



SCHOOL OF ADVANCED TECHNOLOGY
SAT301 FINAL YEAR PROJECT

Property Management System

Final Thesis

In Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Engineering

Student Name	:	Shen Wang
Student ID	:	1824260
Supervisor	:	Kok Hoe Wong
Assessor	:	Soon Phei Tin

Abstract

In the final year, the developer independently designed and developed a property management system based on web application mode. The project uses native web development techniques and the Spring Boot framework. It consists of a user version and an administrator version, which are closely related and serve as platforms for communication. The features include account management, resident management, notice management and message management. For security, access control is performed using sessions.

Due to design defects and technical limitations, the security guarantee is insufficient. In the future, the UI and algorithm framework can be redone with new technologies in combination with the native application to improve non-functional features.

Acknowledgements

I would like to thank my supervisor, Kok Hoe Wong, for his guidance through each stage of the process.

I would like to thank the assessor, Soon Phei Tin, for inspiring my interest in software engineering.

Contents

Abstract	ii
Acknowledgements	iii
Contents	iv
List of Tables	v
List of Figures	vi
List of Acronyms	viii
Chapter 1 Introduction	1
1.1 Motivation, Aims and Objective	1
1.2 Literature Review	3
Chapter 2 Methodology and Results	5
2.1 Methodology.....	5
2.2 Results	7
Chapter 3 Conclusion and Future Work	34
3.1 Conclusion.....	34
3.2 Future Work.....	34
References	36
Appendix A. Study Case: Huihutong.....	38
Appendix B. Code Files Category	45

List of Tables

Table 1 Browser Side Supports.....	8
Table 2 SpringBoot pom.xml.....	8
Table 3 Environment.....	33
Table 4 PC Configuration	33

List of Figures

Figure 1 Plan Schedule	2
Figure 2 Develop Schedule	6
Figure 3 Database File Structure.....	9
Figure 4 Server File Structure.....	9
Figure 5 Browser File Structure.....	9
Figure 6 File Structure	10
Figure 7 User Login	10
Figure 8 Administrator Login	11
Figure 9 ER diagrams for user and admin	11
Figure 10 User Personal Profile.....	13
Figure 11 User Reset Password	14
Figure 12 Administrator Personal Profile	14
Figure 13 Query Area	17
Figure 14 Query Result.....	17
Figure 15 User Management.....	18
Figure 16 User Management: Search.....	18
Figure 17 User Management: Add User	19
Figure 18 Publish Notice	21
Figure 19 Notice table.....	21
Figure 20 Notice Content.....	22
Figure 21 Notice for Users.....	22

Figure 22 ER Diagram for Notice.....	23
Figure 23 User Message: New Message	25
Figure 24 User Message: Received Message.....	26
Figure 25 User Message: Sent Message	26
Figure 26 Administrator Message: New Message	27
Figure 27 Administrator Message: Received Message.....	27
Figure 28 Administrator Message: Sent Message	28
Figure 29 ER Diagram for Notice.....	28
Figure 30 Use Case Diagram	31

List of Acronyms

AJAX	Asynchronous Javascript And XML
CSS	Cascading Style Sheets
HTML	HyperText Mark-up Language
JDBC	Java Database Connectivity
JS	JavaScript
UI	User Interface

Chapter 1 Introduction

1.1 Motivation, Aims and Objective

1.1.1 Description

The project is an individual, non-sponsored software development final year project which was designed and developed by a senior student in the department of Information and Computing Sciences.

The theme of the project is to design and implement a property management system. The project is designed to be a web application with separate front and back ends, and has two versions for users and administrators. In that case, the system is not only a management system, but also a communication platform between users and staffs.

1.1.2 Motivation

With the popularity of personal computers and mobile devices, many property management companies already use softwares to manage houses and homeowners, or to help owners with their homes. However, these management systems often only have an administrator version or a user version. This means that a company needs to use multiple software to meet different functional requirements and platform requirements.

The Huihutong is a study case for typical property management system. For users, this property company has an official website for posting relative news, introducing itself and recruiting staff. There is also a WeChat official account with the same name as the company, which is used to post notices to owners. In addition, this company has an official account called ihuihu to provide residents with features such as water and electricity bill top-ups and repair requests. After a few years, these features were changed to official accounts with the same names as the communities for which the company was responsible. In recent years, a new mini program named weihuihu has been developed to host these functions and new features for resident access management due to the appearance of

COVID-19. In addition to the traditional office system, the company had to manage each of the different user-facing property systems separately.

This case shows the problems associated with a property management system that is not well connected to its administrators and users. In order to adapt to different platforms, such as official accounts and mini programs, the company had to develop multiple versions of the system, which was troublesome for both users and employees.

Considering this situation, the design and implementation of an associated web application with two versions for administrators and users is in line with market trends and has many benefits.

1.1.3 Aim

The aim of the project is to design and develop a property management system on an individual basis. After some research and thought, the project will be a web application divided into an administrator version and a user version.

1.1.4 Objectives

According to the timetable of SAT301, the project was designed during the first semester and realized during the winter holidays and the second semester. The following shows the plan of the development.



Figure 1 Plan Schedule

The work of semester 1 was the same as the plan shown in the figure. In semester 2, more time was spent on the project in view of the extension of the deadline.

The system can be divided into two parts, a management system for staffs and a web application for residents.

For the administrators, there are features:

- Account Management: Login and Personal Profile
- Notice management
- User management
- Message management: New Message, My Received and My Sent

For the user, there are features:

- Account Management: Login, Profile and Password Reset
- Notice
- Query Other Users
- Message: New Message, My Received and My Sent

When originally designed, the administrator's property management system had a number of other features. For example, the office system and schedule management were designed to help employees with their work. However, these functions were modified for a more sensible design.

1.2 Literature Review

With the popularity and use of personal computers and mobile devices, software in various fields has been developed in response to market demand to provide services to administrators and users. For the convenience of property companies and community residents, property management systems are designed and developed to suit different devices and platforms, providing different features for different requirements.

Researchers have focused in different directions. A proportion of contemporary developers are dedicated to designing and implementing marketable projects. They are focused on realizing property management systems using different development approaches and development platforms. Their softwares can be adapted to different devices and combined with diverse external infrastructures to provide services. Zhou, Zheng and Wang [1] had designed and implemented a property management system based on Asterisk, an open-

source communication platform. The project was developed with VoIP soft switch and many technologies, and it achieved community voice intercom, accessing control management, video transmission and processing of community security alarm information. Yao and Gu [2] focused on designing and analyzing databases with Delphi, a rapid application development tool. Tian [3] also designed and implemented a project using the same development platform. Mu and Li [4] developed a property management system based on B/S mode, which is similar to this project.

There is also a part of researchers who try to combine advanced technology with property management. Limited by infrastructure, these results are difficult to apply now, but are promising in the future. Li [5] focused on WebGIS, the result achieved expansions with new functions such as geographical data processing and security alarm.

Due to the limitations of personal ability, this project will not include all the features that users need. However, in the future, it can be combined with advanced technology and become more comprehensive, if the opportunity arises.

Chapter 2 Methodology and Results

2.1 Methodology

The process of the project can be divided into two parts, software design and web application development. The methodologies applied in the design and development will be described here.

2.1.1 Requirement Engineering: Case Study

Requirements engineering is the process of finding out stakeholder needs and summarizing them into detailed requirements that can serve as a basis for all subsequent development activities [6]. The first and most important thing to do when designing a project is to carry out requirements engineering to determine the standards of the system and to use it to plan the features of the project.

The requirement engineering of this project was mainly completed with case studies. By studying different cases, the project requirements that should be available were analysed qualitatively. There will be no specific analysis of each study case, rather a more direct description of the research findings. A case named Huihutong will be shown in the appendix A, which was also mentioned in the introduction.

Firstly, account management is a feature that almost every software contains. The system identifies the current user by the account and provides individual services. Both of two versions of property management system has login and profile features. There is no register and reset password features for administrators. However, a user account can be added by an administrator at the user management area in administrator version. The design of this feature is informed by many other management systems. The user's information is presented visually in a table. In addition, a user can reset the password independently in user version. For the profile, both staffs and owners can modify and save their personal information. The information will be shown if other users query their neighbour. This design was proposed to ensure the security of the account in reality.

The user management feature in the administrator version was design

The notice features are widely presented in many property management systems. For example, in the study case in the appendix, news areas appear both in the website and WeChat mini program. It is necessary that users can know news and notices of the community through an online platform.

As for the message features, it is available in many websites. Users and staffs can communicate with each other freely. In fact, notice management, query user and message management features all have similar communication effects.

2.1.2 Development: Agile Method

According to the time requirement of this project, it was mainly designed in semester 1 and realized in semester 2. The develop method used in this project is mainly similar to the agile method. For the benefits, agile development allows software to be released in iterations that developer can find and fix defects and align expectation early on [7]. Furthermore, it is a familiar development method that I have used before.

