System Architecture

Tech Stack

For this project, we used NodeJS with Express for server side and AngularJS for client side. For the database, we used Oracle's "oracledb" driver for NodeJS.

Documentation on these frameworks can be found here:
NodeJS: https://nodejs.org/docs/latest-v0.12.x/api/
ExpressJS: http://expressjs.com/
AngularJS: https://docs.angularjs.org/guide
OracleDB driver: http://www.oracle.com/technetwork/database/database-technologies/scripting-languages/node_js/index.html

Analysis Module

Description

This module was used to generate OLAP reports for an administrator that displays the number of images within a certain timeframe. The report is based on the combination of a number of factors that the administrator enters: username, subject, and the time that the image was taken (using the "Date Taken" attribute entered when the image is uploaded). There can only be one username and subject entered at a time, but while the username must be the exact username to avoid conflicts with users with similar usernames, the subject will be anything containing the specific keyword. The administrator can also enter a specific timeframe that the images were taken and choose to sort the result weekly, monthly, or yearly if needed. Any of the factors may be left blank, but the administrator must enter the sorting method.

Procedure and SQL Statements

When the administrator presses the "GENERATE" button, all of the information entered is submitted to the javascript controller for analysis. The controller will check that the information is in the correct format before sending it to the javascript service which then posts all of the information to the NodeJS server to make the database queries.

The SQL statement changes depending on the information entered. The query will take information from the SQL view “data_analysis” that contains the columns photo_id (from images), user_name (from persons), subject (from images), and timing (from images). The SELECT statement will have COUNT(photo_id) to calculate the number of images within the query and the dates depending on how the report was chosen to be sorted:

None: this portion is not included in the SELECT statement

Weekly: date = ‘to_char(NEXT_DAY(timing, 'SAT'), 'YYYY-MM-DD') AS " + '"Week", '

Monthly: date = ‘to_char(trunc(timing,'MM'), 'YYYY-MM') AS " + '"Month", '

Yearly: date = ‘to_char(trunc(timing,'yy'), 'YYYY') AS " + '"Year", '

Each of the date formattings will separate the results by their respective formats.

The WHERE clause will have the following operators if the fields were filled out by the administrator as needed:

Username: username_str = "(user_name = :username) "

Subject: subject_str = "(subject LIKE '%" + subject + "%') "

Timeframe: timeframe_str = "(timing BETWEEN TO_DATE (:timeStart, 'yyyy/mm/dd') AND TO_DATE (:timeEnd, 'yyyy/mm/dd')) "

The GROUP BY clause will also change to reflect the format chosen in the SELECT clause:

None: the GROUP BY clause is not included

Weekly: group_by = “CUBE(to_char(NEXT_DAY(timing, 'SAT'), 'YYYY-MM-DD'))”

Monthly: group_by= “CUBE(to_char(trunc(timing,'MM'), 'YYYY-MM'))”

Yearly: group_by= "CUBE(to_char(trunc(timing,'yy'), 'YYYY'))"

As a result, this is the full SQL statement assuming that all fields are filled:

(Parameters: username, timeStart, timeEnd)

SELECT date COUNT(photo_id) FROM data_analysis WHERE username_str AND subject_str AND timeframe_str GROUP BY group_by

After the results matching the query have been computed, they are sent back to the controller so that the information can be displayed to the administrator. The table changes depending on how the report was chosen to be sorted.

None: the total number of images is displayed.

Weekly: the number of images for each week starting on a certain date (week starting on Saturday) is displayed.

Monthly: the number of images for each month is displayed.

Yearly: the number of images for each year is displayed.

The administrator may choose to generate a different report afterwards if needed.

Display Module

Description

The display module shows three sections: a user's personal uploaded images, images that the user is allowed to view (their own, public, and ones that are permitted in the correct group), and popular images. If the user is an administrator, there is an additional section for viewing all images regardless of the image permissions. The user can click on any image to view its full details. If the user is the owner of the image, then the user can also choose to edit the image.

Procedure and SQL Statements

When the user is taken to the display page, the javascript controller retrieves the images and information required to display each of the galleries. The controller sends the user's username to the javascript service for the specific personal and public galleries, while it just calls a function in the service to get the popular and administrator galleries if needed. The service then sends the required information to the NodeJS server to execute the database queries.

