**I. Overview**

**1. Project Title**

Google Scholar Data Visualization System (Scholarview)

**2. Project Summary**

Our project is a data visualization platform that features information about CS professors at UIUC, including short biography, education, interests, classes taught, and a brief timeline, along with information pertaining to their publishings on Google Scholar, such as most commonly used words, the scholar’s top publishers, who they collaborated with ,etc. Upon landing on the home page, users can sign in or sign up, or just go ahead and search up scholars without signing in. Users can also rate a scholar’s courses on several metrics such as quality of lectures, difficulty of exams, etc. The user interfaces demonstrated by Figure 5, Figure 6, Figure 7, and Figure 8 in APPENDIX.

**II. Frontend**

Half of the styles and features (e.g. navigation bar, containers, cards, etc.) of our webpages are imported from Bootstrap, while the other half is Cascading Style Sheets (CSS). We use CSS as a complement to Bootstrap when Bootstrap does provide a needed feature (e.g. the 5-star rating feature), or when we want to adjust Bootstrap styles according to our preferences (e.g. Illini color theme). Besides, we use javascript to modify HTML content according to users’ actions. Looking into Scholar, it has two HTML templates, home.html and scholar.html.

**1. Home.html**

home.html is the home page which comes with sign up and login buttons. To further fulfil the basic requirements, we also added a delete and insert section for deleting a user account and changing the user’s password. Besides the user section, we also have a search bar on the home page. The search bar would allow users to search for specific scholars and view their information.

**2. Scholar.html**

When a user searches for a scholar, Scholarview returns a scholar’s page with the relevant information for the scholar. On the scholar page, we first present a short biography of the professor, followed by a table summarizing his education, research interest, and teachings. Then comes the timeline of the professor. It is built with amChart and would display dynamically about the experience of a professor, mainly the timeline of when they finished each stage of education and were awarded with academic prizes. Following that are several charts displaying information about the professor’s research, citations of their work, and so on. In the end, we have an evaluation section which allows the user to give ratings for a scholar on the courses he has taught. In addition to the main content, we have a navigation bar which is sticky to the top of the scholar’s page. The navigation bar provides a home link and a second search area.

**III. Database and Backend**

**1. MySQL**

Scholarview uses MySQL and MongoDB as the database providers. For MySQL, five tables (i.e., “Articles”, “Scholars”, “Writes”, “Users”, and “Rates”) are created in the database. The relationship between the five tables are demonstrated by Figure 1. “Articles” contain the publication information (e.g. year of publication, number of citations) of a scholar; the data is scraped using the “Publish or Perish” software at <https://harzing.com/resources/publish-or-perish>. A Scrapy web-crawler is developed to verify the correctness of the output of the software. “Scholars” contain the basic information of scholars (e.g. short biography,research interest, teachings, etc.) are manually collected from either a scholar’s homepage or the website hosted by the scholar’s home department. “Writes”, which has two columns columns scId (Scholar ID) and aId (article ID), describes the many-to-many relationship between “Scholars” and “Articles”. For example, if a scholar with scId “10” writes an article with aId “217”, it will appear in “Writes” as a record “(10, 217)”. “Users” holds the account information, namely username and password, of Scholarview users. Once a person registers for an account at Scholarview, the entered username and password will be saved in the table for future login verifications. “Rates” is another many-to-many relation table which contains users’ ratings for the courses of scholars. If a user submits an evaluation form, his ratings will be saved in the “Rates” table. Please refer to Table 1, Table 2, Table 3, Table 4, and Table 5 in APPENDIX for the schema (attribute names and data types) and detailed attribute description of “Articles”, “Scholars”, “Writes”, “Users” and “Rates”, respectively.

