Design & Build 2 Report

E-COMMERCE WITH LAWS

GROUP_28

Jiayi Zhang, Guanzhong Zhang, Xiance Sun

1. Division of Labor and Contribution Percentage

a. Integration

Integration: *Jiayi Zhang.*

b. Backend Development

Backend Development (Login/Registration/Product Search with Filter and Conditions/Show Product Details/Cart/Purchase System and MySQL Database Design): *Jiayi Zhang.*

Backend Development (Individual Information Modification/Order Query System): *Guanzhong Zhang.*

c. Frontend Web Development

Frontend Web Development (JSP/HTML/CSS): *Xiance Sun.*

d. Report Writing

Report Writing: *Jiayi Zhang, Guanzhong Zhang.*

e. Contribution Percentage

Contribution percentage diagram is shown in *Diagram.1.*

	Contribution Percentage
Jiayi Zhang	70%
Guanzhong Zhang	15%
Xiance Sun	15%

Diagram.1. Contribution Percentage

2. Display of Function and Webpage

a. Homepage

At Homepage, user can choose to login, sign up, check company information, searching for product. The page screenshot is shown in *Figure.1*.

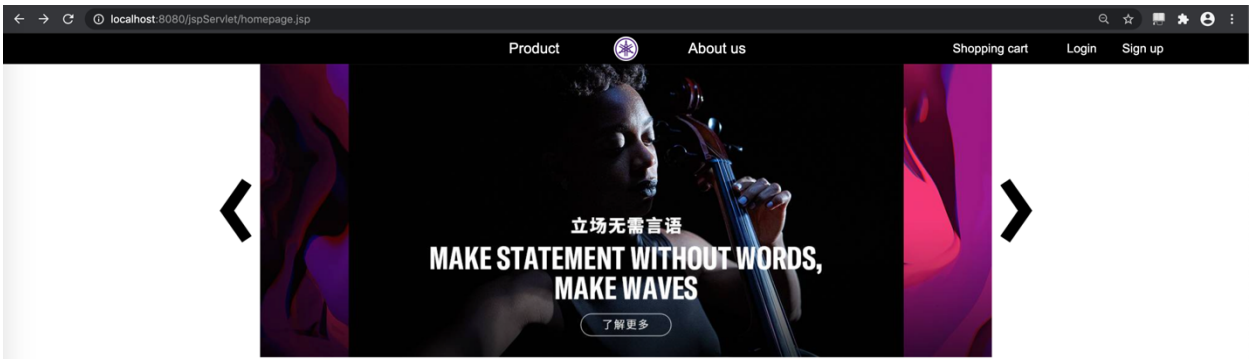

Figure.1. Homepage

b. Login

User are recommended to login by using their account. Click the login button on any page can login by using valid account and password. The page screenshot is shown in *Figure.2*.

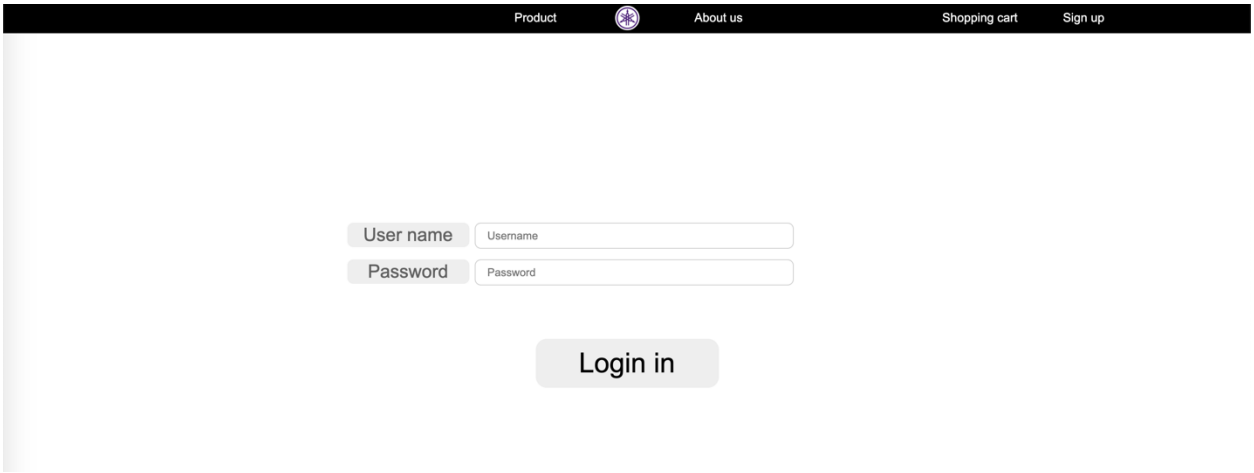

Figure.2. Login Page

c. Registration

Users who don't have an existing account can choose to register. Registration is super easy. Users are only needed to create a username and the corresponding password, and provide their first name, last name and birthday (format constrain: yyyy-mm-dd). And the success registration information will be stored into the MySQL database. Then the user can use login function to login their account. The page screenshot is shown in *Figure.3*. The MySQL proving is shown in *Figure.4*.

