Relationships between files:

Button in Header	File	Permissions	Imports (all files import: "imports.js" "oceanstyler.css")	Calls: (All files call navigate to any other page in the header)	Automatically Redirects to:
Analyze Data	analyze.jsp	Scientist	analyzeSensor.js	analyzeSensor.jsp	
Search Sensors	search.jsp	Scientist	search.js	executeSearch.jsp	
Sensor Subscriptions	subscribe.jsp	Scientist		directSubSensor.jsp subSensor.jsp unSubSensor.jsp	
Upload Data	upload.html	Data Curator	upload.js	uploadToDatabase.jsp	
User Management	usermanage.jsp	Administrator	usermanage.js	createAccount.jsp assocUser.jsp removeThatAccount.jsp modiUser.jsp	
Sensor Management	sensormanage.jsp	Administrator		removeSensor.jsp createSensor.jsp	
Help	help.html	Unauthenticated			
Log Out	login.html	Unauthenticated		login.jsp	
My Info	mymanage.jsp	Authenticated		mymodifyAccount.jsp	
Home	landing.html	Authenticated	landing.js		
	analyzeSensor.jsp	Scientist	analyzeSensor.js	analyzeSensor.jsp (recursive)	
	executeSearch.jsp	Scientist		download.jsp	
	subSensor.jsp	Scientist			sensormanage.jsp
	unSubSensor.jsp	Scientist			sensormanage.jsp
	createAccount.jsp	Administrator			usermanage.jsp
	assocUser.jsp	Administrator		assocAccount.jsp	
	removeThatAccount.jsp	Administrator			usermanage.jsp
	modiUser.jsp	Administrator		modifyAccount.jsp	
	download.jsp	Scientist			
	mymodifyAccount.jsp	Authenticated			landing.html
	uploadToDatabase.jsp	Data Curator			upload.html
	modifyAccount.jsp	Administrator			usermanage.jsp
	assocAccount.jsp	Administrator			usermanage.jsp

analyze.jsp
This is the landing page for Data Analysis. This page lists all of the current user's Subscriptions using the below query(1). A table is generated from these, each with a radio button. When a radio button is selected and the user presses "Analyze Data", the sensor ID will be saved to cookies and the page will move to "analyzeSensor.jsp"

1) *select S.SENSOR_ID, S.LOCATION, S.SENSOR_TYPE, S.DESCRIPTION from SENSORS S, SUBSCRIPTIONS T where S.SENSOR_ID=T.SENSOR_ID and T.PERSON_ID="+pid*

analyzeSensor.jsp / analyzeSensor.js
This is where the majority of the data analysis happens. The code uses sql to create a view of the data cube information based on time and id. After the view is created, we show the average, min, and max of all scalar data values for each year that there is data, using simple sql queries. The user can drill down and select quarter, then month, then week, and finally day, as well as generalize their way back up. This is all done with simple select statements on the created view. When the "Update" button is pressed, the page calls itself and reads the new OLAP details from the cookies.

Year, quarter, month, and day are all grabbed via built in oracle functions. Week is calculated with a small mathematical formula that uses built in functions:
2) *"CREATE OR REPLACE VIEW OLAP_DATA AS SELECT s.sensor_id, EXTRACT (YEAR from s.date_created) AS Year, to_char(s.date_created, 'Q') AS Quarter, EXTRACT (MONTH FROM s.date_created) AS Month, (CEIL((to_char(s.date_created, 'DD') - to_char(s.date_created, 'D'))/7)+1) AS Week, to_char(s.date_created, 'D') AS Day, AVG(s.value) AS Average, MIN(s.value) AS Minimum, MAX(s.value) AS Maximum FROM scalar_data s WHERE s.sensor_id IN (SELECT sensor_id FROM subscriptions WHERE person_id = '" + pid + "') AND s.sensor_id = '" + sid + "' GROUP BY s.sensor_id, ROLLUP(EXTRACT (YEAR from s.date_created), to_char(s.date_created, 'Q'), EXTRACT (MONTH FROM s.date_created), (CEIL((to_char(s.date_created, 'DD') - to_char(s.date_created, 'D'))/7)+1),to_char(s.date_created, 'D'))"*