For the user's personal gallery, the images table is queried for any images that has the owner_name as the username.

(Parameters: userName)

SELECT * FROM images WHERE images.owner_name = :userName

For the user's public gallery, the images table is queried for any images that have their permission set to public (group_id = 1) or has their permission set to a group's id that the user is a part of.

(Parameters: userName)

SELECT * FROM images WHERE images.permitted IN

(SELECT group_id FROM group_lists WHERE group_lists.friend_id = :userName) OR images.permitted = 1

For the user's popular gallery, the image_tracking and images table are queried for the images that have the top five numbers of distinct username clicks.

(Parameters: none)

with order1 as (SELECT * from (select tr.photo_id, COUNT(tr.photo_id) as photo_count from images i, image_tracking tr where i.photo_id = tr.photo_id AND i.permitted != 2 group by tr.photo_id ORDER BY photo_count DESC) where photo_count IN (select distinct COUNT(tr.photo_id) as photo_count from images i, image_tracking tr where i.photo_id = tr.photo_id AND i.permitted != 2 group by tr.photo_id order by photo_count DESC FETCH FIRST 5 ROWS ONLY))

select images.photo_id, images.owner_name, images.permitted, images.subject, images.place, images.timing, images.description, images.thumbnail, images.photo from images, order1 where images.photo_id = order1.photo_id order by order1.photo_count desc

For the administrator's gallery, all images are retrieved from the image table.

(Parameters: none) SELECT * FROM images

The all of the images are added to their respective arrays in the controller and their thumbnails are displayed on the page.

If a user clicks on a thumbnail, the image's full image and description are shown. The controller will also update and keep track of the number of distinct usernames that have clicked on an image. When the thumbnail is clicked, the controller passes the image's photo_id and the user's username into the service, which then sends the information to the server, to update the tracking if required.

The server will first check if the user has clicked on the image before.

(Parameters: photo_id, userName)

SELECT * FROM image_tracking WHERE photo_id = :photo_id AND user_name = :userName

If the entry does not exist in the database, then the server adds it to the image_tracking table.

(Parameters: photo_id, userName)

INSERT INTO image_tracking (PHOTO_ID, USER_NAME) VALUES (:photo_id, :userName)

If the user is the owner of the image, the user also has access to a button that sends the user to the Edit module where the image viewed can be edited.

Display Group Module

Description

This module displays all the members of a specific group, sent from the group chosen by the user in the Group module. The user can add or delete members in the group, as well as delete the entire group. This page can only be accessed by the user that created the group.

Procedure and SQL Statements

When the user enters the page, the names of the members in the group and the dates that they were added are displayed in the table. The group_id is used as a parameter to get the group's information.

(Parameters: groupID)

SELECT friend_id, date_added FROM GROUP_LISTS WHERE GROUP_ID = :groupID

If the user adds a user to the group list, then the controller uses the username added and the group_id to add them to the group list.

(Parameters: group_id, user_name, notice)

INSERT INTO GROUP_LISTS (GROUP_ID, FRIEND_ID, DATE_ADDED, NOTICE)
VALUES (:group_id, :user_name, CURRENT_DATE, :notice)

If the user removes a user from the group list, then the controller uses the username to be removed and the group_id to remove them from the group list.

(Parameters: group_id, user_name, notice)

DELETE from GROUP_LISTS WHERE group_id = :group_id

AND FRIEND_ID = :user_name

If the user chooses to delete the entire group:

All of the images that are sharing to that group are set to private (group_id = 2)

(Parameters: group_id)

UPDATE images SET permitted = 2 WHERE permitted = :group_id

All of the group_list entries with the group's group_id are removed from the group_lists table

(Parameters: group_id) DELETE from GROUP_LISTS WHERE group_id = :group_id

The group is able to be removed from the groups table after the above dependencies are removed

(Parameters: group_id) DELETE from GROUPS WHERE group_id = :group_id

Edit Module

Description

This module allows the user to edit or remove an image. The user may only edit the image's permission, subject, place, timing, and description, and not the image itself. The image's edit page can only be displayed to the user that uploaded the image.

Procedure and SQL Statements

When the user is sent to this module from the display module, the page information is populated with information passed from the display module. From here, the user may make changes to the image's permission, subject, place, timing, and description as required.

