

Xiaotian Bai
N16340028

Project Part II Documentation

0.Preface

The scenario of my project is online bidding system. In the final design, I chose Java Spring framework for the overall project design and source code was mostly written in Java and JavaScript. For the frontend UI design, I used CSS and Bootstrap. The connection between frontend pages and backend database was established by MySQL, Mybatis and Ajax. The whole project was built on Windows 10.

For the tools, I used VS2017 as text editor because of the navigation tools and I ran the project on MyEclipse for the built-in browser. The database was managed by MySQL workbench

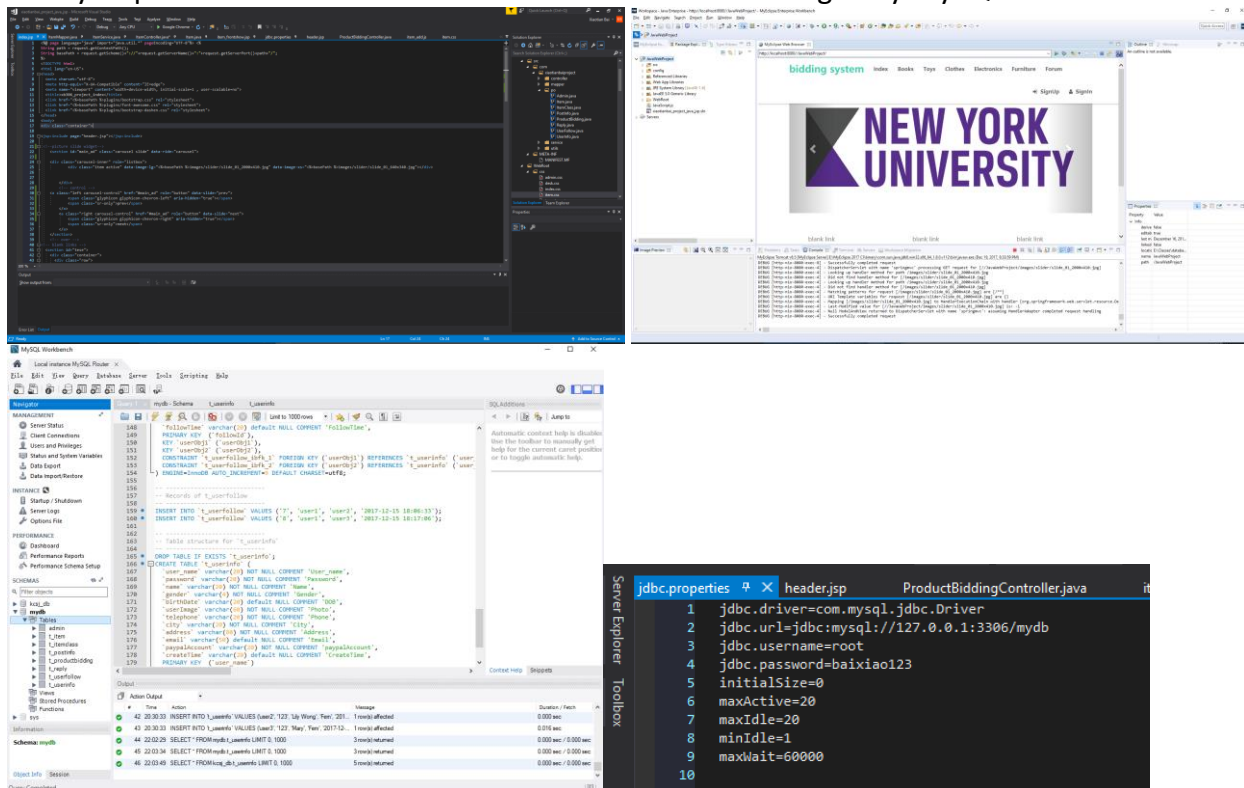


Fig.0 IDEs and JDBC database connection

The website implements an online bidding platform for the users to sell and buy items and a simple forum to support the social functions. The website could run on a local Apache Tomcat server.

1.ER Design

For the bidding system scenario, the goal of the project design is to allow users to communicate and conduct transactions between each other. Hence, the ER model is naturally split into two major parts: social network and bidding system.

