram, timeline

Description automatically generated

### d. Database queries

- getUser(): "Select \* from [User] where (username=? OR email=?) AND password=?"

- checkDupEmail(String email): "select \* from [User] where email='" + email + "'"

- checkDupUsername(String key): "select \* from [User] where username='" + key + "'"

- createNewUser(String name, String gender, String dob, String email, String phone, String username, String password): "insert [User] ([fullname],[gender] ,[dob] , [email], [phone],[username], [password])"

+ " values (N'" + name + "',"

+ "N'" + (gender.equals("Male") ? 1 : 0) + "',"

+ "CAST(N'" + dob+ "' AS Date),"

+ "N'" + email + "',"

+ "N'" + phone + "',"

+ "N'" + username + "',"

+ "N'" + password + "')"

- editProfile(int userid, String name, String gender, String dob, String email, String phone, String address): "update [User] set "

+ " [fullname] = N'" + name+ "' AND"

+ " [gender] = " + (gender.equals("male") ? 1 : 0) + " AND"

+ " [dob] = N'" + dob + "' AND"

+ " [phone] = N'" + phone + "' AND"

+ " [address] = N'"+ address+"'"

+ "where [id] = "+userid

- changePassword(int userid, String newPassword): "update [User] set " + " [password] = N'" +newPassword + "'"+ "where [id] = "+userid

- getNumberUser(): "select count([id]) from [User] where [is\_super] = 'False'"

- getAllUsers(): "Select \* from [User] where [is\_super] = 0"

- getUsername(int userid): "select [username] from [User] where [id] = "+userid

- updateUser(int id, String name, String phone, String address): "update [User] set "

+ " [fullname] = N'" +name + "'"

+ ", [phone] = N'"+phone+"'"

+ ", [address] = N'"+address+"'"

+ "where [id] = "+id

- checkOldPass(int id,String oldpass): "select \* from [User] where [id] = "+id+" AND [password]=N'"+oldpass+"'"

## 3. Login

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### LoginController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getUser()* | *Method used to check if an account record existed in database and get all data of that account.Inputs are username and password.Return an <Code>model.User</Code> represents the user’s data* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram, schematic

Description automatically generated

### d. Database queries

Select \* from [User] where (username=? OR email=?) AND password=?

## 4. Logout

### a. Class Diagram

|  |
| --- |
| LogoutController |
| **processRequest(request: HttpServletRequest, response: HttpServletResponse)**  **doGet()**  **doPost()** |

### b. Class Specifications

**- Logout Controller**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | processRequest(HttpServletRequest request, HttpServletResponse response) | - Destroy session  - Redirect to Homepage |
| 02 | doGet() | Doing processRequest |
| 03 | doPost() | Doing processRequest |

### c. Sequence Diagram(s)

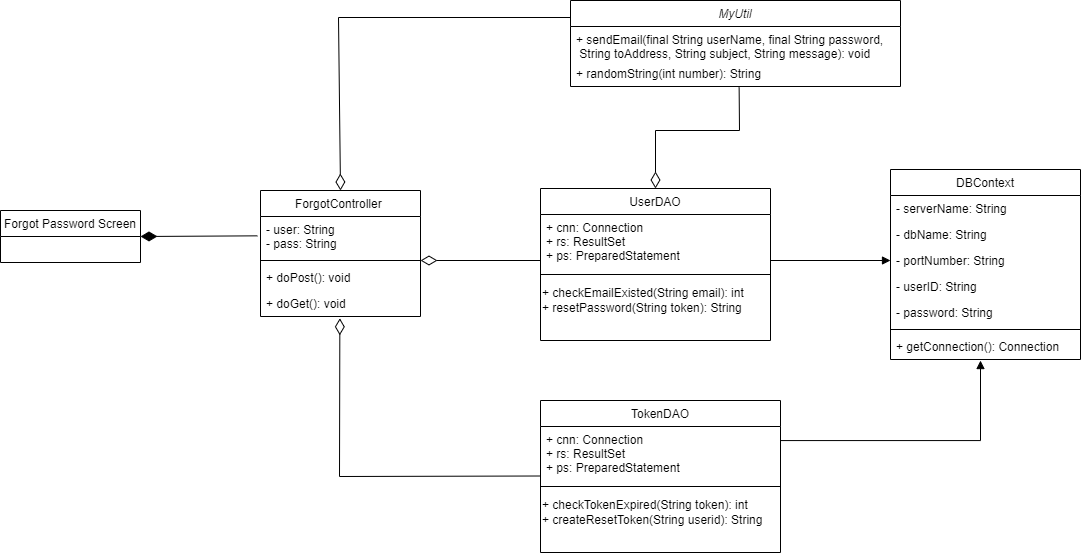
Diagram

Description automatically generated with medium confidence

### d. Database queries

## 5. Forget password

### a. Class Diagram



### b. Class Specifications

#### ForgetController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### UserDAO

#### Class Methods

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *checkEmailExisted* | *Method used to check if an email of a User record existed in the database and get id data of that account. Input is email. Return an Integer represents the account‘s id.* |
| *2* | *resetPassword* | *Method used to update password of a User record if the token input is still in expiration time. Return a String represents the new user’s password* |

#### TokenDAO

#### Class Methods

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *createResetToken* | *Method used to insert a token with expiration time to a Token record . Input is the user's id. Return the generated token as a String* |
| *2* | *checkTokenExpired* | *Method used to check a token valid if the token input is still in expiration time. Return an integer represents userId of that token* |

#### DBContext

***Class Methods***

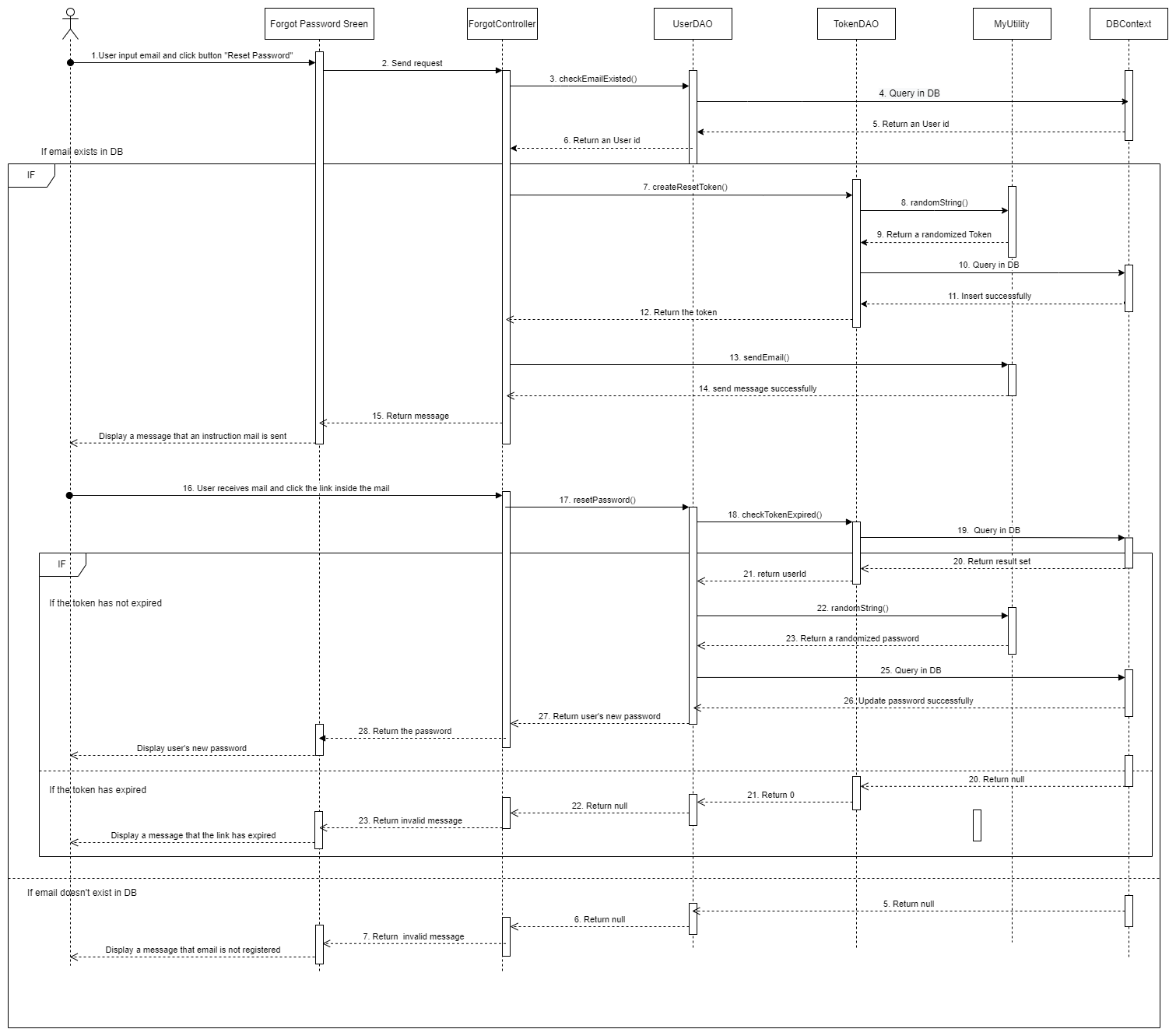
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a connection to the database* |

#### MyUtil

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *sendEmail* | *Method used to send an email of a reset instruction. Inputs include recipient’s email and message .* |
| *2* | *randomString* | *Method used to generate a randomized any-number-character string. Return that string.* |

### c. Sequence Diagram(s)

**

### d. Data queries

* **Get id of a record with specific email from User table**

SELECT id

FROM [User]

WHERE [email] = ?

* **Insert a record with specific user id into Token table**

INSERT INTO [Token]

([userId]

, [token

], [expiredDate])

VALUES ( ?

, ?

, ?)

* **Update password of a record in User table**

UPDATE [User]

SET [password] = ?

WHERE [id] =  ?

## 6. View Admin Profile

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### LoginController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### EditController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getUser()* | *Method used to check if an account record existed in database and get all data of that account.Inputs are username and password.Return an <Code>model.User</Code> represents the user’s data* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Database queries

Select \* from [User] where (username=? OR email=?) AND password=?

## 7. Update Admin Profile

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### EditController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *editProfile()* | *Method used to let admin update the data of that account.Inputs are admin details.Update those to database.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Database queries

update[User] set

[email] = ?,

[fullname] = ?,

[phone] = ?,

[address] = ?

where [id] = ?

## 8. View list book

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### BookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Get categoryID from the HomePage( when user interact with the system ) then start get data from BookDAO to collect books that match the categoryID comefrom Homepage* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBooks()* | *Get all the book in the list book* |
| *2* | *getBookBycID(cID)* | *Get all the books that match the cID* |

#### CategoryDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getCategory()* | *Get the specify category from List Category* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

* select \* from [Category]
* select \* from [Book] where [categoryid] =
* select \* from [Book]

## 9. View book details

### a. Class Diagram

**Diagram

Description automatically generated**

### b. Class Specifications

#### BookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Get bookID from the Homepage(when user interact with the system )then start get the data from BookDAO to collect books that match the bookID come from home page* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBooksByID(ID)* | *Get book match the ID* |

#### CategoryDAO

***Class Methods***

|  |  |  |  |
| --- | --- | --- | --- |
| **No** |  | **Method** | **Description** |
| *1* |  | *getCategory()* | *Get the specify category from List Category* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

**Diagram

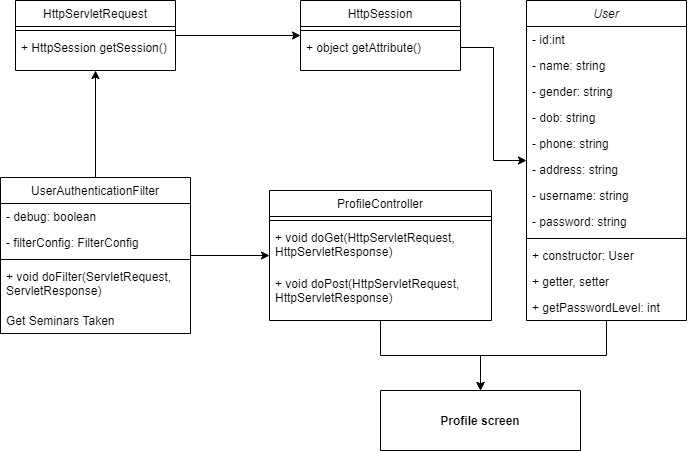
Description automatically generated**

### d. Data queries

* select \* from [Book] where [id] =

## 10. Profile

### a. Class Diagram



### b. Class Specifications

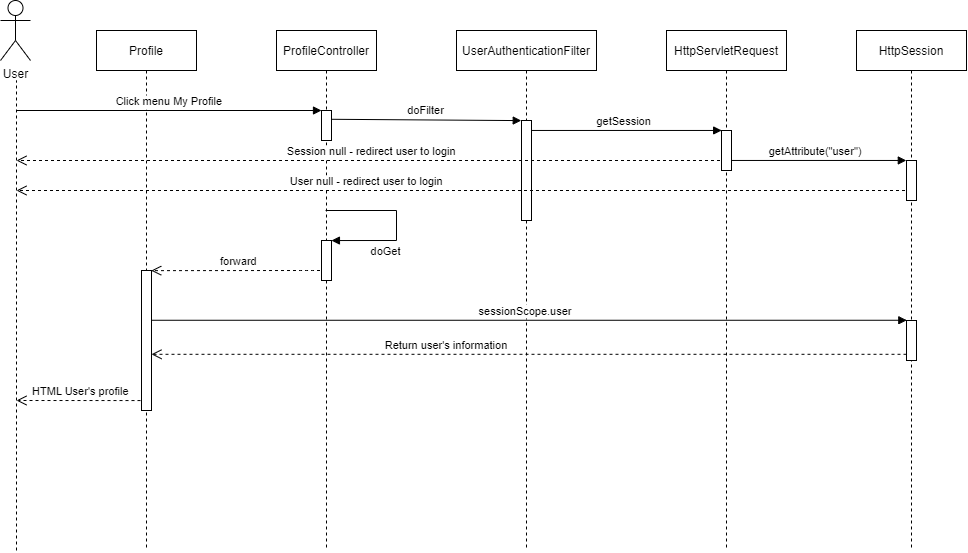
#### ProfileController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### UserAuthenticationFilter class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter | Handles the HTTP <code>GET</code> method. |

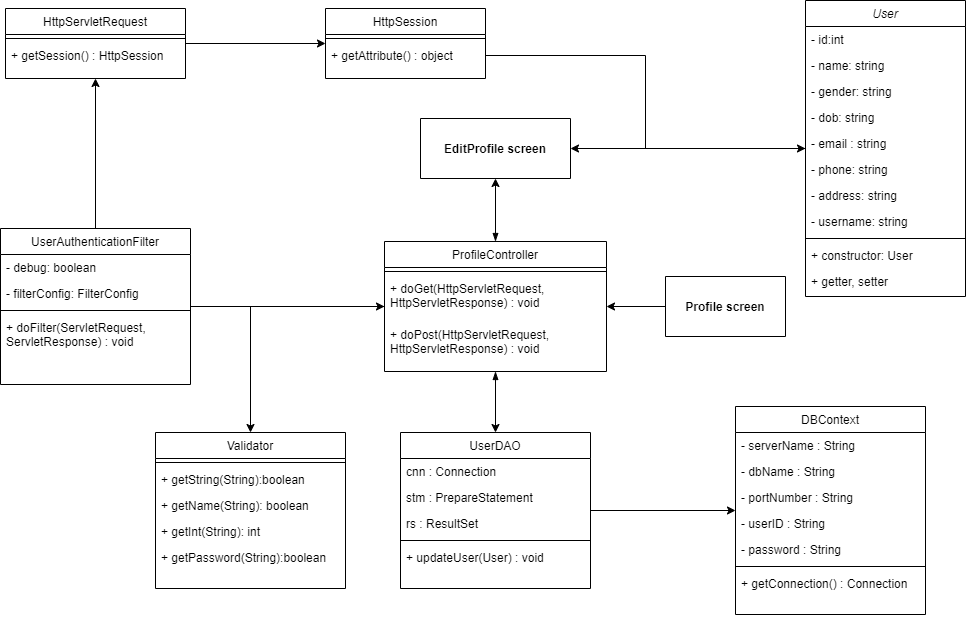
### c. Sequence Diagram(s)



### d. Database queries

## 11. Edit profile

### a. Class Diagram



### b. Class Specifications

#### ProfileController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### UserAuthenticationFilter class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter | Handles the HTTP <code>GET</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getId | Get ID of user |
| 02 | getName | Get name of user |
| 03 | getGender | Get gender of user |
| 04 | getPhone | Get phone number of user |
| 05 | getAddress | Get address of user |
| 06 | getEmail | Get email of user |
| 07 | setName | Set name of user |
| 08 | setPhone | Set phone number of user |
| 09 | setAddress | Set address of user |
| 10 | setGender | Set gender of user |

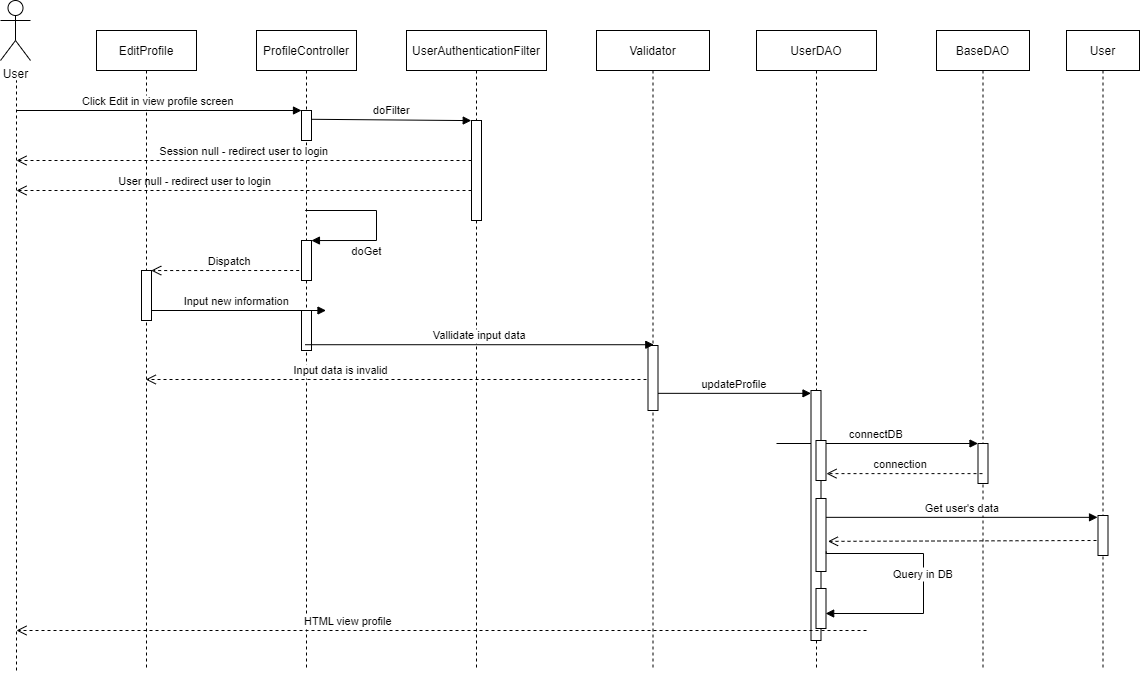
#### UserDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | updateProfile | Update information of user in database |

#### Validator class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getString | Check if string is empty |
| 02 | getName | Check if name is empty or contains digit or so long |

### c. Sequence Diagram(s)

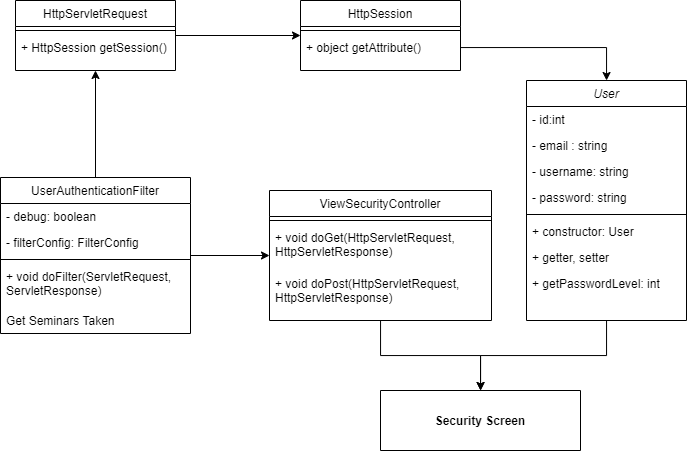


### d. Database queries

update [User] set [fullname] = ?, [phone] = ?, [address] = ?, [gender] = ?, [dob] = ? where [id] = ?

