## ***Note: Part 3 features start on page 3.***

## **Introduction:**

Cytokines are small proteins secreted by the immune system of a person. Secretion patterns of these cytokines especially become important with trauma patients as it is used to determine their treatment process. This application provides the UI to manage and access this data.

## **Functionalities and Access Level Restrictions:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl. No.** | **Functionalities** | **Script details** | **Can be accessed without login** | **Access Levels** |
| 1 | Search the content of this application | [search.php](http://home.php/) | Yes | Guests, Moderators, Admin, Visitors without login |
| 2 | Insert Cytokine data of a patient | input.php (main) processing.php (secondary) | No | Moderators, Admin |
| 3 | Access Cytokine data and heatmap images of the selected demography | get.php (main) query.php (secondary) | No | Guests, Moderators, Admin |
| 4 | Dashboard to track website activities | dashboard.php | No | Admin |
| 5 | User dashboard | User\_dashboard.php | No | Guests, Moderators, Admin |
| 6 | Add Favourities, Create profile vectors and update user search sort order | Query.php (line 247-419) | No | Guests, Moderators, Admin |
| 7 | Recommendations | Recommendation.php | Yes | Guests, Moderators, Admin, Visitors without login |
| 8 | Login and Registration System | login.php registeration.php | Yes | Guests, Moderators, Admin, Visitors without login |
| 9 | logout | logout.php | No | Guests, Moderators, Admin |

## **Home page(home.php)**

Home page (home.php). This page has links to the above functionalities depending on the type of the user. Also, this has the buttons to login and register if the user has not logged in or a button to logout if the user has logged in already.

## **Login and Registration system**

This system includes a table in the database called users with columns – first name, last name, email id(unique key), password, and user type. Any user **registered** through the registration page will be recorded in this table. By default, all users will be first registered as ‘guest’ user and can later be modified as either ‘moderator’ or ‘admin’. Privileges of each of these user types are described in section – ‘Functionalities and Access Level Restrictions’.

Any user trying to **login** with email and password will be verified against the user's table data. Once logged in, all the details of a user will be set as session variables. User\_type and email(session id) will be used on each page of the application to grant or restrict access depending on the access level restrictions. Also, the logins will be recorded in the database.

Logged-in users can **log out** of the application at any time and the logout button is available on each page. Once logged out session will be killed, and session variables will be removed. Also, all logouts will be recorded in the database.

## **Data Insert (input.php)**

This gives the below UI to insert the cytokine data. It includes Html form to demographic details such as gender, age, patient outcome. The ‘action’ of this form or clicking the submit button will lead to ‘processing.php’ which has the database connects and insert query to capture the entered data.

## **Get the cytokine data based on selected demography (get.php)**

This script has PHP script embedded in Html body to capture the below details. Based on the values entered by the user, an appropriate query is executed in script query.php and the results get displayed. Query.php is the ‘action’ of the form in get.php. Also, any data access activity will be logged in the DB.

Based on the action selected either the data get displayed or the correlation heatmap image from the MySQL database.

## **Admin Dashboard**

Activities logged in input, access, search and login/logout pages will be used to display the below information on the dashboard. This page can be accessed only by admin.

1. Registered users for each user type
2. Login counts for each user type
3. Top 5 users with their activity count with regards to data inserts and data accesses
4. Top 5 contents which are accessed by users
5. Top 5 search tokens which gave zero search result. These token if make sense can be used to update the search table’s associated words column to better the search result.

## **User Dashboard**

Activities logged in input, access, search and login/logout pages will be used to display the below information on this user-specific dashboard. This page can be accessed all logged in users.

1. Login counts for each user type
2. User Activity counts with regards to data read and data inserts
3. Top 5 contents accessed by the user.