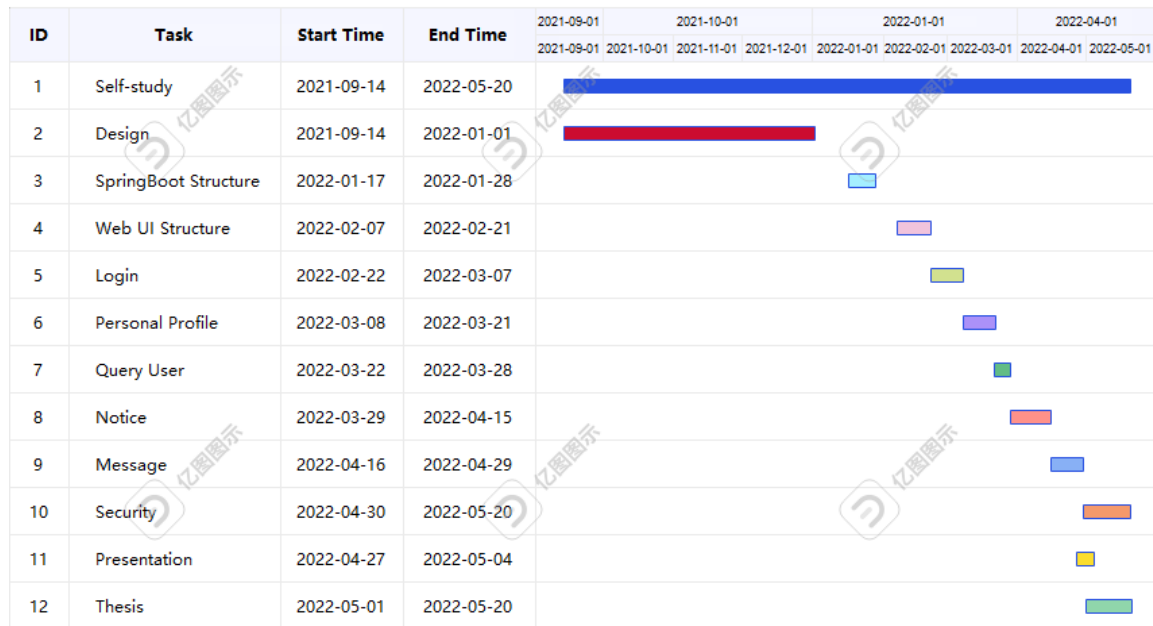


Figure 2 Develop Schedule

The figure above shows the real develop schedule. Self-study continues from the start of the design to the end of the development. After the design, the browser and server program frameworks had been built since winter holiday. Then the functional requirements were realized one by one. The development of each features lasted about one to three weeks depending on how difficult it is.

In each round of development, the programming logic is analyzed and the UI is designed first. Then develop the browser-side and server-side codes and run, test and modify it constantly. These tests and modifications are also iterated over and over until the desired effect is achieved. Finally, I have to develop some non-functional requirements to improve the quality of the project. For example, add security features to the system after all features are developed.

The advantage of using agile method is that errors can be found and debugged in time during continuous iterative development and testing. This approach keeps the entire development process in order and is best suited to the situation.

2.2 Results

This section will focus on the logic and results associated with the implementation of the features, and the relevant code files directories will be listed in the appendix B.

2.2.1 Framework

This web application property management system's browser-side code is composed of HTML, CSS and JS, and the server-side uses SpringBoot framework of Java. The browser and server transfer data through AJAX with jQuery's GET and POST methods. The browser request data and performs operations through the server's interfaces. Since this is a software development project, all testing is done by running the software directly.

The following are support libraries used for browser side:

CSS	JS
Bootstrap	Bootstrap
	jQuery
	TinyMCE

Table 1 Browser Side Supports

Bootstrap is an open-source toolkit from Twitter for front-end development. It is a CSS/HTML framework. jQuery is a fast, concise JavaScript framework code library. TinyMCE [8] is an open-source rich text editor API, which is used for message sending and notice publishing.

The developed web pages have window size adaptive capability. User and administrator versions' UI pages have similar structure. In fact, each set of pages have same headers, footers and navigation bars that link to other features. The different features of each page are written in a container in the body of the page.

The following are dependencies in the SpringBoot frame:

groupId	artifactId
org.projectlombok	lombok
org.springframework.boot	spring-boot-starter-thymeleaf
org.springframework.boot	spring-boot-starter-web
org.springframework.boot	spring-boot-starter-test
org.mybatis.spring.boot	mybatis-spring-boot-starter
org.springframework.boot	spring-boot-starter-jdbc
mysql	mysql-connector-java
com.alibaba	druid
log4j	log4j
com.alibaba	fastjson
org.json	json
org.springframework.session	spring-session-data-redis

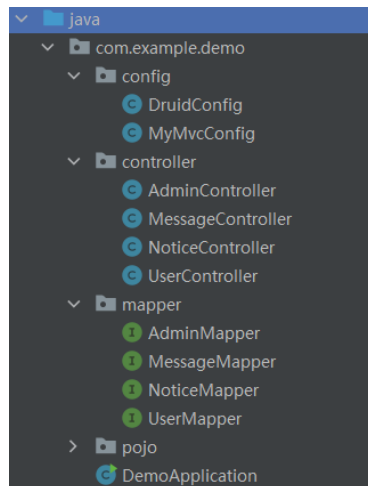
Table 2 SpringBoot pom.xml

MySQL is the database, and it is connected with JDBC and MyBatis. log4j is used to control Java log information. Druid is a JDBC component. Here are the 4 tables in the database.



Figure 3 Database File Structure

The following shows the file structure of the project.

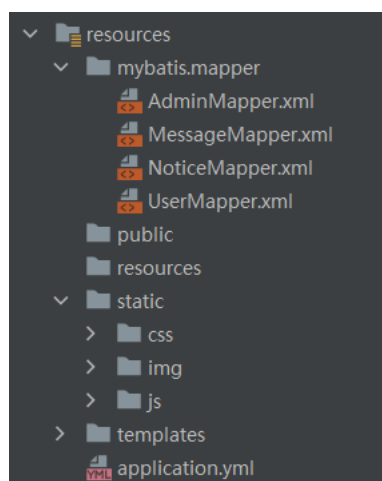


The config folder contains configuration files. MyMvcConfig.java configures the registry of view controller. Only the web pages registered here can be accessed.

The controller folder contains service controllers. The data interfaces and the operation algorithms are here.

The files in mapper, pojo and mybatis.mapper folders work together to realize functions with database operations.

Figure 4 Server File Structure



As mentioned above, the .xml files in mybatis.mapper folder are database operation configuration files.

All static browser codes such as images, CSS and JS files are in the static folder.

All HTML files are in the templates folder. These files show web pages through browser.

The application.yml contains the server configuration, such as port numbers and database properties.

Figure 5 Browser File Structure

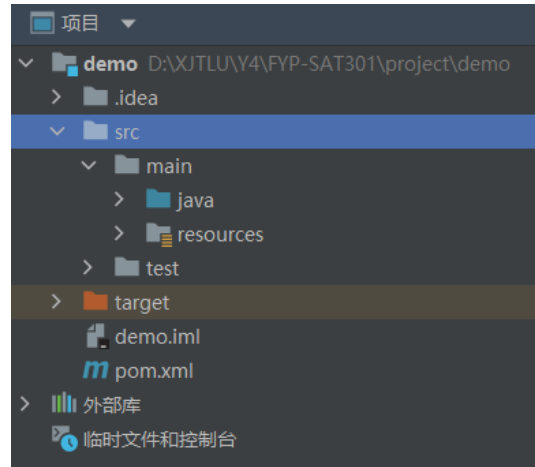


Figure 6 File Structure

Object classes are defined in pojo, Java files in mapper define function interfaces, and XML files in mybatis.mapper define operations on the database. These configured functions are called in controller to perform operations on the data interface.

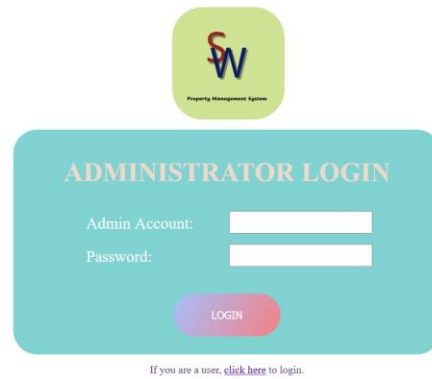
2.2.2 Account Management: Login, Personal Profile and Logout

The account management includes login, personal profile and password reset for users. Users and administrators can login to the system at <http://localhost:8080/userLogin.html> or <http://localhost:8080/adminLogin.html>

Here are the login pages for users and staffs.



Figure 7 User Login



The image shows a web interface for an administrator login. At the top center is a green circular logo with a stylized 'S' and 'W' and the text 'Property Management System' below it. Below the logo is a light blue rounded rectangle containing the title 'ADMINISTRATOR LOGIN' in orange. Underneath the title are two white input fields: 'Admin Account:' and 'Password:'. Below these fields is a red 'LOGIN' button. At the bottom of the blue rectangle, there is a small link: 'If you are a user, [click here](#) to login.'