Like many other social networks, a valid user needs to sign up and provide some personal information to begin with, including name, email, phone number etc. Alternatively, the user can provide more info such as gender, city, and photo(portrait) to share with other users.

1.1 Review on Part I

In project part I, I designed some relations. The ER diagram for this part is given below, the whole diagram is created in MySQL Workbench.

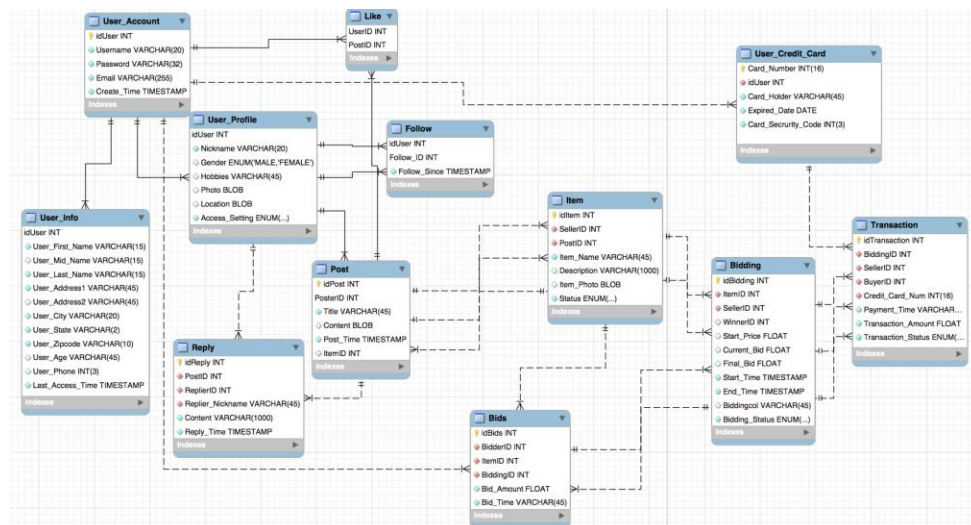


Fig.1 ER diagram in Part I

The relational schema corresponding to the ER is given as:

User_Account (idUser, Username, Password, Email, Create_Time)

User_Info (idUser, User_First_Name, User_Mid_Name, User_Last_Name, User_Address1, User_Address2, User_City, User_State, User_Zipcode, User_Age, User_Phone, Last_Access_Time), idUser is a FK referencing User_Account

User_Profile(idUser, Nickname, Gender, Hobbies, photo, Location, Access_Setting, Nickname, Gender, Hobbies, photo, Location, Access_Setting), idUser is a FK referencing User_Account

Follow (idUser, Follow_ID, Follow_Since), both idUser and Follow_ID are FKs referencing User_Profile

Post (idPost, PosterID, Title, Content, Post_Time, ItemID), PosterID is a FK referencing User_Profile, ItemID is a FK referencing Item

Reply (idReply, PostID, ReplierID, Content, Reply_Time), PostID is a FK referencing Post, ReplierID is a FK referencing User_Profile

Like (UserID, PostID), UserID is a FK referencing User_Profile, PostID is a FK referencing Post

Item (idItem, SellerID, PostID, Item_Name, Description, Item_photo, Status), SellerID is a FK referencing User_Profile, PostID is a FK referencing Post

Bids (idBids, BidderID, ItemID, BiddingID, Bid_Amount, Bid_Time), BidderID is a FK referencing User_Profile, ItemID is a FK referencing Item, BiddingID is a FK referencing Bidding

Bidding (idBidding, ItemID, SellerID, WinnerID, Start_Price, Current_Bid, Final_Bid, Start_Time, End_Time), ItemID is a FK referencing Item, SellerID and WinnerID are FKs referencing User_Profile

User_Credit_Card (Card_Number, UserID, Card_Holder, Expiration_Date, Card_Security_Code), UserID is a FK referencing User_Profile

Transaction (idTransaction, BiddingID, SellerID, BuyerID, Credit_Card_Num, Payment_Time, Transaction_Amount, Transaction_Status), BiddingID is a FK referencing Bidding, SellerID and BuyerID are both FKs referencing User_Profile, Credit_Card_Num is a FK referencing User_Credit_Card

1.2 New Design in Part II

When I tried to implement the ER design on the website, I found some relations and functions not realistic and very difficult to code. Moreover, some information could simply be provided by Java code instead of recording it in database such as the status and end time of biddings. Hence, for this time I redesigned the ER diagram.