# **Part 3 features: Favourites, Search Sort Order, and Recommendations**

## **Favourites [Query.php (line 247-419)]**

Users can mark any content accessed as ‘Favourite’ content for easy access in future logins. When a user clicks on ‘Add to Favourite’ link, below steps take place:

1. the content id is entered in favourites database table against the user
2. User profile Vector gets updated
3. User search sort order gets updated
4. Same content page reloads but now with the option to ‘Remove from Favourites’

Similarly, when someone clicks on ‘remove from favourites’, content id against the user is removed from the favourites table, user profile vector, search sort order gets updated, and the same content page reloaded with ‘add to favourites’ option.

## **User Profile Vector**

User profile vector is a list of keywords which outline the contents of this application, along with their count and average appearance in the favourite contents of a user. Whenever user clicks on ‘Add to favourite’/’remove from favourites’ option for a content, below steps take place to update user\_profiles database table:

1. Words in the content name are broken into tokens
2. Counts of these tokens are incremented/decremented against the user
3. Averages of all keywords get updated against the user. Averages for all keywords are updated because the number of favourites has changed for the user.

Below is the code that can be found in query.php

1. $cont\_name\_lower = strtolower($cont\_name);
2. $cont\_name\_tokens = explode(" ", $cont\_name\_lower);
3. foreach($profile\_words as $word){
4. if(in\_array($word,$cont\_name\_tokens)){
5. $count\_increment\_sql = 'update user\_profiles set count=count+1 where email="'.$user\_id.'" and word="'.$word.'";';
6. mysqli\_query($conn, $count\_increment\_sql);
7. }
8. $avg\_sql = 'update user\_profiles set average=count/'.$favourites\_count.' where email="'.$user\_id.'" and word="'.$word.'";';
9. mysqli\_query($conn, $avg\_sql);
10. }

## **User Search Sort Order**

User Search Sort Order table holds the relation score for each content against each user. This relation score is used in descending order to sort the search results for a given user. Below are the steps followed to update the relation score for each content of the application against a single user whenever a user clicks on on ‘Add to favourite’/’remove from favourites’:

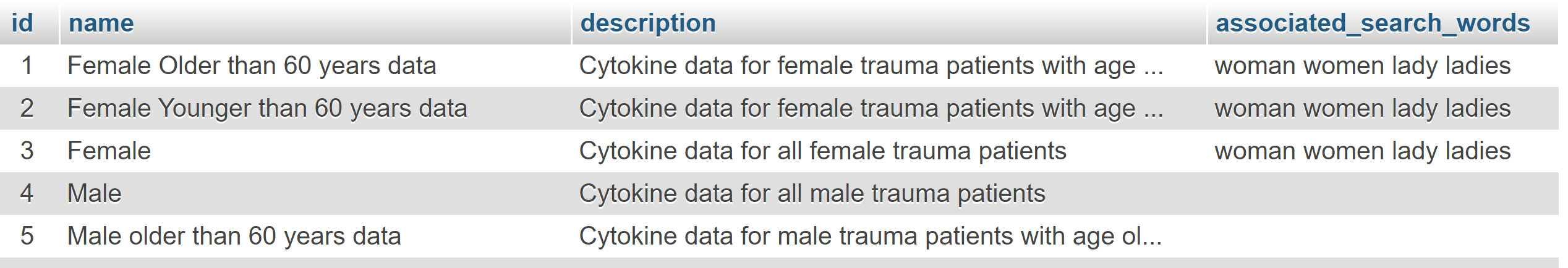
1. Words in the content name are broken into tokens
2. Each of these tokens’ average appearance for a user in the user profile vector table is summed up to get the relation score.