Figure 8 Administrator Login

Database ER diagram for table *user* and *admin*:

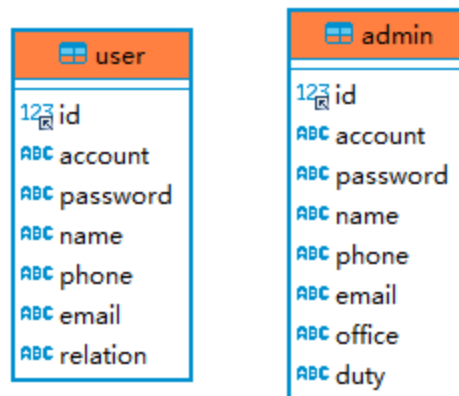


Figure 9 ER diagrams for user and admin

Id is a non-empty increment index. The system can identify the current user or administrator with the unique id.

The account and password are used to check identity and login to the system. Other properties are the user's or administrator's personal information.

When the user or administrator clicks the login button, the browser first checks whether the account and password have been entered.

The following are the JS codes of login button clicked event.

```
1. function checkEmpty(obj) {
2.     if (obj == "") {
3.         return false;
4.     } else {
5.         return true;
6.     }
7. }
8.
9. function userLoginSubmit() {
10.     var account = document.getElementById("userAccount").value;
11.     var password = document.getElementById("userPassword").value
12.     ;
13.     if (checkEmpty(account) && checkEmpty(password)) {
14.         var btnSubmit = document.getElementById("loginButton");
15.         btnSubmit.disabled = "true";
16.         $.post(
17.             url = "/userLogin",
18.             {
19.                 "account": account,
20.                 "password": password
21.             },
22.             function (data) {
23.                 if (data == 0) {
24.                     alert("Please correct the account.");
25.                     btnSubmit.removeAttribute("disabled");
26.                 } else if (data == 1) {
27.                     console.log("Login Successfully.");
28.                     window.location.href="user-index.html";
29.                 } else {
30.                     alert("Please correct the password.");
31.                     btnSubmit.removeAttribute("disabled");
32.                 }
33.             },
34.             type = "text"
35.         )
36.     } else {
37.         alert("Please fill the blank.");
38.         btnSubmit.removeAttribute("disabled");
39.     }
```

If there is no blank space, the browser sends the data to the server, which queries the database to see if the account exists. If the account exists and the password is correct, the id of the current user will be stored in the session. If the account or password is incorrect,

the browser will prompt user or administrator to correct it. After a successful login, the browser redirects to the home page of the user or administrator version.

The following is the server interface and function of login feature.

```
1. @PostMapping("/userLogin")
2. public int userLogin(@RequestParam("account") String account , @R
   equestParam("password") String password){
3.     User user = userMapper.getUserByAccount(account);
4.     if (user == null){
5.         return 0;
6.     } else if (user.getPassword().equals(password)){
7.         HttpSession session = request.getSession();
8.         session.setAttribute("userId",user.getId());
9.         return 1;
10.    } else {
11.        return 2;
12.    }
13. }
```

Here are the personal profiles of user and administrator:

Property Management System User
User Version

Notices Friends Message Personal Information

User Id: 1
House: A010101
Name:
Phone:
Email:
Relation:

Tips:
Here is a final year project: Property management system.
Designed and developed by Shen Wang.
Email: shen.wang18@student.qjtu.edu.cn

Figure 10 User Personal Profile

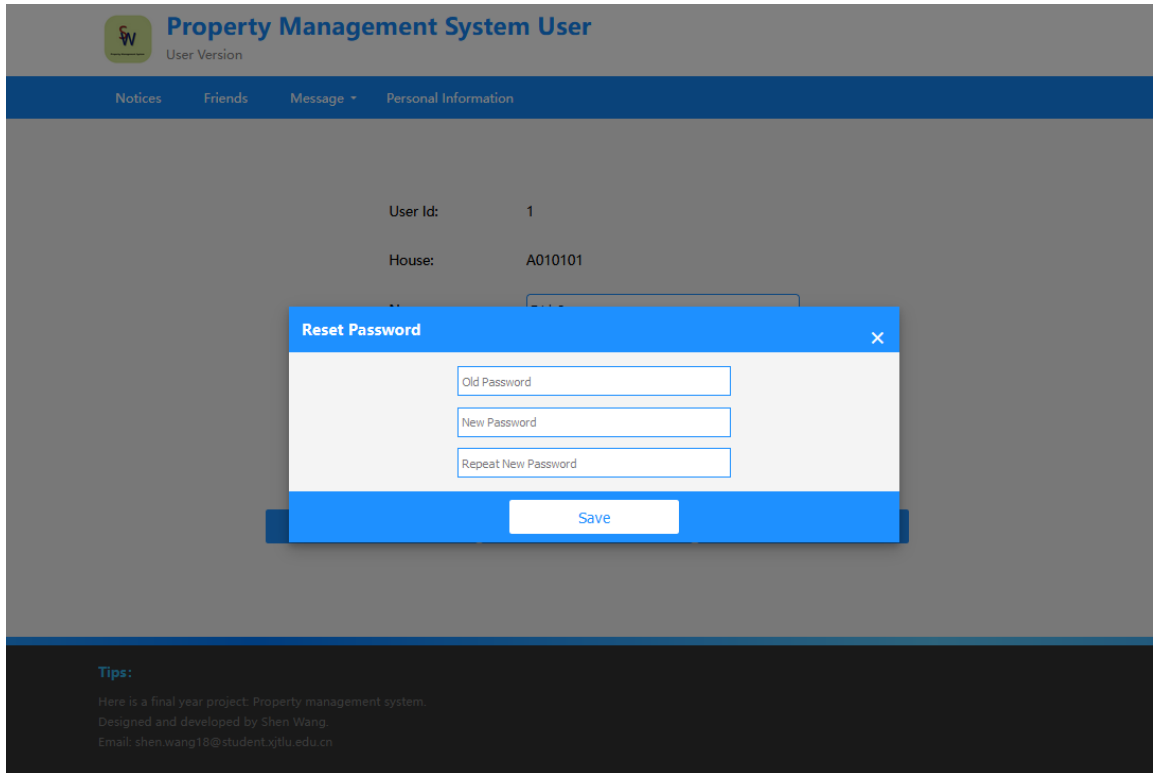


Figure 11 User Reset Password

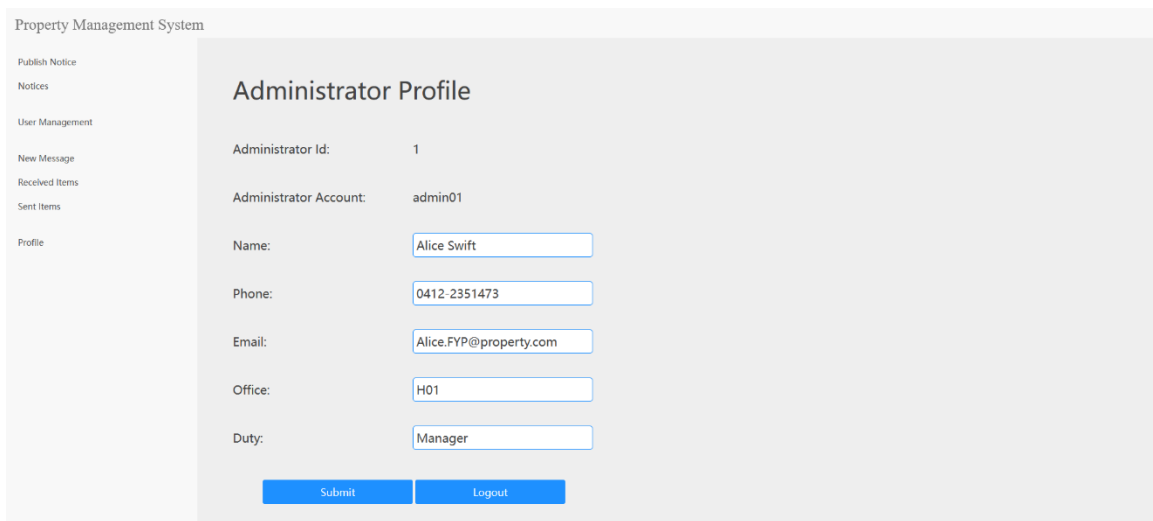


Figure 12 Administrator Personal Profile

As shown in figure 10 and 12, the profile page requests data from the server at loading time. The server reads the current user id in the session and selects personal information based on the id. When the page is loaded, the data will be displayed.