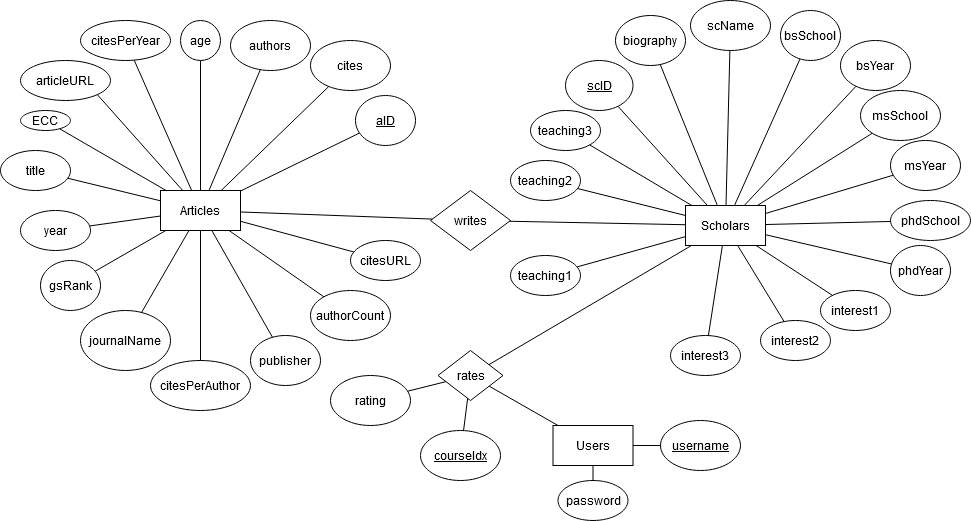


Figure 1. ER Design of the database

**2. MongoDB**

The data from the timeline section for Scholarview is maintained in MongoDB, which is a document-based NoSQL database. A timeline is a sequence of major events in a scholar’s academic career. (An example of a timeline stored in MongoDB is illustrated by Figure 2.) Notice the fact that the number of major events varies from scholar to scholar. For example, a scholar who just became an assistant professor might only have graduations and earning awards during the Ph.D. period as his major events, while a distinguished tenured professor can have tens of honorable awards, fellowships, and research grants. Such variance of the timeline data disqualifies MySQL as a database solution. First, the use of MySQL database requires the maintenance of an extra Event ID column, which increases the difficulty of web development. However, MongoDB stores each scholar’s timeline in an individual document which automatically generates and removes Event ID upon insertion and deletion of a record. Second, the selected data from MySQL cannot be directly sent to the frontend for displaying purposes. It usually needs to be converted to JSON to feed most chart APIs (e.g. amCharts, Chart.js, etc.) In contrast, querying MongoDB directly yields results in the JSON format, thus eliminating the post-processing process.

