CMPUT 391 Project Report

Group 10: Eden Mar, Logan Lin, Sabrina Gannon

[Homepage]

index.jsp

- Provides links to all our other modules
- Redirects to the login page if user is not yet logged in

[User Management Module]

login.jsp ==> doLogin.jsp

- users are automatically redirected from any page to this one if they have not yet logged in
- allows the user to login by providing their username and password
- the user's credentials are submitted to doLogin.jsp via regular form submission
- checks the user's credentials before instantiating a session and setting the session variable "userid" to their username
- SQL query for password checking: "select password from users where user_name = '"+userName+"'"

logout.jsp

- can be accessed from any page via our top bar
- logs out the user's current session
- uses session.invalidate() to clear the current session
- sends the user back to the login page (login.jsp) once they have been logged out

new_user_registration.html ==> new_user_registration.jsp

- can be accessed from the login.jsp page
- allows a user to register an account
- inserts the user's provided information into the database
- the user's information is submitted to new_user_registration.jsp via regular form submission
- SQL query for inserting username and password: "insert into users values ('"+username+"', '"+password+"', SYSDATE)"
- SQL query for inserting optional info: "insert into persons values('"+username+"', '"+firstName+"', '"+lastName+"', '"+address+"', '"+email+"', '"+phoneNumber+"')"
- sends the user back to the homepage after successfully registering

myinfo.jsp

- can be accessed from most pages via our top bar
- allows the user to view their own information
- SQL query for retrieving info: "SELECT first_name, last_name, address, email, phone FROM persons WHERE user_name = '"+userid +"'"
- displays each piece of retrieved information in a text box

help.jsp

- can be accessed from most pages via our top bar
- provides the user with the same information as outlined in userDoc.txt, basic installation and operation information of our site.

[Uploading Module]

upload.jsp ==> uploadOne.jsp

- Can be accessed from the homepage (index.jsp)
- Allows the user to upload a single image (.jpg or .gif file)
- Allows user to select one file and optionally provide information about the image's subject, location, time taken, and description
- Allows user to additionally specify if the image should be set to private, public, or restricted to one group (allows the user to input a group name for the last option)
- Submits the image and associated information to uploadOne.jsp via form submission with enctype="multipart/form-data"

uploadOne.jsp

- Uses Apache's FileUpload java package to extract the image file and information strings out of the encoded parameters
- Inserts the image and associated information into the database using this SQL query: "INSERT INTO images VALUES(" + Integer.toString(pic_id) + ", '" + userid + "', " + permitted + ", '" + subject + "', '" + place + "', TO_DATE('" + when + "','YYYY MM DD'), '" + desc + "', " + " empty_blob(), " + "empty_blob() " + ")"
- uploadOne.jsp then fills in the two empty blobs with the image file's data and the same data but shrunk for the thumbnail
- Uses the BufferedImage class to shrink the image data in order to produce thumbnails
- If the group-only permission was specified along with a group name, we retrieve the id of the specified group using this SQL query: "SELECT group_id from groups where group_name = '" + groupName + "' and " + "user_name = '" + userid + "'"
- SQL query for retrieving the current highest image ID (we add one to this number to get the next unused image ID): "select max(photo_id) from images"
- Also stores the extension of the image using this SQL query: "INSERT INTO photoExt values " + "(" + Integer.toString(pic_id) + ", '" + FilenameUtils.getExtension(fullpath) + "')"

uploadFolder.jsp ==> uploadMultiple.jsp

- Can be accessed from the homepage (index.jsp)
- Allows the user to multiple images at a time (any combination of .jpg and/or .gif files that are in the same folder)
- Allows user to select one or more files, and optionally provide information about the images' subject, location, and time taken (but not description, which will always be initialized to a blank value)
- Allows user to additionally specify if the images should be set to private, public, or restricted to one group (allows the user to input a group name for the last option)
- Submits the images and associated information to uploadMultiple.jsp via form submission with enctype="multipart/form-data"

uploadMultiple.jsp

- Uses Apache's FileUpload java package to extract the image files and information strings out of the encoded parameters
- Inserts each image and the common associated information into the database using this SQL query: "INSERT INTO images VALUES(" + Integer.toString(pic_id) + ", '" + userid + "', " + permitted + ", '" + subject + "', '" + place + "', TO_DATE('" + when + "','YYYY MM DD'), '" + desc + "', " + " empty_blob(), " + "empty_blob() " + ")"
- uploadMultiple.jsp then fills in the two empty blobs with each image file's data and the same data but shrunk for the thumbnail
- Uses the BufferedImage class to shrink the image data in order to produce thumbnails
- If the group-only permission was specified along with a group name, we retrieve the id of the specified group using this SQL query: "SELECT group_id from groups where group_name = '" + groupName + "' and " + "user_name = '" + userid + "'"
- SQL query for retrieving the current highest image ID (we add one to this number to get the next unused image ID's): "select max(photo_id) from images"
- Also stores the extension of each image using this SQL query: "INSERT INTO photoExt values " + "(" + Integer.toString(pic_id) + ", '" + FilenameUtils.getExtension(fullpath) + "')"