The following is the JS that request data from the server. The function will be triggered when the HTML <body> is onload.

```
1. function loaded() {
2.     $.get(
3.         url = "/getUserInfo",
4.         function (data) {
5.             data = JSON.parse(data);
6.             var id = document.getElementById("id");
7.             id.innerHTML = data.id;
8.             var house = document.getElementById("account");
9.             house.innerHTML = data.account;
10.            var name = document.getElementById("name");
11.            name.value = data.name;
12.            var phone = document.getElementById("phone");
13.            phone.value = data.phone;
14.            var email = document.getElementById("email");
15.            email.value = data.email;
16.            var relation = document.getElementById("relation");
17.            relation.value = data.relation;
18.        },
19.        type = "text"
20.    )
21. }
```

The following is the interface that load personal information.

```
1.     @GetMapping("/getUserInfo")
2.     public JSONObject userInfo() {
3.         int userId = Integer.parseInt(String.valueOf(request.getSession().getAttribute("userId")));
4.         User user = userMapper.getUserById(userId);
5.
6.         JSONObject json = new JSONObject();
7.         json.put("id", user.getId());
8.         json.put("account", user.getAccount());
9.         json.put("password", user.getPassword());
10.        json.put("name", user.getName());
11.        json.put("phone", user.getPhone());
12.        json.put("email", user.getEmail());
13.        json.put("relation", user.getRelation());
14.        return json;
15.    }
```

If the user wants to modify the information, just simply enters the new information in the input fields, then clicks the submit button, and the modified information will be sent to the database. The operations of the administrator and the user are the same.

As Figure 11, the user can change the password. When the button is clicked, a pop-up window appears and user needs to enter the old password, then set a new password and repeat it. When user clicks Save button, the browser checks for blank space and compares whether the two new passwords are the same. If it passes, the old password and the new password will be sent to the server. Then server checks whether the old password is correct. If correct, the user's password can be successfully changed. If an error is encountered, the page displays an error message, such as "Please fill blanks.", "Please repeat new password correctly." or "Please correct your old password."

The following is the interface to change password.

```
1.      @PostMapping("/userPasswordReset")
2.      public int resetuserPassword(@RequestParam("pwd1") String pwd
1  , @RequestParam("pwd2") String pwd2, @RequestParam("pwd3") Stri
ng pwd3){
3.          int userId = Integer.parseInt(String.valueOf(request.getS
ession().getAttribute("userId")));
4.          User user = userMapper.getUserById(userId);
5.          if(!Objects.equals(pwd1, user.getPassword())){
6.              return 1;
7.          }else {
8.              user.setPassword(pwd2);
9.              userMapper.updateUser(user);
10.             return 0;
11.         }
12.     }
```

The logout feature is also on this page. When the user or administrator clicks the Logout button, the current id in the session will be deleted and the browser redirects to the corresponding login page.

The following is the interface to logout and remove session data.

```
1. @GetMapping("/userLogout")
2.     public int userLogout(){
3.         request.getSession().removeAttribute("userId");
4.         return 0;
5.     }
```


2.2.3 User Management: Query and User List

A user can query information of neighbors in the “Friend” page.

Property Management System User
User Version

Notices Friends Message Personal Information

Type user here Search

Tips:
Here is a final year project: Property management system.
Designed and developed by Shen Wang.
Email: shen.wang18@student.xjtu.edu.cn

Figure 13 Query Area

Property Management System User
User Version

Notices Friends Message Personal Information

a01 Search

Account	Name	Phone	Email	Relation
A010101	Erick Su	0512-1111111	erick.smith@163.com	House Owner
A010814	Mary White	0533-2647823	hahahaha@gmail.com	Renter

Tips:
Here is a final year project: Property management system.
Designed and developed by Shen Wang.
Email: shen.wang18@student.xjtu.edu.cn

Figure 14 Query Result

Users can enter the keyword of the neighbor's room number that they want to query. When the search button is clicked, the keyword will be sent to the server, according to which the server conducts fuzzy query.

Codes in UserController.java as following.

```
1. @PostMapping("/queryUserList")
2.     public List<User> queryUserList(@RequestParam("acc") String a
   cc) {
3.         List<User> userList = userMapper.queryUserByAccount(acc);
4.         return userList;
5.     }
```

Codes in UserMapper.xml as following.

1. `<select id="queryUserByAccount" resultType="User" parameterType="String">`
2. `select * from user where locate("#{acc}", account) > 0;`
3. `</select>`

The table showing the data is hidden. When the server returns the queried data, it will be filled into the table and displayed. The information includes account, name, phone, email and relation to the house.

UIs for administrators to management users:

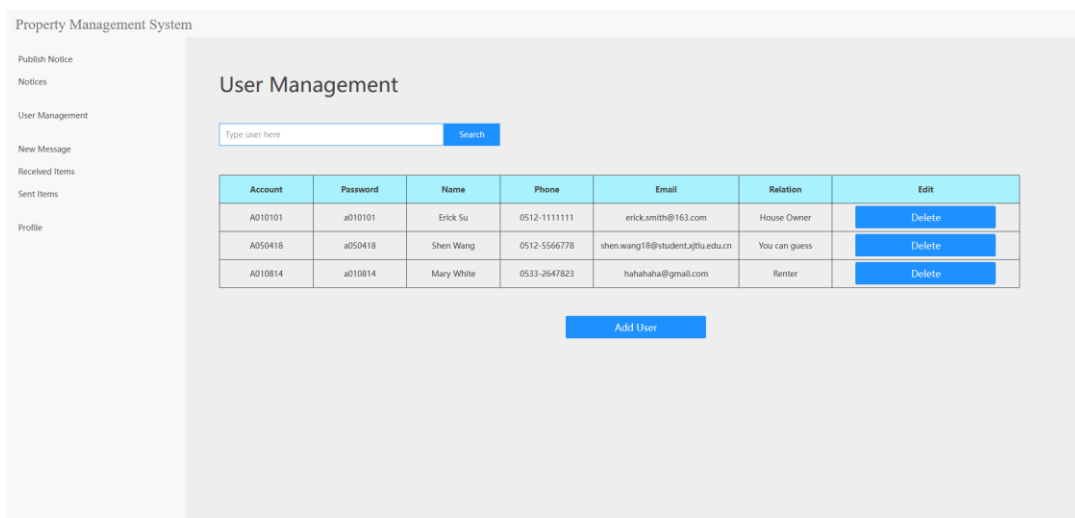


Figure 15 User Management

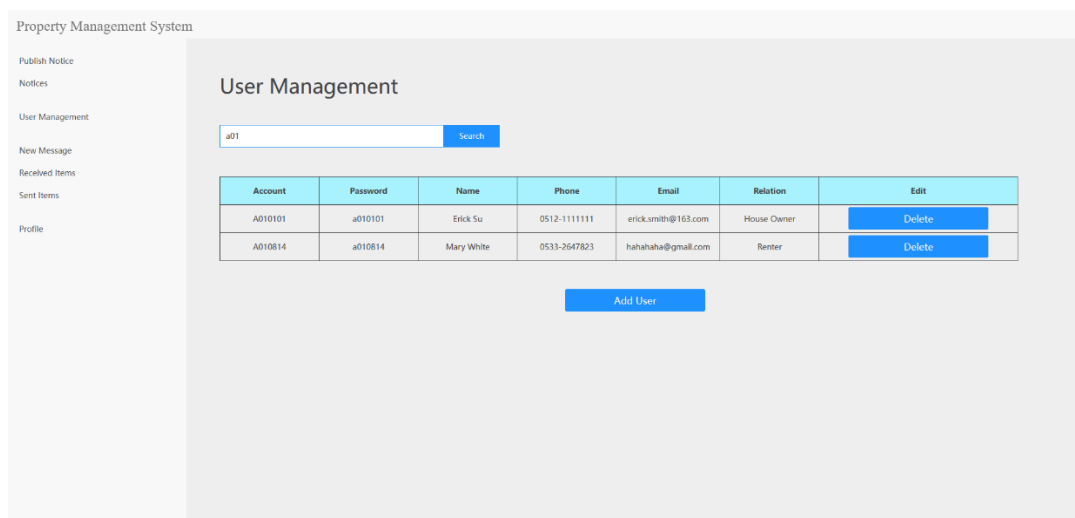


Figure 16 User Management: Search

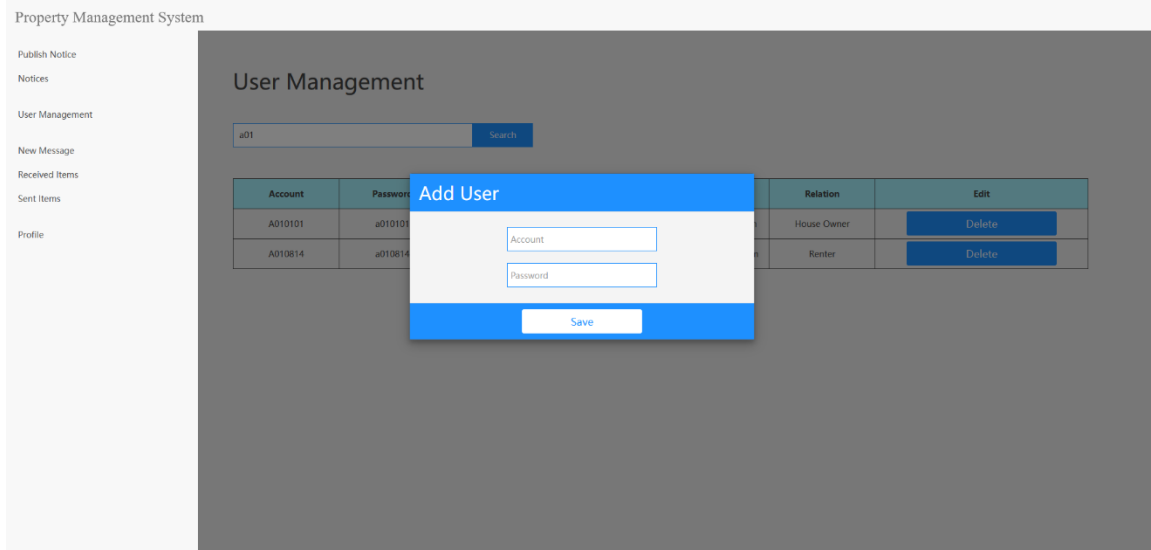


Figure 17 User Management: Add User

When the page loads, all user data is requested and rendered in a table. Administrators can delete or add users.