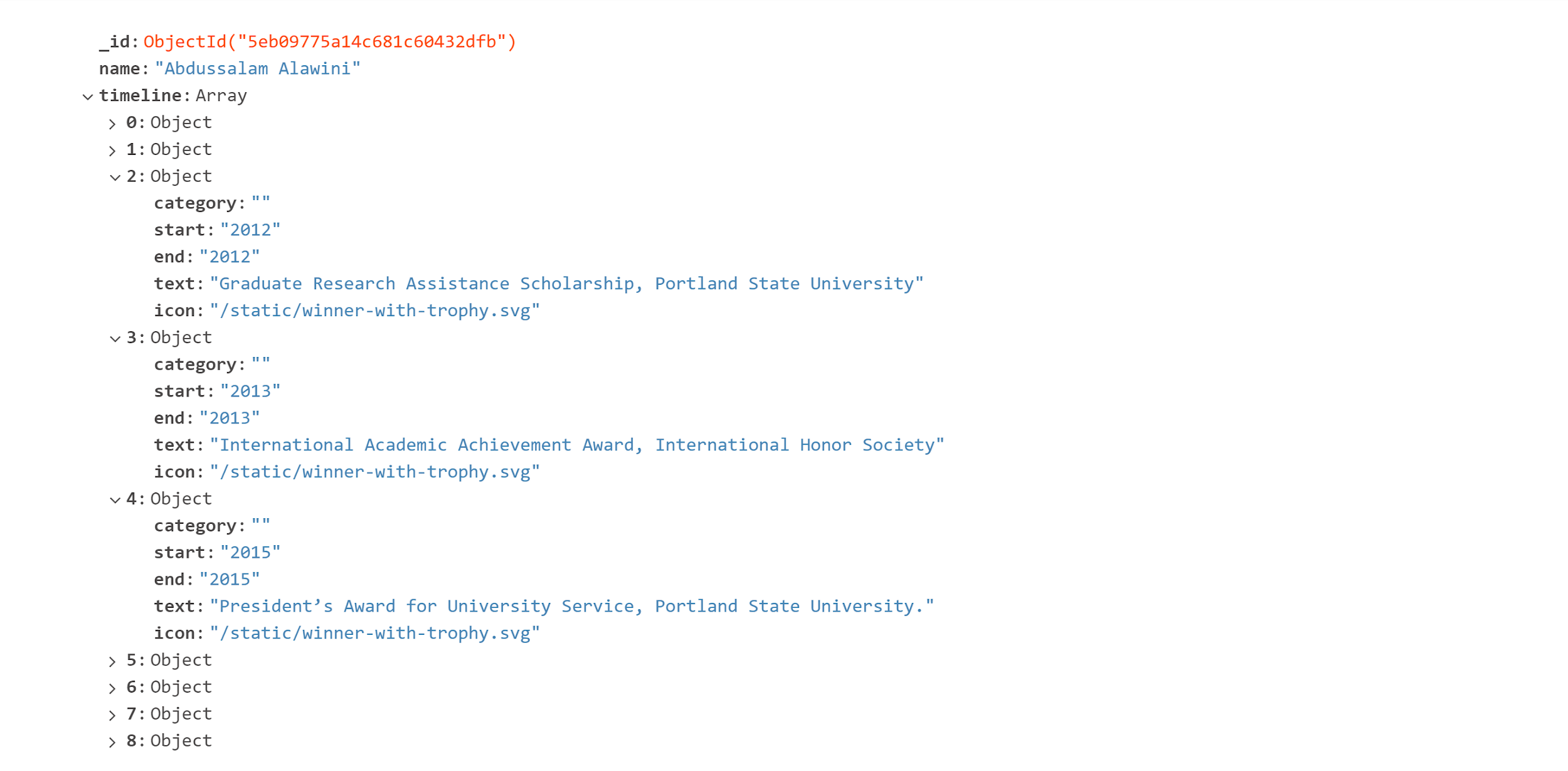


Figure 2. A Timeline of Prof. Abdussalam Alawini in MongoDB

**3. Flask**

The backend of Scholarview is implemented with Flask, a micro web framework written in Python. The “flask”, “mysql” and “pymongo” libraries are imported for backend development. The specific functions and the purposes for using them are summarized in Table 6.

Table 6. Summary of Library Functions Used for Scholarview

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| mysql.connector | Connecting backend with MySQL |
| pymongo.MongoClient | Connecting backend with MongoDB |
| flask.Flask | Basic functionalities of Flask |
| flask.redirect | Redirecting to a route |
| flask.request | Processing HTTP requests from clients |
| flask.render\_template | Rendering a html template |
| flask.session | Maintaining a session for the user account system;  storing the current user’s name (none if users are not logged in) |

Five routes are defined, “/login”, “/logout”, “/signup”, “/home”, “/search”, “/process”, and “/submit”. A function with the same name is developed for each route. An example of a route and function (/login and login()) is shown in the following interactive window of Visual Studio Code.

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

if request.form['btn'] == 'sign\_in':

usrname = request.form['usrname']

psw = request.form['psw']

cursor = connection.cursor()

cursor.callproc('checkinUser', [usrname, psw])

for response in cursor.stored\_results():

response = response.fetchall()[0][0]

print("response is: ", response)

if response == 0:

return redirect(url\_for("home", login\_status="wrong\_pass"))

elif response == 1:

#print("After sign in: ", session['username'])

return redirect(url\_for("home", login\_status="successful", user=usrname))

elif response == 2:

return redirect(url\_for("home", login\_status="user\_dne"))

The above function processes the login requests from Scholarview users and redirects to the “/home” route with a parameter. logout() and signup() have similar duties. home() is responsible for properly rendering “home.html”, the template for the homepage, with the parameters received from login, logout and signup. search() receives users’ searching requests submitted from both the homepage and the scholar’s page; it then looks into the “Ratings” database for the user’s rating status (i.e. if a user has rated a scholar’s course); finally, it redirects to process() with a scholar’s name and his rating status. process() communicates with “Scholars”, “Writes” and “Articles” to collect the data necessary for displaying the visualization charts, and renders “scholar.html” with the parameters received from search(). submit() receives the ratings from a user once he submits an evaluation form on “scholar.html”, saves it to “Ratings”, and redirect to process() which rerenders the scholar’s page.