## 12. View security

### a. Class Diagram



### b. Class Specifications

#### ProfileController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

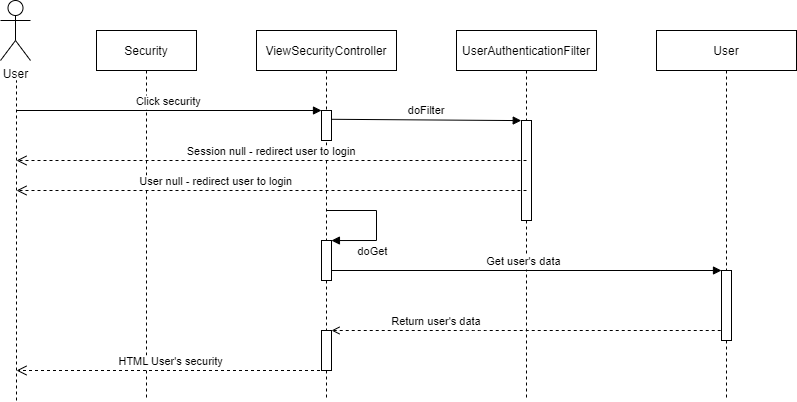
#### UserAuthenticationFilter class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter | Handles the HTTP <code>GET</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getEmail | Get email of user |
| 02 | getPasswordLevel | Get level of password |

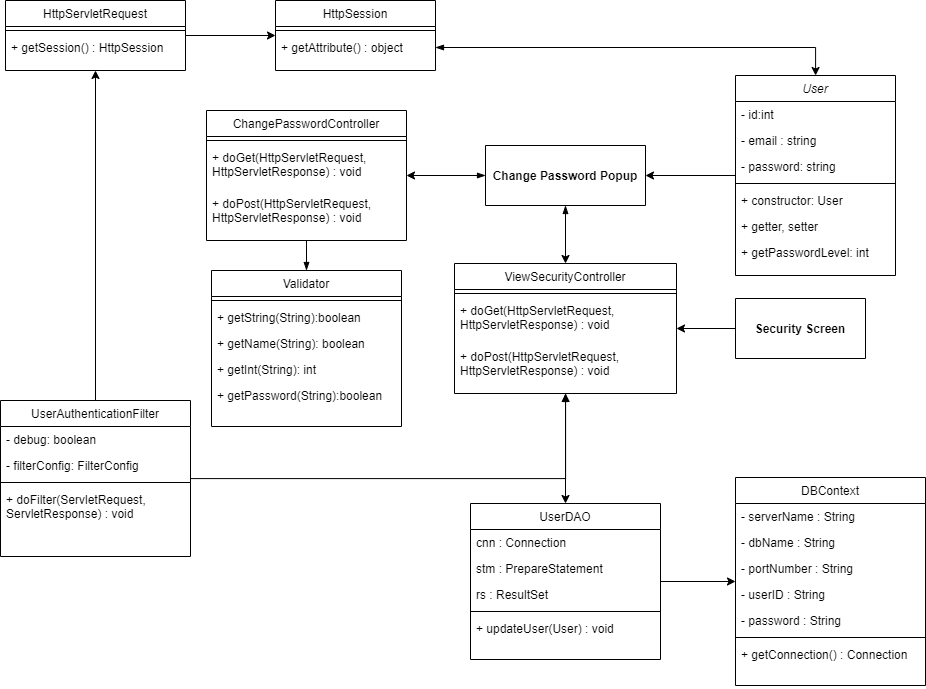
### c. Sequence Diagram(s)



### d. Database queries

## 13. Change password

### a. Class Diagram



### b. Class Specifications

#### ProfileController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### UserAuthenticationFilter class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doFilter | Handles the HTTP <code>GET</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPassword | Get password of user |
| 02 | getId | Get ID of user |

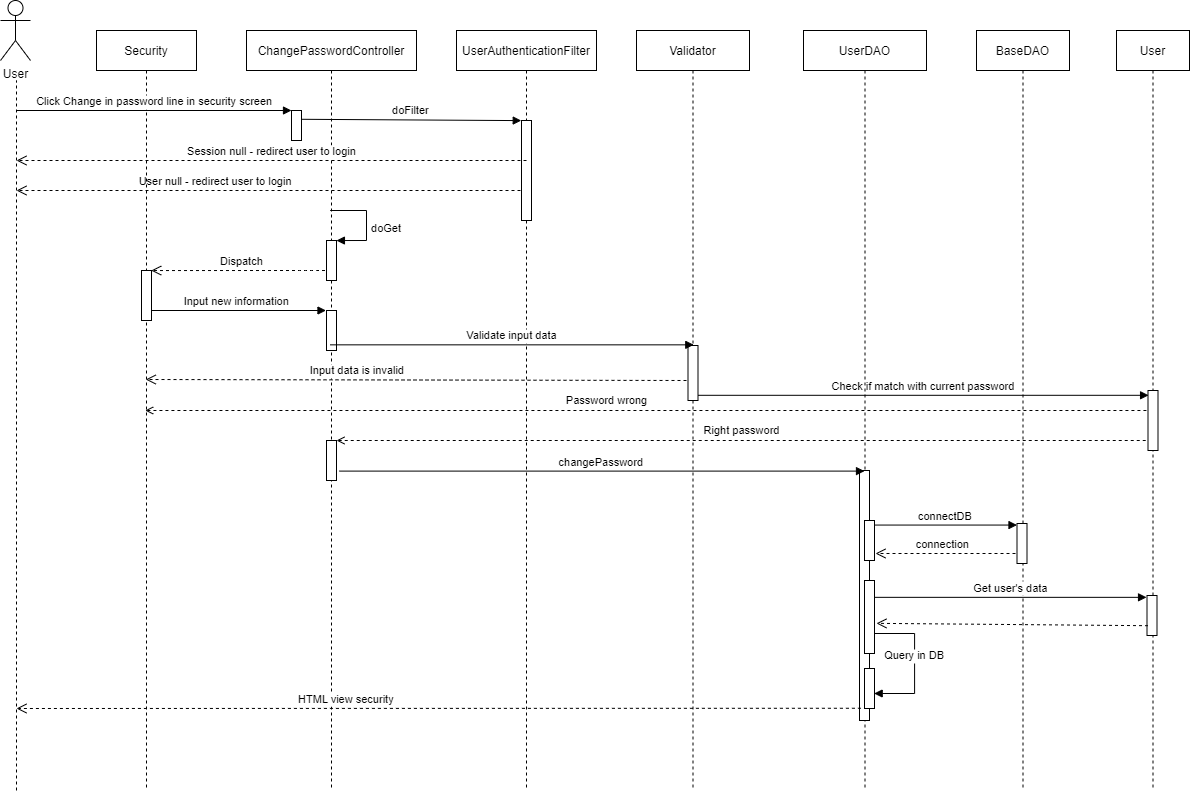
#### UserDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | changePassword | Change password of user by id and new password |

#### Validator class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getString | Check if string is empty |
| 02 | getPassword | Check if password is valid |

### c. Sequence Diagram(s)

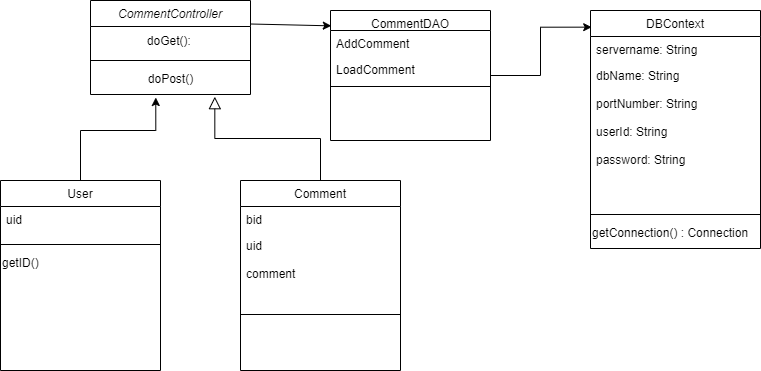


### d. Database queries

update [User] set [password] = ? where [id] = ?

## 14. Comment

### a. Class diagram

****

### b. Class Specifications

#### CommentController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### CommentDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | loadComment | Get all the comment of 1 book |
| 02 | addComment | Insert comment into database |

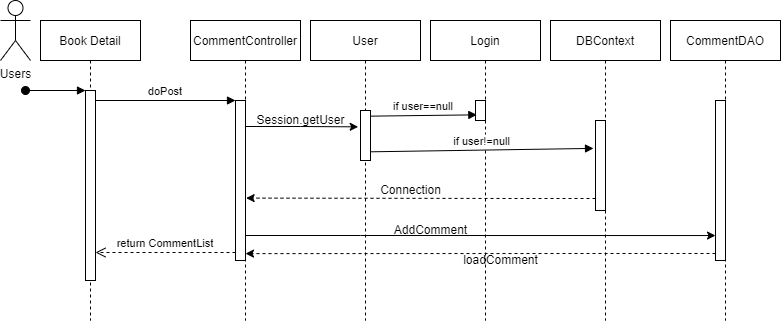
#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getID | Get the ID of the user |

#### DBContext class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | connectDB | Get connection from database |

### c. Sequence Diagram

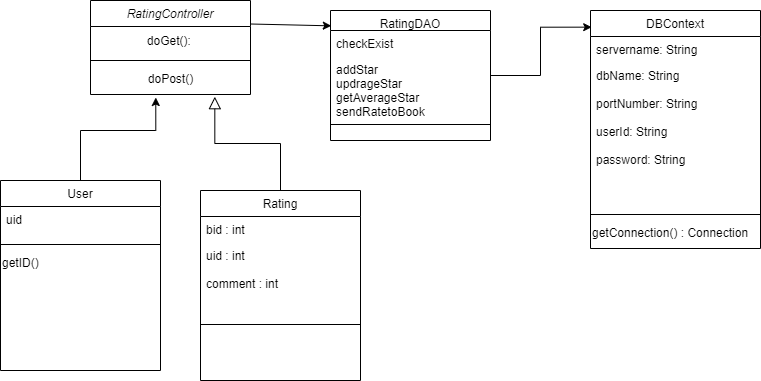
****

### d. Database queries

* INSERT [Comment] VALUES ([bid]='"+bid+"',[uid]='"+uid+"',[comment]='"+comment+"')
* SELECT \* FROM [Comment] WHERE [bid]='"+bid+"'

## 15. Rating Book

### a. Class diagram



### b. Class Specifications

#### RatingController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### RatingDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | checkExist | Check the existance of the rating in the database |
| 02 | addStar | Insert rating into database |
| 03 | updateStar | Update the existed rating in the database |
| 04 | getAverageStar | Take the Rating of the book |
| 05 | sendRatetoBook | Send the average star from [Star] to [Book] |

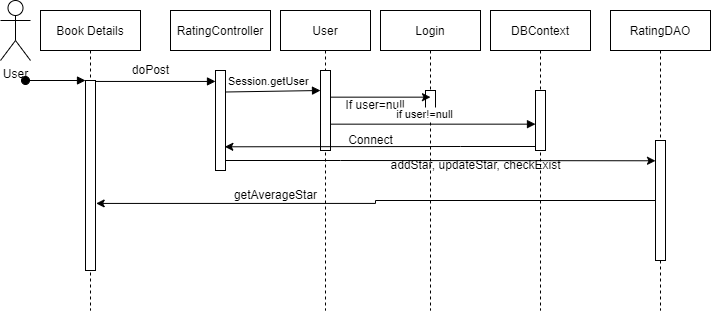
#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getID | Get the ID of the user |

#### DBContext class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | connectDB | Get connection from database |

### c. Sequence Diagram

****

### d. Database queries

- UPDATE [Book] SET [rate]='"+star+"' WHERE [bid]='"+bid+"'

- INSERT INTO [Star] VALUES ([bid]='"+bid+"',[uid]='"+uid+"',[star]='"+star+"')

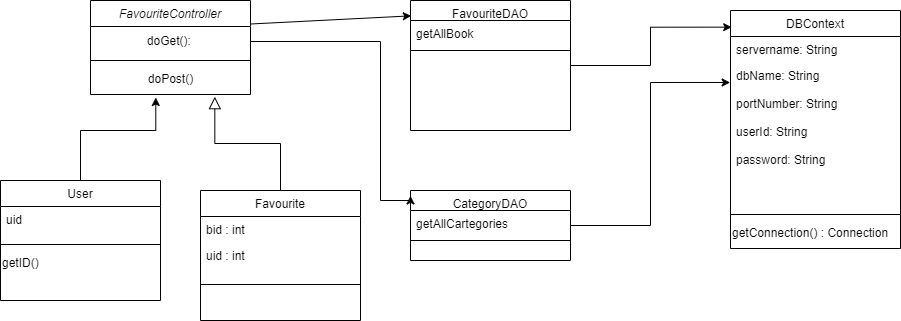
- UPDATE [Star] SET [star]='"+star+"' WHERE [bid]='"+bid+"' and [uid]='"+uid+"'

- SELECT \* FROM [Star] WHERE [bid]='"+bid+"',uid='"+uid+"'

- SELECT \* FROM [Star] WHERE [bid]='"+bid+"'

## 16. Add to Favorite

### a. Class diagram

****

### b. Class Specifications

#### FavouriteController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### FavouriteDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getAllbook | Get all the favourite book of 1 user |

#### BookDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getBookById | Get the specific book by its id |

#### CategoryDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getAllCategories | Get all the categories of the books |

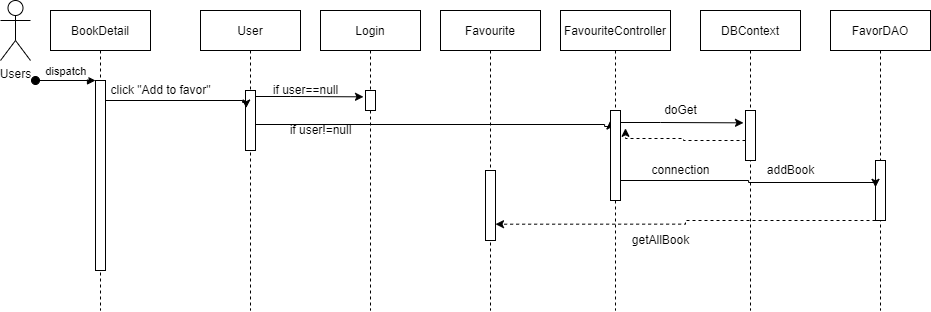
#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getID | Get the ID of the user |

#### DBContext class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | connectDB | Get connection from database |

### c. Sequence Diagram

****

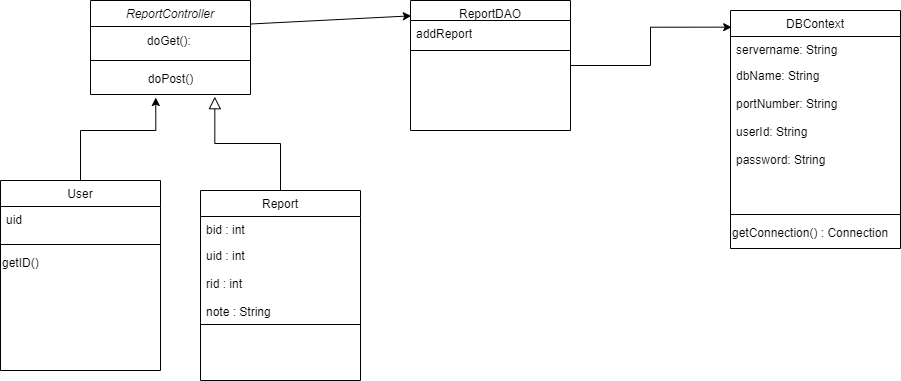
### d. Database queries

- INSERT INTO [Favourite]\n"+"([uid]\n"+",[bid])\n"+ "VALUES\n"+ "("+ uID +",\n" +""+ bID + ")

- SELECT \* FROM [Favourite] WHERE uid='" + uID + "'

## 17. Report Book

### a. Class diagram

****

### b. Class Specifications

#### ReportController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### ReportDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | addReport | Insert report into database |

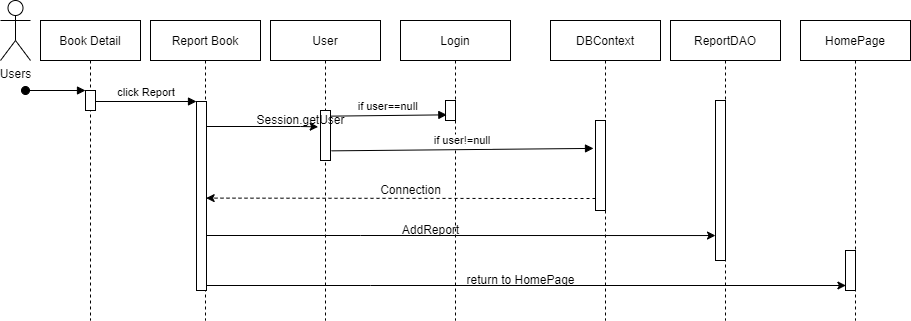
#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getID | Get the ID of the user |

#### DBContext class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | connectDB | Get connection from database |

### c. Sequence Diagram

****

### d. Database queries

**-** INSERT INTO [ReportDetail] (reportId,bookId,userId,note) "+ "VALUES ('"+rid+"','"+bid+"','"+uid+"','"+note+"')

## 18.1 View List Books

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### BooksController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllBook* | *Method used to get all book and its information in database. Return an array list <Book> represents the data of books.* |
| *2* | *getByPage* | *Method used to divide list of books to many page. Return an array list <Book> represents the data of books.* |

#### CategoryDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### 

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

1. getAllBook

SELECT [id]\n"

+ " ,[title]\n"

+ " ,[author]\n"

+ " ,[categoryid]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [dbo].[Book]"

2. getAllCategory

SELECT [id]

,[name]

FROM [dbo].[Category]

## 18.2 Add Book

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### AddBookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *countBookNumber* | *Method used to count the number of books in database. Return the number of books available.* |
| *2* | *addBook* | *Method used to insert the data of a new book into Book table. Return an integer represents result of query execution.* |

#### 

#### CategoryDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### 

### c. Sequence Diagram(s)

*Diagram

Description automatically generated*

### d. Data queries

1. addBook:

INSERT INTO [dbo].[Book]\n"

+ " ([title]\n"

+ " ,[author]\n"

+ " ,[categoryid]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description])\n"

+ " VALUES ( ? "

+ " , ? "

+ " , ? "

+ " , ? "

+ " , ? "

+ " , ? "

+ " , ? )”

2. countNumberBook

SELECT COUNT([id]) FROM [dbo].[Book]

## 18.3 Update Book

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### UpdateBookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

**BookDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |
| *2* | *editBook* | *Method used to update Book into database and return an integer represents sql query result* |

**CategoryDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### 

### c. Sequence Diagram(s)

*Diagram

Description automatically generated*

### 

### d. Data queries

1.*editBook*

UPDATE [Book] "

+ "SET [title]= ?"

+ ", [author] = ? "

+ ", [categoryid] = ?"

+ ", [price] = ?"

+ ", [is\_sale] = ?"

+ ", [image] = ?"

+ ", [description] = ?"

+ " WHERE [id] = ? "

2. getBookById

SELECT [id]\n"

+ " ,[title]\n"

+ " ,[author]\n"

+ " ,[categoryid]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [dbo].[Book]"

+ "WHERE [id] = ?"