When the delete button is clicked, the browser sends the id of the user to be deleted to the server, which deletes the user's data from the database based on the id.

When the add user button is clicked, a pop-up window appears that allows the administrator to enter a new account and password.

The following is the script that controls the pop-window.

```

1. var modal = document.getElementById("simpleModal");
2. var modalBtn = document.getElementById("modalBtn");
3. modalBtn.addEventListener("click", openModal);
4. window.addEventListener("click", outsideClick);
5. function openModal() {
6.     modal.style.display = "block";
7. }
8. function outsideClick(e) {
9.     if (e.target == modal) {
10.         modal.style.display = "none";
11.     }
12. }
13. function closeModal() {
14.     modal.style.display = "none";
15. }

```

The data is then sent to the server, where the account is first checked to see if it already exists. If the account exists, the browser will remind the administrator to change the account based on the feedback from the server. If the same account does not exist in the database, the user can be added to the user table.

Codes in UserController.java as following.

```
1.      @PostMapping("/addUser")
2.      public int addUser(@RequestParam("acc") String acc, @RequestParam("pass") String pass) {
3.          User checkUser = userMapper.getUserByAccount(acc);
4.          if (checkUser == null) {
5.              User user = new User();
6.              user.setAccount(acc);
7.              user.setPassword(pass);
8.              userMapper.addUser(user);
9.              return 0;
10.         } else {
11.             return 1;
12.         }
13.     }
```

Codes in UserMapper.xml as following.

```
1. <select id="getUserByAccount" resultType="User" parameterType="String">
2.     select * from user where account = #{account};
3. </select>
```

In addition, administrators can enter keywords to query users vaguely in the database. When the search button is clicked, the keyword is sent to the server, where the user's account is fuzzily queried. The queried user data will be returned and displayed in a table.

2.2.4 Notice Management

Here are UIs for the administrators:

Property Management System

Publish Notice
Notices
User Management
New Message
Received Items
Sent Items
Profile

Publish Notices

Title: News

File Edit View Format

↶ ↷ Paragraph ▼ **B** *I*

POWERED BY Tiny

Save

Figure 18 Publish Notice

Property Management System

Publish Notice
Notices
User Management
New Message
Received Items
Sent Items
Profile

Notices List

Title	Author	Publish Time	Category	Edit
Cryptanalysis of Short RSA Secret Exponents	admin01	2022-05-04 01:00:10	news	Delete
Teaching Digital Circuit Design With a 3-D Video Game: The Impact of Using In-Game Tools on Students' Performance	admin01	2022-05-01 08:18:40	news	Delete
Application design of community property management system based on Delphi	admin01	2022-05-01 05:18:14	notice	Delete

Figure 19 Notice table

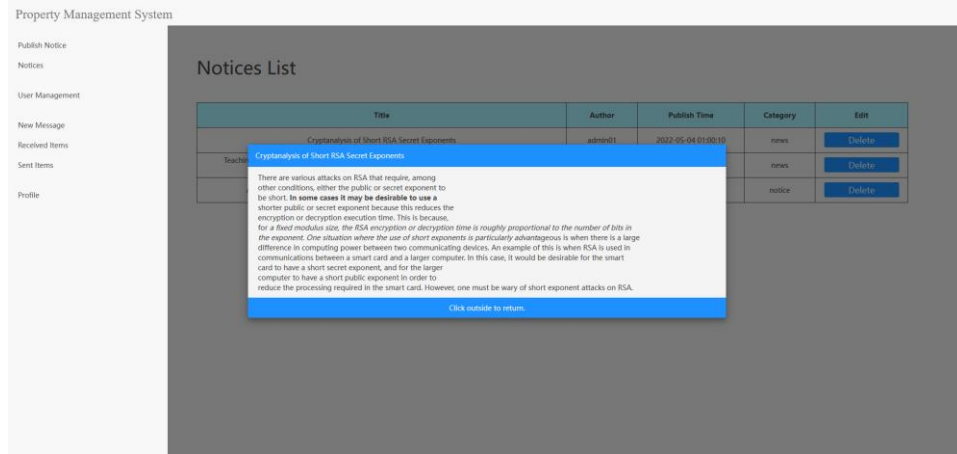


Figure 20 Notice Content

Here are the UI for users:



Figure 21 Notice for Users

Database ER diagram for table *notice*:

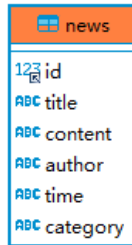


Figure 22 ER Diagram for Notice

As figure 15, administrators can publish notices. A notice must contain a title, category, and content. A rich text editor plug-in, TinyMCE, is referenced to help save rich text formats. The administrator can edit the title and content in the input areas and change the category to news or notice.

The following is the HTML and JS for rich text editor.

```

1. <div class="titleLabel">
2.     <tr>
3.         <td>Title:</td>
4.         <td>
5.             <select class="newsCategory" id="category">
6.                 <option value="news">News</option>
7.                 <option value="notices">Notice</option>
8.             </select>
9.             <input type="text" class="newsTitle" id="newsTitle">
10.        </td>
11.    </tr>
12. </div>
13.     <div class="editor">
14.         <p class="feedback" id="feedback" style="display: none;"></p>
15.         <textarea class="editor"></textarea>
16.     </div>
17.     <script>
18.         tinymce.init({
19.             selector: 'textarea',
20.             plugins: 'allychecker advcode casechange export form
atpainter image editimage linkchecker autolink lists checklist me
dia mediaembed pageembed permanentpen powerpaste table advtable t
ableofcontents tinycomments tinymce_spellchecker',
21.             menubar: 'file edit view format',

```

```

22.         toolbar: 'undo redo | styleselect | bold italic',
23.         toolbar_mode: 'floating',
24.         tinymcecomments_mode: 'embedded',
25.         tinymcecomments_author: 'Author name',
26.     });
27. </script>

```

When the administrator clicks the Save button, the browser checks whether the title and content are empty. If there is blank space, the page will prompt the administrator to add text, for example, “Please fill in the title.” or “Please add content.”

If the check passed, the browser sends the data to the server, which receives data, escapes the HTML tags, and adds it to the database. At the same time, the server reads the administrator id in the session and saves it as the author, then reads the current time and saves it as the publish time. The data type of content is longtext.

Similarly, there will be feedback “Article submitted successfully.” when the data is saved.

The following is the interface that add notice to the database.

```

1. @PostMapping("/adminAddArticle")
2.     public int addNotice(@RequestParam("title") String title, @Re
   questParam("category") String category, @RequestParam("content")
   String content){
3.         Notice notice = new Notice();
4.         notice.setTitle(title);
5.
6.         String temp = HtmlUtils.htmlEscapeHex(content);
7.         notice.setContent(temp);
8.
9.         int adminId = Integer.parseInt(String.valueOf(request.get
   Session().getAttribute("adminId")));
10.        Admin admin = adminMapper.getAdminById(adminId);
11.        String author = admin.getAccount();
12.        notice.setAuthor(author);
13.
14.        Date date = new Date();
15.        SimpleDateFormat dateFormat= new SimpleDateFormat("yyyy-
   MM-dd hh:mm:ss");
16.        String time = dateFormat.format(date);
17.        notice.setTime(time);
18.
19.        notice.setCategory(category);
20.
21.        noticeMapper.addNotice(notice);
22.        return 0;
23.    }

```


For pages that display notices, the browser requests all notices data from the server and renders it dynamically when the page loads. Administrator's version, the data is displayed in a table, and the content text is displayed in a pop-up window when the title is clicked. For the user's version, notice with a blue background and news with a green background, which distinguishes the types of information.

Administrators can delete notices. When the button is clicked, the id of the notice will be sent to the server, which deletes data from the database based on the id. Then the browser will reload the page and show new notice list.

2.2.5 Message Management

UIs for users:

The screenshot shows the 'Property Management System User' interface. At the top, there is a blue header bar with the title 'Property Management System User' and 'User Version' below it. Below the header, there is a navigation bar with links: 'Notices', 'Friends', 'Message', and 'Personal Information'. The 'Message' link is highlighted. The main content area is a light gray box containing a form for sending a message. The form has two input fields: 'Receiver:' and 'Subject:'. Below these fields is a text editor with a menu bar (File, Edit, View, Format) and a toolbar with buttons for undo, redo, paragraph, bold, and italic. The text editor area is large and empty. At the bottom of the text editor, there is a 'Send' button. Below the main content area, there is a dark gray footer bar with the text 'Tips: Here is a final year project: Property management systems. Designed and developed by Shen Wang. Email: shen.wang18@student.xjtu.edu.cn'.