[Security Module]

groups.jsp ==> editGroup.jsp, doCreategroup.jsp

- Can be accessed from the homepage (index.jsp)
- Allows the user to create a new group, and manage the groups they have created
- Obtains the list of owned groups with this SQL query: "select group_name from groups where user_name = '" +userid +"'"
- Displays each group the user has created, with a View/Edit button beside the group name allowing the user to manage the specific group
- Submits the name of the group to manage to editGroup.jsp via POST using Javascript
- If the user wishes to create a new group, the page gets their desired group name using a Javascript prompt() popup box, and then submits this new group name to doCreategroup.jsp again via POST using

Javascript
- The page allows the user to navigate back to the homepage (index.jsp)

doCreategroup.jsp

- Inserts the new group into the database using this SQL query: "insert into groups values (" + Integer.toString(newGroupId) + ", '" + userid + "', '" + groupname + "', SYSDATE)"
- SQL query for retrieving the current highest group ID (we add one to this number to get the next unused group ID): "select max(group_id) from groups"
- Also inserts the group owner into the database as a member of the group he has just created using this SQL query: "INSERT INTO group_lists values (" + Integer.toString(newGroupId) + ", '" + userid + "', SYSDATE, 'This is me'" + ")"
- After informing the user of success or failure, the page allows the user to navigate back to the Groups page (groups.jsp)

editGroup.jsp ==> addGroupMember.jsp, deleteGroupMember.jsp

- Allows the user to add members to a group they own, or remove existing members
- Retrieves all members in the specified group using this SQL query: "select friend_id from group_lists where group_id in " + "(select group_id from groups where group_name = '" + groupName + "' and user_name = '" + userid + "')"
- Displays the current list of group members, with a Delete button next to each member's name
- Submits the name of a member to be deleted to deleteGroupMember.jsp via POST using Javascript
- If the user wishes to add a new member, the page gets the name of the new member using a Javascript prompt() popup box, and then submits the new member's name to addGroupMember.jsp via POST using Javascript
- The page allows the user to navigate back to the Groups page (groups.jsp)

addGroupMember.jsp

- Inserts the new group member into the database using this SQL query (uses a select statement nested in the insert statement to get the group ID from the group name): "insert into group_lists " + "select group_id, " + "'" + newName + "', SYSDATE, '" + notice + "' from groups where group_name = '" + groupName + "' and user_name = '" + userid + "'"
- After informing the user of success or failure, the page allows the user to navigate back to the group management page (editGroup.jsp); the page resubmits the group name to the management page via POST using Javascript

deleteGroupMember.jsp

- Deletes the specified group member from the group's roster with this

SQL query (again there is a nested select statement in the delete statement in order to obtain the group ID from the group name): "delete from group_lists where group_id in " + "(select group_id from groups where group_name = '" + groupName + "' and user_name = '" + userid + "')" + " and friend_id = '" + deletedMember + "'"
- After informing the user of success or failure, the page allows the user to navigate back to the group management page (editGroup.jsp); this page resubmits the group name to the management page via POST using Javascript

[Display And Search Module]

viewImages.jsp ==> doViewImages.jsp

- Can be accessed from the homepage (index.jsp)
- Allows the user to view images in a variety of ways:
  - View all images the user has uploaded
  - View the most popular images (at least 5 images, possibly more if there are ties in popularity)
  - Search for images either by keyword with optional sort by date, or by date
- Submits the user-provided parameters for image-viewing to doViewImages.jsp via regular form submission
- The page allows the user to navigate back to the homepage (index.jsp)

doViewImages.jsp ==> viewOneImage.jsp

- Retrieves and displays a list of image thumbnails corresponding to the submitted viewing parameters
- For owned images, uses the SQL query: "select photo_id from images where owner_name = '" + userid + "'"
- For popular images, uses the SQl query: "select p.photo_id, i.permitted, count*() from " + "images" + " i, popularity p where i.photo_id = p.photo_id " + " GROUP BY p.photo_id, i.permitted ORDER BY count*()* DESC"
- For keyword search, uses the SQL query: "select photo_id, permitted from (select (6*score(1) + 3*score(2) +1*score(3)) as Score, photo_id, permitted from images where contains(subject, ?, 1) >0 or contains(place, ?, 2)>0 or contains(description, ?, 3)>0)"
- For date search, the SQL query is: "select photo_id, permitted, timing from images where timing between TO_DATE('"+lower+"','YYYY MM DD') and TO_DATE('"+upper+"','YYYY MM DD') order by timing asc"
- Additionally, the page enforces security by ensuring that the user can never see the thumbnail of an image they do not have permission to view
- The page allows the user to click a thumbnail, which submits the corresponding image ID to viewOneImage.jsp via POST using Javascript, where the user will be able to view the full resolution version of the image as well as its associated information
- Fetching the actual thumbnails is delegated to the image-fetching servlet GetOnePic.class (GetOnePic.java)
- The page allows the user to navigate back to the View Image page