If the user chooses to edit the image's information, then the image's photo_id and information is sent to the server to be updated in the database.

(Parameters: photo_id, permitted, subject, place, timing, desc)

UPDATE images SET permitted = :permitted, subject = :subject, place = :place, timing = TO_DATE (:timing, 'yyyy/mm/dd'), description = :desc WHERE photo_id = :photo_id

If the user chooses to delete the image:

The image's tracking information is removed from the image_tracking table

(Parameters: photo_id)

DELETE from IMAGE_TRACKING WHERE PHOTO_ID = :photo_id

The image can be removed from the images table after the above dependency is removed

(Parameters: photo_id)

DELETE from IMAGES WHERE PHOTO_ID = :photo_id

Groups Module

Description

This module displays groups that the user owns and is a member of. Groups that the user owns can be accessed in further detail by clicking on a group in the Owned Groups section, which sends the user to the Display Group module. Groups that the owner is only a member of cannot be accessed in further detail. The user is also able to create a new group from this page.

Procedure and SQL Statements

When the user is sent to this module, the javascript controller uses the user's username as a parameter to retrieve the following information:

The groups that the user owns

(Parameters: userName)

SELECT * FROM GROUPS WHERE GROUPS.USER_NAME = :userName

The groups that the user is only a member of

(Parameters: userName)

SELECT l.group_id, g.group_name FROM GROUP_LISTS l, GROUPS g

WHERE l.FRIEND_ID = :userName AND l.GROUP_ID = g.GROUP_ID

If the user chooses to create a new group, then the controller does the following:

Retrieves the current maximum group_id under the groups table and adds 1 to create a new group_id

(Parameters: none)
SELECT GROUP_ID FROM groups ORDER BY GROUP_ID DESC
Uses the new group_id, group_name, user's username, and the current date to create a new group in the groups table

(Parameters: groupID, userName, groupName)
INSERT INTO GROUPS (GROUP_ID, USER_NAME, GROUP_NAME, DATE_CREATED)
VALUES (:groupID, :userName, :groupName, CURRENT_DATE)

Uses the new group_id, user's username, and the current date to create a new entry in the group_lists table

(Parameters: groupID, userName, notice)
INSERT INTO GROUP_LISTS (GROUP_ID, FRIEND_ID, DATE_ADDED, NOTICE)
VALUES (:groupID, :userName, CURRENT_DATE, :notice)

Homepage Module

Description

This module is displayed when the user logins to the application and shows navigation buttons to the modules Upload, Display, Groups, and Search. If the logged in user is an administrator, then a navigation button to the Analysis module is also shown. The user is also able to logout from this page.

Procedure

When the user is sent to this module, the javascript controller checks if the user is logged in to the application. If not, then it sends the user back to the login module. From this page, the user is able to click on a navigation button to be taken to the Upload, Display, Groups, and

Search modules. The controller also checks if the user is an administrator and if so, then the navigation button to the Analysis module is also shown. If the user chooses to logout of the session, then the local storage is cleared of the username and the user is taken back to the login module.

Login Module

Description

This module allows the user to login to the application. It also displays a navigation button to take the user to the Register module.

Procedure and SQL Statements

When the user clicks the "LOGIN" button, then the javascript controller takes the entered information and sends it to the server to try and login the user. In order to login successfully, the user needs to have an entry in the users table. To check this, the following SQL query is used:

(Parameters: user_name, password)

```
SELECT *  FROM users WHERE user_name = :user_name AND password = :password
```

If this returns a result row, then the user is logged in to the application and the user's username is saved in the local storage to indicate that the user is logged in.

If the user chooses to register a new user, then they are sent to the Register module.

Register Module

Description

This module allows the user to register a new user into the system with the given username, password, first name, last name, address, email, and phone number. The username and email must be unique and not already in the database.

Procedure and SQL Statements

When the user submits the registration information, the controller sends all of the information to the server. To register a user the following steps must be performed:

Check if username and email already exists in the database

(Parameters: user_name, email)

SELECT * FROM persons WHERE persons.user_name = :user_name OR persons.email = :email

If this returns a result row, then the username or email already exists in the system and an error occurs. The user must enter a different username or email. If not, then the next step is executed.