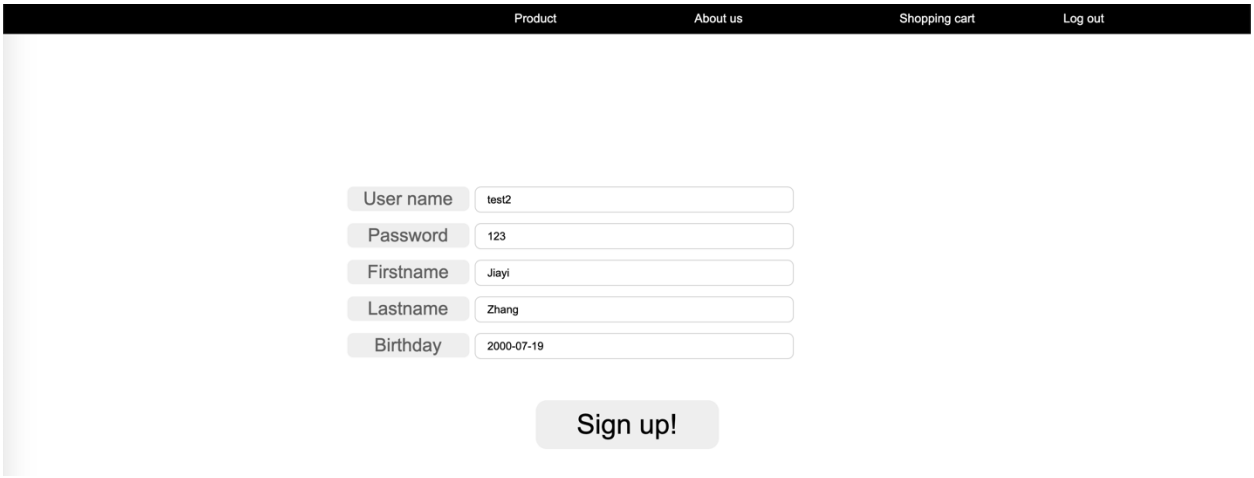

Figure.3. Registration Page

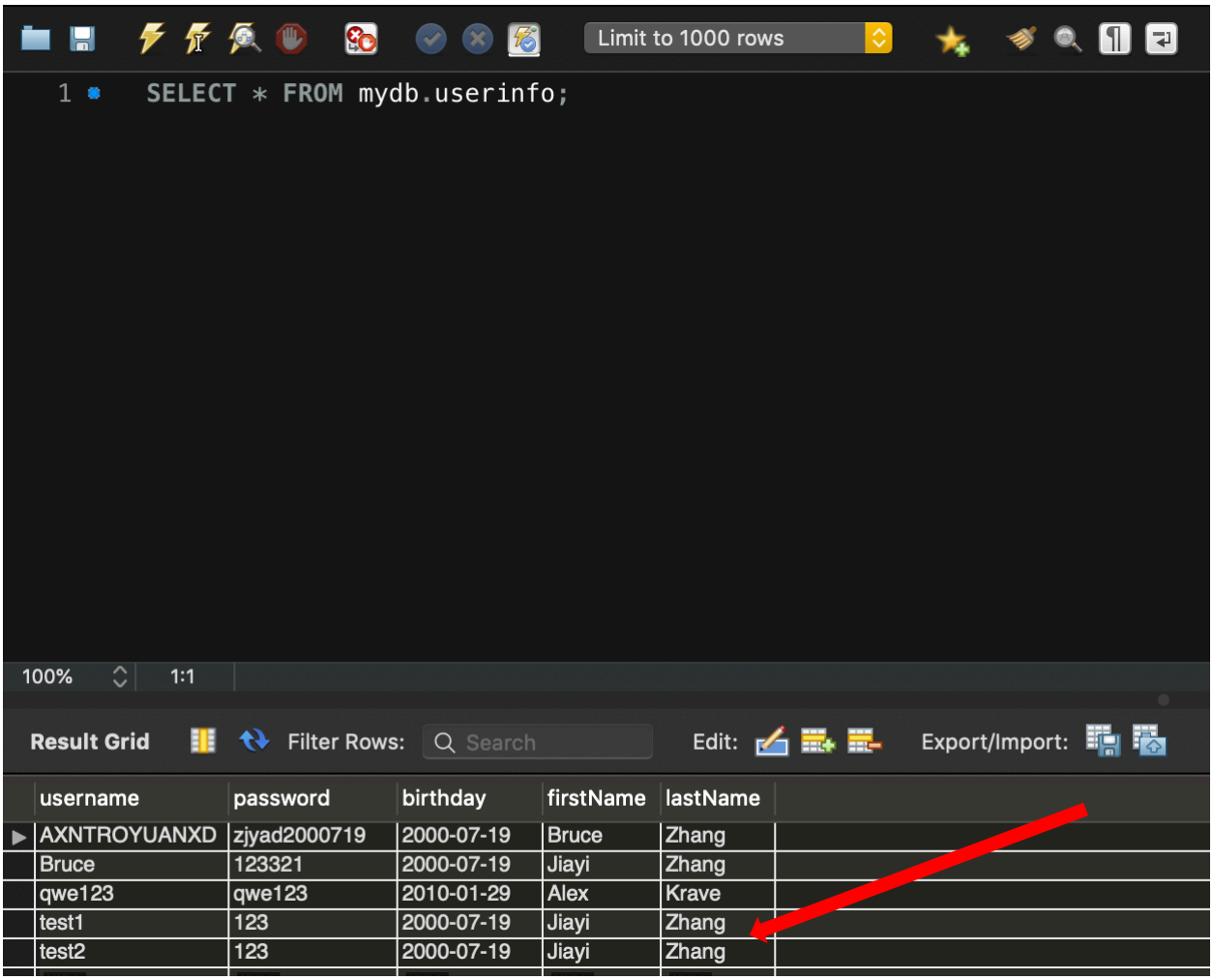

Figure.4. Registration Data in MySQL

d. Homepage (Logined Status)

Once a user successfully login. The homepage and any other pages will all display the user's username. The page screenshot is shown in *Figure.5*.

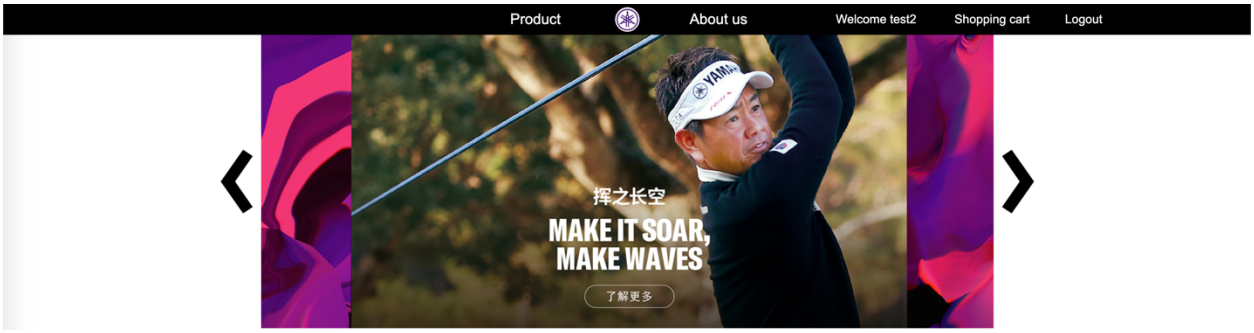

Figure.5. Logined Homepage Displaying the Username

e. Product Search and Browse

User can enter the keyword to search the related products and browse them on the page. Also, user can choose the display order: by "default-order" or by "latest-released-order", and choose filter: "all" or "classic" or "electronic". The page screenshot is shown in *Figure.6* and *Figure.7*.