(viewImages.jsp)

viewOneImage ==> GetOnePic.java, updateImage.jsp

- Displays the full resolution version of a requested image, along with the image's associated information (subject, place, time, description, and permissions info), and allows the user to update all such information if they are the owner of the image
- Delegates the task of fetching the image to the image-fetching servlet GetOnePic.class (GetOnePic.java)
- Retrieves the image's associated information with this SQL query: "select" + " owner_name, permitted, subject, place, timing, description " + "from images where photo_id = " + Integer.toString(photoId)
- As a security feature, the page checks if the user has permission to view the requested image before fetching the image
    - if the image can only be viewed by a certain group, the page checks if the user belongs to that group using this SQL query: "SELECT 1 from group_lists where group_id = " + permitted + " and friend_id = '" + userid + "'"
- The page will update the image's popularity rating if it is the user's first time viewing the image, using the following SQL query: "INSERT INTO popularity (photo_id, username) SELECT " + Integer.toString(photoId) + ", '" + userid + "'" + " from dual where NOT EXISTS (" + "SELECT * from popularity where photo_id = " + Integer.toString(photoId) + " and username = '" + userid + "'" + ")"
- If the user owns the image, the page displays Edit/Change buttons beside each piece of associated information for the photo
- When the user wishes to Edit a piece of information, the page gets the desired new value for that field using a Javascript prompt() popup box, and then submits the new value to updateImage.jsp via POST using Javascript
- The page allows the user to navigate back to the View Image page (viewImages.jsp). Alternatively, the user may press the back button on their browser to navigate back to their View Image Results page (doViewImages.jsp)

GetOnePic.java

- This servlet retrieves and returns images from the database; the image ID is specified in the URL
- Retrieves and returns a single image thumbnail using this SQL query: "select thumbnail from images where photo_id=" + picid
- If the "big" prefix is used, the servlet retrieves and returns the full resolution version of the image using this SQl query: "select photo from images where photo_id=" + picid
- In order to set the response's content type correctly, the servlet retrieves the image's original extension from the database using the following SQL query: "select extension from photoExt where photo_id = " + picid

updateImage.jsp

- Updates the specified field (the "attribute" parameter) to the specified new value using the following SQL query: "UPDATE images SET " + setter + " WHERE photo_id = " + imageId
  - where the setter variable is one of the following:
  - for updating time: "timing = " + "TO_DATE('" + newValue + "','YYYY MM DD')"
  - for updating permissions: "permitted = " + permitted
  - for all other attributes: attribute + " = '" + newValue + "'"
- if the permissions on the image are being changed to group-only, the page converts the specified group name to a group ID using the following SQL query: "SELECT group_id from groups where group_name = '" + groupName + "' and " + "user_name = '" + userid + "'"
- After informing the user of success or failure, the page allows the user to navigate back to the View One Image page (viewOneImage.jsp); this page resubmits the image ID to the View One Image page via POST using Javascript

[Data Analysis Module]

adminModule.jsp ==> Olap.jsp

- Can be accessed from the homepage (index.jsp), but only by the "admin" user
- If a user other than admin attempts to access this page, they will be redirected to the homepage
- Allows the administrator to perform OLAP operations on the "images" table of the database
- The admin is able to analyze the table by image uploader, image subject, and image time. He may later drill down or roll up this analysis
- the parameters for analysis are submitted to Olap.jsp via regular form submission
- The page allows the user to navigate back to the homepage (index.jsp)

Olap.jsp

- Performs an OLAP operation using the specified parameters and displays the results in a table
- An example query for analyzing by all three parameters (owner name, subject, and timing), broken down by week: select owner_name, subject, to_number(to_char(timing, 'ww')) as Week, count(*) as count from images group by owner_name, subject, to_number(to_char(timing, 'ww')) order by count asc
- For analyzing by only subject and timing, broken down by month: select '__', subject, to_number(to_char(timing, 'mm')) as Month, count(*) as count from images where owner_name = 'username' group by subject, to_number(to_char(timing, 'mm')) order by count asc
- And for analyzing by timing only, broken down by year: select '**,** '', to_number(to_char(timing, 'mm')) as Month, count(*) as count from images where owner_name = 'username' and subject = 'subject' group by to_number(to_char(timing, 'mm')) order by count asc
- Allows the user to drill down or roll up along the time dimension by providing a form for the user to re-specify the time period out of week,

month, and year
- The page would then resubmit the newly selected time frame and all the same old parameters back to Olap.jsp via POST using Javascript
- The page allows the user to navigate back to the homepage (index.jsp)