Data is collected by running a simple select statement. Our code checked what level of search the user wants, and uses a series of if statements to decide what query to use. An example of a simple select query: Retrieve the data of all months for a given quarter.
3) *"select MONTH, AVERAGE, MINIMUM, MAXIMUM from OLAP_DATA where QUARTER = " + grabQU + " AND YEAR = " + grabYR + " AND WEEK IS NULL";*

createAccount.jsp/assocAccount.jsp/assocUser.jsp
When the “Create New User” form is filled out in usermanage.js and submitted, it calls createAccount.jsp. Alternatively, they can add an associated user if they press the “Add Associated User” button in usermanage.js and fill out the form in assocUser.jsp. Both assocAccount and createAccount are nearly identical. The difference is that assocAccount creates a new users, whereas createAccount creates a new user AND a new person.

createAccount:
4) select person_id, first_name, last_name, address, email, phone from PERSONS";

createAccount, assocAccount;
5) select USER_NAME, PASSWORD, ROLE, DATE_REGISTERED, PERSON_ID from USERS";
6) select USER_NAME, SALT from SALTS";
Select U.USER_NAME from USERS U, SALTS S where U.USER_NAME=? and U.USER_NAME=S.USER_NAME

createSensor.jsp
createSensor is called when the Create New Sensor form is filled out in sensormanage.jsp. Create sensor checks if the chosen sensor ID is unique (8), and if it is, it adds the sensor into the database.
7) select SENSOR_ID, LOCATION, SENSOR_TYPE, DESCRIPTION from SENSORS";
8) select SENSOR_ID from SENSORS where SENSOR_ID = "+sid;

subSensor.jsp
subSensor accepts input from a sensor with a checked radio in the table on sensormanage.jsp, add creates a subscription entry for it.
Subscribed before adding a subscription (10)
9) select SENSOR_ID, PERSON_ID from SUBSCRIPTIONS";
10) select SENSOR_ID from SUBSCRIPTIONS where SENSOR_ID="+sid+" and PERSON_ID="+pid;

download.jsp

When a download button is pressed on the corresponding entry within search results, it will search for the matching blob and download the file. Values are directly coded into each button, so there is no room for error. The file will be named after the it's type and it's ID. E.g. "sensor_image_id124.jpeg"
11) select recoreded_data from images where image_id=?
12) select recorded_data from audio_recordings where recording_id=?

executeSearch.jsp
This module is where the queries for the search module are formed, after the user inputs the search parameters. It starts this by first reading the user's person_id from a cookie, and then creates an sql query that displays all images, audio recordings, and scalar data that comes from sensors the user is subscribed to. For each additional search parameter the user provides, the code tacks on additional conditions via adding the "AND" operator. If sensor type and/or location is a search parameter, the query is modified to make sure that the data's associated sensor has that sensor type and/or location. If a date range is provided, the query makes sure that all data happens between that range. If keywords is a required field, then the sql statement checks that either the data's description (in the case of audio or image data) matches the keyword parameter, OR the associated sensor's description matches the keywords. After all search parameters are checked, the program inputs crafted queries into sql, and then displays any data that matches the search conditions.
In this page you can directly download the scalar data: every scalar data result will be saved in into a batch together in a CSV file. CSV file is called "scalar-data_batch_date_after_until_date_before.csv".

SQL STATEMENTS:
Note: Anything that occurs after a + sign is only added if the user is searching with that parameter. If they are not, the "AND" statement is not added to the query.
13) "SELECT Distinct a.recording_id, a.sensor_id, a.date_created, a.length, a.description FROM sensors sen, audio_recordings a WHERE a.sensor_id in (SELECT sensor_id FROM subscriptions WHERE person_id = "+pid+") + " AND sen.sensor_id = a.sensor_id AND sen.sensor_type = '"+ sensorType+""+" AND sen.location = '"+ location+"" +" AND (a.description = '"+keywords+"' OR sen.description = '"+ keywords+"')" + " AND a.date_created <= TO_DATE('"+dateBefore+"','dd/mm/yyyy HH24:MI:SS')" + " AND a.date_created >= TO_DATE('"+dateAfter+"','dd/mm/yyyy HH24:MI:SS')"