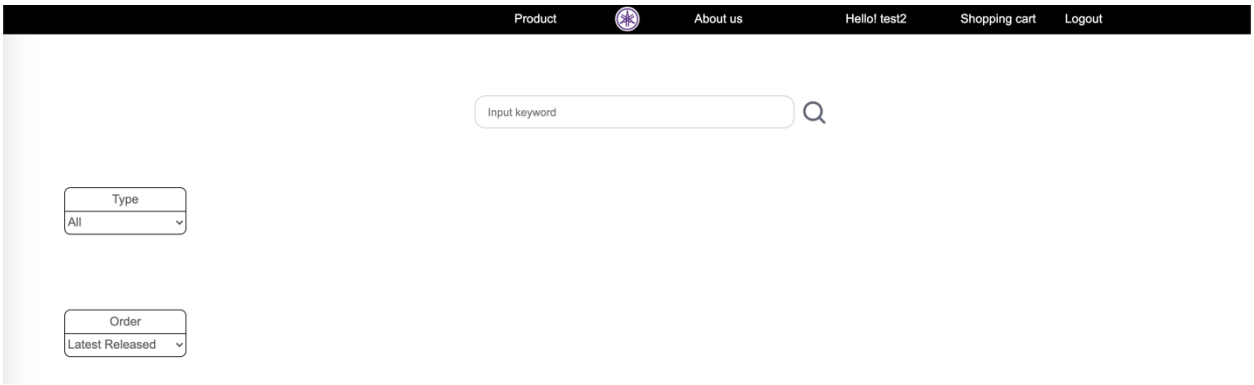

Figure.6. Initial Search Page

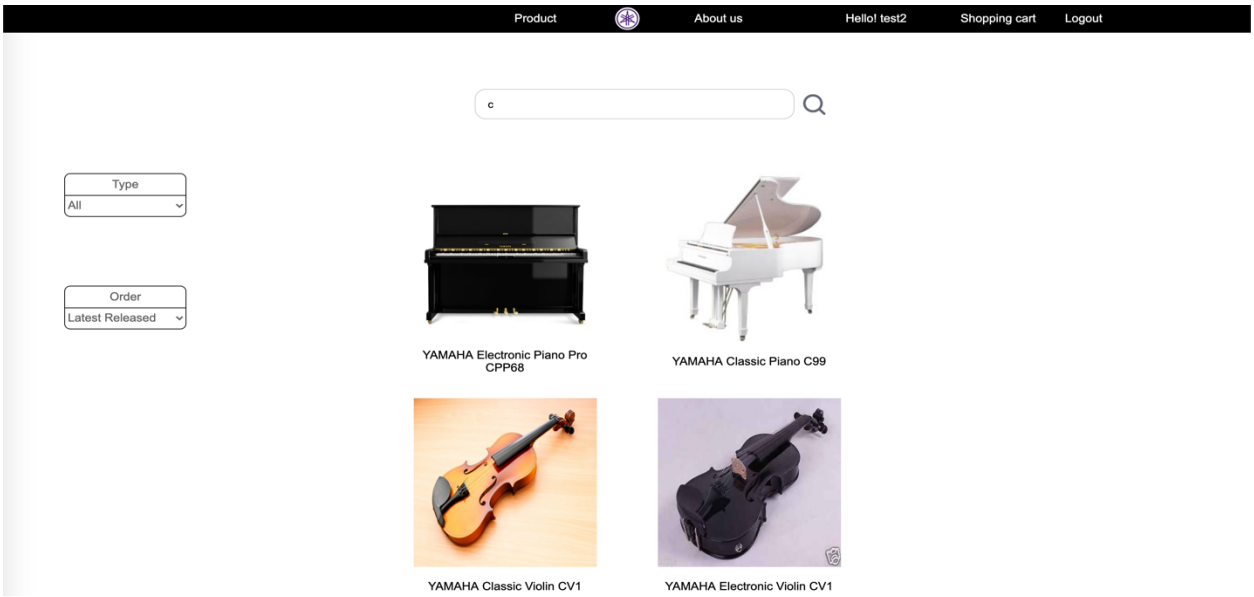

Figure.7. Search Page After Searching with Keyword "c" and Order "Latest-Released"

f. Product Detail Demostration

User can click the pictures of the searched products and look through the details. If user want to buy the product, user can select quantity and click the cart logo to add the specific numbers of product to the user's cart. Also, it will update user's cart in MySQL database. The page screenshot is shown in *Figure.8*.

Figure.8. Display of Product Details

g. Cart

After adding the fovorite products to the cart, user can goto the cart and purchase them. Click "BUY NOW" to purchase all items in the cart. The page screenshot is shown in *Figure.9*. The MySQL proving is shown in *Figure.10*.