As I switched from Mac OS to Windows 10 for the free tools, the following diagrams may look different from those in Part I, but the main attributes and overall structure remain the same. Here, to separate the tables from the actual objects, I used "t_" as prefix for each table and for those objects as foreign keys to corresponding tables, I added postfix "Obj".

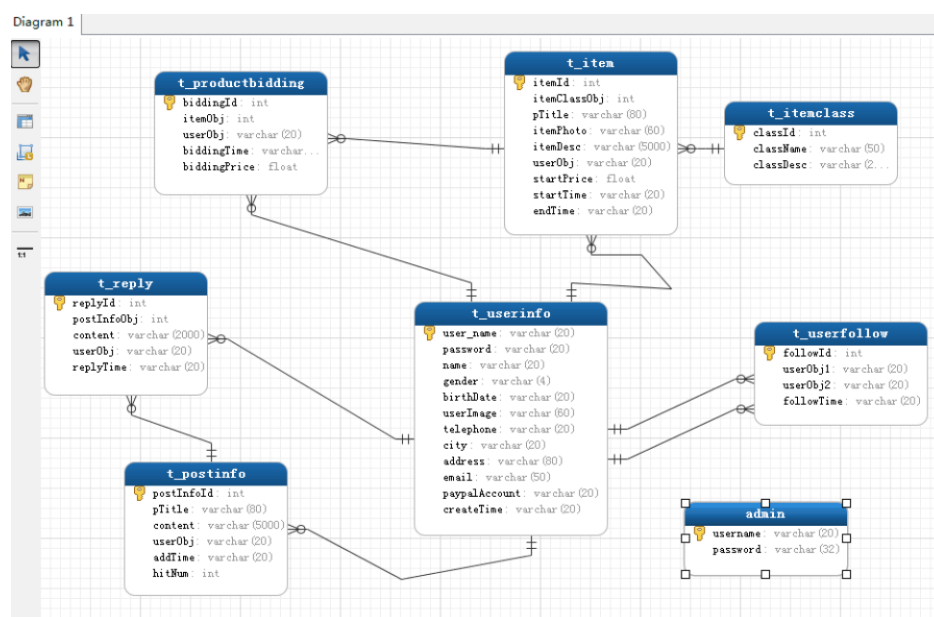


Fig.2 ER diagram in Part II

The new relational schema is given as:

t_userinfo (user_name, password, name, gender, birthdate, userImage, phone, city, address, email, paypalAccount)

t_userfollow (followId, userObj1, userObj2, followTime), userObj1 and userObj2 are foreign keys referencing t_userinfo

t_postinfo (postInfoId, pTitle, content, userObj, addTime, hitNum), userObj is a foreign key referencing t_userinfo

t_reply (replyId, postInfoObj, content, userObj, replyTime), postInfoObj is a foreign key referencing t_postinfo

t_item (itemId, itemClassObj, Title, itemPhoto, itemDesc, userObj, startPrice, startTime, endTime), itemClassObj is a foreign key referencing t_itemclass, userObj is a foreign key referencing t_userinfo

t_itemclass (classId, className, classDesc)

t_productbidding (biddingId, itemObj, userObj, biddingTime, biddingPrice), itemObj is a foreign key referencing t_item, userObj is a foreign key referencing t_userinfo

The **admin** table is set for the purpose of backend administrators, it's still under development and not related to any actual implementation in the project.

Comparing the new ER design to the old one in Part I, a lot of attributes in the old design have been removed in exchange for convenience in coding. For example, the 'Credit Card' and 'Transaction' tables were replaced by a single attribute 'paypalAccount' in t_userinfo because the credit card info is not supposed to be stored in server for security and the transaction requires a lot of services and APIs from banks. To make things simple, I introduced PayPal as the 3rd party transaction platform and assume the service is covered by PayPal just like EBay.