Figure 23 User Message: New Message

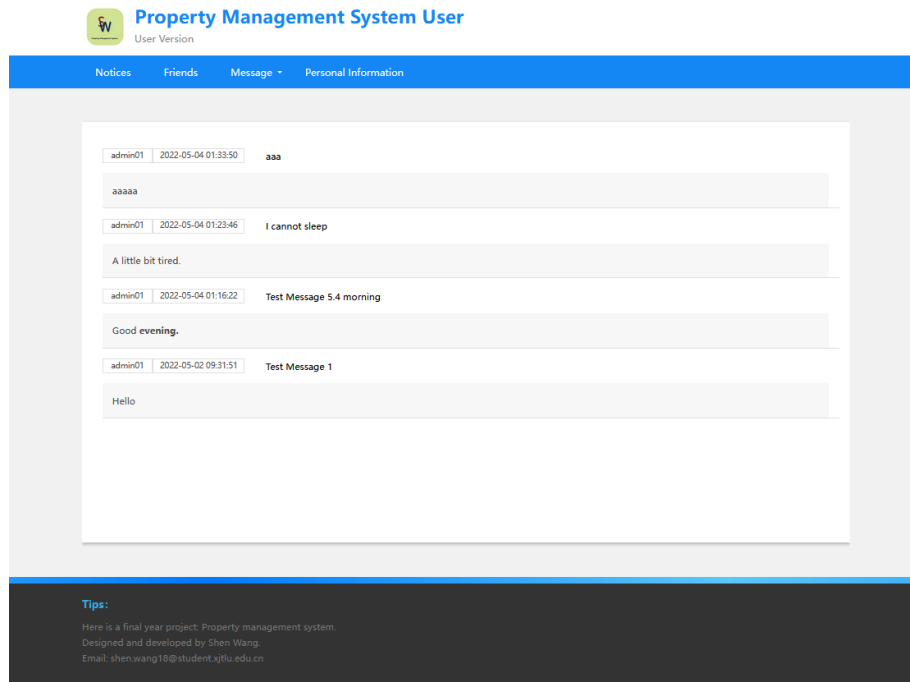


Figure 24 User Message: Received Message

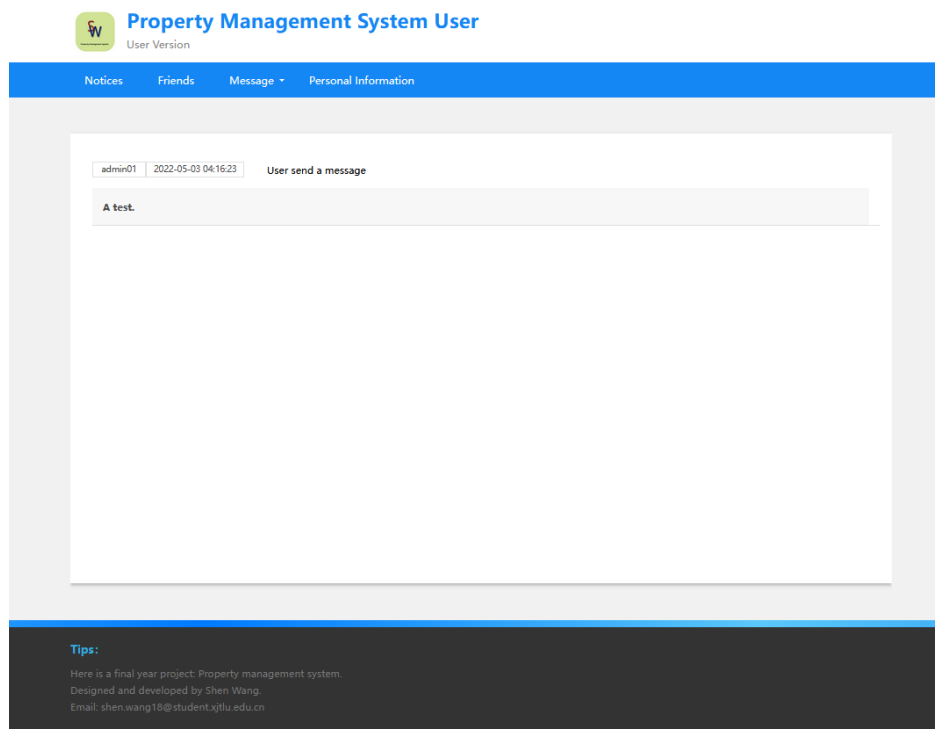


Figure 25 User Message: Sent Message

UIs for administrators:

The screenshot shows the 'New Message' form within the 'Property Management System'. On the left is a sidebar menu with options: Publish Notice, Notices, User Management, New Message (selected), Received Items, Sent Items, and Profile. The main content area is titled 'New Message' and contains a 'Receivers:' text input field, a 'Subject:' text input field, and a large text area for the message body. Above the text area is a rich text editor toolbar with menus for File, Edit, View, and Format, and buttons for undo, redo, Paragraph, Bold (B), and Italic (I). A 'Send' button is located at the bottom right of the form. The text area has a small 'POWERED BY TINY' watermark in the bottom right corner.

Figure 26 Administrator Message: New Message

The screenshot shows the 'Received Items' view within the 'Property Management System'. The sidebar menu is the same as in Figure 26, with 'Received Items' selected. The main content area is titled 'Received Items' and displays a single message item. The message header shows the ID 'A210101', the timestamp '2002-05-01 04:16:23', and the sender 'User send a message'. The message body contains the text 'A test.'.

Figure 27 Administrator Message: Received Message

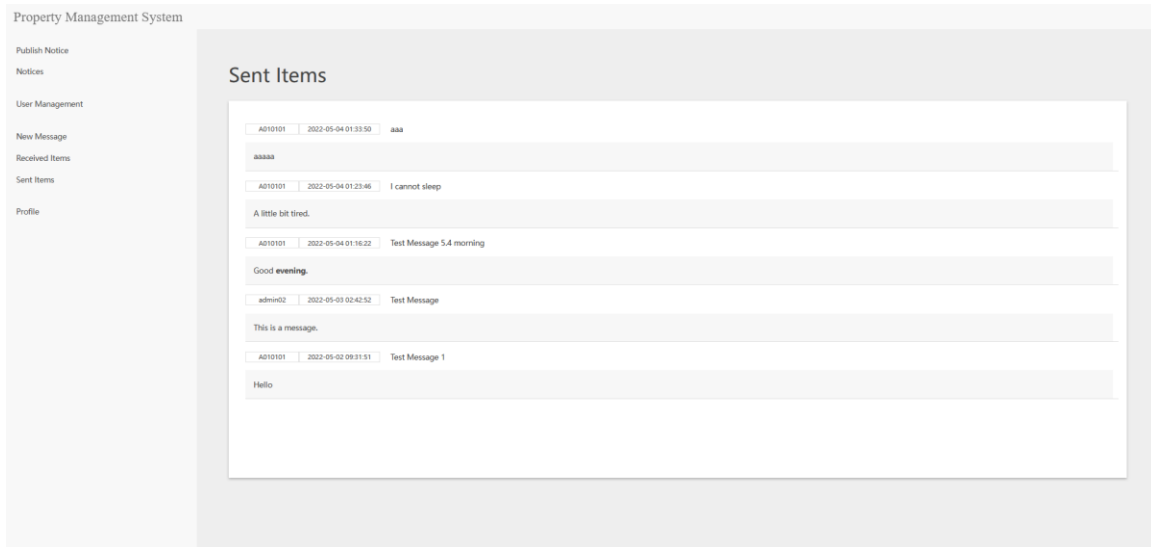


Figure 28 Administrator Message: Sent Message

Database ER diagram for table *message*:

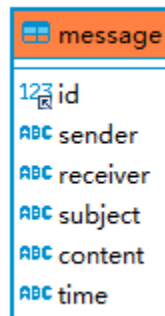


Figure 29 ER Diagram for Notice

The message management functions of both the user and administrator versions can be divided into three parts: new message, received messages, and sent messages. The implementation logic is exactly the same for both versions.

New message is similar to publishing a notice in that both reference a rich text editor plugin. The user must fill in the receiver, subject and content to pass the browser inspection. When the send button is clicked, the browser checks for blank space. The data is then sent

to the server, where the existence of the receiver account is first checked. If the recipient exists, the server records the sender id from session and the current time and adds the message to the database.