## 18.4. Disable/Enable Book

### a. Class Diagram

Text

Description automatically generated with low confidence

### b. Class Specifications

**ChangeStatusController**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

**BookDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *changeStatusById* | *Method used to update status column of the book tuple in database that has the input id.* |

**CategoryDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

*Diagram

Description automatically generated*

### d. Data queries

- changeStatus

UPDATE [Book]"

+ " SET [status] = 1 ^ [status] "

+ " WHERE [id] = ?"

## 19.1 View Table of Contents

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

**TOCController**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

**BookDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |

**VolumeDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllVolume* | *Method used to get all volume and their information of a book has the input id in database. Return an array list <Volume> represents the data of volumes.* |
| *2* | *getVolumeById* | *Method used to get the data of volume that has the input id and return the volume* |

**ChapterDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllChapter* | *Method used to get all chapter and its information of a book has the input id in database. Return an array list <Chapter> represents the data of chapters.* |
| *2* | *getChapterById* | *Method used to get the data of chapter that has the input id and return the chapter* |

**DBContext**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram, schematic

Description automatically generated

### d. Data queries

1. getAllVolume

SELECT [id]\n"

+ " ,[volume]\n"

+ " ,[volumeName]\n"

+ " ,[summary]\n"

+ " FROM [dbo].[Volume]"

+ " WHERE [bookId] = ? "

+ " ORDER BY [volume] asc"

2. getAllChapter

SELECT [id]\n"

+ " ,[volumeId]\n"

+ " ,[chapter]\n"

+ " ,[chapterName]\n"

+ " ,[status]\n"

+ " FROM [dbo].[Chapter]"

+ " WHERE [bookId] = ?"

+ " ORDER BY [chapter] asc"

3. getVolumeById

SELECT [id]\n"

+ " ,[bookId]\n"

+ " ,[volume]\n"

+ " ,[volumeName]\n"

+ " ,[summary]\n"

+ " FROM [dbo].[Volume]"

+ " WHERE [id] = ? "

4. getChapterById

SELECT [id]\n"

+ " ,[bookId]\n"

+ " ,[volumeId]\n"

+ " ,[chapter]\n"

+ " ,[chapterName]\n"

+ " ,[status]\n"

+ " ,[content]\n"

+ " FROM [dbo].[Chapter]"

+ " WHERE [id] = ?"

## 19.2 Add Volume

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### AddVolumeController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |

#### VolumeDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllVolume* | *Method used to get all volume and their information of a book has the input id in database. Return an array list <Volume> represents the data of volumes.* |
| *2* | *addVolume* | *Method used to insert the data of new volume into database. Input is volume model. Return an integer represents sql query result.* |
| *3* | *getNewVolume* | *Methos used to get the newest volume in database. Return a Volume represents the data of volume.* |

#### ChapterDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllChapter* | *Method used to get all chapter and its information of a book has the input id in database. Return an array list <Chapter> represents the data of chapters.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

#### - addVolume

INSERT INTO [dbo].[Volume]\n"

+ " ([bookId]\n"

+ " ,[volume]\n"

+ " ,[volumeName]\n"

+ " ,[summary])\n"

+ " VALUES ( ?"

+ " , (SELECT COUNT([volume]) FROM [dbo].[Volume] WHERE [bookId] = ?)+1"

+ " , ?"

+ " , ? )"

- getNewVolume

SELECT TOP 1 [id]\n"

+ " ,[bookId]\n"

+ " ,[volume]\n"

+ " ,[volumeName]\n"

+ " ,[summary]\n"

+ " FROM [dbo].[Volume]"

+ " WHERE [bookId] = ? "

+ " ORDER BY [volume] DESC"

## 19.3. Edit Volume

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### EditVolumeController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |

#### VolumeDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllVolume* | *Method used to get all volume and their information of a book has the input id in database. Return an array list <Volume> represents the data of volumes.* |
| *2* | *editVolume* | *Method used to update the data of a Volume in database. Input is volume model. Return an integer represents sql query result.* |
| *3* | *getVolumeById* | *Method used to get the data of volume that has the input id and return the volume* |

#### ChapterDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllChapter* | *Method used to get all chapter and its information of a book has the input id in database. Return an array list <Chapter> represents the data of chapters.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

- editVolume

UPDATE [dbo].[Volume]\n"

+ " SET [volumeName] = ? "

+ " ,[summary] = ? "

+ " WHERE [id] = ? "

## 19.4. Delete Volume

### a. Class Diagram

Text

Description automatically generated with medium confidence

### b. Class Specifications

#### DeleteVolumeController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### VolumeDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *deleteVolumeAndItsChapter* | *Method used to delete the data of a volume and all of its chapter in database. Input is volume id.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

- deleteVolumeAndItsChapter

DELETE FROM [dbo].[Volume]\n"

+ " WHERE [id] = ? "

## 19.5 Add Chapter

### a. Class Diagram

Table

Description automatically generated with medium confidence

### b. Class Specifications

**AddChapterController**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

**BookDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |

**VolumeDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllVolume* | *Method used to get all volume and their information of a book has the input id in database. Return an array list <Volume> represents the data of volumes.* |
| *2* | *getVolumeById* | *Method used to get the data of volume that has the input id and return the volume* |

**ChapterDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllChapter* | *Method used to get all chapter and its information of a book has the input id in database. Return an array list <Chapter> represents the data of chapters.* |
| *2* | *addChapter* | *Method used to insert the data of new chapter into database. Input is chapter model. Return an integer represents sql query result.* |
| *3* | *getNewChapter* | *Methos used to get the newest chapter in database. Return a Chapter object represents the data of chapter.* |

**DBContext**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

- addChapter

INSERT INTO [dbo].[Chapter]\n"

+ " ([bookId]\n"

+ " ,[volumeId]\n"

+ " ,[chapter]\n"

+ " ,[chapterName]\n"

+ " ,[status]\n"

+ " ,[content])\n"

+ " VALUES\n"

+ " ( ? "

+ " , ? "

+ " , (SELECT COUNT([chapter]) FROM [dbo].[Chapter] WHERE [volumeId] = ?)+1 "

+ " , ? "

+ " , ? "

+ " , ? )"

- getNewChapter

SELECT TOP 1 [id]\n"

+ " ,[volumeId]\n"

+ " ,[chapter]\n"

+ " ,[chapterName]\n"

+ " ,[status]\n"

+ " ,[content]\n"

+ " FROM [dbo].[Chapter]"

+ "WHERE [bookId] = ? "

+ "ORDER BY [chapter] DESC "

## 19.6 Edit Chapter

### a. Class Diagram

Diagram

Description automatically generated with medium confidence

### b. Class Specifications

**EditChapterController**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

**BookDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |

**VolumeDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllVolume* | *Method used to get all volume and their information of a book has the input id in database. Return an array list <Volume> represents the data of volumes.* |
| *2* | *getVolumeById* | *Method used to get the data of volume that has the input id and return the volume* |

**ChapterDAO**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllChapter* | *Method used to get all chapter and its information of a book has the input id in database. Return an array list <Chapter> represents the data of chapters.* |
| *2* | *editChapter* | *Method used to update the data of a chapter in database. Input is chapter object. Return an integer represents sql query result.* |

**DBContext**

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

- editChapter

UPDATE [dbo].[Chapter]\n"

+ " SET [chapterName] = ? "

+ " ,[status] = ?"

+ " ,[content] = ? "

+ " WHERE [id] = ?

## 19.7 Delete Chapter

### a. Class Diagram

Graphical user interface

Description automatically generated with low confidence

### b. Class Specifications

#### DeleteChapterController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### ChapterDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| **1** | **deleteChapter** | *Method used to delete the data of a chapter in database. Input is chapter id.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

Diagram

Description automatically generated

### d. Data queries

- deleteChapter

DELETE FROM [dbo].[Chapter]\n"

+ " WHERE [id] = ?

## 20. View List of User Account

### a. Class Diagram

Diagram

Description automatically generated

### 

### b. Class Specifications

#### UserManagement

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### UserDAO

***Class Methods***

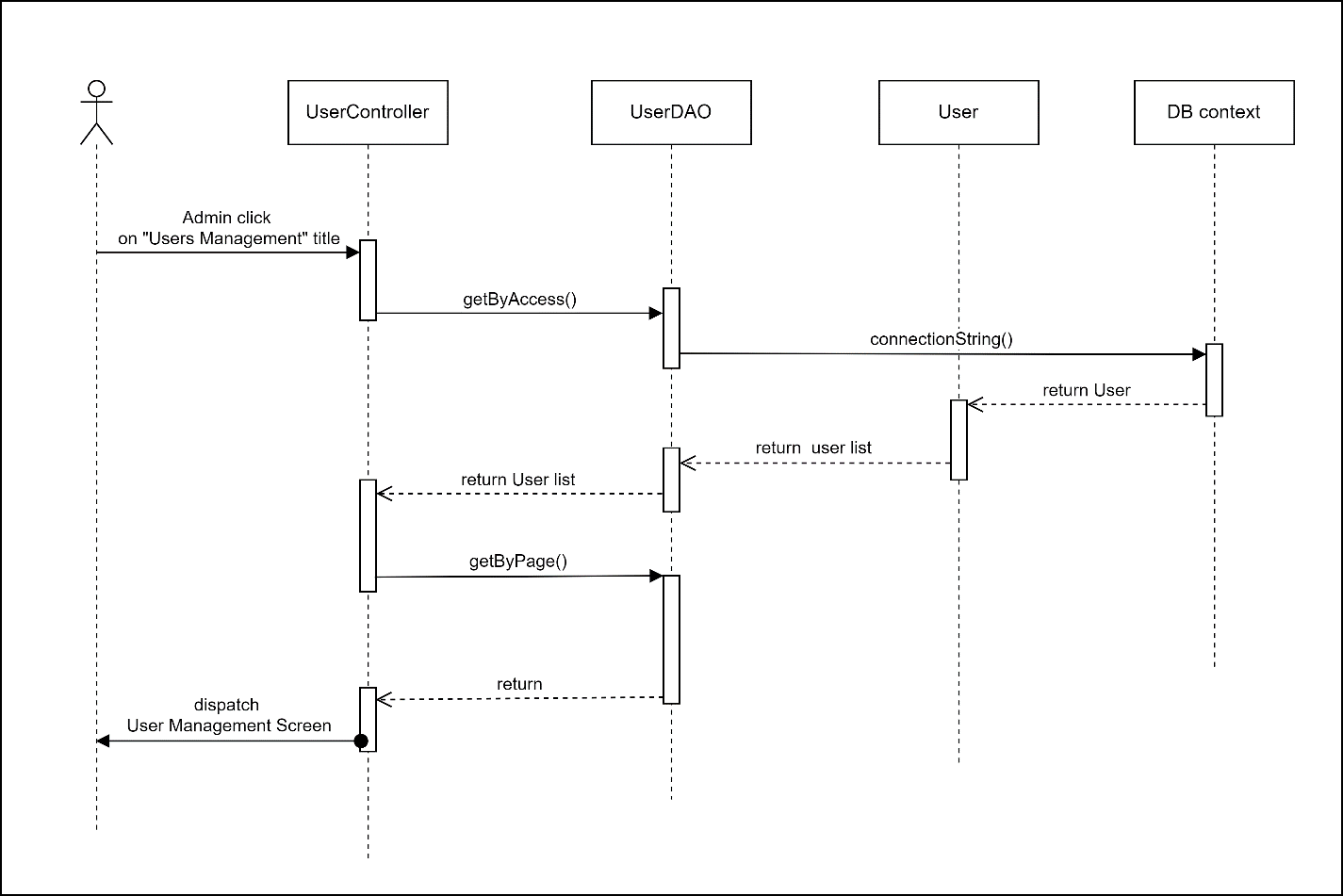
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getByAccess()* | *Method used to return the list of user accounts with lower level than admin account login .Inputs are admin level .Return a List of <Code>model.User</Code> represents the user’s data* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)



### 

### d. Database queries

Select \* from [User] where [is\_super] < ?

## 

## 21. View List of User Account

### a. Class Diagram

Diagram

Description automatically generated

Diagram

Description automatically generated

### 

### b. Class Specifications

#### UserManagement

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### LoadUser

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### DBContext

***Class Methods***

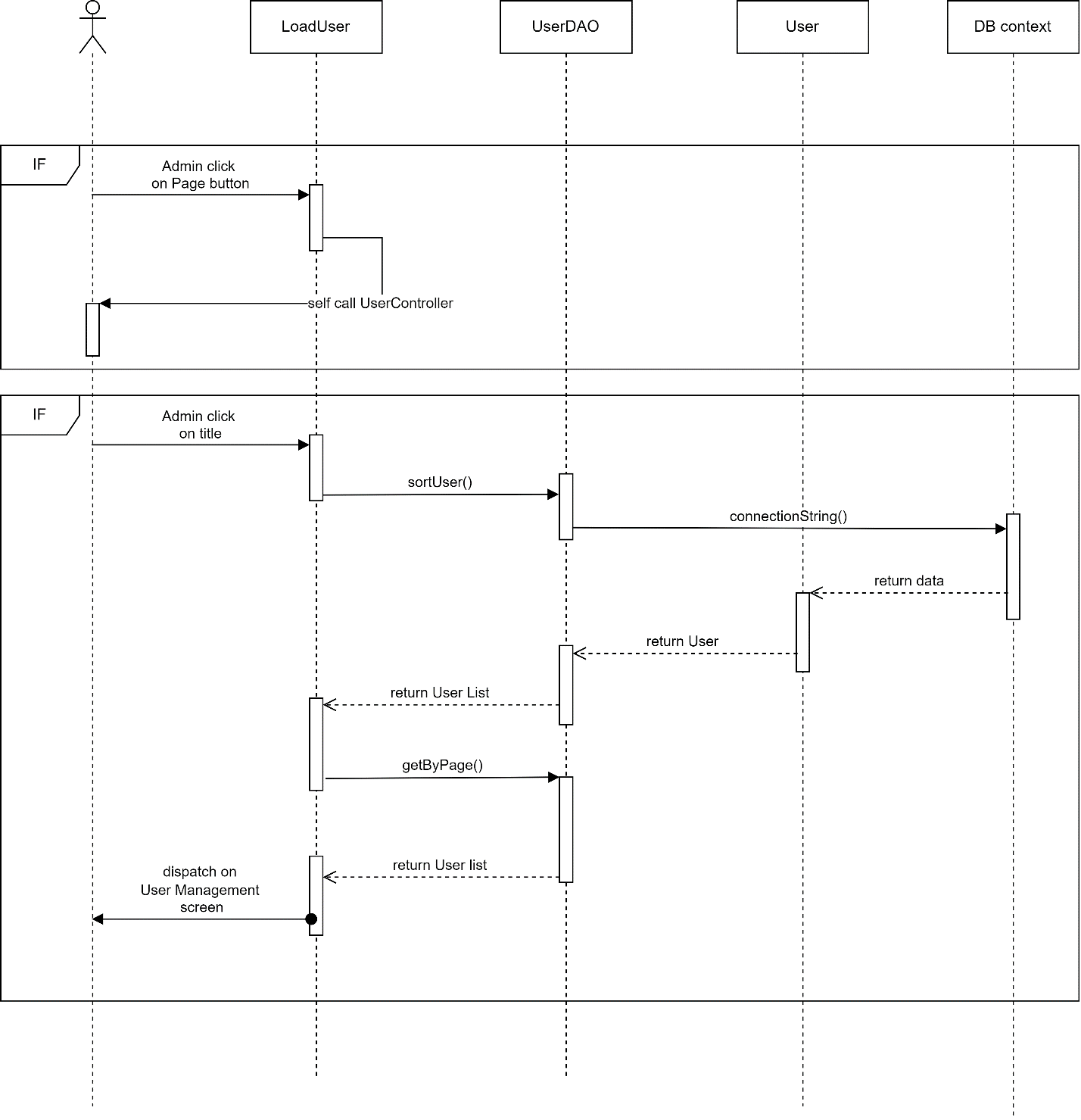
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

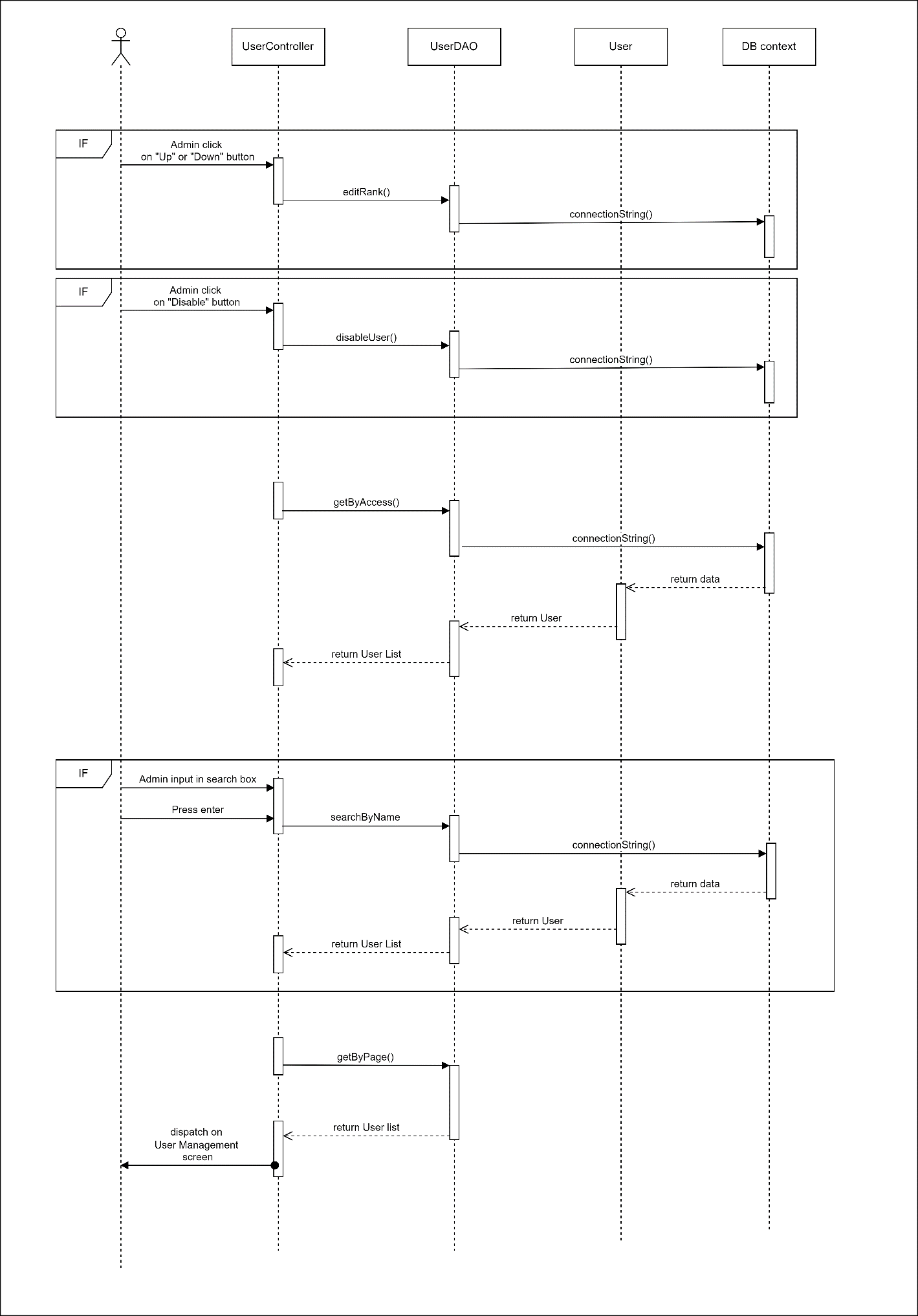
#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getByAccess ()* | *Method used to return the list of user accounts with lower level than admin account login .Inputs are admin level .Return a List of <Code>model.User</Code> represents the user’s data* |
| *2* | *editRank()* | *Method used to increase or decrease the level of an user account .*  *Input is the id of the user.* |
| *3* | *disableUser()* | *Method used to set the account level lowest . Input is the id of the user.* |
| *4* | *searchByName()* | *Method used to search accounts have the detail has the similarities with text input . Input are the admin level , text need to search . Return a List of <Code>model.User</Code> represents the user’s data* |
| *5* | *sortUser()* | *Method used to sort accounts depend on title . Input are the admin level , title type . Return a List of <Code>model.User</Code> represents the user’s data* |

### c. Sequence Diagram(s)





### 

### d. Database queries

getByAccess() : Select \* from [User] where [is\_super] < ?

disableUser() : update [User] set [is\_super] = 0 where [id] = ?

editRank() : update [User] set [is\_super] = ? where [id] = ?

searchByName(): select \* from [User] where [is\_super] < ?

and [fullname] like ?

or [username] like ?

or [email] like ?