I combined User_Account and User_Info into a single table t_userinfo. The original design was to keep the User_Account private but User_Info public, but now that the project is able to query for the specific data in the table, the private part and public part don't need to be separated.

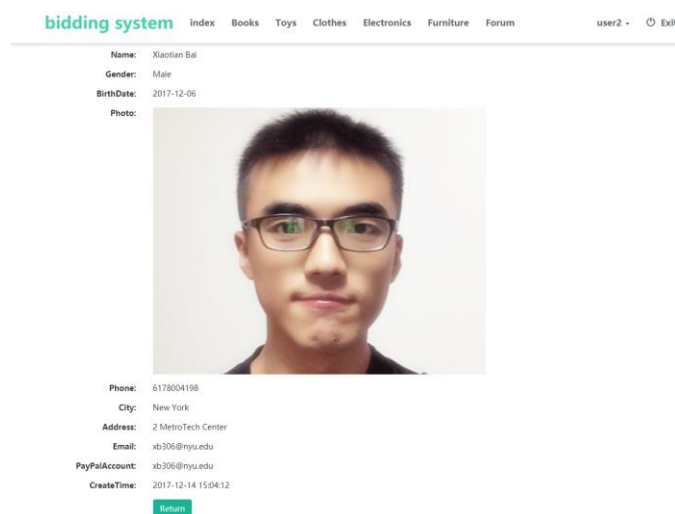


Fig.3 Userinfo

From the picture above, we can see that username and password are hidden from the user profile.

I also combined the Item and Bidding table since a bidding session is associated with an item and vice versa. There is no need to separate the item and its bidding session.

The old and new schemas are shown below. Clearly the new schema is simpler and has a better format.

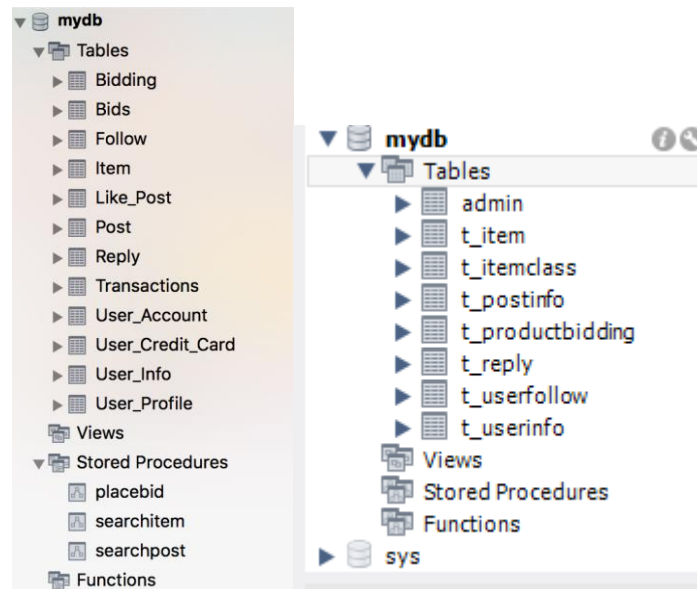
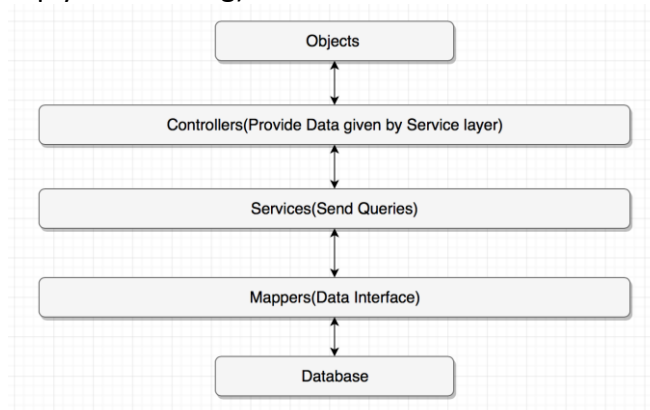