Codes in the interface as following.

```
1. @PostMapping("/userAddMessage")
2.     public int userAddMessage(@RequestParam("receiver") String re
   ceiver,
3.                               @RequestParam("subject") String sub
   ject,
4.                               @RequestParam("content") String con
   tent){
5.         Admin checkAdminReceiver = adminMapper.getAdminByAccount(
   receiver);
6.         User checkUserReceiver = userMapper.getUserByAccount(rece
   iver);
7.         if (checkAdminReceiver == null && checkUserReceiver == nu
   ll){
8.             return 1;
9.         } else {
10.             Message message = new Message();
11.
12.             int userId = Integer.parseInt(String.valueOf(request
   .getSession().getAttribute("userId")));
13.             User user = userMapper.getUserById(userId);
14.             String sender = user.getAccount();
15.             message.setSender(sender);
16.             message.setReceiver(receiver);
17.             message.setSubject(subject);
18.
19.             String temp = HtmlUtils.htmlEscapeHex(content);
20.             message.setContent(temp);
21.
22.             Date date = new Date();
23.             SimpleDateFormat dateFormat= new SimpleDateFormat("y
   yy-MM-dd hh:mm:ss");
24.             String time = dateFormat.format(date);
25.             message.setTime(time);
26.
27.             messageMapper.addMessage(message);
28.             return 0;
29.         }
30.     }
```

The received items and sent items pages load data from the server first. The difference is that the received items pages select messages according to the receiver id. And the sent items pages select messages according to the sender id. Then the data will be rendered to the container.

Interface load message according to receiver as following.

```
1. @GetMapping("/userGetMessageListByReceiver")
2.     public List<Message> userGetMessageListByReceiver() {
3.         int userId = Integer.parseInt(String.valueOf(request.getSession().getAttribute("userId")));
4.         User user = userMapper.getUserById(userId);
5.         List<Message> messageList = messageMapper.getMessageListByReceiver(user.getAccount());
6.         for(int i=0;i<messageList.size();i++){
7.             String content = messageList.get(i).getContent();
8.             messageList.get(i).setContent(HtmlUtils.htmlUnescape(content));
9.         }
10.        return messageList;
11.    }
```

Interface load message according to sender as following.

```
1.     @GetMapping("/userGetMessageListBySender")
2.     public List<Message> userGetMessageListBySender() {
3.         int userId = Integer.parseInt(String.valueOf(request.getSession().getAttribute("userId")));
4.         User user = userMapper.getUserById(userId);
5.         List<Message> messageList = messageMapper.getMessageListBySender(user.getAccount());
6.         for(int i=0;i<messageList.size();i++){
7.             String content = messageList.get(i).getContent();
8.             messageList.get(i).setContent(HtmlUtils.htmlUnescape(content));
9.         }
10.        return messageList;
11.    }
```

So far, most features are covered in the section. The following is a use case diagram shows the relation.

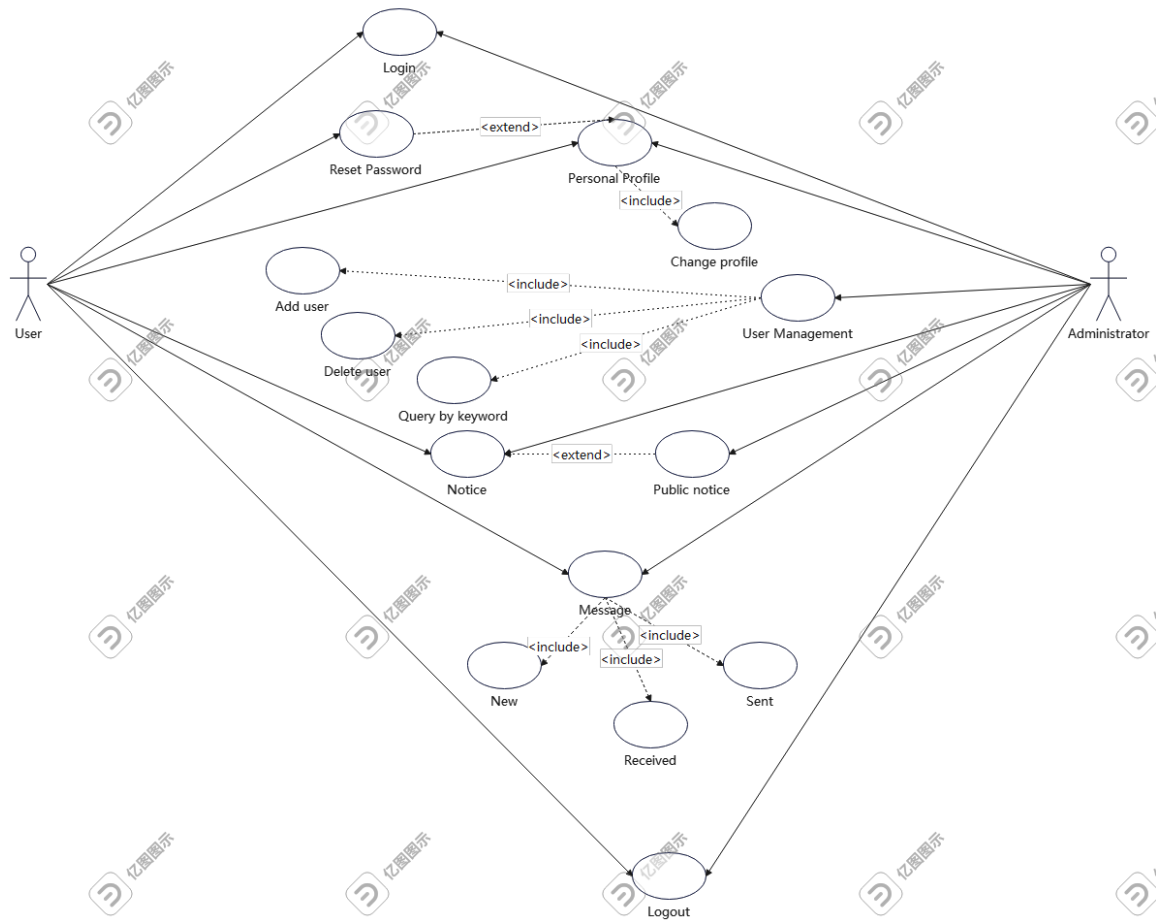


Figure 30 Use Case Diagram

2.2.6 Security

There are controls for page access. As mentioned earlier, when the user logouts, the session data will be deleted. Every page except the login page needs to be checked by a session before loading. If the user has not login to the system, the login page will be automatically displayed. This ensures that only logged-in users can access the application's functionality.

Here are the JS codes.

```

1. <script>
2.   $.get(

```

```

3.         url = "/getUserSession",
4.         function (data) {
5.             if(data==0){
6.                 window.location.href="userLogin.html";
7.             }
8.         },
9.         type = "text"
10.     )
11. </script>

```

Here are the interface codes.

```

12. @GetMapping("/getUserSession")
13.     public int getAdminSession(){
14.         HttpSession session = request.getSession();
15.         if(request.getSession().getAttribute("userId")==null){
16.             return 0;
17.         }else {
18.             return 1;
19.         }
20.     }

```

In addition, using input with a password type and AJAX with the POST method helps keep the system secure.

2.2.7 Alteration

In the original design, the administrator's version would have a schedule management feature. However, this feature actually has little to do with property management. Therefore, this superfluous feature was removed from the development plan.

2.2.8 Develop Environment

Environment	Version
Spring Boot	2.6.3
Java	15.0.2
Apache Tomcat	9.0.56
Maven	4.0.0
Mybatis	2.2.2
MySQL	10.4.22-MariaDB

Table 3 Environment

Component	Property
CPU	Intel Core i5-10210U CPU@ 1.60GHz
GPU	NAVID GeForce MX250 2GB
Memory	8GB (1867 MHz)
Browser	Firefox 100.0.2

Table 4 PC Configuration

Chapter 3 Conclusion and Future Work

3.1 Conclusion

In general, in the past academic year, the property management system was independently designed and developed. The requirement engineering was completed with case studies, and the project was iteratively implemented with agile method. The system can be divided into user version and administrator version, and it is also a platform for communication between two user groups.

The project involved a complete design and development process, so it was a valuable exercise and experience. However, there is also reflection on this development. During the design and development process, I only focused on the design of various features and neglected the planning of overall attributes, especially the non-functional requirements. Since custom configurations do not meet Spring Security requirements, it is difficult to integrate adequate security guarantees late in development. This is because the system was not designed with maintainability and modifiability in mind.

In addition to new techniques and valuable experience, what I learned from this project is how to better plan and design the next project. If the developer focus on non-functional requirements such as modifiability, maintainability, and downward compatibility at the beginning of design and development, the finished product can meet not only current but future needs [9].

3.2 Future Work

As mentioned above, the security of the property management system is inadequate. It is difficult to integrate Spring Security because the project configuration is not standardized enough. What the project has done on security is to limit access to users who are not logged in and to use secure data transfer methods.

In the short term, what can be done is to use encryption to encrypt data in transit. In the long term, the structure and configuration of Spring Boot can be reworked to integrate

Spring Security for more comprehensive authentication and authorization management. At the same time, database access control can also better protect data security.

In addition, this project is a web application, which is limited by the browser and is difficult to call local resources. If the architecture of the native application can be partly used, the dependence on the network and browser can be reduced [10]. This requires a redesign of the software framework. Moreover, current UI layout and new scripting techniques can be used to build responsive, up-to-the-minute pages. This improves the aesthetics of the software and takes advantage of new technologies and frameworks.