Below is the code that can be found in query.php

1. $all\_contents = array(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,28,29);
2. $delete\_search\_sort\_sql = 'delete from user\_search\_sort where email="'.$user\_id.'";';
3. mysqli\_query($conn, $delete\_search\_sort\_sql);
4. foreach($all\_contents as $cid){
5. $cont\_name\_sql = 'select name from search\_table where id='.$cid.';';
6. $result = mysqli\_query($conn, $cont\_name\_sql);
7. $row = mysqli\_fetch\_assoc($result);
8. $cont\_name = $row["name"];
10. $cont\_name\_lower = strtolower($cont\_name);
11. $cont\_name\_tokens = explode(" ", $cont\_name\_lower);
12. $relation\_score = 0.0;
13. foreach($cont\_name\_tokens as $token){
14. if(in\_array($token,$profile\_words)){
15. $score\_sql = 'select average from user\_profiles where email="'.$user\_id.'" and word="'.$token.'";';
16. $result = mysqli\_query($conn, $score\_sql);
17. $row = mysqli\_fetch\_assoc($result);
18. $score = $row["average"];
19. $relation\_score = $relation\_score + $score;
20. }
21. }
22. $search\_sort\_sql = 'INSERT INTO user\_search\_sort(email, content\_id,relation\_score)values("'.$user\_id.'",'.$cid.','.$relation\_score.');';
23. mysqli\_query($conn, $search\_sort\_sql);
24. }

## **Search content (search.php)**

First search names and descriptions are stored in the database as shown below. Any search token will be queried against this table to return search results. Search queries with zero results will be logged in the database.



Based on the search token entered in search.php, search results are the result of below 5 queries:

1. Exact match with name column
2. Starting with the search token value in name column
3. Name column containing search token value
4. Description column containing search token value
5. Associated words column containing token value

Search.php has the database connection and parameterized query to get the above results.

**Data Sanitization:** Any search token will be sanitized by removing the special characters and common words in the search.php. Only then these search tokens will be queried against the database to return search results.

In the case of **multiple tokens**, after data sanitization, it gets split into multiple tokens. For each token above 5 search queries is used and search results will be displayed for the below criteria:

1. Results common for all tokens
2. Results with only the first token
3. Results with only the second token and so on.

When displaying search results link to the corresponding webpage(php script) is added through href.

**Lazy login feature:** While the search results can be viewed by anyone and even without login. Accessing the displayed link requires a login and the application displays a‘ login required’ message if anyone tries to access content without login.

## **Search Sort Order**

Once the contents resulting from search queries is obtained, each of the resulting content’s relation score for the user is looked up from the user search sort table. This score is used in descending order to sort the results. Sort order is applied only for search results of the registered logged-in users.

## **Recommendations**

Below steps are followed to get recommendations for a user:

1. Get the list of favourite contents of a user.
2. For the favourite contents from step 1, get the list of users who have it as favourites
3. For each of the users from step 2, get all of their favourite contents
4. From the list of contents from step 3, remove the contents which are already marked as favourite by the user whose recommendations are being worked on.
5. List of contents from step 4, is displayed as recommendations.

If the user does not have favourites or if the user has not logged in **(lazy log in feature),** display top 5 favourite contents among registered users as recommendations.

Below is the query with recommendation logic:

SELECT id, name, description, count(\*)

FROM

favourites a

join search\_table b

     on a.content\_id = b.id

where email in (select email from favourites

                where content\_id in (select content\_id

                                     from favourites where email="'.$email.'"))

and content\_id not in (select content\_id from favourites

                       where email="'.$email.'")

group by id, name, description order by count(\*) desc;

## **Problems faced:**

1. Designing user profile table
2. Designing user search sort order process
3. Using the user search sort order table data in sorting the search results

Below is how the generated search query for the search token “women heatmap”:

Select id, name, description from

((

(Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name)="women" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "% women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(description) like "% women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(associated\_search\_words) like "% women%" and b.email="admin@gmail.com")

INTERSECT

(Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name)="heatmap" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "heatmap%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "% heatmap%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(description) like "% heatmap%" and b.email="admin@gmail.com"

UNION Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(associated\_search\_words) like "% heatmap%" and b.email="admin@gmail.com")

)

UNION

(Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name)="women" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "% women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(description) like "% women%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(associated\_search\_words) like "% women%" and b.email="admin@gmail.com")

UNION

(Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name)="heatmap" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "heatmap%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(name) like "% heatmap%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(description) like "% heatmap%" and b.email="admin@gmail.com"

UNION

Select id, name, description, relation\_score from search\_table a join user\_search\_sort b on a.id=b.content\_id

where lower(associated\_search\_words) like "% heatmap%" and b.email="admin@gmail.com")

) x order by relation\_score desc;