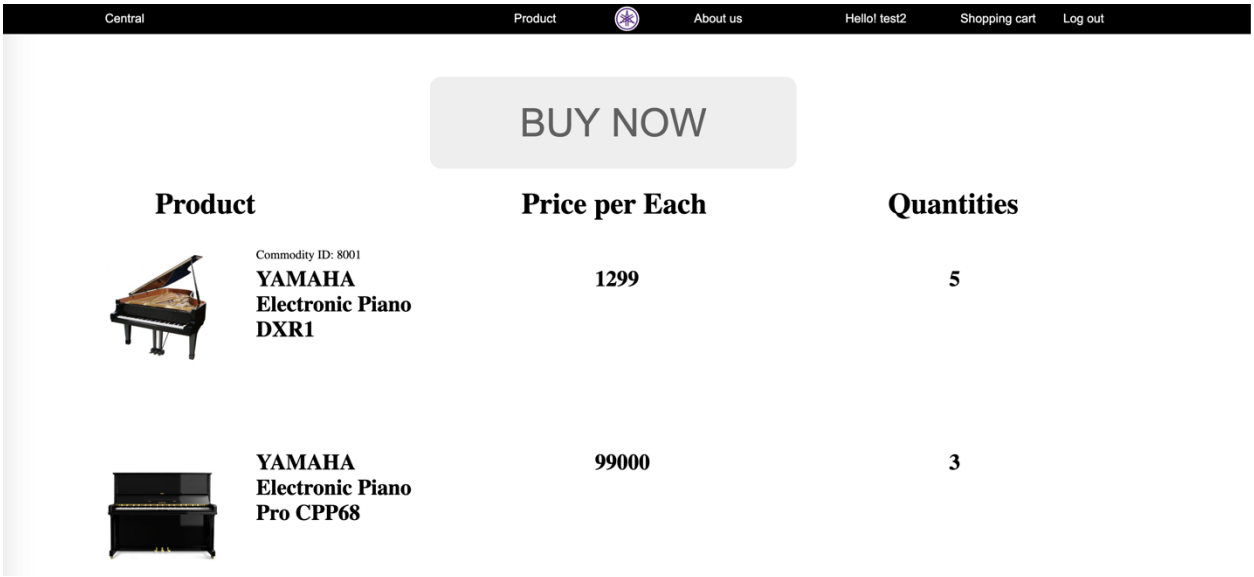

Figure.9. Cart Page

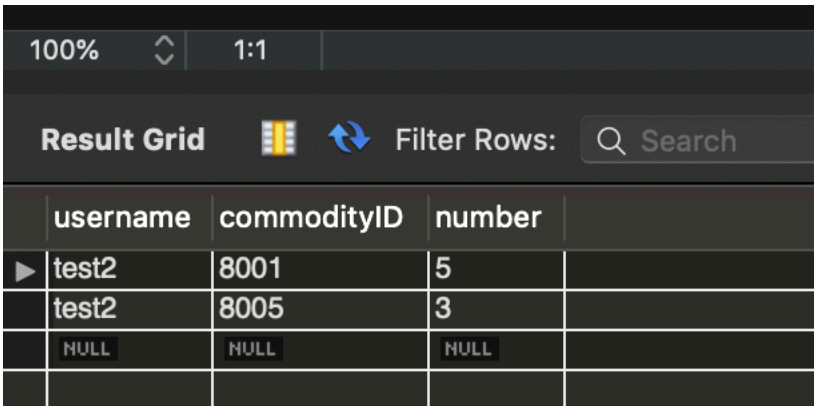

Figure.10. Cart Data in MySQL

h.Purchase Success and Total Price Display

The system will automatically calculate the total price and display. In the meantime, the system will automatically generate an order for the customer after successfully purchased. The page screenshot is shown in *Figure.11*. The MySQL proving is shown in *Figure.12*.

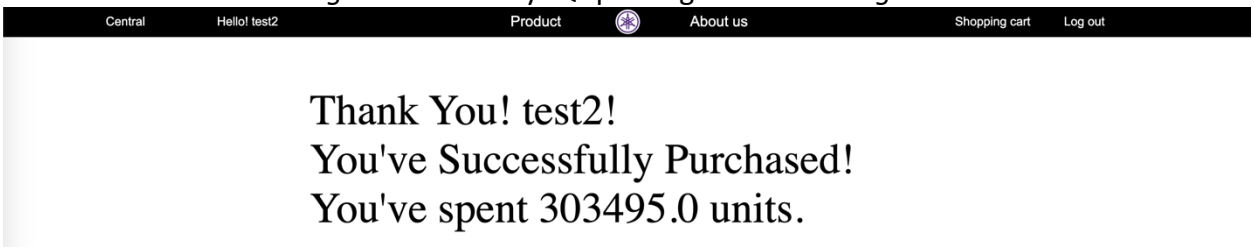

Figure.11. Purchase Success Page and Display the Total Price

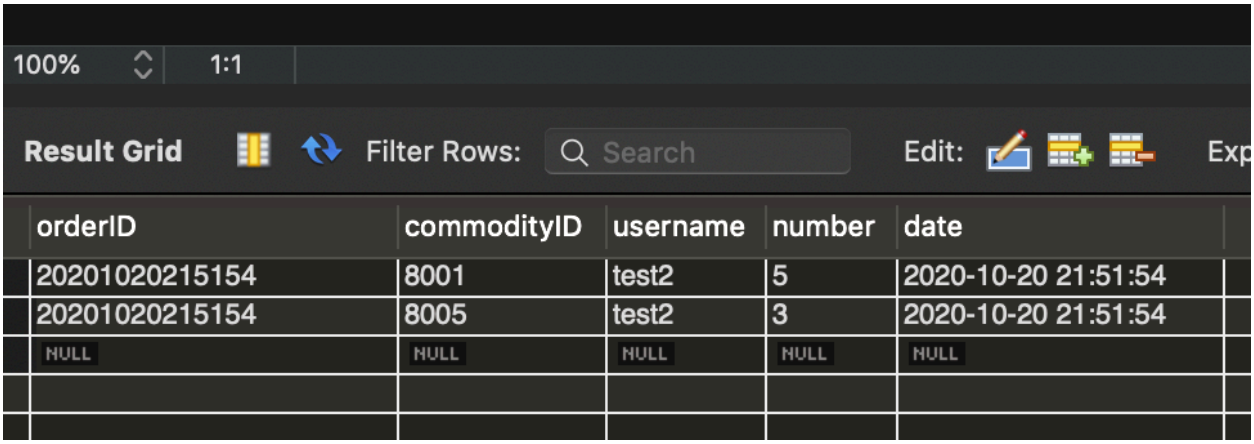

Figure.12. Automatically Generated Order in MySQL

3. Function Realization

In the D&B2, we adopted layering thought of development. In *Figure.13.*, it demonstrates the package structure and the naming method of class or interface where vo means Value Object, dao means Data Access Object, dao.impl means dao's implement, DB means Database Connection.

a. User Related

```java
package jspservlet.dao;

import jspservlet.vo.User;

public interface UserDAO {
    public int queryByUsername(User user) throws Exception;
    public int insertByUsername(User user) throws Exception;
    public int identyByUsername(User user) throws Exception;
    public User getAllUserInfo(User user) throws Exception;
    public int updatePassword(User user) throws Exception;
    public int updateInformation(User user) throws Exception;
}
```