Fig.4 Old schema and new schema

2.Function Design

2.1 Overall Design

For the function design, I used Java Spring framework, which is very popular and resourceful. According to the guidance of SpringMVC I designed several layers from bottom to top, namely, mapper layer, service layer and controller layer. Each layer has a dedicated file for each object class (user, item, post, reply and bidding).




```

ItemController.java
238 request.setAttribute("pTitle", pTitle);
239 request.setAttribute("userObj", userObj);
240 request.setAttribute("startTime", startTime);
241 List<ItemClass> itemClassList = itemClassService.queryAllItemClass();
242 request.setAttribute("itemClassList", itemClassList);
243 List<UserInfo> userInfoList = userInfoService.queryAllUserInfo();
244 request.setAttribute("userInfoList", userInfoList);
245 return "item_frontshowquery_result";
246 }
247
248 /**Frontend Search ItemInfo*/
249 @RequestMapping(value="/{itemId}/frontshow",method=RequestMethod.GET)
250 public String frontshow(@PathVariable Integer itemId,Model model,HttpServletRequest request) throws Exception {
251     //UserKeyItemIdsGetItemObj*/
252     Item item = itemService.getItem(itemId);
253     List<ItemClass> itemClassList = itemClassService.queryAllItemClass();
254     request.setAttribute("itemClassList", itemClassList);
255     List<UserInfo> userInfoList = userInfoService.queryAllUserInfo();
256     ArrayList<ProductBidding> biddingList = biddingService.queryProductBidding(item, null, "");
257     request.setAttribute("userInfoList", userInfoList);
258     request.setAttribute("item", item);
259     request.setAttribute("biddingList", biddingList);
260     return "item_frontshow";
261 }
262
263 /**Ajax: Display Item and Edit Item Page*/
264 @RequestMapping(value="/{itemId}/update",method=RequestMethod.GET)
265 public void update(@PathVariable Integer itemId,Model model,HttpServletRequest request,HttpServletResponse response) throws Exception {
266     //UserKeyItemIdsGetItemObj*/
267     Item item = itemService.getItem(itemId);
268     response.setContentType("text/json;charset=utf-8");
269     PrintWriter out = response.getWriter();
270     JSONObject jsonItem = item.getJSONObject();
271     out.println(jsonItem.toString());
272     out.flush();
273     out.close();
274 }

```

Fig.7 Controller layer code

```

po
├── Admin.java
├── Item.java
├── ItemClass.java
├── PostInfo.java
├── ProductBidding.java
├── Reply.java
├── UserFollow.java
└── UserInfo.java

```

```

8 public class Item {
9     /*itemid*/
10    private Integer itemId;
11    public Integer getItemId(){
12        return itemId;
13    }
14    public void setItemId(Integer itemId){
15        this.itemId = itemId;
16    }
17
18    /*itemClass*/
19    private ItemClass itemClassObj;
20    public ItemClass getItemClassObj() {
21        return itemClassObj;
22    }
23    public void setItemClassObj(ItemClass itemClassObj) {
24        this.itemClassObj = itemClassObj;
25    }
26
27    /*itemtitle*/
28    @NotEmpty(message="item title cannot be empty")
29    private String pTitle;
30    public String getPTitle() {
31        return pTitle;
32    }
33    public void setPTitle(String pTitle) {
34        this.pTitle = pTitle;
35    }
36
37    /*item photo*/
38    private String itemPhoto;
39    public String getItemPhoto() {
40        return itemPhoto;
41    }

```

Fig.8 Item class

The po (project objects) files declares all the methods to get value for each attribute (e.g. title, id, name). These methods are later called by the server pages and other methods that require data for the objects. Note that these methods do not fetch data from database, they simply return the values of the attributes from a constructed object.

```

<div class="row bottom15">
    <div class="col-md-2 col-xs-4 text-right bold">Itemid:</div>
    <div class="col-md-10 col-xs-6"><%=item.getItemId()%></div>
</div>
<div class="row bottom15">
    <div class="col-md-2 col-xs-4 text-right bold">ItemClass:</div>
    <div class="col-md-10 col-xs-6"><%=item.getItemClassObj().getClassName() %></div>
</div>
<div class="row bottom15">
    <div class="col-md-2 col-xs-4 text-right bold">Item Title:</div>
    <div class="col-md-10 col-xs-6"><%=item.getPTitle()%></div>
</div>

```

Fig.9 Project object class methods called by page files

2.2 Signup, Login and Logout

To start with, users must sign up with the website before sign in.