14) "SELECT Distinct i.image_id, i.sensor_id, i.date_created, i.description FROM sensors sen, images i WHERE i.sensor_id in (SELECT sensor_id FROM subscriptions

WHERE person_id = "+pid+") + " AND sen.sensor_id = i.sensor_id AND sen.sensor_type = '"+ sensorType+"'" +" AND sen.location = '"+ location+"'"+" AND (i.description = '"+keywords+"' OR sen.description = '"+ keywords+"')" + " AND i.date_created <= TO_DATE('"+dateBefore+"','dd/mm/yyyy HH24:MI:SS')" + " AND i.date_created >= TO_DATE('"+dateAfter+"','dd/mm/yyyy HH24:MI:SS')"

15) SELECT Distinct s.id, s.sensor_id, s.date_created, s.value FROM sensors sen, scalar_data s WHERE s.sensor_id in (SELECT sensor_id FROM subscriptions WHERE person_id = "+pid+") + " AND sen.sensor_id = s.sensor_id AND sen.sensor_type = '"+ sensorType+"'"+" AND sen.location = '"+ location+"'" + " AND sen.description = '"+ keywords+"'" + " AND s.date_created <= TO_DATE('"+dateBefore+"','dd/mm/yyyy HH24:MI:SS')" + " AND s.date_created >= TO_DATE('"+dateAfter+"','dd/mm/yyyy HH24:MI:SS')"

help.html

A basic help file that displays the help document via Google Drive embedding. A hard copy is also located in the submission. This file uses no SQL

imports.js

imports.js is a javascript file that contains any function that wants to be imported into every page.The functions it contains are:

- getRole(): returns the current users role from cookies
- getUser(): returns the current users username from cookies
- getPID(): returns the current users unique Person ID from cookies
- updateNav(): on page load, the header is generated and placed in the “header” on the page. It maps the links to all buttons, then pulls the users role and removes links to all pages they should not have access to. The 5 roles are Scientist, Data Curator, Administrator, Loggedout (Unauthenticated) and Universal (Authenticated). If it detects a user is not logged in on any pages that require authentication, a message will appear saying they must log in.
- checkPermissions(role): After the nav is updated, this will check the users role and wipe the main body and replace it with a message saying the user does not have permission.

landing.html/landing.js

landing displays basic credits and some information on what the user may view. There is no SQL.

login.html/login.jsp

login.html is the main login form, that calls login.jsp. The function uses MD5 hashing and salt tables to check encrypted passwords.

16) select U.USER_NAME, U.PASSWORD, U.ROLE, U.PERSON_ID, S.SALT from USERS U, SALTS S where U.USER_NAME=? and U.USER_NAME=S.USER_NAME

modiUser.jsp/modifyAccount.jsp/mymanage.jsp/mymodifyAccount.jsp

modiUser.jsp and mymanage.jsp are form pages for modifying your information. modiUsers is for admins to edit anyone, whereas mymanage allows users to modify some of their own details, which are then pass to modifyAccount and myModifyAccount, respectively. Both use the same backend, but accept different fields and changes. The majority of SQL statements are update and insert statements, with the other two main statements below. If a password is changed, a new MD5 salt must be used. In modifyAccount, if you change a userID, it will affect other users attached to the same person

17) select USER_NAME from USERS where USER_NAME=? AND USER_NAME<>?
18) select U.PASSWORD, S.SALT, P.email from USERS U, SALTS S, PERSONS P where U.USER_NAME=S.USER_NAME and P.PERSON_ID=U.PERSON_ID and U.USER_NAME=?

oceanstyler.css

A basic css file that controls our sites theme.

removeSensor.jsp

This is called when “Remove Sensor” is selected on sensormanage.jsp. It removes the sensor from the Sensor listings that has the “Remove Sensor” radio checked. In order to delete a sensor, all subscriptions, scalar data, images and audio recordings must be removed.