Figure.14. UserDAO interface

Figure.15. and *Figure.16.* shows the methods of determination of validation of users' login information.

```java
public int queryByUsername(User user) throws Exception {
    // Check username and password if they match to the database.
    int flag = 0;
    String sql = "SELECT * FROM userinfo WHERE username=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getUsername());
        ResultSet rs = pstmt.executeQuery();
        if(rs.next())
        { // Checking password.
            if(rs.getString("password").equals(user.getPassword())) {
                flag = 1;
                }
        }
        rs.close();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}
```

Figure.15. Query User

```java
public int identyByUsername(User user) throws Exception
{// Checking if there has current username.
    int flag = 0;
    String sql = "SELECT * FROM userinfo WHERE username=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getUsername());
        ResultSet rs = pstmt.executeQuery();
        if(rs.next())
        {
                flag = 1;
        }
        rs.close();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}
```

Figure.16. Validation Determination

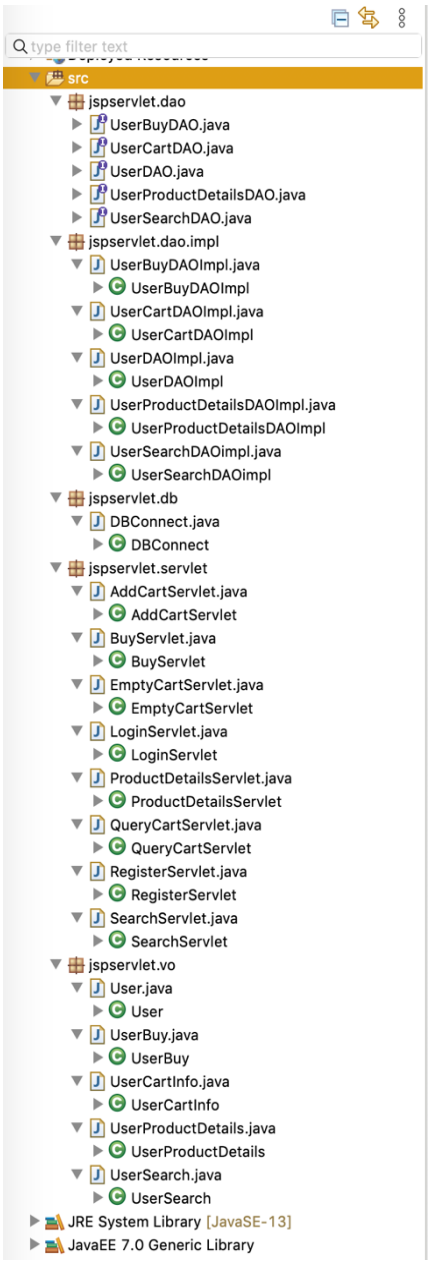

Figure.13. Package Structure and Naming

Figure.17. shows the method of insertion of new user. Figure.18. shows the method of updating users' information.

```java
public int insertByUsername(User user) throws Exception
{
    int flag = 0;
    int rs = 0;
    flag = this.identyByUsername(user);
    if(flag == 1)
        return flag;

    String sql = "Insert INTO userinfo VALUE(?,?,?,?,?)";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getPassword());
        pstmt.setString(2, user.getUsername());
        pstmt.setString(3, user.getBirthday());
        pstmt.setString(4, user.getFirstname());
        pstmt.setString(5, user.getLastname());
        rs = pstmt.executeUpdate();// Affected row numbers.
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return rs;
}
```

Figure.17. New User Insertion

```java
public int updatePassword(User user) throws Exception
{
    int flag = 0;
    String sql = "UPDATE userinfo SET password=? WHERE username=? AND password=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getNewpassword());
        pstmt.setString(2, user.getUsername());
        pstmt.setString(3, user.getPassword());
        flag = pstmt.executeUpdate(); // Affected row numbers.
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}

public int updateInformation(User user) throws Exception
{
    int flag = 0;
    String sql = "UPDATE userinfo SET firstname=?,lastname=?,birthday=? WHERE username=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getFirstname());
        pstmt.setString(2, user.getLastname());
        pstmt.setString(3, user.getBirthday());
        pstmt.setString(4, user.getUsername());
        flag = pstmt.executeUpdate();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}
}
```

Figure.18. Update Users' Information

b. Search Related

Search by content only is nested within search by search content, sort, and filter, and is not used alone.

```java
public ArrayList<String> searchContent(UserSearch content) throws Exception;
public ArrayList<String> searchFliter(UserSearch content, UserSearch sortMode, UserSearch fliter) throws Exception;
```

Figure.19. UserSearchDAO Interface

Search by search content, sort, and filter criteria.

Note: The content is too long, the screenshot is short, please see the original code for details.