The sign-up page consists of several input bar, including one file input.

The file input is implemented by `<input type="file"/>`, which is supported by commons-fileupload.jar package.

The following pictures are the sign-up interface and t_userinfo table which stores the input information. The 'romil' user was registered during the demo.

The image shows a web application's signup page and a database table. The signup page has fields for Username, Password, Name, Gender, BirthDate, Photo, Phone, City, Address, Email, and PayPalAccount. The Username field is filled with 'user4' and has a green checkmark. The Password field is filled with '...'. The Name field is 'Input Name', Gender is 'Input Gender', BirthDate is 'Choose BirthDate', Photo is 'Choose File' (No file chosen), Phone is 'Input Phone', City is 'Input City', Address is 'Input Address', Email is 'Input Email', and PayPalAccount is 'Input PayPalAccount'. There is an 'Add' button at the bottom. Below the form is a table with the following data:

user_name	password	name	gender	birthDate	userImage	telephone
romil	12345	romil	male	2017-12-12	upload/04d77d15-9bac-479d-8ba8-f69615f451...	123
user 1	123	Xiaotian Bai	Male	2017-12-06	upload/a3ece24a-7314-4d9e-9f5e-663165db1b...	6178004198
user2	123	Lilv Wong	Fem	2017-12-07	upload/user2.jpg	1001112233
user3	123	Marv	Fem	2017-12-01	upload/user3.jpg	1001112244
user4	123	Xiaotian Bai	Male	2017-12-06	upload/4cd74078-0674-4333-9478-cd75eaa2e4...	6178006003
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig.10 Signup page and t_userinfo entries

For the login and logout function, I simply verify the username and password pair stored in the database. If the pair is valid, the verified username is stored and used for all the operations after such as post and reply, until next logout.

```
session.setAttribute("username");  
session.removeAttribute("username");
```

These are the two methods used for login and logout. They helped the server to store the username for further purpose.

```
public void fromLogin(@RequestParam("username")String username,@RequestParam("password")String password,HttpServletRequest response,HttpSession session) throws Exception {  
    boolean success = true;  
    String msg = "";  
  
    if (userService.checkLogin(username, password)) {  
        msg = userService.getMessage();  
        success = false;  
    }  
    if(success) {  
        session.setAttribute("user_name", username);  
    }  
  
    response.setContentType("text/json;charset=UTF-8");  
    PrintWriter out = response.getWriter();  
  
    JSONObject jsonObj = new JSONObject();  
    jsonObj.put("success", success);  
    jsonObj.put("msg", msg);  
    out.println(jsonObj.toString());  
    out.flush();  
    out.close();  
}
```

Fig.11 Login function


```

session.removeAttribute("user_name");
session.removeAttribute("user_name");
session.invalidate();
out.println("<script>top.location='" + basePath + "index.jsp';</script>");

```

Fig.12 Logout function

2.3 Search

In Part I, I designed several stored procedures and functions, they are further implemented by Java and JavaScript code in Project II with many other new features.

For instance, the search function I wrote in project I only support search by name. Now with a more powerful query method written in service, one can search items or posts by their names, categories, posters and time.

```

/* Search Item*/
public ArrayList<Item> queryItem(ItemClass itemClassObj,String pTitle,UserInfo userObj,String startTime) throws Exception {
    String where = "where 1=1";
    if(null != itemClassObj && itemClassObj.getClassId() != null && itemClassObj.getClassId() != 0) where += " and t_item.itemClassObj=" + itemClassObj.getClassId();
    if(!pTitle.equals("")) where = where + " and t_item.pTitle like '%" + pTitle + "%'";
    if(null != userObj && userObj.getUser_name() != null && !userObj.getUser_name().equals("")) where += " and t_item.userObj=" + userObj.getUser_name() + "";
    if(!startTime.equals("")) where = where + " and t_item.startTime like '%" + startTime + "%'";
    return itemMapper.queryItemList(where);
}

```

Fig.13 Item search query

The query method above is in accordance with the layer structure I mentioned earlier. With the help of Mappers, this Server layer method can use multiple keywords as parameters and find the results that fit the combination of different keywords.

