**CMPUT 391 Project Report**

Tiancheng Shen

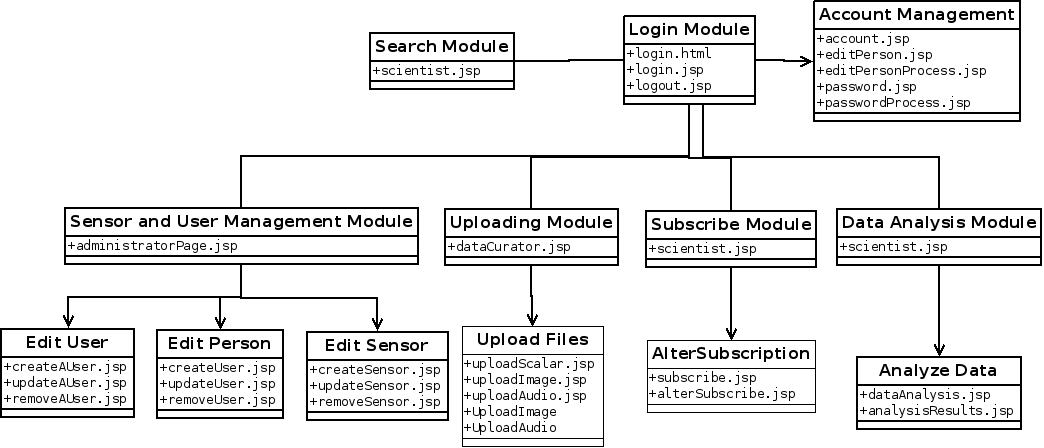
Hong Wang

Di Zhang

**Introduction:**

This project is a three-tiered web-based Ocean Observation System. The three tiers are the database server, the web server and the client. These components run on different machines and are connected via the Internet. JSP files are used to build the interface between the website and the database server. Oracle SQL is utilized for the database server, and Tomcat is utilized for the web server. Our clients currently have the authority to access this system through Google-Chrome on Linux systems.

**Architecture:**



**Modules:**

**Login Module:**

All users can use Login Module. It enables users to log in the Ocean Observation System with proper privileges. They can modify their personal information or password. It also set session so that other pages can use session to detect if user has privileges to access the page.

login.html:

This is the start page of our Ocean Observation System. User can input username and password at this page. It will also detect error if user input nothing and click “Log In”.

login.jsp:

This is the page after user login. It uses SQL statement "select \* from users where users.user\_name = '" + LogName + "'" to check user's username and password, where the “LogName” is the username we got from login.html. Although it shows “Loading”, it actually doesn't load anything, because we just need a page to detect the username and password error. If user input wrong username or password, it will show the error and go to login.html. If user input correct username and password, it will show welcome and go to the homepage of the user's role automatically. It also set session so that other pages can use session to detect if user has privileges to access the page.

account.jsp:

This page shows user's personal information. It uses “String UserID = (String) session.getAttribute("userid")” to get user's Person Id, and uses SQL statement "select \* from persons where persons.person\_id = '" + UserID + "'" to get other detailed information of user.

It also provide link for modifying personal information (“Modify Personal Information” button), changing password (“Change Password” button), entering user's homepage (“Go to HomePage” button), and logging out (“Log Out” button).

editPerson.jsp:

This page shows up when user click “Modify Personal Information” button at account.jsp page. It enables users to edit his information expect Person ID. Person ID is the primary key, so only administrators can change it. Similar to account.jsp, it uses "select \* from persons where persons.person\_id = '" + UserID + "'" statement, where UserID is got from session, to get detailed information and shows them at text blank, so user can edit it easier. After user modifying it, clicking “Save” button to submit or user can just cancel the modifying by clicking “Cancel” button.

editPersonProcess.jsp:

After user click “Save” button at editPerson.jsp page, the editPersonProcess.jsp page will get new information and use "update persons set first\_name = '" + UserFN + "', last\_name = '" + UserLN + "', address = '" + UserAddr + "', email = '" + UserEmail + "', phone = '" + UserPhone + "' where persons.person\_id = '" + UserID + "'" statement to update the information. “UserFN”, “UserLN”, “UserAddr”, “UserEmail”, and “UserPhone” are all got from editPerson.jsp page. Then it will go to account.jsp page automatically.

password.jsp:

This page shows up when user click “Change Password” button at account.jsp page. It lets user input new password twice. It will detect error if user inputs nothing or these two inputs are not matched. After user inputs correctly, clicking “Save” button to submit or user can just cancel the modifying by clicking “Cancel” button.

passwordProcess.jsp:

After user click “Save” button at password.jsp page, the passwordProcess.jsp page will get the new password and use "update users set password = '" + newPW + "' where users.user\_name = '" + LogName + "'" statement to update the password. “newPW” is got from password.jsp, and “LogName” is got from session so that we can make sure we change the password for right person. Then it will go to account.jsp page automatically.

logout.jsp:

This page shows up when user click “Log Out” button at account.jsp page. It will update user login status to session, so user cannot access any pages except login.html page.

**Sensor and User Management Module:**

administratorPage.html

This page is composed of several buttons that promts administrators to choose from “edit user” and “edit sensor”.

editSensor.jsp

This page displays a table of all existing sensors and several empty editTexts that promts user to enter information of the sensor that he wants to manage.

removeSensor.jsp

By entering a sensor id , system will check if the sensor exists in table by “select \* from sensors” and assert” resultset.getInt ("sensor\_id") == sensor id” ; Also, system will ensure sensor id is non-empty and numeric. And then system will delete sensor with sensor id that user entered and all its related data by using “delete from tables where senor\_id = sensor\_id”; (table includes sensors, subscriptions...)

createSensor.jsp

System will check if the sensor exists in table by “select \* from sensors” and assert “resultset.getInt("sensor\_id") != sensor id”; Also, system will ensure sensor id is non-empty and numeric, type is within (“a”,”i”,”s”). After enter correct and new sensor id，system will “insert into sensors value (?,?,?,?,?), stmt.setInt(1,id)…”.

editUser.jsp

This page displays a table of all existing users and several empty editTexts that promts administrators to enter information of the user that he wants to manage.

removeUser.jsp

By entering a sensor id , system will check if the user exists in table by “select \* from persons” and assert” resultset.getInt ("id") ==id” ; Also, system will ensure id is non-empty and numeric. And then system will delete user with the id that administrators entered and its related data by using “delete from tables where id = id”; (table includes persons, users...)

createUser.jsp

System will check if the user exists in table by “select \* from users” and assert “resultset.getInt("id") !=id”; Also, system will ensure id is non-empty and numeric. And them system will “insert into persons value (?,?,?,?,?), stmt.setInt(1,id)…”.

UpdateUser.jsp

After checking validity as createUser, use query="UPDATE persons SET first\_name = ?,last\_name=?,address = ?,email=?,phone = ? WHERE person\_id =id, stmt.setString(1,first); …” to update user.

**Subscribe Module:**

subscribe.jsp:

This page displays a table of all existing sensors and the sensor\_ids that the logged in scientist subscribed to. The SQL statement to select all sensors is “SELECT \* FROM sensors”, then “while(rset.next())” will output all sensor information in the database. The scientist's person\_id is retrieved with “(String)session.getAttribute(“userid”);”, and “userid” is set in login.jsp. Now to extract the sensor\_ids this scientist subscribed to, person\_id can be used to get this information from the subscriptions table, the SQL statement “SELECT sensor\_id FROM subscriptions WHERE person\_id=”+person\_id; and “while(rset.next())” will output all sensor\_ids this scientist subscribed to.

To add or remove a sensor, input the sensor\_id the scientist wishes to add or remove and select the appropriate radio button. Once the “submit” button is clicked, the scientist will be directed to alter\_subscribe.jsp to find out whether the sensor is added or removed.

alter\_subscribe.jsp:

This page displays whether the scientist has successfully added or removed a subscription to a sensor. Unsuccessful attempts to add/remove a sensor subscription include: sensor does not exist, sensor already subscribed when adding subscription, sensor not subscribed when trying to remove. The SQL queries for alter\_subscribe.jsp file is the same as the queries for subscribe.jsp. The scientist can choose to return to subscribe page or account page.