19) select SENSOR_ID from SENSORS where SENSOR_ID = "+sid;
20) select SENSOR_ID from SUBSCRIPTIONS where SENSOR_ID="+sid;
21) select SENSOR_ID from SCALAR_DATA where SENSOR_ID="+sid;
22) select SENSOR_ID from AUDIO_RECORDINGS where SENSOR_ID="+sid;
23) select SENSOR_ID from IMAGES where SENSOR_ID="+sid;

removeThatAccount.jsp

This is called when “Remove User” is selected on usermanage.jsp. It removes the user from the user listings that has the “Remove User” radio checked. Note that since a person can have multiple users, the associated person is not removed unless the account being removed is that person’s only account.

24) *select USER_NAME from USERS where USER_NAME=\'" + user +"\'";*
25) select USER_NAME from SALTS where USER_NAME=\'" + user +"\'";

search.jsp/search.js
A form that allows users to enter their search parameters. If a user searches by type they must also search by date. If no type is entered, a blank date assumes that all data is searched. This calls executeSearch.jsp. It has no SQL statements itself.

sensormanage.jsp
Sensor manage parses the list of all sensors in the system and creates a table out of them. The table contains all information on the sensor, as well as a radio button for deletion. The page also contains a form for adding a new sensor.

26) select SENSOR_ID, LOCATION, SENSOR_TYPE, DESCRIPTION from SENSORS
subscribe.jsp
Generates tables based on a users subscriptions. The first table shows all subscriptions with a radio to unsubscribe, and the second table shows all other sensors with an option to subscribe.

27) select S.SENSOR_ID, S.LOCATION, S.SENSOR_TYPE, S.DESCRIPTION from SENSORS S, SUBSCRIPTIONS T where S.SENSOR_ID=T.SENSOR_ID and T.PERSON_ID="+pid;
28) select distinct S.SENSOR_ID, S.LOCATION, S.SENSOR_TYPE, S.DESCRIPTION from SENSORS S, SUBSCRIPTIONS T where S.SENSOR_ID not in (select S.SENSOR_ID from SENSORS S, SUBSCRIPTIONS T where S.SENSOR_ID=T.SENSOR_ID and T.PERSON_ID="+pid+")";

unSubSensor.jsp
Removes a subscrition from a user-sensor pairing.
select SENSOR_ID from SUBSCRIPTIONS where SENSOR_ID = "+sid+" and PERSON_ID="+pid;
upload.html/upload.js/uploadToDatabase.jsp
upload.html contains a form that allows for JPEG, WAV or CSV files to be uploaded, and the data associated with them. It can tell what you're uploading and will make certain fields required based on that. It has no SQL, but calls uploadToDatabase.

uploadToDatabase parses the files. For images and audio, it creates the uploadeable blob, and for sensors it splits the csv file into lines and adds new entries for each IDs are assigned by querying the database for the highest previous ID, and adding 1.

29) select coalesce(MAX(i.image_id), 0), coalesce(MAX(a.recording_id), 0), coalesce(MAX(d.id), 0) from images i, audio_recordings a, scalar_data d

30) insert into IMAGES (image_id, sensor_id, date_created, description, thumbnail, recoreded_data) values (?,?,TO_DATE(?,'DD/MM/YYYY HH24:MI:SS'),?,?,?)

31) insert into scalar_data (id, sensor_id, date_created, value) values (?,?,TO_DATE(?,'DD/MM/YYYY HH24:MI:SS'),?)

32) insert into IMAGES (image_id, sensor_id, date_created, description, thumbnail, recoreded_data) values (?,?,TO_DATE(?,'DD/MM/YYYY HH24:MI:SS'),?,?,?)

usermanage.jsp/usermanage.js

User manage parses the list of all users in the system and creates a table out of them. The table contains all information in the user table, as well as radio buttons for modification, associating other users and deletion. These can be used to access and modify the Persons table as well. This page also contains a form for adding new users & persons at the same time.

33) select USER_NAME, ROLE, PERSON_ID from USERS order by PERSON_ID, USER_NAME, ROLE