Design idea: The search content entered by the user, the selected sort and filter conditions were obtained from the front end, and nested IF-else statements were used to determine how to use SQL statements to search the corresponding results from the library. Take StringBuffer and set special symbols to get and separate the search results and return them to the Servlet in the form of ArrayList<String>, where the different attributes of each search result are again separated and returned to the front end in turn.

c. Display of Product and Details Related

The design idea is basically the same as the design idea of search. The method is shown in *Figure.20.*

```
@Override
public String showProductDetails(UserProductDetails userproductdetails) throws Exception {
    // TODO Auto-generated method stub
    // Return particular product information.
    String commodityID = userproductdetails.getCommodityID();
    String sql = "SELECT * FROM commodity WHERE commodityID=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    String str = null;
    StringBuffer sb = new StringBuffer();

    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, commodityID);
        ResultSet rs = pstmt.executeQuery();
        sb.append(rs.getString("commodityID"));
        sb.append("*");
        sb.append(rs.getString("cname"));
        sb.append("*");
        sb.append(rs.getString("price"));
        sb.append("*");
        sb.append(rs.getString("introduction"));
        sb.append("*");
        sb.append(rs.getString("order_orderID"));
        sb.append("*");
        sb.append(rs.getString("store_storeID"));
        sb.append("*");
        sb.append(rs.getString("type"));
        sb.append("*");
        sb.append(rs.getString("addedDate"));
        sb.append("*");
        sb.append(rs.getString("picName"));
        sb.append("%"); // End of the row.
        str = sb.toString();
        rs.close();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }

    return str;
```

Figure.20. Display Product Details Method

d. Cart Related

```
public interface UserCartDAO {
    public int addToCart(UserCartInfo info) throws Exception;
    public int queryCart(UserCartInfo info) throws Exception;
    public ArrayList<String> currentCart(UserCartInfo info) throws Exception;
    public int resertCart(UserCartInfo info) throws Exception;
}
```

Figure.21. UserCartDAO Interface

In *Figure.22.* shows to display current cart items.

```
public ArrayList<String> currentCart(UserCartInfo info) throws Exception
{


    ArrayList<String> cartItem = new ArrayList<String>();
    cartItem.clear();

    StringBuffer sb = new StringBuffer();
    // Temporary string form data of all the matched info.
    String str = null;

    PreparedStatement pstmt = null;
    DBConnect dbc = null;

    try
    {
        dbc = new DBConnect();

        String sql = "SELECT * FROM cart NATURAL JOIN commodity WHERE username=?";
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, info.getUsername());

        ResultSet rs = pstmt.executeQuery();

        if(!rs.next()) { // No Item In Cart.
            cartItem.add("NoItemInCart");

        }
        else {
            rs.previous();// Back to the first row.
            while (rs.next()) {
                // Put search result behind the current result.
                // * and % are the marks for splitting.
                sb.append(rs.getString("commodityID"));
                sb.append("*");
                sb.append(rs.getString("cname"));
                sb.append("*");
                sb.append(rs.getString("price"));
                sb.append("*");
                sb.append(rs.getString("picName"));
                sb.append("*");
                sb.append(rs.getString("number"));
                sb.append("%"); // End of the row.
            }

            str = sb.toString();
            // Divided as encountered "%" and store in ArrayList
            String[] params = str.split("\\%");
            for(int i=0;i<params.length;i++) {
                cartItem.add(params[i]);
            }
        }
        rs.close();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return cartItem;
}
```

Figure.22. Display Current Cart

In *Figure.23.* shows adding to cart function.

```java
@Override
public int addToCart(UserCartInfo info) throws Exception {
    int flag = 0;
    String sql = null;
    PreparedStatement pstmt = null;
    DBConnect dbc = null;

    flag = this.queryCart(info); // If cart already has the item.
    System.out.println("flag = " + flag);
    System.out.println(info.getNumber() + info.getUsername() + info.getCommodityID() + "_____________________-");
    if(flag == 1) // Has
    {
        dbc = new DBConnect();
        if(info.getNumber() == 0)
        {
            sql = "DELETE FROM cart WHERE username=? AND commodityID=?";
            pstmt = dbc.getConnection().prepareStatement(sql);
            pstmt.setString(1, info.getUsername());
            pstmt.setString(2, info.getCommodityID());
        }

        else
        {
            sql = "UPDATE cart SET number=? WHERE username=? AND commodityID=?";
            pstmt = dbc.getConnection().prepareStatement(sql);
            pstmt.setInt(1, info.getNumber());
            pstmt.setString(2, info.getUsername());
            pstmt.setString(3, info.getCommodityID());
        }
        flag = pstmt.executeUpdate();
    }
    else // Do not have.
    {
        dbc = new DBConnect();

        if(info.getNumber() == 0)
        {// Do not add.
            flag = 1;
        }
        else
        {
            sql = "INSERT INTO cart VALUES(?,?,?)";
            pstmt = dbc.getConnection().prepareStatement(sql);
            pstmt.setString(1, info.getUsername());
            pstmt.setString(2, info.getCommodityID());
            pstmt.setInt(3, info.getNumber());

            boolean f = pstmt.execute(); // TRUE: execute query FALSE: execute update/delete/insert...
            System.out.println("++++++++"+f);
            if(!f)// Make sure it execute insert.
                flag = 1;
        }

    }
    pstmt.close();
    return flag;
}
```

Figure.23. Add to Cart

```java
public int queryCart(UserCartInfo info) throws Exception
{ // If it is currently existing.
    int flag = 0;
    String sql = "SELECT * FROM cart WHERE username=? AND commodityID=?"
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, info.getUsername());
        pstmt.setString(2, info.getCommodityID());
        ResultSet rs = pstmt.executeQuery();
        if(rs.next())
            flag = 1;
        rs.close();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}
```