**IV. Functionality**

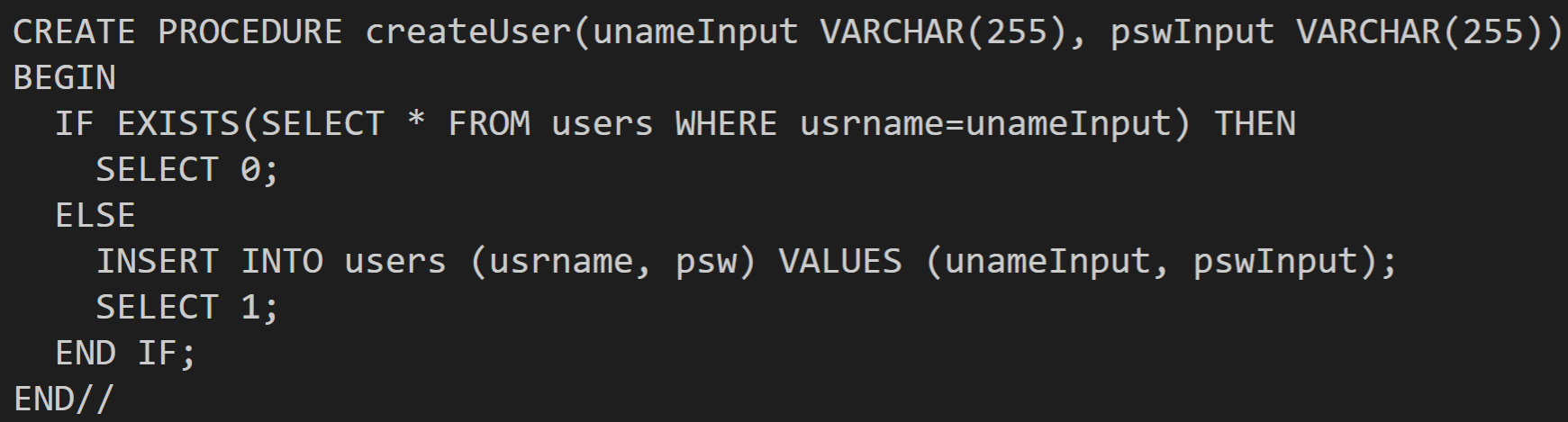
**1. Overview**

Scholarview provides the following list of functionalities. The category of functions (basic function/advanced function) each functionality demonstrates is appended with parenthesis to the subtitles.

1. User System (Advanced Function)
   1. Signup (Basic Function & Advanced Function)
      1. Alerting upon duplicate usernames
      2. Alerting upon successful signup
   2. Login (Basic Function & Advanced Function)
      1. Alerting upon user DNE (does not exist)
      2. Alerting upon successful login
      3. Alerting upon wrong password
   3. Signout (Basic Function & Advanced Function)
      1. Alerting upon signout
2. Searching
   1. Searching from homepage
   2. Searching from scholar’s page
3. Visualizing (Basic Function)
   1. Displaying biography, along with education, research interests, and teachings
   2. Displaying the timeline including the major events in a scholar’s career
   3. Visualizing research work of a scholar with 6 charts (Basic Function)
      1. Keywords of the scholar
      2. Favorite publishers
      3. Citation percentage of top journals
      4. # of papers by # of authors
      5. # of papers by yearly citations
      6. Scholars who published with the scholar
4. Rating (Advanced Function)
   1. Providing evaluation forms
      1. Displaying a thank-you message if already evaluated a course
      2. Displaying a please-login message if user has not logged in
   2. Evaluating courses
      1. 5-star ratings
      2. Allowing evaluation for each course given by the scholar
      3. Alerting upon submission of a evaluation form