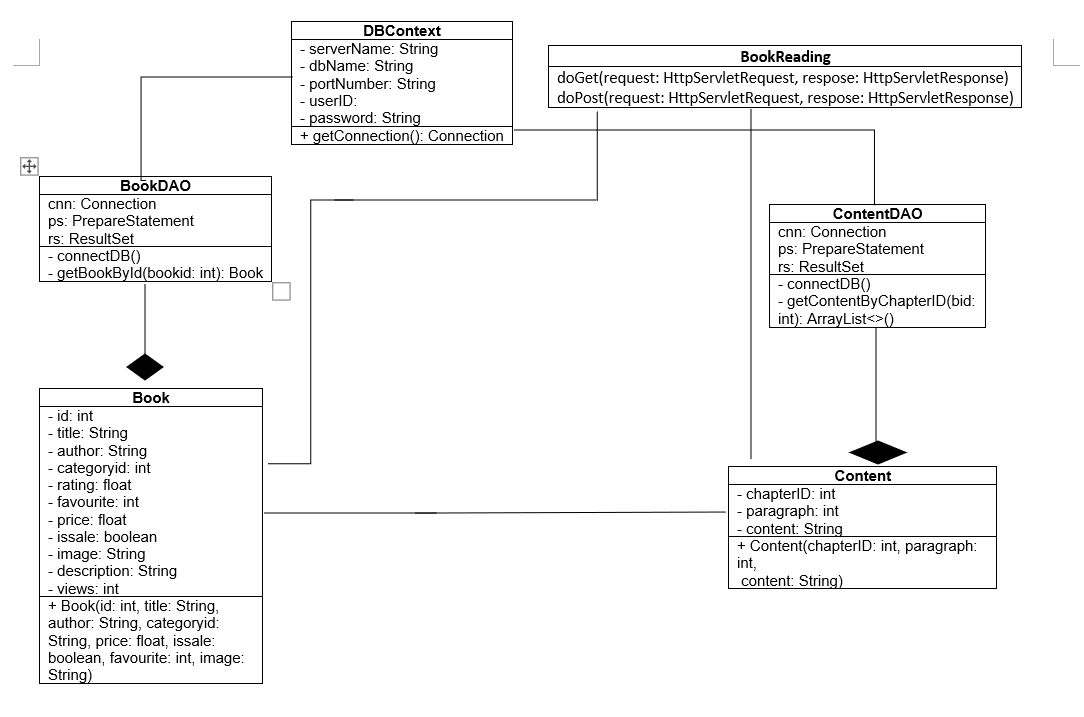
or [phone] like ?

or [address] like ?;

sortUser(): select \* from [User] where is\_super < ? order by ?

## 22. Reading interface

### a. Class Diagram



### b. Class Specifications

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Book(int id, String title, String author, float price, boolean issale, int favourite, String image) | Get general information of a books for display on homepage |

**- Book class**

**- DBContext class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

**- BookDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getBookByBookId(bookid: int) | Get general information of a book by its BookID |

**- ContentDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connecting with database |
| 02 | getContentByChapterID | Get information of Content by ChapterID |

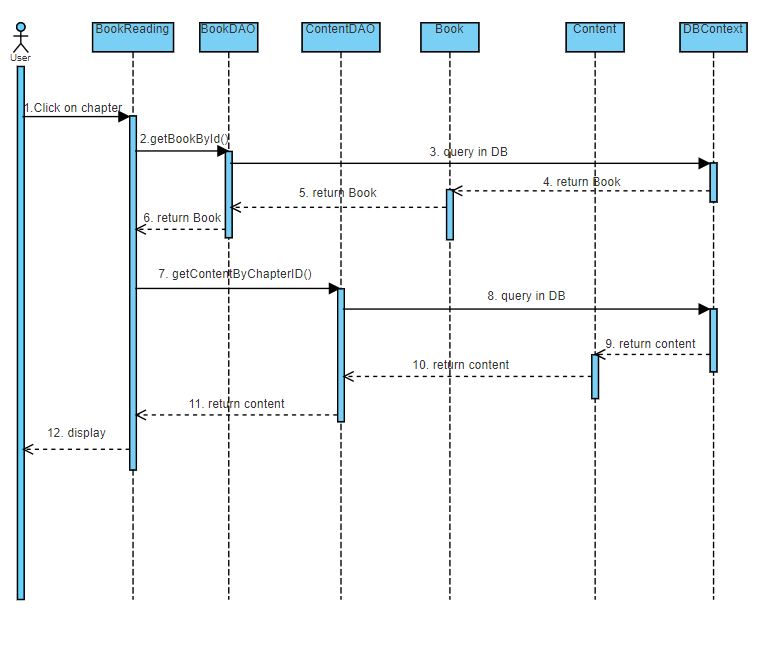
**- Content class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Content(chapterID: int, paragraph: int, content: String) |  |

**- BookReading class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | doGet() | Handles the HTTP <code>GET</code> method. |
| 02 | doPost() | Handles the HTTP <code>GET</code> method. |

### c. Sequence Diagram(s)



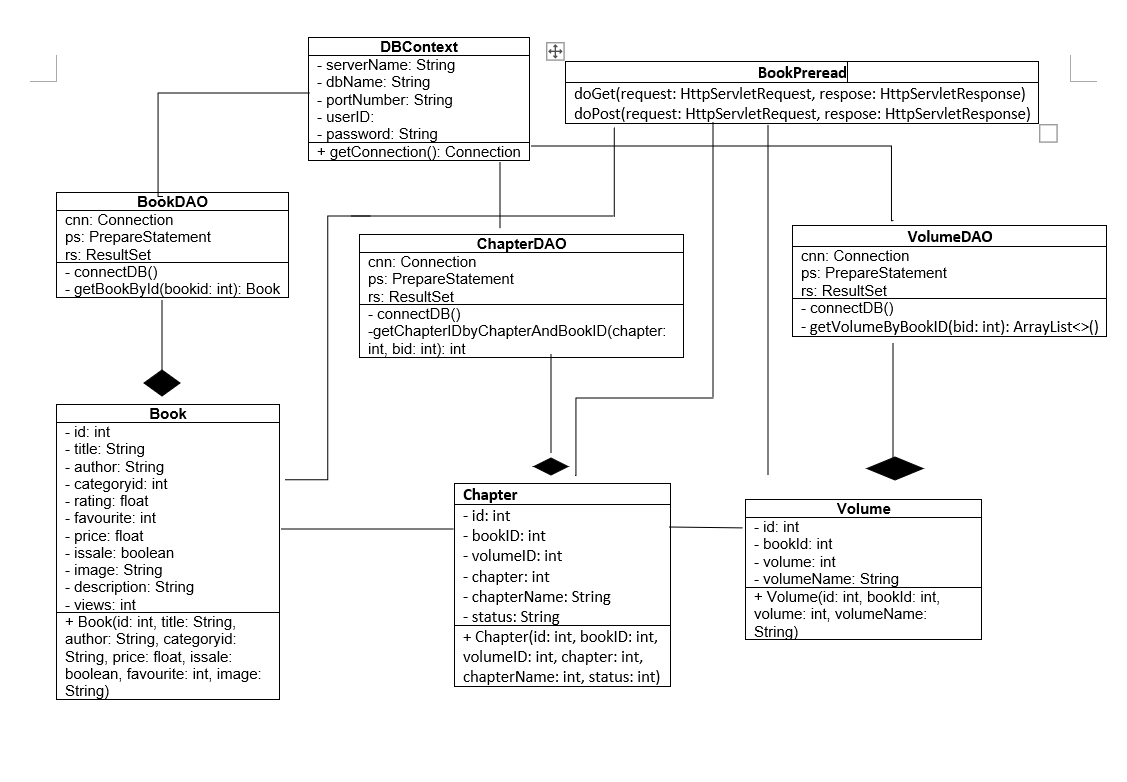
### d. Database queries

- getBookById(bookid: int): "select \* from [Book] where [id] = ?"

- getContentByChapterID(bid: int): "select \* from Content where chapterId = ?"

## 23. Pre-read

### a. Class Diagram



### b. Class Specifications

**- Book class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Book(int id, String title, String author, float price, boolean issale, int favourite, String image) | Get general information of book |

**- BookDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | Book(id: int, author: String, categoryid: String, price: float, issale: boolean, image: String ) | Get general information of a book |

**- DBContext class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

**- ChapterDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | getChapterIDbyChapterAndBookID | Get the first chapter of a book |

**- Chapter class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Chapter(id: int, bookID: int, volumeID: int, chapter: int, chapterName: int, status: int) | Get information of a chapter |

**- VolumeDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | getVolumeByBookID(bid: int) | Get all volumes of a book by its bookID |

**- VolumeDAO class**

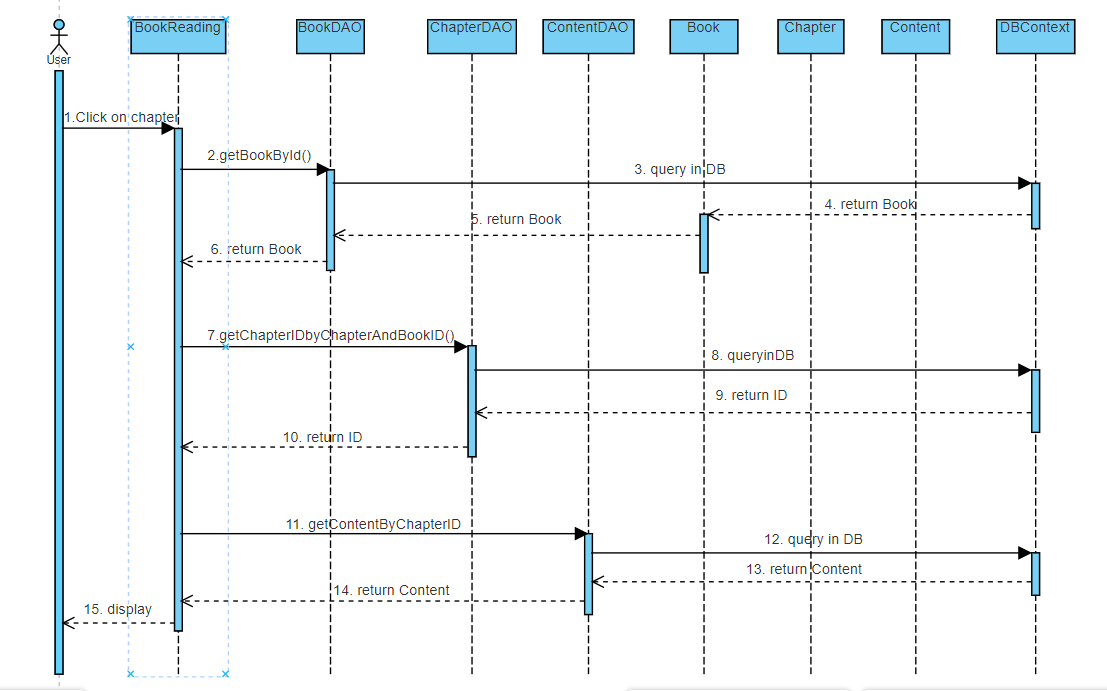
|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Volume(id: int, bookid: int, volume: int, volumeName: String) |  |

**- BookPreread class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | doGet() | Handles the HTTP <code>GET</code> method. |
| 02 | doPost() | Handles the HTTP <code>GET</code> method. |

### c. Sequence Diagram(s)

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



### d. Database queries

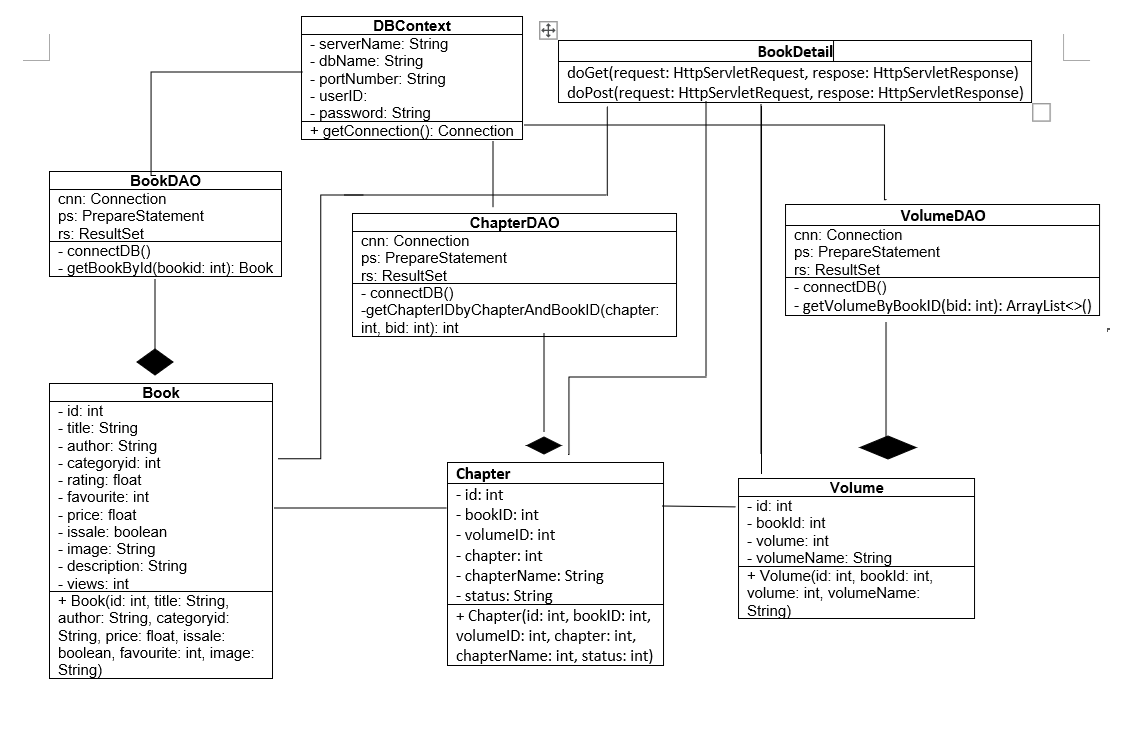
- getBookById(bookid: int): "select \* from [Book] where [id] = ?"

- getChapterIDByChapterAndBookID(chapter: int, bid: int): “select c.id from Chapter c where chapter = ? and bookId = ?”

- getContentByChapterID(bid: int): "select \* from Content where chapterId = ?"

## 24. Table of Contents

### a. Class Diagram



### b. Class Specifications

**- Book class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Book(int id, String title, String author, float price, boolean issale, int favourite, String image) | Get general information of book |

**- BookDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | Book(id: int, author: String, categoryid: String, price: float, issale: boolean, image: String ) | Get general information of a book |

**- DBContext class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

**- ChapterDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | getChapterIDbyChapterAndBookID | Get the first chapter of a book |

**- Chapter class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Chapter(id: int, bookID: int, volumeID: int, chapter: int, chapterName: int, status: int) | Get information of a chapter |

**- VolumeDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | connectDB() | Connect database |
| 02 | getVolumeByBookID(bid: int) | Get all volumes of a book by its bookID |

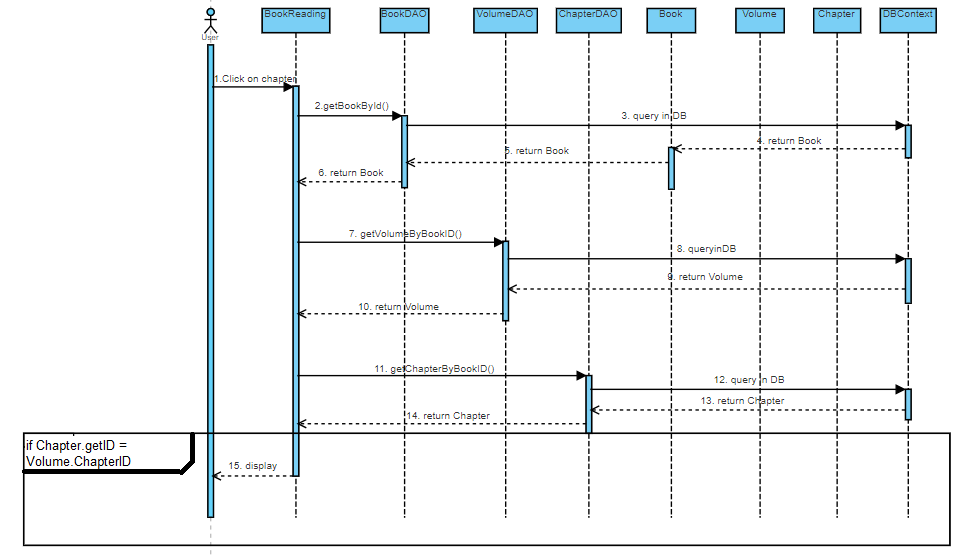
**- VolumeDAO class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | Volume(id: int, bookid: int, volume: int, volumeName: String) |  |

**- BookDetails class**

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | doGet() | Handles the HTTP <code>GET</code> method. |
| 02 | doPost() | Handles the HTTP <code>GET</code> method. |

### c. Sequence Diagram(s)



### d. Database queries

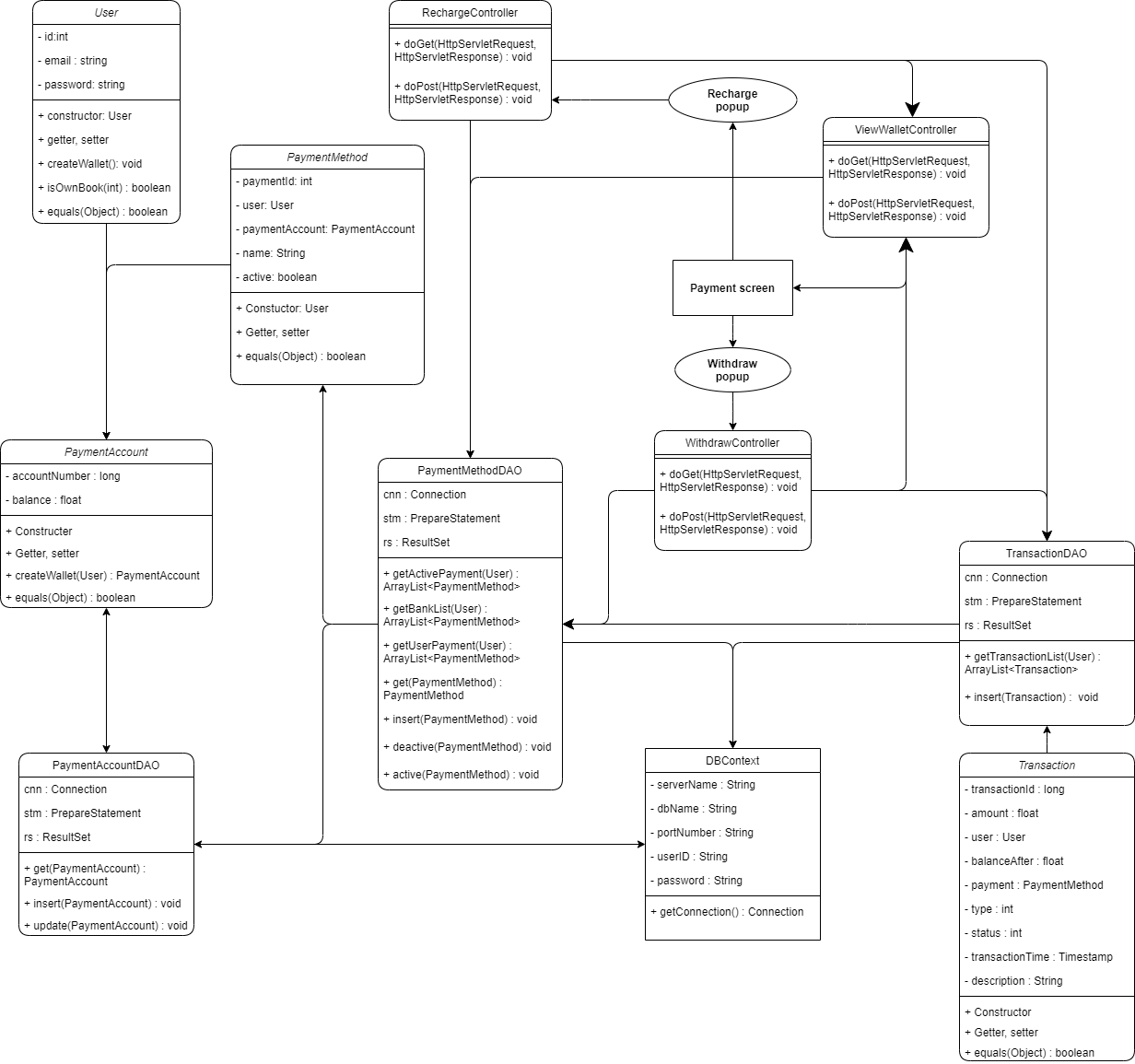
- getBookById(bookid: int): "select \* from [Book] where [id] = ?"