Fig.14 Item search interface

2.4 Bidding: Post and Bid

As for the bidding function, a user should be able to post his/her items and also place bid on the items other users posted. Note that there must be some constraints, such as the user can't place bid on his/her own items or items whose bidding sessions have already been expired, a user can only place bid with a higher price than the current highest bid etc.

The post item function is related to insertion queries in service layer, a user should provide the information of the item as below:

Index / AddItem

ItemClass: Book

Item Title: Input Item Title

ItemPhoto: Choose File No file chosen

ItemDescription: Input ItemDescription

StartPrice: Input StartPrice

AddItem

Fig.15 Post item for bidding

To make it simple for illustration, when an item is post, the program will automatically add 5 minutes to the start time and get an end time of the bidding:

```
Date endDate = new java.util.Date(startDate.getTime() + (5*60*1000));
```

Bidding action is associated with insertion to the t_productbidding table. As mentioned previously, this action has many constraints.

ItemDescription: it's a nice chair

Poster: romil

StartPrice: 20.0

Start Time: 2017-12-17 18:04:34

EndTime: 2017-12-17 18:09:34

BiddingPrice:

PlaceBid FollowPoster Return

Fig.16 Bidding interface

The bidding function, as shown above, triggered by a button on the item bidding page, collects price from an input bar, and information from the user and finally return the result using Json messages (succeeded or failed).

```
//Cannot Bid On Your Own Item
if(item.getUserObj().getUser_name().equals(userName)) {
    message = "Cannot Bid On Your Own Item";
    writeJsonResponse(response, success, message);
    return;
}

//Judge if Bidding is Over
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String biddingTime = sdf.format(new java.util.Date());

if(biddingTime.compareTo(item.getEndTime()) > 0) {
    message = "Bidding is Over";
    writeJsonResponse(response, success, message);
    return;
}
```

localhost:8080 says:

Bidding is Over

OK

Fig.16 Constraints and error message

2.5 Forum: Post and Reply

The integrated forum is basically a list of the posts with very simple interface. Very similar to the items, the forum also has a search function. The only difference between forum and the item list is the layout. The forum follows the same workflow as the item list: fetching data from the database and displaying it on the page.

[Index](#) [PostList](#) [AddPost](#)

No.	Postid	PostTitle	Poster	Post Time	Views	Operations
1	1	Hello, World!	Xiaotian Bai	2017-12-14 15:16:36	68	i Inspect
2	2	2234	Lily Wong	2017-12-15 16:32:32	5	i Inspect
3	4	test	romil	2017-12-17 18:05:27	3	i Inspect

[«](#) [1](#) [»](#)

Records: 3 , Page: 1/1

Search for Posts

PostTitle:

Poster :

Post Time:

Fig.17 Forum interface


[PostList](#) [AddPost](#)

PostTitle:

PostContent:


Fig.18 Add post interface

Postid: 1
PostTitle: Hello, World!
PostContent: I like this website!
Poster: Xiaotian Bai
Post Time: 2017-12-14 15:16:36
Views: 68




Hi Nice to meet you!

2017-12-14 15:16:45



Hi Where are you from?

2017-12-15 17:29:03



1231323

2017-12-17 18:05:39

ReplyContent:

Fig.19 Reply list

Due to insufficient time and lack of knowledge and resources, the forum interfaces and functions are poorly designed. Users can only post and reply with plain text, no pictures or emojis available, it's even worse than most free forum template. Also, the expected built-in map is not implemented here.

2.6 Other Functions

Although a lot of features were not successfully implemented, I added the follow user button on the bidding item page and allow users to follow the poster. Also, users are able to view their follow list via the user menu.

FollowPoster

```
181 function doFollow(){
182     $.ajax({
183         url : basePath + "UserFollow/userAdd",
184         type : "post",
185         data: {
186             "userFollow.userObj1.user_name": "<%=item.getUserObj().getUser_name() %>"
187         },
188         success : function (data, response, status) {
189             if(data.success){
190                 alert("Successful!");
191             }else{
192                 alert(data.message);
193             }
194         }
195     });
196 }
```

Fig.21 Follow function

Index		MyFollowerList			
No.	Recordid	Followed	FollowTime	Operations	
1	7	Xiaotian Bai	2017-12-15 18:06:33	Inspect	Cancel

Records: 1 , Page: 1/1

Fig.21 follow list

The follow list reused the template of forum and added a Cancel button to remove the entries.