Figure.24. Query the Existence in Cart

```java
public int resertCart(User user) throws Exception
{
    int flag = 0;
    String sql = "DELETE FROM cart WHERE username=?";
    PreparedStatement pstmt = null;
    DBConnect dbc = null;
    try
    {
        dbc = new DBConnect();
        pstmt = dbc.getConnection().prepareStatement(sql);
        pstmt.setString(1, user.getUsername());
        flag = pstmt.executeUpdate();
        pstmt.close();
    }catch(SQLException e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {
        dbc.close();
    }
    return flag;
}
```

Figure.25. Empty Cart

e. Purchase Related

```
public class UserBuyDAOImpl implements UserBuyDAO {

    @SuppressWarnings("null")
    @Override
    public double buyItem(UserBuy uci) throws Exception {
        // TODO Auto-generated method stub


        ArrayList<String> commodityID = new ArrayList<String>();

        double sum = 0;

        String currentTime= null;

        PreparedStatement pstmt = null, pstmt1 = null, pstmt3 = null, pstmt4 = null;
        DBConnect dbc = null;


        try
        {
            dbc = new DBConnect();

            String sql2 = "SELECT * FROM cart WHERE username=?" ;
            pstmt = dbc.getConnection().prepareStatement(sql2);
            pstmt.setString(1, uci.getUsername());
            StringBuffer sb = new StringBuffer();
            // Temporary string form data of all the matched info.
            String str = null;
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                // Put search result behind the current result.
                // * and % are the marks for splitting.
                sb.append(rs.getString("commodityID"));
                System.out.println(rs.getString("commodityID"));
                sb.append("%"); // End of the row.
                str = sb.toString();
                System.out.println("str in buyImpl::::::" + str);
            }
            // Divided as encountered "%" and store in ArrayList
             String[] params = str.split("\\%");
             commodityID.clear();
                for(int i=0;i<params.length;i++) {
                    commodityID.add(params[i]);
                }
            rs.close();

            if(!commodityID.get(0).equals("EmptyCart")) { // Not empty.
                for(int i = 0; i < commodityID.size(); i++) {
                    String sql3 = "SELECT * FROM commodity WHERE commodityID=?" ;
                    pstmt3 = dbc.getConnection().prepareStatement(sql3);
                    System.out.println("))))))))))))))"+commodityID.get(i));
                    pstmt3.setString(1, commodityID.get(i));
                    ResultSet rs1 = pstmt3.executeQuery();
                    double tempPrice = 0;
                    if(rs1.next())
                    tempPrice = Double.parseDouble(rs1.getString("price"));


                    String sql4 = "SELECT * FROM cart WHERE username=? AND commodityID=?" ;
                    pstmt4 = dbc.getConnection().prepareStatement(sql4);
                    pstmt4.setString(1, uci.getUsername());
                    pstmt4.setString(2, commodityID.get(i));
                    int tempnumber = 0;
                    ResultSet rs2 = pstmt4.executeQuery();
                    if(rs2.next())
                    tempnumber = Integer.parseInt(rs2.getString("number"));

                    sum = sum + tempnumber*tempPrice; //!!!!!
                    System.out.println("SUM::::::::::::::::" + sum);

                    String sql1 = "INSERT INTO `order` VALUES(?,?,?,?,?)";
                    pstmt1 = dbc.getConnection().prepareStatement(sql1);

                    SimpleDateFormat df0 = new SimpleDateFormat("yyyyMMddHHmmss");// Set format.
                    String currentTime0 = df0.format(new Date());
                    SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");// Set format.
                    currentTime = df.format(new Date());
                    String tempernum = String.valueOf(tempnumber);
                    pstmt1.setString(1, currentTime0);
                    pstmt1.setString(2, commodityID.get(i));
                    pstmt1.setString(3, uci.getUsername());
                    pstmt1.setString(4, tempernum);
                    pstmt1.setString(5, currentTime);

                    boolean f = pstmt1.execute();

                    rs1.close();
                    rs2.close();
                    pstmt3.close();
                    pstmt4.close();
                }


                pstmt1.close();
                pstmt.close();
            } else {
                sum = 0;
            }

        }catch(SQLException e) {
            System.out.println(e.getMessage());
        }
        finally
        {
```

Figure.26. Purchase Method

In *Figure.26.,* users can buy the goods in the shopping cart and calculate the total price. Design idea: First, get the ID and quantity of the corresponding user's goods on the cart from the shopping cart database, then get the commodity price from the commodity database, convert String into double type and calculate the total price.

4. Runtime Environment

OPERATING SYSTEM

MACOS MOJAVE VERSION 10.14.4

JAVA

JDK 13

IDE

MYECLIPSE ENTERPRISE WORKBENCH

VERSION: 2020.5.18

MYSQLWORKBENCH 8.0

VERSION 8.0.19

BUILD 15713499 CE (64 BITS)

SERVER

MYECLIPSE TOMCAT V8.5