- getVolumeByBookID(bid: int): ”select \* from [Volume] where [bookId] = ?”

- getChapterByBookID(bid: int): "select \* from [Chapter] where [bookId] = ?"

## 25. Wallet

### a. Class Diagram



### b. Class Specifications

#### ViewWalletController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### RechargeController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### WithdrawController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentAccount | Get wallet of user |
| 02 | getId | Get ID of user |
| 03 | createWallet | Create new wallet for new signup user |
| 04 | getPassword | Get password of user |
| 05 | setPaymentAccount | Set payment account as wallet of user |

#### PaymentAccount class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | createWallet | Create new wallet of user |
| 02 | setAccountNumber | Set number as account number of payment account |
| 03 | getAccountNumber | Get account number of payment account |
| 04 | getBalance | Get balance of payment account |
| 05 | setBalance | Set number as balance of payment account |

#### PaymentMethod class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentId | Get ID of payment method |
| 02 | setPaymentId | Set ID of payment method |
| 03 | isActive | Check is payment method is active |
| 04 | setActive | Set status of payment method is active or not |
| 05 | getName | Get name of payment method |
| 06 | setName | Set name for payment method |
| 07 | setUser | Set user who own this payment method |
| 08 | getUser | Get user who own this payment pethod |
| 09 | getPaymentAccount | Get payment account which linked in this payment method |
| 10 | setPaymentAccount | Set payment account which linked in this payment method |

#### Transaction class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionId | Get ID of transaction |
| 02 | setTransactionId | Set ID for transaction |
| 03 | getAmount | Get amount of transaction |
| 04 | setAmount | Set amount for transaction |
| 05 | getPayment | Get payment method of transaction |
| 06 | setPayment | Set payment method for transaction |
| 07 | getBalanceAfter | Get balance in wallet after transaction finish |
| 08 | setBalanceAfter | Set balance in wallet after transaction finish |
| 09 | getUser | Get user who have this transaction |
| 10 | setUser | Set user as who have this transaction |
| 11 | getType | Get type of transaction |
| 12 | setType | Set type of transaction |
| 13 | getStatus | Get status of transaction |
| 14 | setStatus | Set status of transaction |
| 15 | getTransactionTime | Get time created transaction |
| 16 | setTransactionTime | Set time as created time of transaction |
| 17 | getDescription | Get description of transaction |
| 18 | setDescription | Set description of transaction |

#### TransactionDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionList | Get all transaction of user from database |
| 02 | insert | Add new transaction into database tabe of transaction |

#### PaymentMethodDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deactive | Set payment method as not active in database |
| 02 | get | Get payment method from database base on paymentId or user and paymentAccount |
| 03 | active | Set payment method as active in database |
| 04 | getActivePayment | Get list of all active payment methods of user |
| 05 | getBankList | Get list of all payment methods of user except user’s wallet |
| 06 | insert | Add new payment method into database |

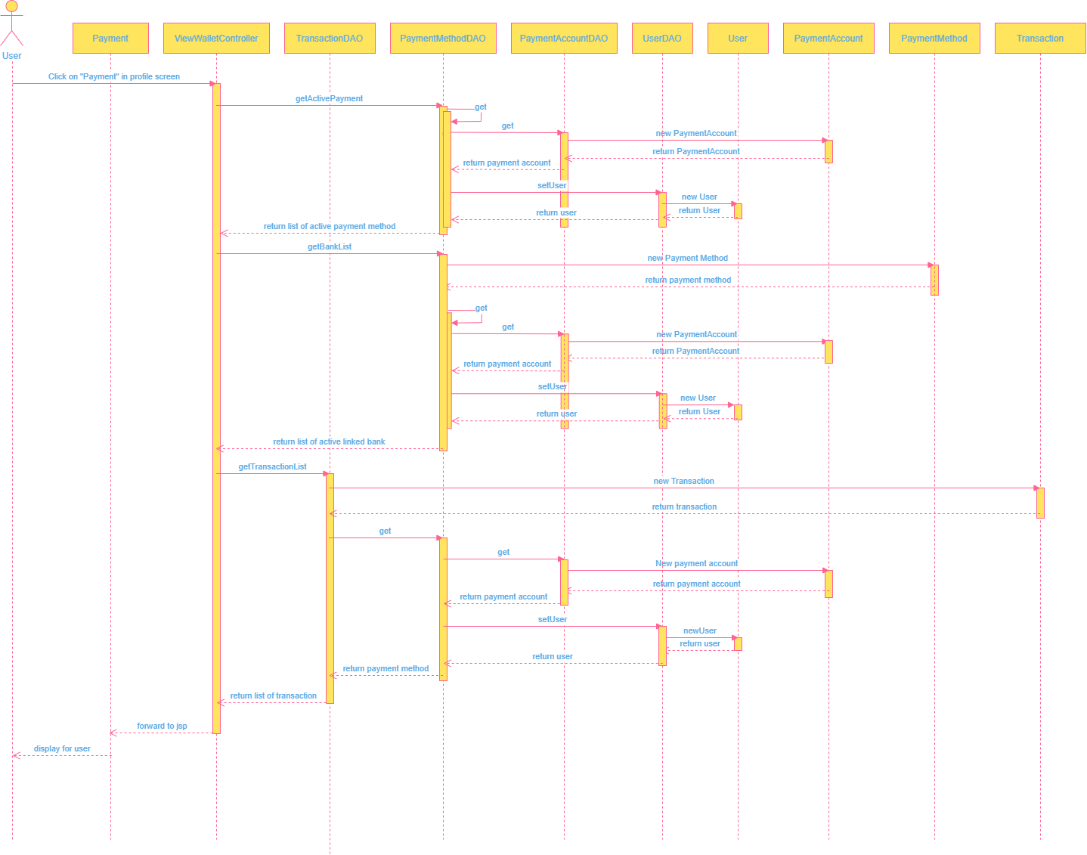
#### UserDAO class

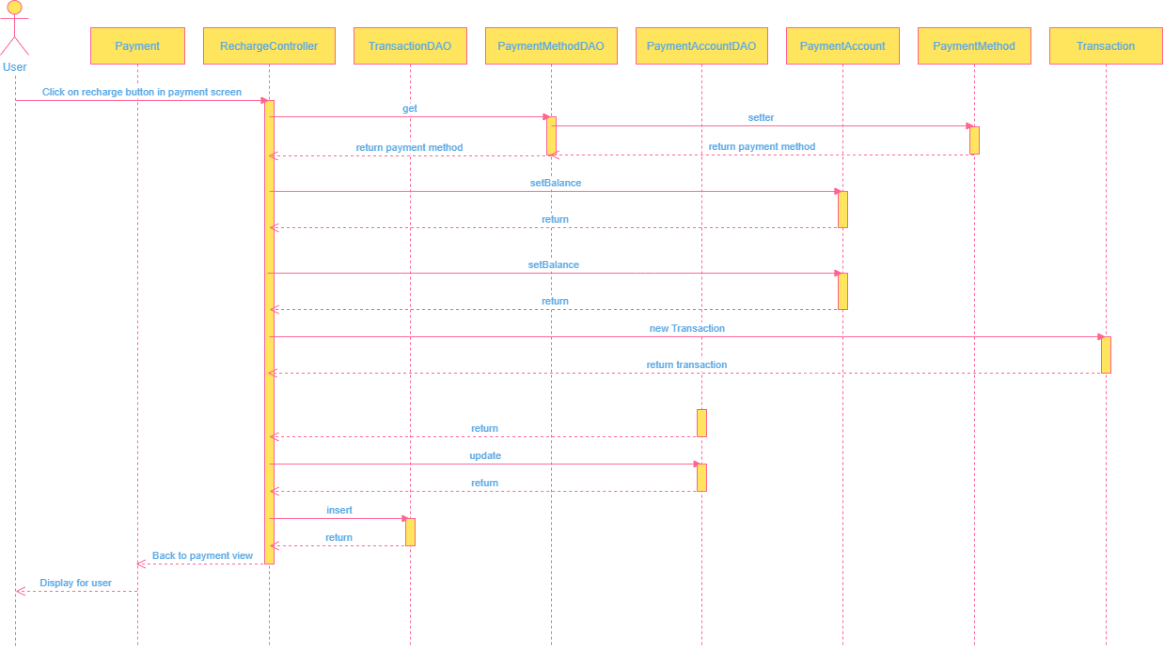
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | setWalletNumber | Set account number of payment account as wallet number of user in database |

#### PaymentAccountDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | get | Get payment account from database base on account number |
| 02 | getWallet | Get payment account which is input user’s wallet |
| 03 | insert | Add new payment account into database |
| 04 | update | Update balance of payment account in database |

### c. Sequence Diagram(s)





### d. Database queries

UPDATE [Payment\_Account]

SET [balance] = ?

WHERE [accountNumber] = ?

INSERT INTO [Transaction

([userId]

,[amount]

,[balanceAfter]

,[type]

,[status]

,[description]

,[paymentId]

,[transactionTime])

VALUES

(? ,? ,?,?,?,?,?,?)

select pa.accountNumber, balance, paymentId, userId, pm.[name]

from Payment\_Account pa

inner join Payment\_Method pm

on pa.accountNumber = pm.accountNumber

where pa.accountNumber not in

(select walletNumber from [User]

inner join Payment\_Account

on [User].walletNumber = Payment\_Account.accountNumber

) and userId = ?

SELECT paymentId, userId, pm.accountNumber, [name], [balance], [active]

FROM Payment\_Method pm inner join Payment\_Account pa

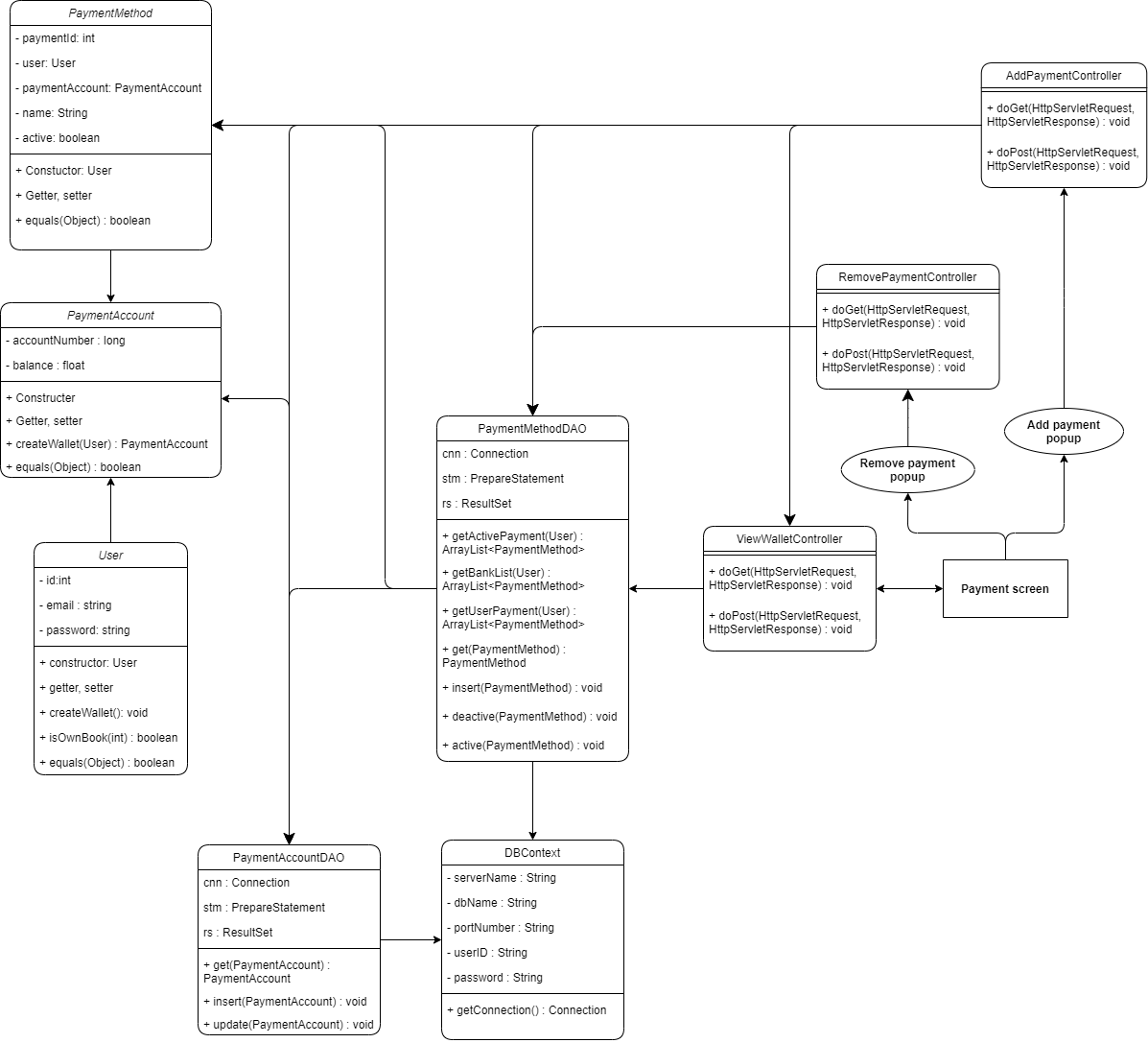
ON pm.accountNumber = pa.accountNumber

WHERE userId = ?

AND [active] = 1

## 26. Payment

### a. Class Diagram



#### b, Class Specification

#### ViewWalletController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### AddPaymentController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### RemovePaymentController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentAccount | Get wallet of user |
| 02 | getId | Get ID of user |
| 03 | createWallet | Create new wallet for new signup user |
| 04 | getPassword | Get password of user |
| 05 | setPaymentAccount | Set payment account as wallet of user |

#### PaymentAccount class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | createWallet | Create new wallet of user |
| 02 | setAccountNumber | Set number as account number of payment account |
| 03 | getAccountNumber | Get account number of payment account |
| 04 | getBalance | Get balance of payment account |
| 05 | setBalance | Set number as balance of payment account |

#### PaymentMethod class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentId | Get ID of payment method |
| 02 | setPaymentId | Set ID of payment method |
| 03 | isActive | Check is payment method is active |
| 04 | setActive | Set status of payment method is active or not |
| 05 | getName | Get name of payment method |
| 06 | setName | Set name for payment method |
| 07 | setUser | Set user who own this payment method |
| 08 | getUser | Get user who own this payment pethod |
| 09 | getPaymentAccount | Get payment account which linked in this payment method |
| 10 | setPaymentAccount | Set payment account which linked in this payment method |

#### Context.TransactionDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionList | Get all transaction of user from database |
| 02 | insert | Add new transaction into database table of transaction |

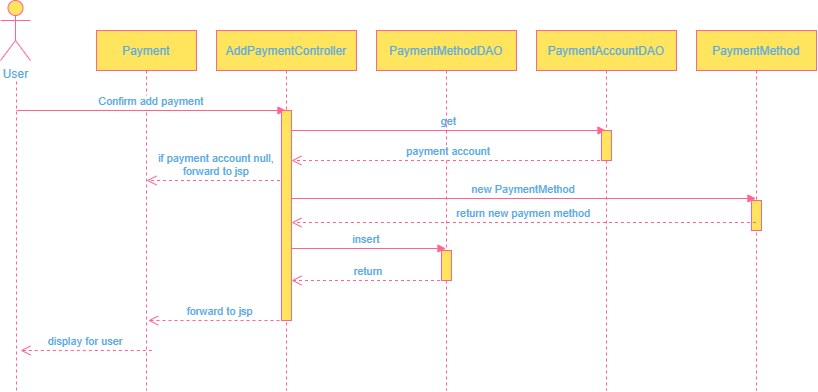
#### Transaction class

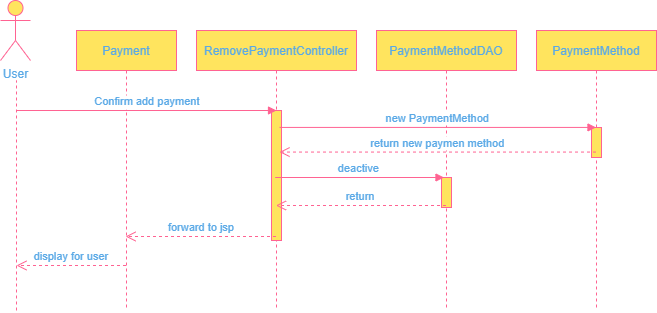
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionId | Get ID of transaction |
| 02 | setTransactionId | Set ID for transaction |
| 03 | getAmount | Get amount of transaction |
| 04 | setAmount | Set amount for transaction |
| 05 | getPayment | Get payment method of transaction |
| 06 | setPayment | Set payment method for transaction |
| 07 | getBalanceAfter | Get balance in wallet after transaction finish |
| 08 | setBalanceAfter | Set balance in wallet after transaction finish |
| 09 | getUser | Get user who have this transaction |
| 10 | setUser | Set user as who have this transaction |
| 11 | getType | Get type of transaction |
| 12 | setType | Set type of transaction |
| 13 | getStatus | Get status of transaction |
| 14 | setStatus | Set status of transaction |
| 15 | getTransactionTime | Get time created transaction |
| 16 | setTransactionTime | Set time as created time of transaction |
| 17 | getDescription | Get description of transaction |
| 18 | setDescription | Set description of transaction |

#### PaymentMethodDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deactive | Set payment method as not active in database |
| 02 | get | Get payment method from database base on paymentId or user and paymentAccount |
| 03 | active | Set payment method as active in database |
| 04 | getActivePayment | Get list of all active payment methods of user |
| 05 | getBankList | Get list of all payment methods of user except user’s wallet |
| 06 | insert | Add new payment method into database |

### c. Sequence Diagram(s)





### d. Database queries

INSERT INTO [Payment\_Method]

([userId]

,[accountNumber]

,[name]

,[active])

VALUES (?, ?, ?, ?)