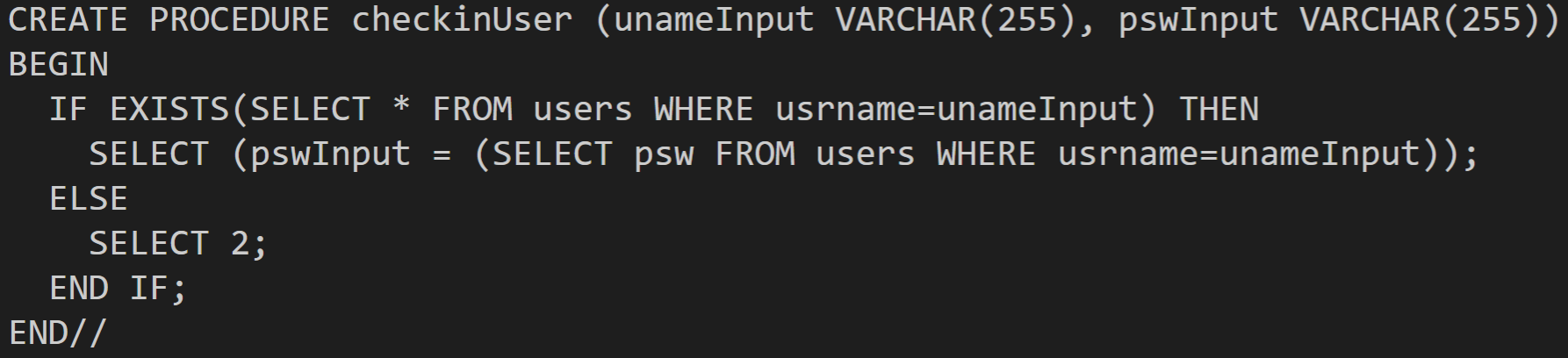
**2. Basic Functions**

The basic functions of Scholarview are insert, delete, update and search which are implemented throughout the user account system and the visualization system. When a user registers for an account at Scholarview, the request will be sent to the backend, and a stored procedure “createUser”, as illustrated by Figure 3, will be called. The procedure first checks if the username that the user has entered is already in the database. It will either return 0 if username already exists or return 1 after inserting the user account record into “Users” table if username does not exist.



**Figure 3. Stored Procedure “createUser” with INSERT function**

Upon user login, Scholarview will call the “checkinUser” procedure, as illustrated by Figure 4. The procedure first checks if the user exists in the “Users” table or not. If not, the procedure returns 2, which represents “user does not exist status”. If the user does exist, the procedure will further check whether the password entered by the user is the same as the password stored in the “Users” table. If they are the same, the procedure returns 1; otherwise, the procedure returns 0. A single SELECT command is used to realize this functionality.



**Figure 4. Stored Procedure “checkinUser”**

Besides the basic login and signup functions, we provide options for the users to delete the user account and update their passwords. The following SQL queries are used for such purposes.

"DELETE FROM users WHERE usrname = '" + usrname + "';"

"UPDATE users SET psw = '" + psw + "' WHERE usrname = '" + usrname + "';"

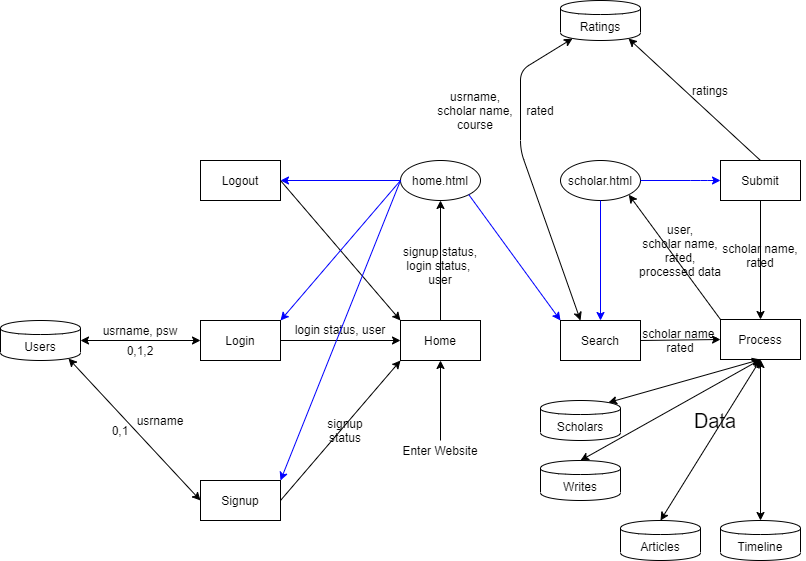
We have multiple search functions in the visualization part, which includes 6 charts empowered by amCharts. A summary of the name (purpose) of a chart, the SQL query used to retrieve raw data, and the post-processing process is given by Table 2.