References

- [1] X. Zhou, Y. Zheng and A. Wang, "Design of property management system server based on the Asterisk," in *2014 9th International Conference on Computer Science & Education*, 2014, pp. 606-609, doi: 10.1109/ICCSE.2014.6926534.
- [2] Q. Yao and C. Gu, "Design and implementation of community property management system based on Delphi," in *2010 The 2nd International Conference on Industrial Mechatronics and Automation*, 2010, pp. 565-568, doi: 10.1109/ICINDMA.2010.5538243.
- [3] J. Tian, "Application design of community property management system based on Delphi," in *2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, 2010, pp. 85-88, doi: 10.1109/CCTAE.2010.5543541.
- [4] Z. Mu and X. Li, "Analysis and Design of Property Management System Based on B/S," in *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, 2019, pp. 381-384, doi: 10.1109/ICITBS.2019.00100.
- [5] Y. Li, L. Cao, Y. Qian, W. Shi and A. Liu, "A property management system using WebGIS," in *2010 2nd International Conference on Computer Engineering and Technology*, 2010, pp. V2-683-V2-685, doi: 10.1109/ICCET.2010.5485699.
- [6] Z. Jin, "Requirements Engineering Methodologies," in *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*, Morgan Kaufmann, 2018, ch.2, pp.13-27.
- [7] S. Al-Saqqah, S. Sawalha and H. AbdelNabi, "Agile Software Development: Methodologies and Trends," *Int. J. Interact. Mob. Technol.*, vol. 14, no. 11, pp. 246-270, 2020.
- [8] TinyMCE. [Online]. <https://www.tiny.cloud/>
- [9] U. Garg and A. Singhal, "Software requirement prioritization based on non-functional requirements," *2017 7th International Conference on Cloud Computing, Data Science*

& Engineering - Confluence, 2017, pp. 793-797, doi:
10.1109/CONFLUENCE.2017.7943258.

- [10] R. Nunkesser, "Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development," 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2018, pp. 214-218.

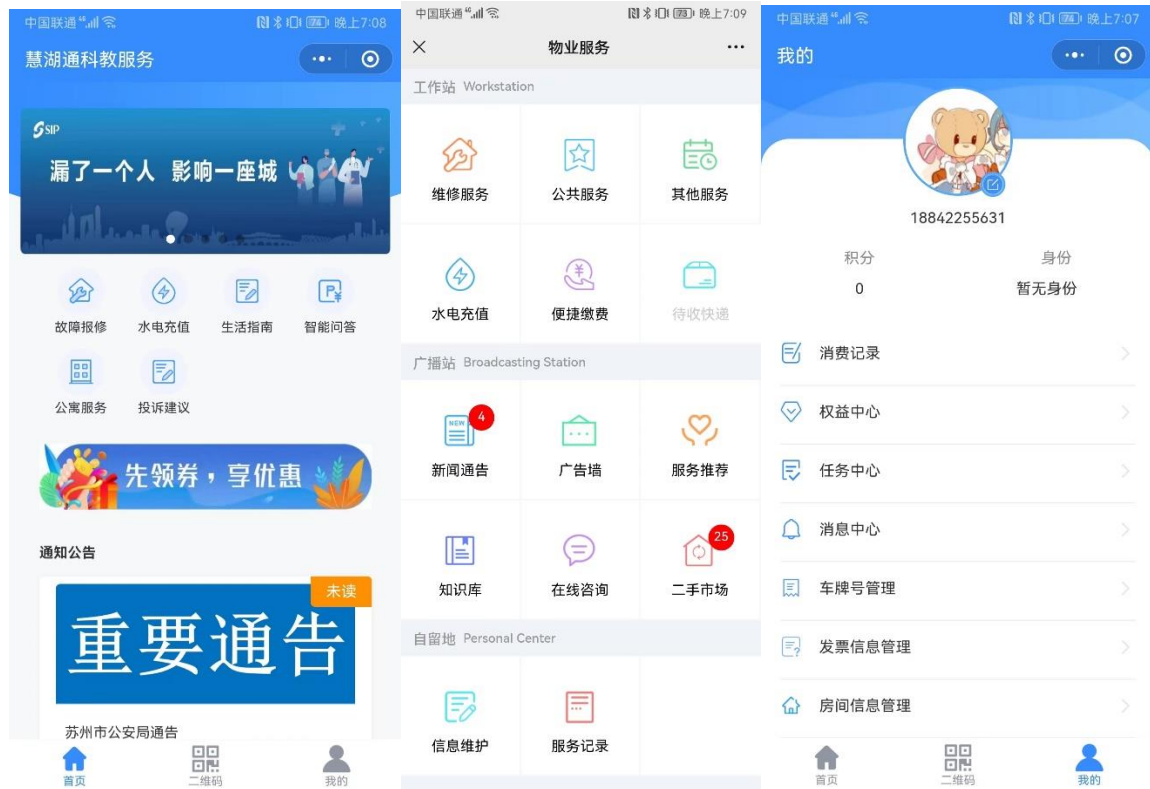
Appendix A. Study Case: Huihutong

1. Huihutong Website:



This site is mainly categorized to display news and announcements from the company. It is divided into sections such as news from the community, notices from the government, and introductions to the company.

2. WeChat Mini Program:



Here are the home page, services and account management of WeChat mini program.

For the first home page, here shows notices and news. And there are buttons link to different services pages.

The second figure is the services list of the system. Users can book repair visits, pay utility bills, check delivery, view the online second-hand goods market and so on.

The third figure is the account management page. Users can view the history and manage personal information here.

Appendix B. Code Files Category

Browser-side files:

1. User login page:

HTML	src/main/resources/templates/userLogin.html
CSS	src/main/resources/static/css/userLogin.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/login.js

2. Administrator login page:

HTML	src/main/resources/templates/adminLogin.html
CSS	src/main/resources/static/css/adminLogin.css
JS	src/main/resources/static/js/login.js
	src/main/resources/static/js/jquery-3.5.1.js

3. User personal profile page:

HTML	src/main/resources/templates/user-info.html
CSS	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-info.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-information.js

4. Administrator personal profile page:

HTML	src/main/resources/templates/admin-info.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/admin-info.css
JS	src/main/resources/static/js/jquery-3.5.1.js

	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-info.js

5. User query neighbors page:

HTML	src/main/resources/templates/user-query.html
CSS	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-query.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-query.js

6. Administrator user management page:

HTML	src/main/resources/templates/admin-user-manage.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/admin-user-manage.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-user-manage.js

7. Administrator publish notice page:

HTML	src/main/resources/templates/admin-publish-news.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/admin-publish-news.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-publish-news.js

8. Administrator notice page:

HTML	src/main/resources/templates/admin-news.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/admin-news.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-news.js

9. User notice page:

HTML	src/main/resources/templates/user-index.html
CSS	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-news.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-news.js

10. Administrator new message page:

HTML	src/main/resources/templates/admin-new-message.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/admin-new-message.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-new-message.js

11. Administrator received message page:

HTML	src/main/resources/templates/admin-received-items.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css

	src/main/resources/static/css/admin-received-items.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-received-items.js

12. Administrator sent message page:

HTML	src/main/resources/templates/admin-sent-items.html
CSS	src/main/resources/static/css/bootstrap.min.css
	src/main/resources/static/css/admin-style.css
	src/main/resources/static/css/ admin-received-items.css
JS	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/bootstrap.min.js
	src/main/resources/static/js/admin-sent-items.js

13. User new message page:

HTML	src/main/resources/templates/user-new-message.html
	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-new-message.css
	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-new-message.js

14. User received message page:

HTML	src/main/resources/templates/user-received-items.html
	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-received-items.css
	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-received-items.js

15. User sent message page:

HTML	src/main/resources/templates/user-sent-items.html
	src/main/resources/static/css/base.css
	src/main/resources/static/css/user-received-items.css
	src/main/resources/static/js/jquery-3.5.1.js
	src/main/resources/static/js/user-sent-items.js

Above are all 15 HTML pages and their corresponding CSS and JS files.

Server-side files:

src/main/java/com/example/demo/config/MyMvcConfig.java register 15 pages that can be accessed.

User files:

controller	src/main/java/com/example/demo/controller/UserController.java
mapper	src/main/java/com/example/demo/mapper/UserMapper.java
pojo	src/main/java/com/example/demo/pojo/User.java
mybatis.mapper	src/main/resources/mybatis/mapper/UserMapper.xml

Admin files:

controller	src/main/java/com/example/demo/controller/AdminController.java
mapper	src/main/java/com/example/demo/mapper/AdminMapper.java
pojo	src/main/java/com/example/demo/pojo/Admin.java
mybatis.mapper	src/main/resources/mybatis/mapper/AdminMapper.xml

Notice files:

controller	src/main/java/com/example/demo/controller/NoticeController.java
mapper	src/main/java/com/example/demo/mapper/NoticeMapper.java

pojo	src/main/java/com/example/demo/pojo/Notice.java
mybatis.mapper	src/main/resources/mybatis/mapper/NoticeMapper.xml

Message files:

controller	src/main/java/com/example/demo/controller/MessageController.java
mapper	src/main/java/com/example/demo/mapper/MessageMapper.java
pojo	src/main/java/com/example/demo/pojo/Message.java
mybatis.mapper	src/main/resources/mybatis/mapper/MessageMapper.xml