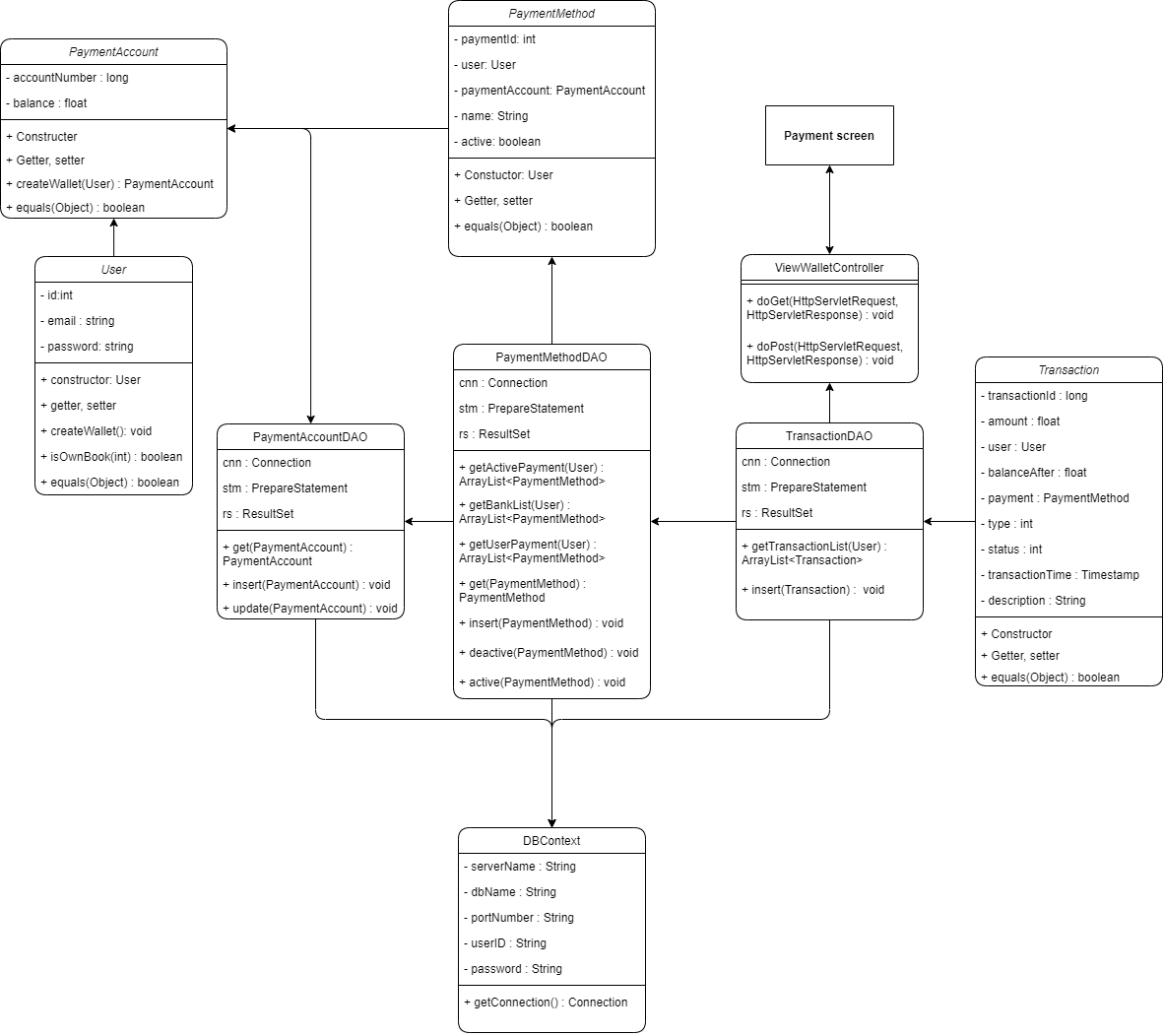
UPDATE [dbo].[Payment\_Method]

SET [active] = 0

WHERE paymentId = ?

## 27. View purchase history

### a. Class Diagram



### b. Class Specifications

#### ViewWalletController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### PaymentAccount class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | createWallet | Create new wallet of user |
| 02 | setAccountNumber | Set number as account number of payment account |
| 03 | getAccountNumber | Get account number of payment account |
| 04 | getBalance | Get balance of payment account |
| 05 | setBalance | Set number as balance of payment account |

#### PaymentMethod class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentId | Get ID of payment method |
| 02 | setPaymentId | Set ID of payment method |
| 03 | isActive | Check is payment method is active |
| 04 | setActive | Set status of payment method is active or not |
| 05 | getName | Get name of payment method |
| 06 | setName | Set name for payment method |
| 07 | setUser | Set user who own this payment method |
| 08 | getUser | Get user who own this payment pethod |
| 09 | getPaymentAccount | Get payment account which linked in this payment method |
| 10 | setPaymentAccount | Set payment account which linked in this payment method |

#### Transaction class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionId | Get ID of transaction |
| 02 | setTransactionId | Set ID for transaction |
| 03 | getAmount | Get amount of transaction |
| 04 | setAmount | Set amount for transaction |
| 05 | getPayment | Get payment method of transaction |
| 06 | setPayment | Set payment method for transaction |
| 07 | getBalanceAfter | Get balance in wallet after transaction finish |
| 08 | setBalanceAfter | Set balance in wallet after transaction finish |
| 09 | getUser | Get user who have this transaction |
| 10 | setUser | Set user as who have this transaction |
| 11 | getType | Get type of transaction |
| 12 | setType | Set type of transaction |
| 13 | getStatus | Get status of transaction |
| 14 | setStatus | Set status of transaction |
| 15 | getTransactionTime | Get time created transaction |
| 16 | setTransactionTime | Set time as created time of transaction |
| 17 | getDescription | Get description of transaction |
| 18 | setDescription | Set description of transaction |

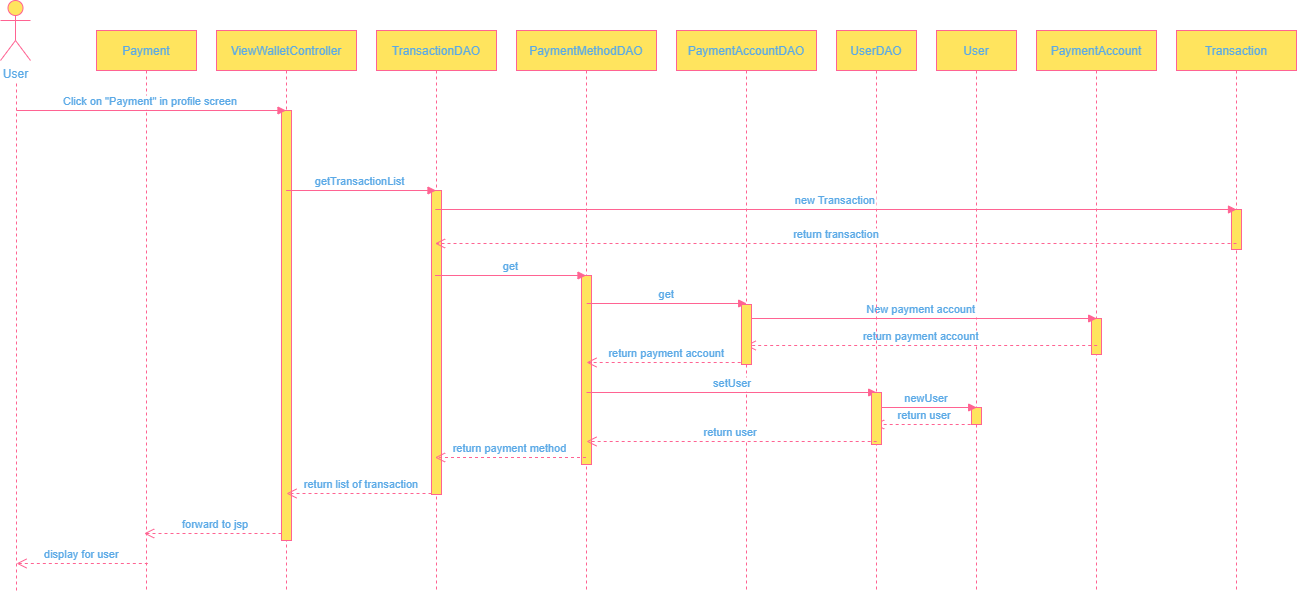
#### TransactionDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionList | Get all transaction of user from database |
| 02 | insert | Add new transaction into database tabe of transaction |

#### Context.PaymentMethodDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deactive | Set payment method as not active in database |
| 02 | get | Get payment method from database base on paymentId or user and paymentAccount |
| 03 | active | Set payment method as active in database |
| 04 | getActivePayment | Get list of all active payment methods of user |
| 05 | getBankList | Get list of all payment methods of user except user’s wallet |
| 06 | insert | Add new payment method into database |

### c. Sequence Diagram(s)



### d. Database queries

SELECT [transactionId

,[userId]

,[amount]

,[balanceAfter]

,[transactionTime]

,[type]

,[status]

,[description]

,[paymentId]

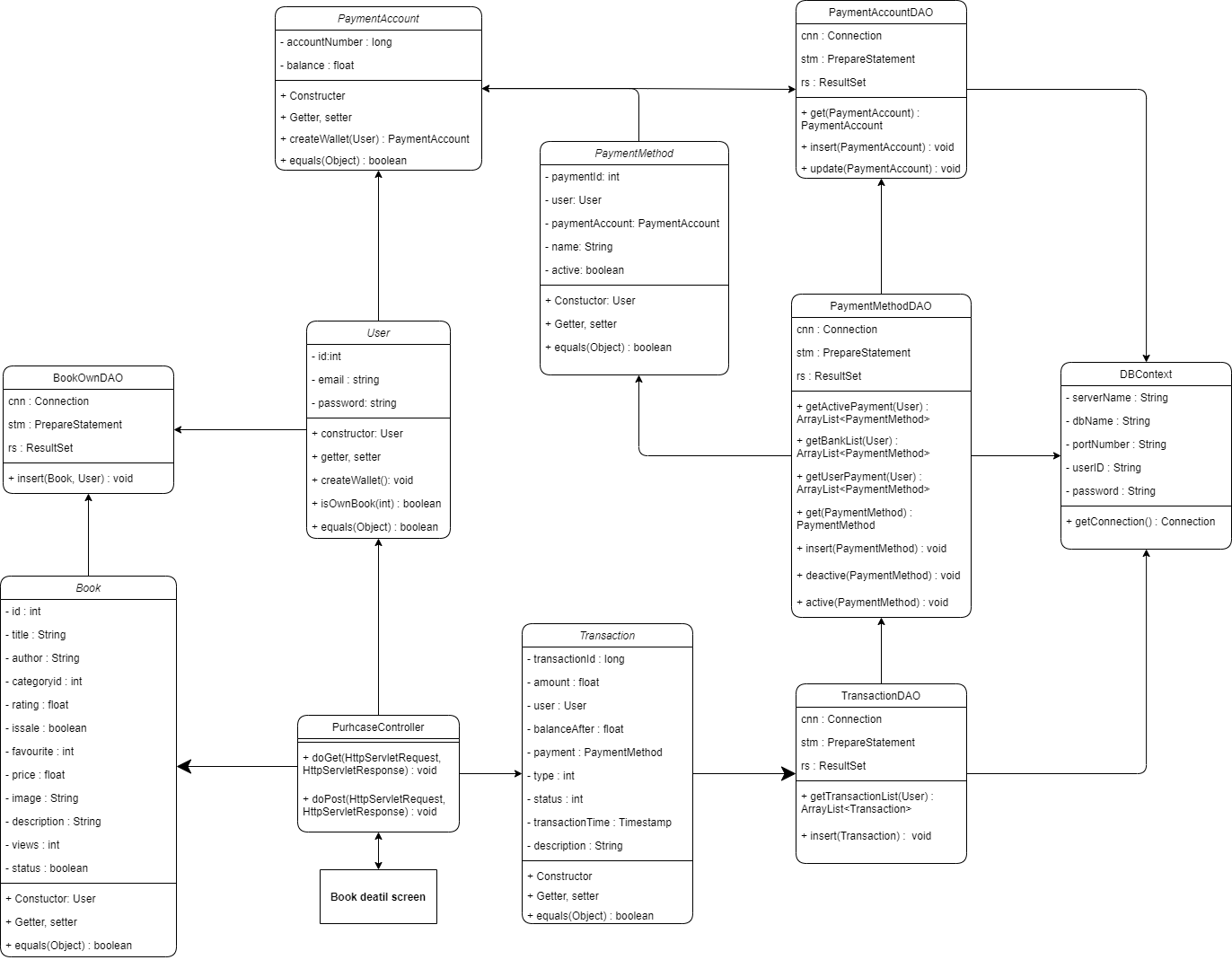
FROM [Transaction]

WHERE userId = ?

ORDER BY transactionTime desc

## 28. Purchase a book

### Class Diagram



### b. Class Specifications

#### PaymentAccount class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | createWallet | Create new wallet of user |
| 02 | setAccountNumber | Set number as account number of payment account |
| 03 | getAccountNumber | Get account number of payment account |
| 04 | getBalance | Get balance of payment account |
| 05 | setBalance | Set number as balance of payment account |

#### PaymentMethod class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getPaymentId | Get ID of payment method |
| 02 | setPaymentId | Set ID of payment method |
| 03 | isActive | Check is payment method is active |
| 04 | setActive | Set status of payment method is active or not |
| 05 | getName | Get name of payment method |
| 06 | setName | Set name for payment method |
| 07 | setUser | Set user who own this payment method |
| 08 | getUser | Get user who own this payment pethod |
| 09 | getPaymentAccount | Get payment account which linked in this payment method |
| 10 | setPaymentAccount | Set payment account which linked in this payment method |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getId | Get ID of user |
| 02 | getPaymentAccount | Get wallet of user |
| 03 | setPaymentAccount | Set payment account as wallet of user |

#### Book class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getId | Get ID of book |
| 02 | getTitle | Get title of book |
| 03 | getPrice | Get price of book |

#### Transaction class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionId | Get ID of transaction |
| 02 | setTransactionId | Set ID for transaction |
| 03 | getAmount | Get amount of transaction |
| 04 | setAmount | Set amount for transaction |
| 05 | getPayment | Get payment method of transaction |
| 06 | setPayment | Set payment method for transaction |
| 07 | getBalanceAfter | Get balance in wallet after transaction finish |
| 08 | setBalanceAfter | Set balance in wallet after transaction finish |
| 09 | getUser | Get user who have this transaction |
| 10 | setUser | Set user as who have this transaction |
| 11 | getType | Get type of transaction |
| 12 | setType | Set type of transaction |
| 13 | getStatus | Get status of transaction |
| 14 | setStatus | Set status of transaction |
| 15 | getTransactionTime | Get time created transaction |
| 16 | setTransactionTime | Set time as created time of transaction |
| 17 | getDescription | Get description of transaction |
| 18 | setDescription | Set description of transaction |

#### BookDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | isOwn | Check if user own this book |
| 02 | getOwnBooks | Get all books that user own |
| 03 | getBookById | Get a book by it’s ID |

#### BookOwnDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | insert | Add new own status of an user with a book |

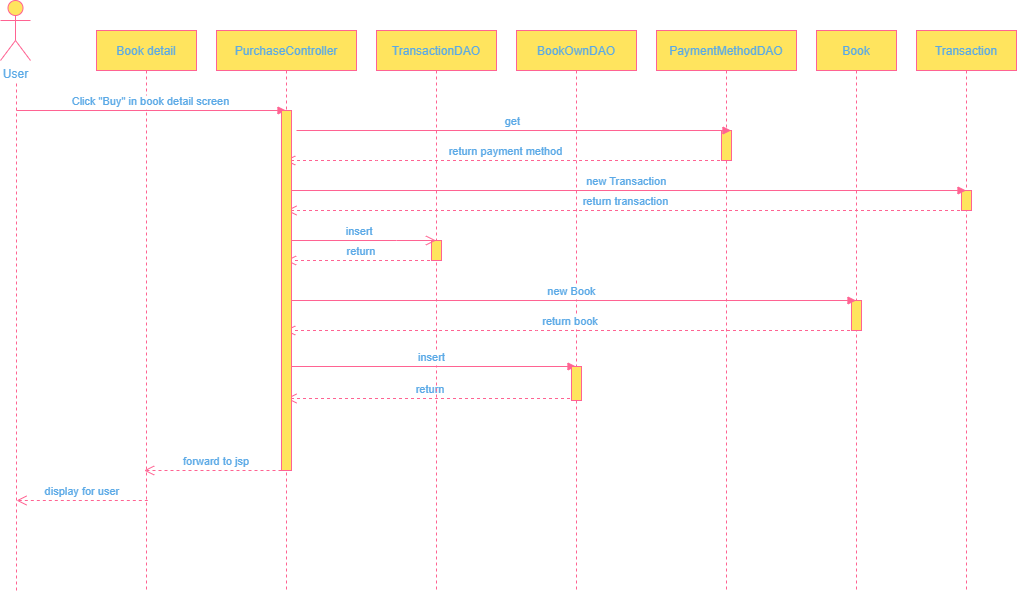
#### TransactionDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getTransactionList | Get all transaction of user from database |
| 02 | insert | Add new transaction into database tabe of transaction |

#### ViewWalletController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

### c. Sequence Diagram(s)



### d. Database queries

select bo.bookId from [User] u

inner join [Book\_Own] bo

on bo.userId = u.id

inner join Book b

on bo.bookId = b.id

where u.id = ?

INSERT INTO [Book\_Own]

([userId]

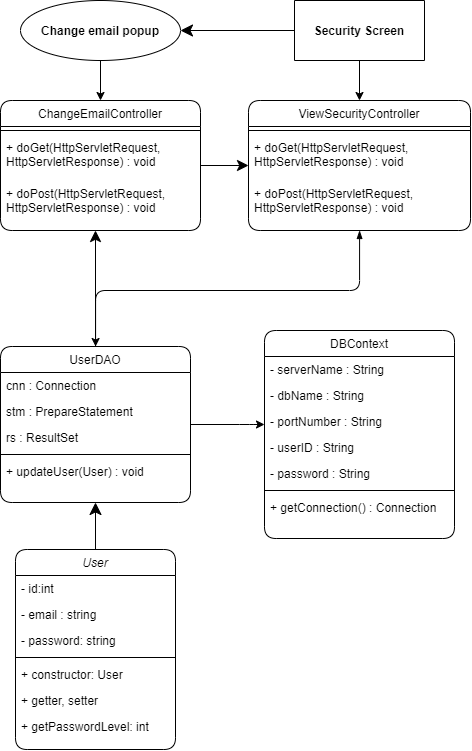
,[bookId])

VALUES

(? ,?)

## 29. Change email

### a. Class Diagram



### b. Class Specifications

#### ProfileController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### User class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getId | Get ID of user |
| 02 | getName | Get name of user |
| 03 | getGender | Get gender of user |
| 04 | getPhone | Get phone number of user |
| 05 | getAddress | Get address of user |
| 06 | getEmail | Get email of user |
| 07 | setEmail | Set name of user |

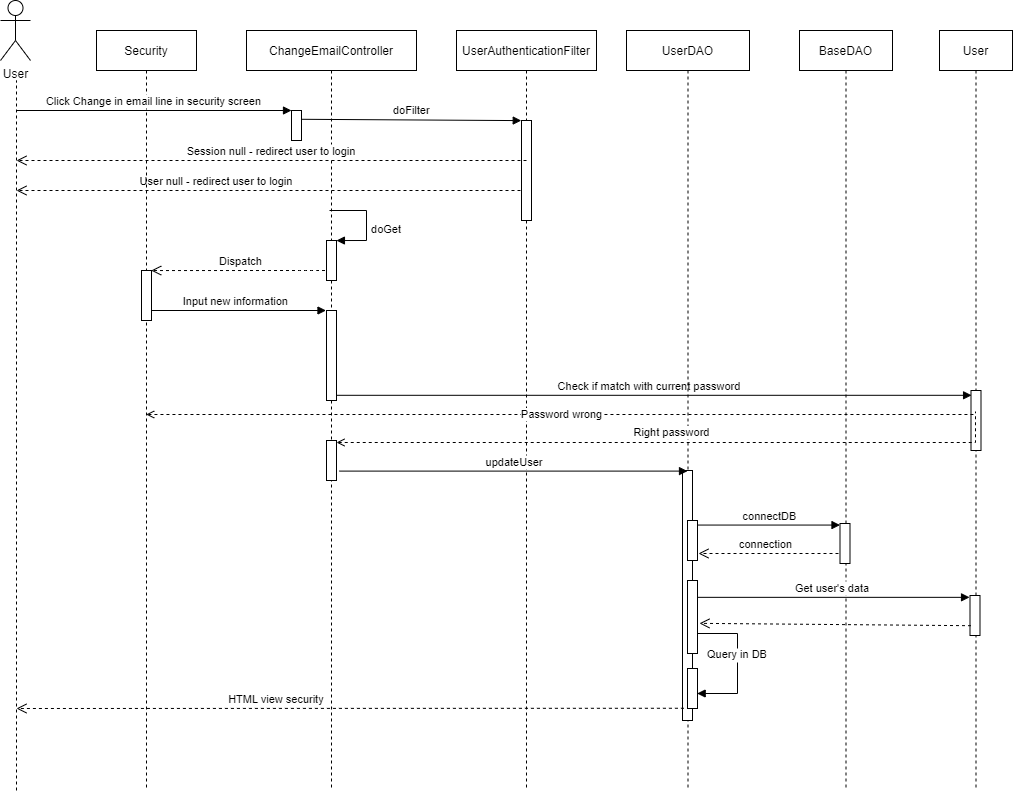
#### UserDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | updateProfile | Update information of user in database |

#### Validator class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getString | Check if string is empty |
| 02 | getName | Check if name is empty or contains digit or so long |

### c. Sequence Diagram(s)



### d. Database queries

update [User] set

[fullname] = ?,[phone] = ?,[address] = ?, [gender] = ?, [dob] = ? ,[email] = ?

where [id] = ?