Table 7. A summary of the visualization charts

|  |  |  |
| --- | --- | --- |
| **Chart** | **SQL Query** | **Post-processing** |
| Keywords of the scholar | "SELECT title FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';") | 1. Titles are split up into words; punctuation marks (e.g., commas, colons, periods, etc.) are removed  2. Meaningless words are removed (e.g. prepositions, conjunctions)  3. Each word is counted; the frequencies of the top keywords are reported |
| Favorite publishers | "SELECT publisher FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';" | 1. Each publisher is counted  2. The frequencies of the top publishers are reported |
| Citation percentage of top journals | "SELECT aId, cites, authorCount FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';") | 1. aId’s (Article ID’s) are sorted in the decreasing order of cites (# of citations)  2. The total # of citations for the top10%, 10%-30%, 30%-70% papers are computed and reported |
| # of papers by # of authors | "SELECT aId, cites, authorCount FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';") | 1. aId’s (Article ID’s) are grouped by authorCount (number of authors)  2. Each group is divided into subgroups by the % level of aId (top10%, 10%-30%, 30%-70%)  3. The total # of papers for each group and subgroup is computed and reported |
| # of papers by yearly citations | "SELECT aId, citesPerYear FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';" | 1. citesPerYear (yearly citations) are rounded to the nearest integer  2. aId is grouped by citedPerYear  3. aId’s with 0 yearly cites are ignored  4. aId’s with 10 or more yearly cites are grouped together  4. The total # of aId’s for each group are computed and reported |
| Scholars who published with the scholar | "SELECT authors, cites FROM scholars NATURAL JOIN writes NATURAL JOIN articles WHERE scName = '" + scholar\_name + "';" | 1. The total # of citations is computed for each author, and appended to a list along with the author  2. The list is sorted, and the top terms are reported |

**3. Advanced Function**

The advanced function of Scholarview is demonstrated by the rating system and the user account system. They together allow a user to rate a scholar’s courses. The challenge of implementing a rating system comes from the complicated software logics and information flow, as illustrated by Figure 4. In this figure, the disks represent the databases; the rectangles represent the route and functions in the backend; the ellipses represent the HTML templates; blue lines describe information flows through HTTP requests, while black lines describe information flow between backend functions and databases.



**Figure 5. Information Flow in the System**

As an example, once a user enters the website, the home function will be activated so the user is directed to the home.html page, where he can logout, login, signup, or search by submitting forms. If he logs in or signs up, the corresponding functions will be called; the functions will communicate with the “Users” database, retrieve some information, process it, and rerender home.html with certain parameters. If the user searches for a scholar, the search function will first ask “Ratings” if the user has rated any of the scholar’s courses, the answer from ratings “rated” will be sent back to search(). Then, search() will call process(), passing the scholar’s name and “rated” parameters to it. Next, process() will communicate with MySQL and NoSQL databases to obtain the proper data. Finally, process() will send the processed data and “rated” to scholar.html for rendering. Depending on the value of “rated”, scholar.html will either show or hide an evaluation form. For example, if “rated” tells scholar.html that the user has already rated for the scholar on the course, the evaluation form for the course will be hidden and a thank-you message will be displayed.

While complicated software logic accounts for a challenge, the team struggled with the maintenance of user status (i.e. logged in, logged out), too. In the beginning, we tried to dynamically maintain the user status, meaning we initialize a user status upon website visit, and pass it from function to function during users’ interactions with our website. However, it causes some unresolvable problems. For example, when a user has already logged in from the home page, and for some reason he revisits the home page, his login status will be lost so he needs to go through the login process again. Aiming to have a robust user system, we have applied flask.session function to our backend program. The session function was able to maintain a global variable until the termination of a Flask server session. Thus, user status becomes unaffected by revisits, so a smooth user experience is promised. Therefore, we would suggest future teams which try to implement a user system that flask.session or a similar function should be used.

**V. Conclusion**

**1. Labor Division**

The workload for this project is distributed evenly across all the teammates. Table 8 describes our labor division in detail. In order to ensure work efficiency and product quality, we assign at least 2 teammates to develop the relatively important features (e.g. visualization charts, evaluation, etc.). The final report is crafted and proofread by all team members.