**Upload Module:**

upload.jsp:

This page displays the types of data the data curator can upload. Upon choosing a data type and hit “Add”, the data curator will be directed to uploadData.jsp to upload data.

uploadData.jsp:

This page displays an html “multiart/form-data” form for file uploading. The data curator uploads a file that matches the file type chosen in upload.jsp. Scalar data accepts .csv files, image accepts .jpg files and audio accepts .wav files. Each form can only upload one file at a time. If a .csv file is uploaded, the data curator will be directed to uploadScalar.jsp. If a .jpg file is uploaded, the user will be directed to UploadImage class. If a .wav file is uploaded, the user will be directed to UploadAudio class.

uploadScalar.jsp:

This page displays whether the .csv file is read and the data is inserted into the database. If file input is empty, the user will be directed to uploadFinish.jsp and receive file empty message. If a correct file is uploaded, each line is split into three parts and inserted into the database using the query “INSERT INTO scalar\_data VALUES(“+id+”,”+sensor\_id+”,to\_date(“+date+”,'dd/mm/yyyy hh24:mi:ss'),”+value+”)”;

UploadImage:

This class inserts an uploaded image and its thumbnail to the database. This class processes the uploaded image and creates a thumbnail by shrinking the original image. With the query “SELECT SEQ\_IMAGE\_ID.NEXTVAL FROM dual”, a unique image\_id is generated and this id corresponds to the uploaded image. After the id is generated, it is used to create a row in images table with default information with the exception of image\_id. “INSERT INTO images VALUES(“+image\_id+”3333,SYSDATE','testdesc',empty\_blob(),empty\_blob())” is the SQL query. Then, the class updates the empty\_blob() to its corresponding image with “SELECT \* FROM images WHERE image\_id=”+image\_id+” FOR UPDATE”. The image and thumbnail are converted into type blob and updated in the row with corresponding image\_id. Upon completion, the data curator is redirected to uploadImage.jsp.

uploadImage.jsp:

This page displays all image sensors and let the data curator upload image information. The SQL query to display all image sensors is “SELECT \* FROM sensors WHERE sensor\_type='i'”. The data curator then enters the sensor\_id of the image, date, time and description. When the upload button is clicked, the information is updated in the database using the SQL statement "update images set date\_created=TO\_DATE('"+datetime+"','dd/mm/yyyy hh24:mi:ss'),sensor\_id="+sensor\_id+",description='"+description+"' where image\_id="+image\_id). Upon completion, the user knows that the file is uploaded.

UploadAudio

This class uploads an audio file to the database. A unique recording\_id is generated with the SQL query “SELECT recording\_id.nextval FROM dual”. The uploaded file is processed and the binary blob file is inserted into a row with its corresponding recording\_id and default information in the audio\_recordings table. Upon completion, the data curator is redirected to uploadAudio.jsp.

uploadAudio.jsp:

This page displays all audio sensors and let the data curator upload audio information. The SQL query to display all audio sensors is “SELECT \* FROM sensors WHERE sensor\_type='a'”. The data curator then enters the sensor\_id of the audio, date, time, length, and description. When the upload button clicked, the information is updated in the database using the SQL statement "update audio\_recordings set date\_created=TO\_DATE('"+datetime+"','dd/mm/yyyy hh24:mi:ss'),sensor\_id="+sensor\_id+",description='"+description+"',length="+length+" where recording\_id="+recording\_id). Upon completion, the user knows that the file is uploaded.

UploadFinish.jsp:

This page displays a message that file input for upload module is empty.

**Search Module:**

search.jsp

This page prompts users to enter information about keywords, location, type and time period to specify search conditions

searchResult.jsp

After receiving search conditions, system first check if type is within {“s”,”a”,”i”)and if kewords and time period is valid and non-empty.

And then system first searches all sensors that user subscribed to and store it into a new table by using SQL “create table sensors1 as (Select sensors.sensor\_id, sensors.location, sensors.sensor\_type, sensors.description FROM sensors, subscriptions WHERE sensors.sensor\_id = subscriptions.sensor\_id AND subscriptions.person\_id ="+person\_id+").

*Case 1:* If there are sensors having description, type and location matched:

1. store sensors into a new table by query ="create table matched as (SELECT \* FROM sensors1 WHERE sensors1.description LIKE '%+keywords+%'AND sensors1.location LIKE '%"+location+%'AND sensors1.sensor\_type = '"+type.charAt(0)+"')";
2. display images, audio recordings and scalar data of these senosrs created within time period by

queryImage= “SELECT images.image\_id,… WHERE images.sensor\_id = matched.sensor\_id AND images.date\_created between to\_date('" +startTime+ "', 'DD/MM/YYYY') AND to\_date('" + endTime + "', 'DD/MM/YYYY')";

queryAudio ="SELECT audio\_recordings.recording\_id,… FROM audio\_recordings,matched WHERE audio\_recordings.sensor\_id = matched.sensor\_id AND audio\_recordings.date\_created between to\_date('" +startTime+ "', 'DD/MM/YYYY') AND to\_date('" + endTime + "', 'DD/MM/YYYY')";

queryScalar ="SELECT scalar\_data.id,scalar\_data.sensor\_id,scalar\_data.date\_created,scalar\_data.value FROM scalar\_data,matched WHERE scalar\_data.sensor\_id = matched.sensor\_id AND scalar\_data.date\_created between to\_date('" +startTime+ "', 'DD/MM/YYYY') AND to\_date('" + endTime + "', 'DD/MM/YYYY')";

*Case 2:* If there is no sensors have description matched with keywords

1. store sensors that have type and location matched into a new table by query= "create table partialMatched as (SELECT \* FROM sensors1 WHERE sensors1.location LIKE '%"+location+"%' AND sensors1.sensor\_type = '"+ type.charAt(0)+"')";

2. display sensors’ images and audio recordings that having description matched with kewords by query:

queryImage ="SELECT images.image\_id… FROM images,partialMatched WHERE images.sensor\_id = partialMatched.sensor\_id AND images.description LIKE '%"+keywords+"%' AND images.date\_created between to\_date('" +startTime+ "', 'DD/MM/YYYY') AND to\_date('" + endTime + "', 'DD/MM/YYYY')";

queryAudio ="SELECT audio\_recordings.recording\_id,… FROM audio\_recordings,partialMatched WHERE audio\_recordings.sensor\_id = partialMatched.sensor\_id AND audio\_recordings.description LIKE '%"+keywords+"%' AND audio\_recordings.date\_created between to\_date('" +startTime+ "', 'DD/MM/YYYY') AND to\_date('" + endTime + "', 'DD/MM/YYYY')";

DownloadImage

System first get image by "select \* from images where image\_id = "+ imageId; Blob b = rset.getBlob("recorded\_data");” and then use response and outputStream to enable users to download image.

DownloadAudio

System first get audio recordings by query = "select \* from audio\_recordings where recording\_id = "+ audioId; Blob b = rset.getBlob("recorded\_data");” and then use response and outputStream to enable users to download audio recordings.

DownloadScalar

System first get scalar file by query = "select \* from scalar\_data where id = "+ Id; int sid = rset.getInt("id"); sensor\_id = rset.getInt("sensor\_id");

Date create = rset.getDate("date\_created");float value = rset.getFloat("value");”

And then convert them to bytes, use response and outputStream to enable users to download scalar.csv

**Data Analysis Module:**

Only scientists can access this module. It enables scientists to view the average, minimum, and maximum values of the selected sensor on five levels of time hierarchies.

dataAnalysis.jsp:

This page shows up when user click “Data Analysis” button at scientist.jsp page. Firstly, it uses SQL statement "select \* from sensors" to show all sensors information and "select sensor\_id from subscriptions where person\_id='"+UserID+"'" to show the sensors user has subscribed, where “UserID” is got from session. Since user can only choose the subscribed sensors, it also provide a link (“Go To Subscribe” button) to subscribe.jsp page so that user can edit his subscribed sensors. After that, user can select which sensor he want to view and the time hierarchies, and then click “Submit” button.

analysisResult.jsp:

After user click “Submit” button at dataAnalysis.jsp page, the analysisResult.jsp page will get the sensor id in “Sensor” and time hierarchies in “Time”. Firstly, it uses SQL statement "select \* from sensors where sensor\_id = '"+Sensor+"'" to show the detailed information of the selected sensor. Then it uses SQL statement "SELECT to\_char(date\_created, '(time format)'), avg(sd.value), max(sd.value), min(sd.value) FROM scalar\_data sd WHERE sd.sensor\_id = '"+Sensor+"' GROUP BY to\_char(date\_created, '(time format)') ORDER BY to\_char(date\_created, '(time format)')" to show the average, minimum, and maximum values according to the time hierarchies. The “(time format)” can be “yyyy”, “yyyy-q”, “yyyy-mm”, “yyyy-ww”, or “yyyy-mm-dd”, depending on which time hierarchy user chooses. After viewing the scalar values of the sensor, user can click “Back” button to go back to dataAnalysis.jsp page to select the sensor and/or time hierarchies again.