## 30. Search Engine

**a. Class Diagram**

**Diagram

Description automatically generated**

**b. Class Specifications**

#### BookController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### BookDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getByPage(books, start, end) | Get books in each page |
| 02 | getBooks(search,author,iCate) | Get all the books that fit the requirements |

#### DBContext

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection() | Get access to the database |

#### CategoryDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getAllCategory() | Get all the category of the books |

**c. Sequence Diagram**

**Diagram, calendar

Description automatically generated**

**d. Database queries**

* String sql = "SELECT [Book].[id] as [bookId]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name] as [authorName]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [Book]"

+ " INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]"

+ " INNER JOIN [CategoryBook] cb ON [Book].[id] = cb.[bookId]"

+ " WHERE [Book].[title] LIKE ? "

+ " AND [Author].[name] LIKE ? "

+ " AND [status] != 0 "

+ " AND [Author].[userId] IS NULL";

int idCateLength = idCategory.length;

if (idCategory[0] != -1) {

sql += "AND cb.[categoryId] in (? ";

for (int i = 1; i < idCateLength; i++) {

sql += ",? ";

}

sql += ") GROUP BY [Book].[id]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ "having count(distinct cb.[categoryId]) = ? ";

} else {

sql += " GROUP BY [Book].[id]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n";

}

* "SELECT [Book].[id]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [Book]"

+ " INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]"

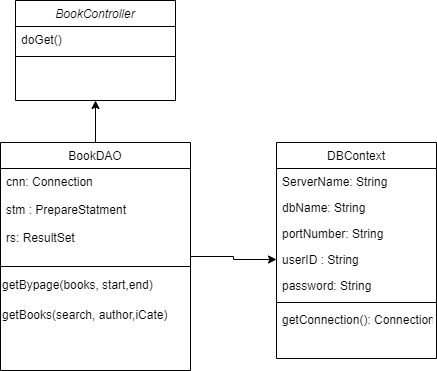
+ " INNER JOIN [CategoryBook] cb ON [Book].[id] = cb.[bookId] "

+ " WHERE cb.[categoryId] = ?"

+ " AND [status] != 0 "

## 31. Paging

**a. Class Diagram**

****

**b. Class Specifications**

#### BookController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

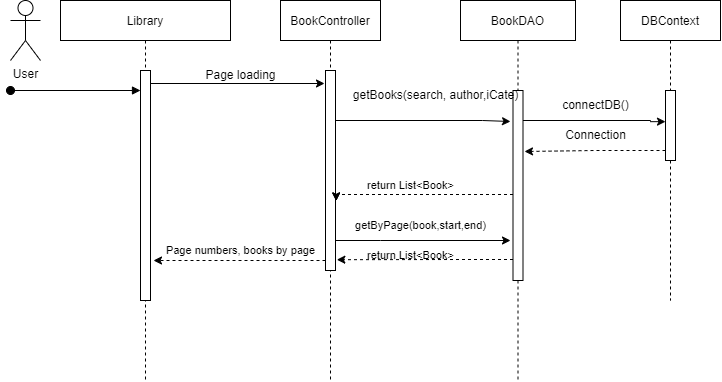
#### BookDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getByPage(books, start, end) | Get books in each page |
| 02 | getBooks(search,author,iCate) | Get all the books that fit the requirements |

#### DBContext

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection() | Get access to the database |

**c. Sequence Diagram**

****

**d. Database queries**

* "SELECT [Book].[id] as [bookId]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name] as [authorName]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [Book]"

+ " INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]"

+ " INNER JOIN [CategoryBook] cb ON [Book].[id] = cb.[bookId]"

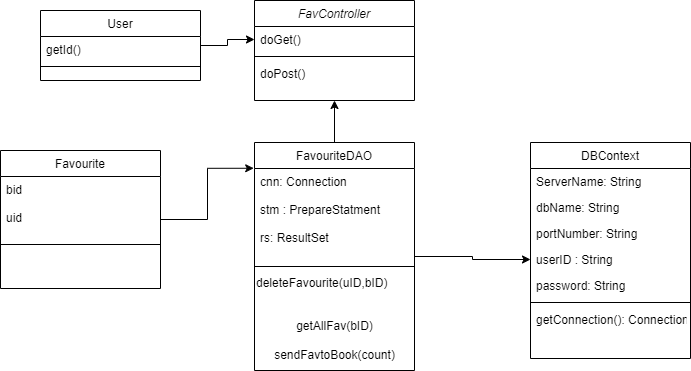
+ " WHERE [Book].[title] LIKE ? "

+ " AND [Author].[name] LIKE ? "

+ " AND [status] != 0 "

## 32. Delete From Favourite

**a. Class Diagram**

****

**b. Class Specifications**

#### FavController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

#### User

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getId() | Get User ID |

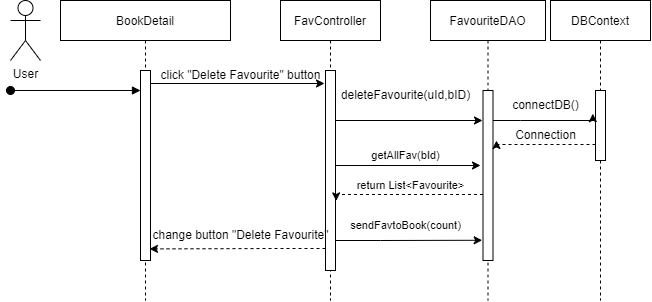
#### FavouriteDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | deleteFavourite() | Delete the specific book from the Favourites of the user |
| 02 | getAllFav() | Get all the user that add this book as favourite |
| 03 | sendFavtoBook() | Store the number of users that add this book as favourite to the database |

#### DBContext

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getConnection() | Get access to the database |

**c. Sequence Diagram**

****

**d. Database queries**

- "DELETE FROM [dbo].[Favourite]\n"

+ " WHERE uid='" + uID + "' and bid='" + bID + "'"

* “SELECT \* FROM [Favourite] WHERE bid='" + bID + "'"
* "INSERT INTO [dbo].[Book]\n"
* + " (\n"
* + " [favourite])\n"
* + " VALUES\n"
* + " ('" + count + "'"
* + " )"

## 33.1 View my own novels

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### BooksController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getUserNovels* | *Method used to get all novels and its information of specific user in database. Return an array list <Book> represents the data of books.* |
| *2* | *getByPage* | *Method used to divide list of books to many page. Return an array list <Book> represents the data of books.* |

#### CategoryDAO

***Class Methods***

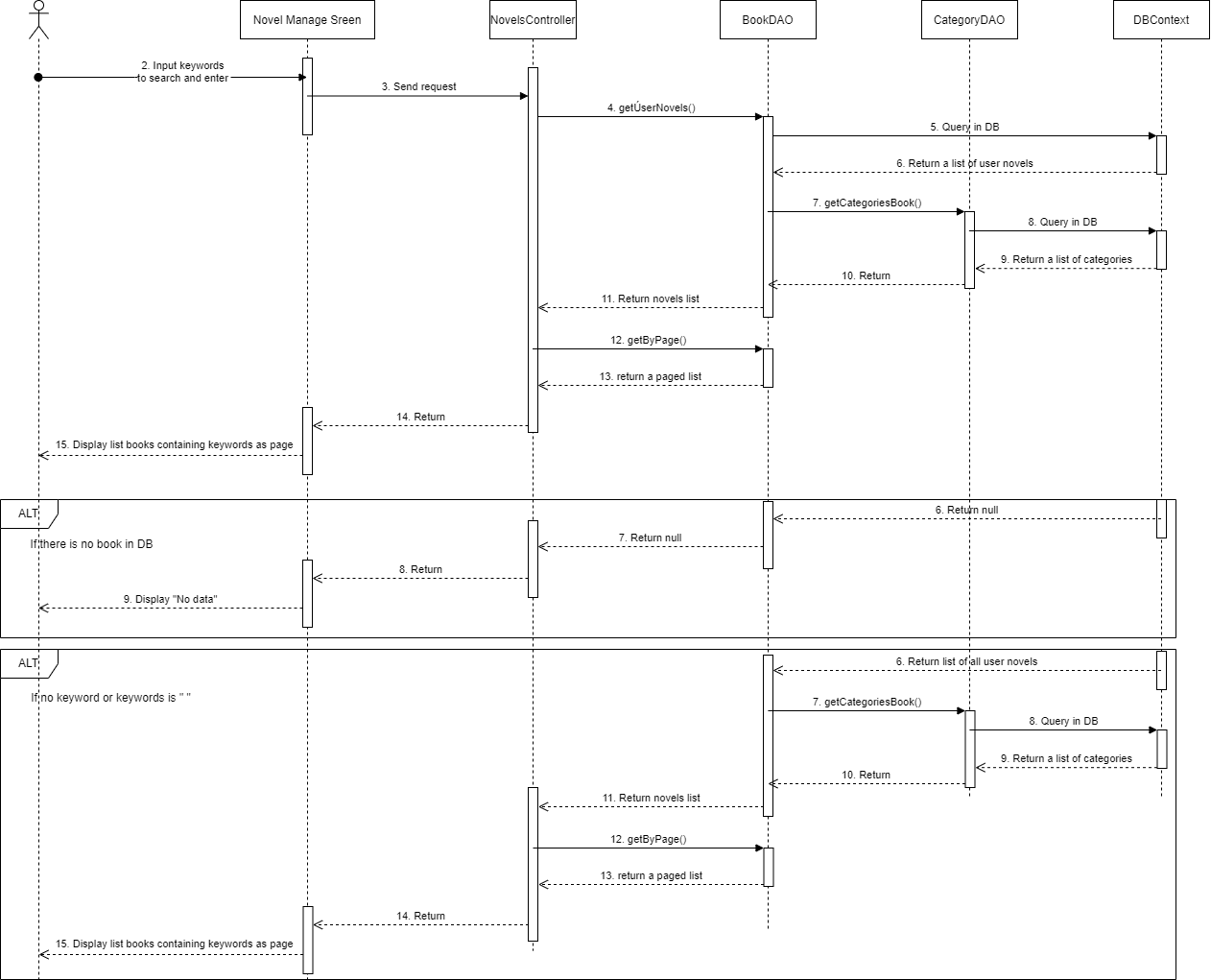
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |
| *2* | *getCategoriesBook* | *Method used to get all genres of a book in database. Return an array list <Category>* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)



### d. Data queries

1. getUserNovels

SELECT [Book].[id]

,[title]

,[authorId]

,[Author].[name]

,[rating]

,[favourite]

,[price]

,[is\_sale]

,[image]

,[description]

,[views]

,[status]

FROM [Book]

INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]

WHERE [Author].[userId] = ?

AND [Book].[title] LIKE ?

2. getAllCategory

SELECT [id]

,[name]

FROM [dbo].[Category]

3. getCategories

SELECT [categoryId]

,[name]

FROM [CategoryBook] cb

INNER JOIN [Category] c ON cb.[categoryId] = c.[id]

WHERE [bookId] = ?

ORDER BY [name] ASC

## 33.2 Create Novel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### AddBookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *addNovel* | *Method used to insert the data of a new novel into Book table. Return an integer represents id value of inserted <Book>.* |

#### CategoryDAO

***Class Methods***

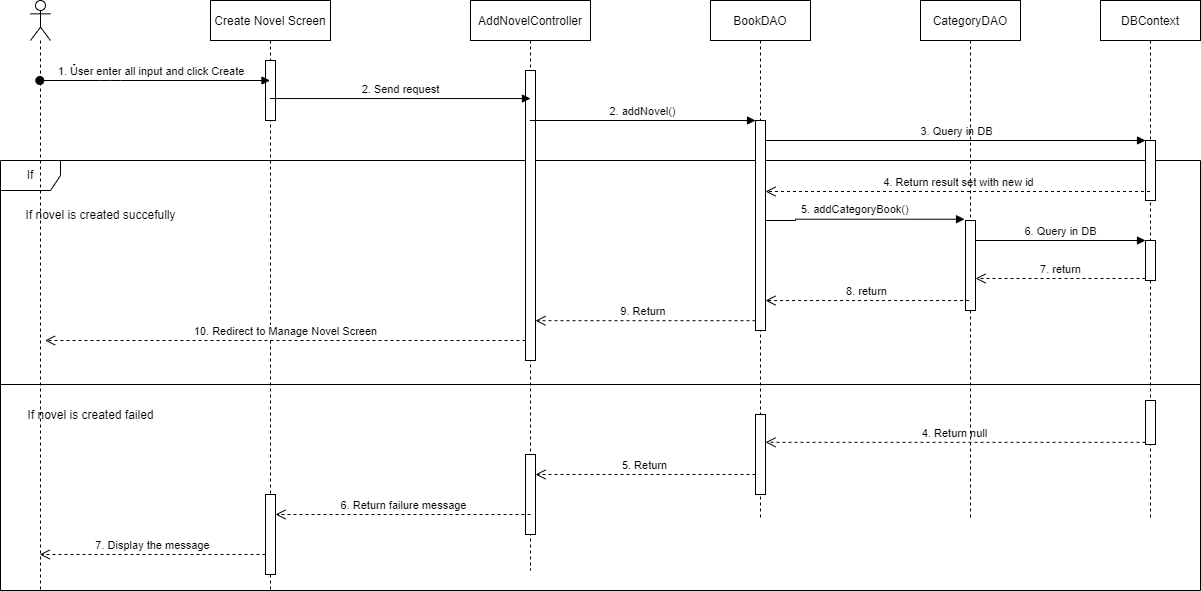
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |
| *2* | *addCategoryBook* | *Method used to insert the data of related genre of a novel into CategoryBook table. Return an integer represents execution function result.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

**

### d. Data queries

1. addNovel:

INSERT INTO [dbo].[Book]

([title]

,[authorId]

,[price]

,[is\_sale]

,[image]

,[description])

OUTPUT Inserted.[id]"

VALUES ( ? "

, (SELECT [id] FROM [Author] WHERE [userId] = ?) "

, ? "

, ? "

, ? "

, ? )

2. addCategoryBook

INSERT INTO [CategoryBook] ([bookId], [CategoryId])

VALUES ( ?, ?)

## 33.2 Update Novel

### a. Class Diagram

Diagram, table

Description automatically generated

### b. Class Specifications

#### UpdateBookController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getBookById* | *Method used to get the data of book that has the input id and return the book* |
| *2* | *editNovel* | *Method used to update Novel into database and return an integer represents sql query result* |

#### CategoryDAO

***Class Methods***

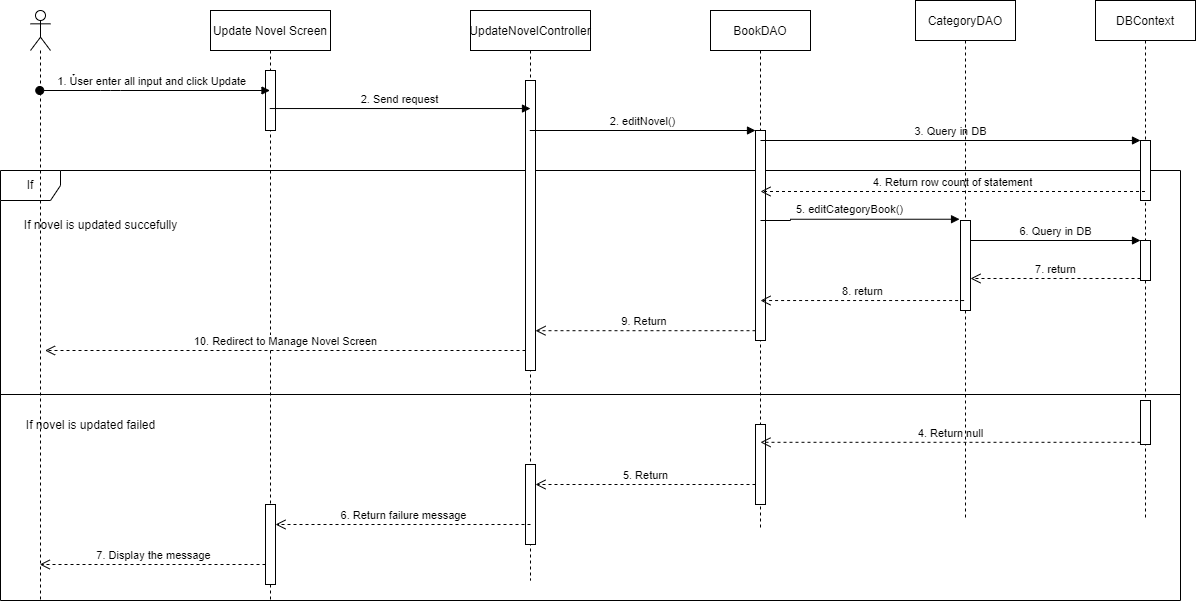
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |
| *2* | *editCategoryBook* | *Method used to insert new data of related genres of a novel into CategoryBook table. Return an integer represents execution function result.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

**

### d. Data queries

1. *editNovel*

UPDATE [Book] "

SET [title]= ?"

, [price] = ?"

, [is\_sale] = ?"

, [image] = ?"

, [description] = ?"

WHERE [id] = ? "

2. getBookById

SELECT [Book].[id]

,[title]

,[authorId]

,[userId]"

,[Author].[name]

,[rating]

,[favourite]

,[price]

,[is\_sale]

,[image]

,[description]

,[views]

,[status]

FROM [Book]"

INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]"

WHERE [Book].[id] = ?

3. edit Category

INSERT INTO [CategoryBook] ([bookId], [CategoryId])"

VALUES ( ? ,? )

## 33.3 Delete/Disable/Enable Novel

### a. Class Diagram

Text

Description automatically generated

### b. Class Specifications

#### ChangeStatusController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* | *Handles the HTTP <code>POST</code> method.* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *changeStatusById* | *Method used to update status column of the book tuple in database that has the input id.* |
| *2* | *deleteBook* | *Method used to delete entire record of a novel in database. Input is novel id.* |

#### CategoryDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllCategory* | *Method used to get all category and its information in database. Return an array list <Category> represents the data of books.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

### c. Sequence Diagram(s)

*Diagram

Description automatically generated*

### d. Data queries

1. changeStatus

UPDATE [Book]"

SET [status] = 1 ^ [status] "

WHERE [id] = ?"

2. deleteBook

DELETE FROM [Book]

WHERE [id] = ?

## 34 View List Author

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### AuthorController

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### AuthorDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllAuthor ()* | *Method used to return the list of authors which are generated.Return a List of <Code>model.Author</Code> represents the author’s data* |
| *2* | *getAllUser()* | *Method used to return the list of authors which are user sign up .Return a List of <Code>model.Author</Code> represents the author’s data* |
| *3* | *getByPage()* | *Method used to resize the list of user accounts .Inputs are list of author , start-point and end-point .Return a List of <Code>model.Author</Code> represents the user’s data* |

### c. Sequence Diagram(s)

### d. Database queries

getAllAuthor() : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NULL

*getAllUser* () : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NOT NULL

## 35 Edit Author

### a. Class Diagram

Diagram

Description automatically generated

Diagram

Description automatically generated

### b. Class Specifications

#### AuthorController

***Class Methods***

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Method** |  | **Description** |
| *1* | *doGet()* |  | *Handles the HTTP <code>GET</code> method.* |
| *2* | *doPost()* |  | *Handles the HTTP <code>POST</code> method.* |

#### LoadAuthor

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### AuthorDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllAuthor ()* | *Method used to return the list of authors which are generated.Return a List of <Code>model.Author</Code> represents the author’s data* |
| *2* | *getAllUser()* | *Method used to return the list of authors which are user sign up .Return a List of <Code>model.Author</Code> represents the author’s data* |
| *3* | *getByPage()* | *Method used to resize the list of user accounts .Inputs are list of author , start-point and end-point .Return a List of <Code>model.User</Code> represents the user’s data* |
| *4* | *updateAu()* | *Method used to update the new name for a author generated . Input is author name .* |
| *5* | *searchByAname()* | *Method used to return the list of authors which are generated with the name need search .Input is a string name . Return a List of <Code>model.Author</Code> represents the author’s data* |
| *6* | *searchByUname ()* | *Method used to resize the list of user accounts with the name need search .Inputs is a string name .Return a List of <Code>model.User</Code> represents the user’s data* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllBooksByAuthorId ()* | *Method used to return the list of books belong to an author. Input is the id of author . Return a List of <Code>model.Book</Code> represents the author’s data* |
| *2* | *getByPage ()* | *Method used to resize the list of user accounts .Inputs are list of book , start-point and end-point .Return a List of <Code>model.Book</Code> represents the user’s data* |

### c. Sequence Diagram(s)

### d. Database queries

getAllAuthor() : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NULL

*getAllUser* () : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NOT NULL

*updateAu* () : UPDATE [dbo].[Author] SET [name] = ?