Table 8. Labor Division For Project Track I

|  |  |
| --- | --- |
| **Work** | **Responsible Team Members** |
| Homepage layout design (HTML) | Ilie Vartic |
| Scholar’s page layout design (HTML) | Ilie Vartic |
| Biography and basic info | Akhil Bhamidipati |
| Timeline chart and features | Longji Li, Ilie Vartic |
| Visualization charts i, ii, and iii and features | Jiashuo Tong, Akhil Bhamidipati |
| Visualization charts iv, v, and vi and features | Jiashuo Tong, Longji Li |
| Evaluation (Advanced function) | Longji Li, Akhil Bhamidipati, Jiashuo Tong |
| Manual data collection | Ilie Vartic, Jiashuo Tong |
| Data scraping from Google Scholar | Akhil Bhamidipati, Longji Li |
| Video Demo | Akhil Bhamidipati |
| Final Report Crafting and Proofreading | All team members |

**2. About Original Plan**

One of our previous attempts was to use React.js to implement both frontend and backend. It did not turn out to be fruitful because we have encountered tremendous problems in connecting the script with either SQL or MongoDB databases. Also, the learning curve for React.js is quite steep. Therefore, we have decided to switch to Flask and HTML/CSS/JS. For data visualization, we first attempted to interface with chart.js. However, the variety of the charts at Chart.js is limited; in addition, the formal style of Chart.js does not meet our expectations. Therefore, we finally opted for amCharts because it provides more options, demonstrates more liveliness, provides more, and allows more interactions with the charts. Empowered by amCharts, our visualization part is able to fully convey the diversity we have expected from Scholarview.

**3. Outcomes**

At the close of the project, we have realized a data visualization system for the Google Scholar’s data of scholars at CS@Illinois. All the team members have obtained practical knowledge of the following topics: RDBMS, ER/UML designs, NoSQL, and web development. What’s more, all of the team members have become more proficient in using the following database solutions and web framework: MySQL, MongoDB, Flask, and React.js.

**APPENDIX**

Table 1: Schema and Description for the “Articles” MySQL Table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| aId | INT NOT NULL | Article ID |
| cites | INT | # of cites of article |
| authors | VARCHAR(255) | Authors of article |
| title | VARCHAR(255) | Title of article |
| year\* | INT | Year when article is published |
| journalName\* | VARCHAR(255) | Name of journal |
| publisher | VARCHAR(255) | Name of publisher |
| articleURL\* | LONGTEXT | URL for article |
| citesURL\* | LONGTEXT | URL for citation |
| gsRank\* | INT | Ranking of the # of citations for article |
| ECC | INT | # of cites of article (Same as “cites”) |
| citesPerYear | INT | # citations per year for article |
| citesPerAuthor | INT | # citations per author for article |
| authorCount | INT | # authors of an article |
| age | INT |  |
| Notes:   * Asterisk (\*) denotes that the data in the column is not used in the product. * Underscore (\_) denotes that the column is part of the primary key. * 1800+ records scraped from Google Scholar | | |

Table 2: Schema and Description for the “Scholars” MySQL Table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| scId | INT NOT NULL | Scholar ID |
| biography | LONGTEXT | A short biography |
| scName | VARCHAR(255) | Scholar name |
| bsSchool | VARCHAR(255) | School where scholar is awarded B.S. degree |
| bsYear | INT | Year of graduation from B.S. school |
| msSchool | VARCHAR(255) | School where scholar is awarded M.S. degree |
| msYear | INT | Year of graduation from M.S. school |
| phdSchool | VARCHAR(255) | School where scholar is awarded Ph.D. degree |
| phdYear | INT | Year of graduation from Ph.D. school |
| interest1 | VARCHAR(255) | Scholar’s first academic interest |
| interest2 | VARCHAR(255) | Scholar’s second academic interest |
| interest3 | VARCHAR(255) | Scholar’s third academic interest |
| teaching1 | VARCHAR(255) | Scholar’s first teaching |
| teaching2 | VARCHAR(255) | Scholar’s second teaching |
| teaching3 | VARCHAR(255) | Scholar’s third teaching |
| Notes:   * Underscore (\_) denotes that the column is part of the primary key. * 25 manually collected records from UIUC’s websites and scholars’ home pages | | |

Table 3: Schema and Description for the “Writes” MySQL Table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| scId | INT NOT NULL | Scholar ID, referencing scholars(scId) |
| aId | INT NOT NULL | Article ID, referencing articles(scId) |
| Notes:   * Underscore (\_) denotes that the column is part of the primary key. | | |

Table 4: Schema and Description for the “Users” MySQL Table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| Usrname | VARCHAR(255) NOT NULL | The username of user |
| psw | VARCHAR(255) NOT NULL | The password of user |
| Notes:   * Underscore (\_) denotes that the column is part of the primary key. | | |