3.Overall UI design

For the user interface I used CSS and bootstrap for the layout design and I put some icons and pictures to make the UI colorful.

Some resources and materials (mostly icons and buttons) are downloaded from w3school.com, material.io and some other websites.

The pages consist of a header, a foot and its main page

bidding system index Books Toys Clothes Electronics Furniture Forum user4 ⚙ Exit

Fig.22 The header bar

The header is a navigate bar, providing links to different pages and login/logout functions.

The links with item classes are associated with search functions, they basically call the search method on item classes and display the results.

The user menu list is integrated in the nav bar.

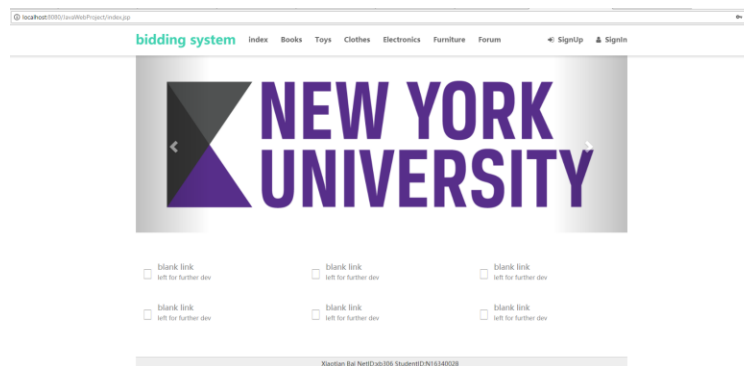


Fig.25 Index page

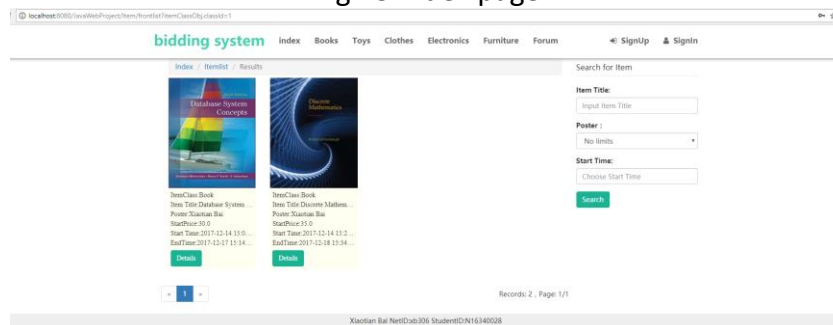


Fig.26 Item list (Books)

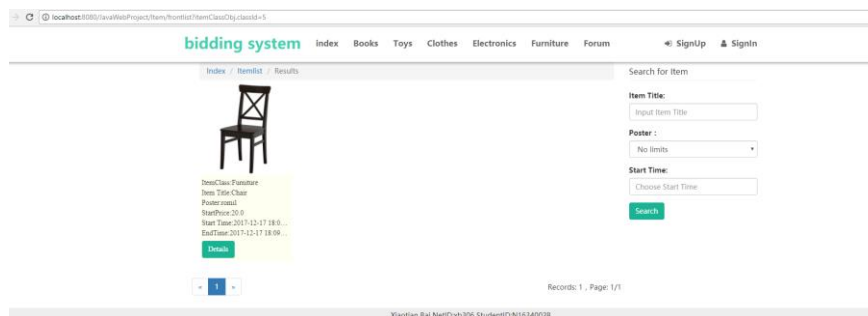


Fig.27 Item list (Furniture)

4.Summary

The project implemented basic functions of a simple online bidding system with an integrated simple forum as required.

Both the UI design and function design are imperfect and immature. Limited in knowledge and time, some required functions such as built-in map and user group settings were not implemented in the design.

As an ECE student, working alone on the project was a big challenge. Thanks to the project, I learned a lot about website development (for both frontend and backend).

From the project I learned how database works in a web-based application and how the application works around the database. I will keep working and learning and make the project better.