WHERE [Author].[id] = ?

*searchByAname* SELECT \* FROM [dbo].[Author]

WHERE WHERE [name] LIKE ? AND [userId] IS NULL

*searchByUname* (): SELECT \* FROM [dbo].[Author]

WHERE WHERE [name] LIKE ? AND [userId] IS NOT NULL

*getAllBooksByAuthorId():* SELECT \* FROM [Book]

INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]

WHERE [authorId] = ? AND [status] != 0

## 36 Admin Panel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### DashBoard

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### AuthorDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllAuthor ()* | *Method used to return the list of authors which are generated.Return a List of <Code>model.Author</Code> represents the author’s data* |
| *2* | *getAllUser()* | *Method used to return the list of authors which are user sign up .Return a List of <Code>model.Author</Code> represents the author’s data* |
| *3* | *getNumQuater ()* | *Method used to get the number of sign up account in each quarter from 2 year nearly. Input are year now and formal. Return a List of <Code>Integer</Code> represents the data* |
| *4* | *getNumYear ()* | *Method used to get the number of sign up account in 1month current year. Input are month and current year. Return <Code>Integer</Code> represents the data* |

#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllUsers ()* | *Method used to return the list all user .Return a List of <Code>model.User</Code> represents the user’s data* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the number of book . Return <Code>Integer</Code> represents the data* |

#### ReportDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the report of book. . Return <Code>Integer</Code> represents the data* |

#### CommentDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the comment of book. Return <Code>Integer</Code> represents the data* |

### c. Sequence Diagram(s)

### d. Database queries

getAllAuthor() : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NULL

*getAllUser* () : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NOT NULL

*getNumQuater* () :

SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] <= '" + ? + "-1-1')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-1-1' AND '" + ? + "-3-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-4-1' AND '" + ? + "-6-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-7-1' AND '" + ? + "-9-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-10-1' AND '" + ? + "-12-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-1-1' AND '" + ? + "-3-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-4-1' AND '" + ? + "-6-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-7-1' AND '" + ? + "-9-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-10-1' AND '" + ? + "-12-31')

*getNumYear :* (SELECT COUNT([Author].[id])

FROM [dbo].[Author]

WHERE YEAR([date]) = ? AND MONTH([date]) = ?)

*getAllUsers* (): Select \* from [User] where [is\_super] < 5

*count():* SELECT COUNT([id])

FROM [dbo].[Book]

*count():* SELECT COUNT([id])

FROM [dbo].[Comment]

*count():* SELECT COUNT([id])

FROM [dbo].[ReportDetail]

## 37 Admin Panel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

#### DashBoard

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *doGet()* | *Handles the HTTP <code>GET</code> method.* |

#### DBContext

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getConnection* | *Method used to connect to the database. Return a <Code>java.sql.Connection</Code> represents a connection to the database.* |

#### AuthorDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllAuthor ()* | *Method used to return the list of authors which are generated.Return a List of <Code>model.Author</Code> represents the author’s data* |
| *2* | *getAllUser()* | *Method used to return the list of authors which are user sign up .Return a List of <Code>model.Author</Code> represents the author’s data* |
| *3* | *getNumQuater ()* | *Method used to get the number of sign up account in each quarter from 2 year nearly. Input are year now and formal. Return a List of <Code>Integer</Code> represents the data* |
| *4* | *getNumYear ()* | *Method used to get the number of sign up account in 1month current year. Input are month and current year. Return <Code>Integer</Code> represents the data* |

#### UserDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *getAllUsers ()* | *Method used to return the list all user .Return a List of <Code>model.User</Code> represents the user’s data* |

#### BookDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the number of book . Return <Code>Integer</Code> represents the data* |

#### ReportDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the report of book. . Return <Code>Integer</Code> represents the data* |

#### CommentDAO

***Class Methods***

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| *1* | *count ()* | *Method used to return the comment of book. Return <Code>Integer</Code> represents the data* |

### c. Sequence Diagram(s)Diagram, schematic Description automatically generated

### d. Database queries

getAllAuthor() : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NULL

*getAllUser* () : SELECT \* FROM [dbo].[Author]

WHERE [userId] IS NOT NULL

*getNumQuater* () :

SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] <= '" + ? + "-1-1')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-1-1' AND '" + ? + "-3-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-4-1' AND '" + ? + "-6-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-7-1' AND '" + ? + "-9-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-10-1' AND '" + ? + "-12-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-1-1' AND '" + ? + "-3-31')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-4-1' AND '" + ? + "-6-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-7-1' AND '" + ? + "-9-30')

UNION ALL

(SELECT COUNT([Author].[date])

FROM[dbo].[Author]

WHERE[date] BETWEEN '" + ? + "-10-1' AND '" + ? + "-12-31')

*getNumYear :* (SELECT COUNT([Author].[id])

FROM [dbo].[Author]

WHERE YEAR([date]) = ? AND MONTH([date]) = ?)

*getAllUsers* (): Select \* from [User] where [is\_super] < 5

*count():* SELECT COUNT([id])

FROM [dbo].[Book]

*count():* SELECT COUNT([id])

FROM [dbo].[Comment]

*count():* SELECT COUNT([id])

FROM [dbo].[ReportDetail]

## 38 List book of this author

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

- BookDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getAllBooksByAuthorId(aid: int) | Get a list of books from database by authorId |

- AuthorDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getAuthorById(aid: int) | Get author’s information by their ID |

- DBContext class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

-BookAuthor class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |

### c. Sequence Diagrams(s)

Chart, box and whisker chart

Description automatically generated

### d. Database queries

- getAuthorById(aid: int): SELECT \* from Author where aid = ?

- getAllBooksByAuthorId(aid: int): "SELECT [Book].[id]\n"

+ " ,[title]\n"

+ " ,[authorId]\n"

+ " ,[Author].[name]\n"

+ " ,[rating]\n"

+ " ,[favourite]\n"

+ " ,[price]\n"

+ " ,[is\_sale]\n"

+ " ,[image]\n"

+ " ,[description]\n"

+ " ,[views]\n"

+ " ,[status]\n"

+ " FROM [Book]"

+ " INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]"

+ " WHERE [authorId] = ?\n"

+ " AND [status] != 0 "

## 39 Add chapter for user’s novel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specifications

- VolumeDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getVolumeByBookId(bookId: int) | Get volume list of a book by its ID |
| 02 | getVolumeById(volId: int) | Get volume’s information by its ID |

- ChapterDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getChapterById(chapterID: int) | Get chapter’s information by its ID |
| 02 | getChapterByBookId(bookId: int) | Get chapter list of a book by its ID |
| 03 | addChapter(chapter: Chapter) | Add chapter to the database within information |

- BookDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getBookById(bookid: int) | Get book’s information by its ID |

- DBContext class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

- AddChapterController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

### c. Sequence Diagram

Diagram

Description automatically generated

### d. Data queries

- addChapter(): INSERT INTO [dbo].[Chapter]\n"

+ " ([volumeId]"

+ " ,[no]\n"

+ " ,[title]\n"

+ " ,[status]\n"

+ " ,[content])\n"

+ " OUTPUT [Inserted].[id]"

+ " VALUES\n"

+ " ( ? "

+ " , (SELECT COUNT([no]) FROM [dbo].[Chapter] WHERE [volumeId] = ?)+1 "

+ " , ? "

+ " , ? "

+ " , ? )

## 40 Edit chapter for user’s novel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class specification

- ChapterDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getChapterById(chapterID: int) | Get chapter’s information by its ID |
| 02 | getChapterByBookId(bookId: int) | Get chapter list of a book by its ID |
| 03 | editChapter(chapter: Chapter) | Change chapter’s information |

- DBContext class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

- EditChapterController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

### c. Sequence diagram

Diagram

Description automatically generated

### d. Database queries

- editChapter(): "UPDATE [dbo].[Chapter]\n"

+ " SET [title] = ? "

+ " ,[status] = ?"

+ " ,[content] = ? "

+ " WHERE [id] = ? "

## 41 Delete chapter for user’s novel

### a. Class Diagram

Diagram

Description automatically generated

### b. Class Specification

- ChapterDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getChapterById(chapterID: int) | Get chapter’s information by its ID |
| 02 | getChapterByBookId(bookId: int) | Get chapter list of a book by its ID |
| 03 | deleteChapter(chapter: Chapter) | Delete chapter’s all information |

- DBContext class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | getConnection() | Connecting with database |

- DeleteChapterController class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

### c. Sequence diagram

Diagram

Description automatically generated

### d. Database queries

- deleteChapter(ChapterId): "DELETE FROM [dbo].[Chapter] WHERE [id] = ? "

## 42.1 View Category

#### a. Class Diagram

Diagram

Description automatically generated

#### b. Class Specifications

##### Controller.admin.category.ViewCategoryController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

##### Model.product.Category

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | getName() : String | Get name of category |
| 02 | getId : int | Get ID of category |
| 03 | setId(int id) : void | Set number as ID of category |
| 04 | setName(String name) : String | Set string as name of category |

##### context.product.CategoryDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | countBook(Category category) : int | Count number of books in input category |
| 02 | getAllCategory\_SortById() : ArrayList<Category> | Get all categories from database order by id ascending |

#### c. Sequence Diagram(s)

Calendar

Description automatically generated

#### d. Database queries

##### getAllCategory\_SortById()

SELECT [id] , [name] FROM [Category] ORDER BY [id] ASC

##### countBook(Category category)

select COUNT (\*) as total from Category c

INNER JOIN CategoryBook cb ON c.id = cb.categoryId

where c.id = ?

## 42.2 Add Category

#### a. Class Diagram

Diagram

Description automatically generated

#### b. Class Specifications

##### Controller.admin.category.AddCategoryController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

##### context.product.CategoryDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | addCategory(String name): int | Add new category into database |

#### c. Sequence Diagram(s)

A picture containing calendar

Description automatically generated

#### d. Database queries

##### addCategory(String name)

insert [Category] ([name]) values (?)

## 42.3 Update Category

#### a. Class Diagram

Diagram

Description automatically generated

#### b. Class Specifications

##### Controller. admin.category.UpdateCategoryController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

##### context.product.CategoryDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | editCategory(int id, String editName) : int | Rename category have input id |

#### c. Sequence Diagram(s)

Diagram, calendar

Description automatically generated with medium confidence

#### d. Database queries

##### editCategory(int id, String editName)

update [Category] set [name]= ? where [id]=?

## 42.4 Remove Category

#### a. Class Diagram

Diagram

Description automatically generated

#### b. Class Specifications

##### Controller. admin.category.RemoveCategoryController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

##### context.product.CategoryDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | delBookCategory(int id) : void | Remove category have input id from all books which have this category |
| 02 | : void | Remove category have input id from database |

#### c. Sequence Diagram(s)

Calendar

Description automatically generated

#### d. Database queries

##### delBookCategory(int id)

DELETE FROM [CategoryBook] WHERE [categoryId] = ?

##### delCategory(int id)

delete from [Category] where [id] = ?

## 43 Bookshelf

#### a. Class Diagram

Diagram

Description automatically generated

#### b. Class Specifications

##### Controller.payment.ViewWalletController

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | doGet | Handles the HTTP <code>GET</code> method. |
| 02 | doPost | Handles the HTTP <code>POST</code> method. |

##### Context.product.BookDAO

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | + getNovelsPagging(User user, int pageSize, int pageIndex) : ArrayList<Book> | Get pageSize novels in pageIndex from database |
| 02 | countNovel(User user) : int | Count number of novels that input user wrote |

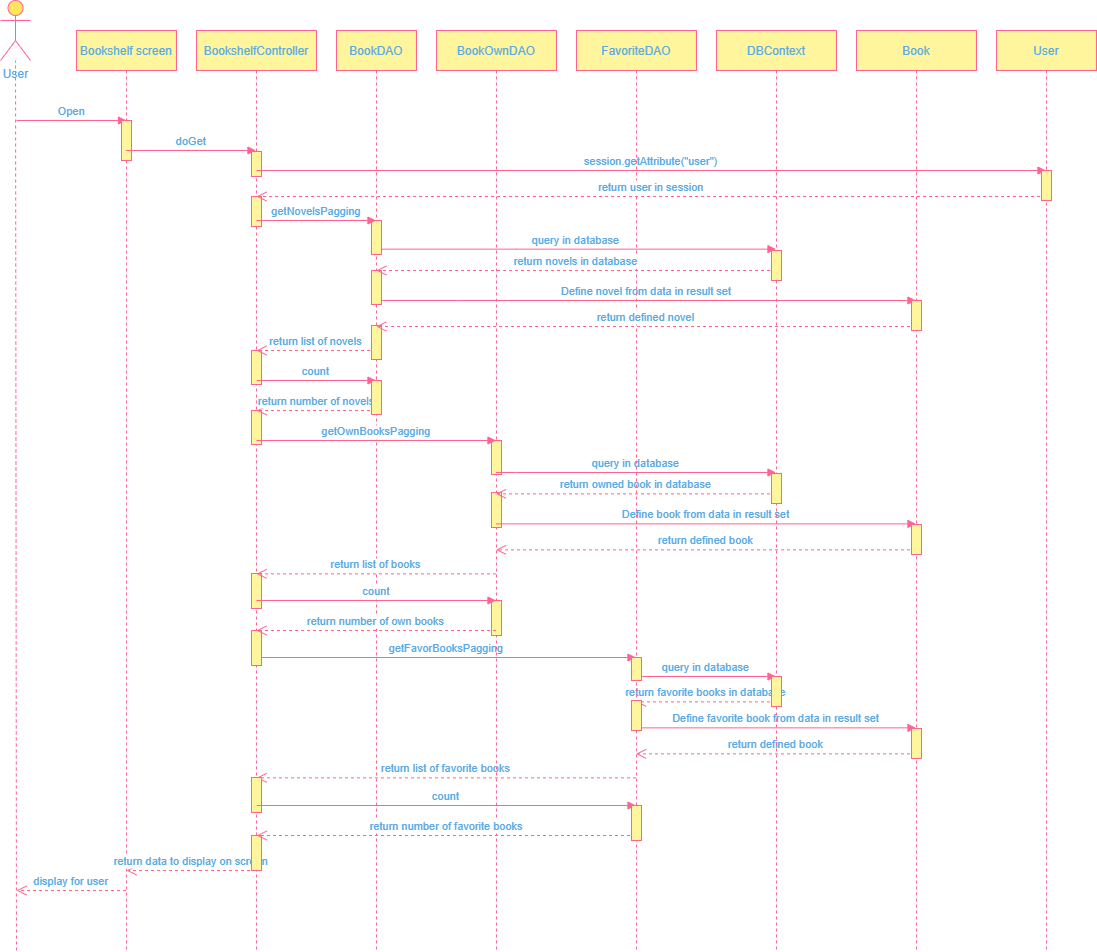
##### Context.product.BookOwnDAO

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Method** | **Description** | |
| 01 | + getOwnBooksPagging(User user, int pageSize, int pageIndex) : ArrayList<Book> | | Get pageSize books that user bought in pageIndex from database |
| 02 | Count(User user) : int | | Count number of books that user bought |

##### Context.product.FavoriteDAO

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Method** | **Description** | |
| 01 | + getFavorBooksPagging(User user, int pageSize, int pageIndex) : ArrayList<Book> | | Get pageSize books that user added to favorite in pageIndex from database |
| 02 | count(User user) : int | | Count number of books that user added to favorite |

#### c. Sequence Diagram(s)



#### d. Database queries

##### getFavorBooksPagging(User user, int pageSize, int pageIndex)

select b.id, b.[title], b.[authorId], b.[rating], b.[favourite]

,b.[price], b.[is\_sale], b.[image], b.[description], b.[views], b.[status]

from [Favourite] f inner join Book b on f.bid = b.id

where f.uid = ?

ORDER BY b.id ASC OFFSET ? \* (?-1) ROWS FETCH NEXT ? ROWS ONLY

##### FavoriteDAO.count(User user) : int

SELECT COUNT(\*) as total FROM [Favourite] WHERE uid = ?

##### getOwnBooksPagging(User user, int pageSize, int pageIndex)

select bo.bookId, b.[title], b.[authorId], b.[rating], b.[favourite], b.[price]

,b.[is\_sale], b.[image], b.[description], b.[views], b.[status]

from [Book\_Own] bo inner join Book b on bo.bookId = b.id

where bo.userId = ?

ORDER BY b.id ASC OFFSET ? \* (?-1) ROWS FETCH NEXT ? ROWS ONLY

##### getNovelsPagging(User user, int pageSize, int pageIndex)

SELECT [Book].[id], [title], [authorId],[Author].[name] ,[rating] ,[favourite]

,[price],[is\_sale],[image],[description] ,[views] status]

FROM [Book] INNER JOIN [Author] ON [Book].[authorId] = [Author].[id]

WHERE [Author].[userId] = ? ORDER BY [Book].id ASC

OFFSET ? \* (?-1) ROWS FETCH NEXT ? ROWS ONLY

##### BookOwnDAO.count(User user)

SELECT COUNT(\*) as total FROM [Book\_Own] WHERE userId = ?

##### countNovel(User user)

SELECT COUNT(\*) as total FROM [Book] INNER JOIN [Author] on [Book].authorId = [Author].id

WHERE [Author].userId =

# III. Database Design

## 1. Database Schema

Diagram

Description automatically generated

## 2. Table Description

|  |  |  |
| --- | --- | --- |
| **No** | **Table** | **Description** |
| 01 | Book | - Primary key: ID  - Foreign key :<none>  - Attribute: tittle, author, categoryID, rating, favourite, price, is\_sale, image, description, views |
| 02 | User | - Primary key: ID  - Foreign key: <none>  - Attribute: fullname, gender, dob, email, phone, address,username, password, is\_super |
| 03 | Favourite | - Primary key: uID  - Foreign key: uID(User: ID), bID(Book: ID)  - Attribute: <none> |
| 04 | Category | - Primary key: ID  - Foreign key: ID(Book: categoryID)  - Attribute: name |
| 05 | Star | - Primary key: bID  - Foreign key: bID(Book: ID)  - Attribute: <none> |
| 06 | ResetToken | - Primary key: uID  - Foreign key: uID(Book: ID)  - Attribute: token, expired data |
| 07 | Volume | - Primary key: bID, vol  - Foreign key: bID(Book: ID)  - Attribute: vol\_name |
| 08 | Chapter | - Primary key: bID, vol, chapter  - Foreign key: bID(Book: ID), vol(Volumn: vol)  - Attribute: chapter\_name, content |
| 09 | Transaction | - Primary key: userId  - Foreign key: userId(User: ID)  - Attribute: balanceBefore, amount, balanceAfter, transactionTime, type, status |
| 10 | Comment | - Primary key: bookId, userID  - Foreign key: bookId(Book: ID), userId(User: ID)  - Attribute: comment |
| 11 | Content | - Primary key: chapterId  - Foreign key: chapterId(Chapter: ID)  - Attribute: paragraph, content |
| 12 | Report | - Primary key: id  - Foreign key: <none>  - Attribute: title |
| 13 | ReportDetail | - Primary key: id  - Foreign key: reportId(Report: ID), bookId(Book: ID), userId(User: ID)  - Attribute:note |