Register the new user in the user table using the username and password

(Parameters: username, password)

INSERT INTO users (USER_NAME, PASSWORD, DATE_REGISTERED) VALUES (:username, :password, CURRENT_DATE)

Register the new user into the persons table using the username, first name, last name, address, email, and phone number

(Parameters: username, first_name, last_name, address, email, phone)

INSERT INTO persons (USER_NAME, FIRST_NAME, LAST_NAME, ADDRESS, EMAIL, PHONE) VALUES (:username, :first_name, :last_name, :address, :email, :phone);

Once the new user has been registered into the database, then the user is sent back to the Login module to login with the new credentials.

Search Module

Description

This module will be used by a registered user to specify the condition to search the database for a list of desired images by a list of key words, and/or a time period. Results will be based on permissions and the following ranking algorithm:

Rank(photo_id) = 6*frequency(subject) + 3*frequency(place) + frequency(description)

Where frequency(column_name) represents the number of occurrences of the key word(s) in the respective column (as returned by the SCORE function of the corresponding CONTAINS function

Procedure and SQL Statements

There are 3 possible scenarios.

Scenario 1: User searches by keyword only.

```
SELECT photo, thumbnail, description, timing, place, subject, permitted, owner_name, photo_id, score(1) , score(2) , score(3)  FROM IMAGES WHERE (images.permitted IN (SELECT group_id FROM group_lists WHERE group_lists.friend_id = 'nopics') OR images.permitted = 1 OR (images.permitted = 2 AND images.owner_name = 'nopics')) AND (contains(subject, 'user1', 1) > 0 OR contains(place, 'user1', 2) > 0 OR contains(description, 'user1', 3) > 0 )  ORDER BY score(1) * 6 + score(2) * 3 + score(3) * 1  DESC
```

Scenario 2: User searches by timing only.

```
SELECT * FROM IMAGES WHERE (timing BETWEEN TO_DATE ('2016/03/01', 'yyyy/mm/dd') AND TO_DATE ('2016/04/01', 'yyyy/mm/dd'))
```

Scenario 3: User searches by keyword and timing.

```
SELECT photo, thumbnail, description, timing, place, subject, permitted, owner_name, photo_id, score(1) , score(2) , score(3)  FROM IMAGES WHERE (timing BETWEEN TO_DATE ('2016/03/01', 'yyyy/mm/dd') AND TO_DATE ('2016/04/01', 'yyyy/mm/dd')) AND (images.permitted IN (SELECT group_id FROM group_lists WHERE group_lists.friend_id = 'nopics') OR images.permitted = 1 OR (images.permitted = 2 AND images.owner_name = 'nopics')) AND (contains(subject, 'user1', 1) > 0 OR contains(place, 'user1', 2) > 0 OR contains(description, 'user1', 3) > 0 )  ORDER BY score(1) * 6 + score(2) * 3 + score(3) * 1 DESC
```

If a User is not an admin, the following condition is added (where usernameFromStorage is the User's username):

```
(images.permitted IN (SELECT group_id FROM group_lists WHERE group_lists.friend_id = ' usernameFromStorage OR images.permitted = 1 OR (images.permitted = 2 AND images.owner_name = usernameFromStorage))
```

Upload Module

Description

A registered user may use this module to upload images to the database, and to enter and to update descriptive information for uploaded images.

Procedure and SQL Statements

Lobs and blobs were incredibly difficult to parse with NodeJS and Oracledb. We required many asynchronous tasks to successfully write to the database. Fortunately, Oracle provided lots of documentation on how to properly save image data.

Step 1: Insert an empty blob where our images need to be saved

Step 2: Return the empty blob and write our base 64 representation of the image onto the blob. This required writing to an array buffer stream. Once we have finished writing both photo and thumbnail, we move to step 3.

Step 3: Update the row, replacing the empty blob with our new image data lob. Using our base 64 representation in Step 2, we can pipe the image data onto the lob.

Query:

```
INSERT INTO images (photo_id, owner_name, permitted, subject, place, timing, description, thumbnail, photo) VALUES (:photo_id, :owner_name, :permitted, :subject, :place, TO_DATE (:timing, 'yyyy/mm/dd'), :description, EMPTY_BLOB(), EMPTY_BLOB()) RETURNING photo, thumbnail INTO :lobbv, :lobtn
```