Table 5: Schema and Description for the “Rates” MySQL Table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| usrname | VARCHAR(255) NOT NULL | Username, referencing user(usrname) |
| scname | VARCHAR(255) NOT NULL | Scholar’s name, referencing |
| courseIdx | INT NOT NULL | Index of the course (1, 2 or 3) rated |
| rating | VARCHAR(255) | Rating on a scholar’s course given by user |
| Notes:   * Underscore (\_) denotes that the column is part of the primary key. | | |

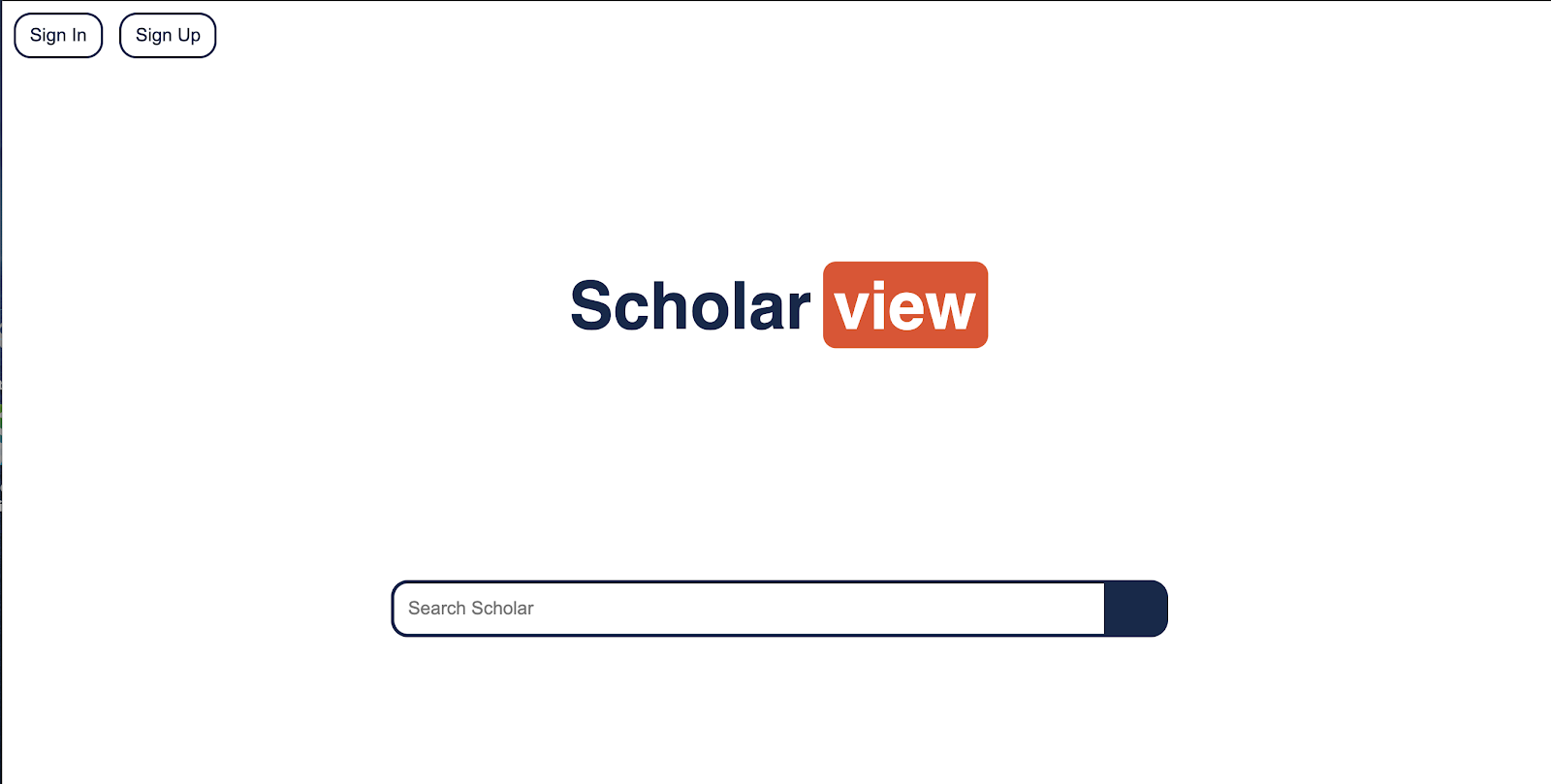


Figure 5. Home Page of Scholarview

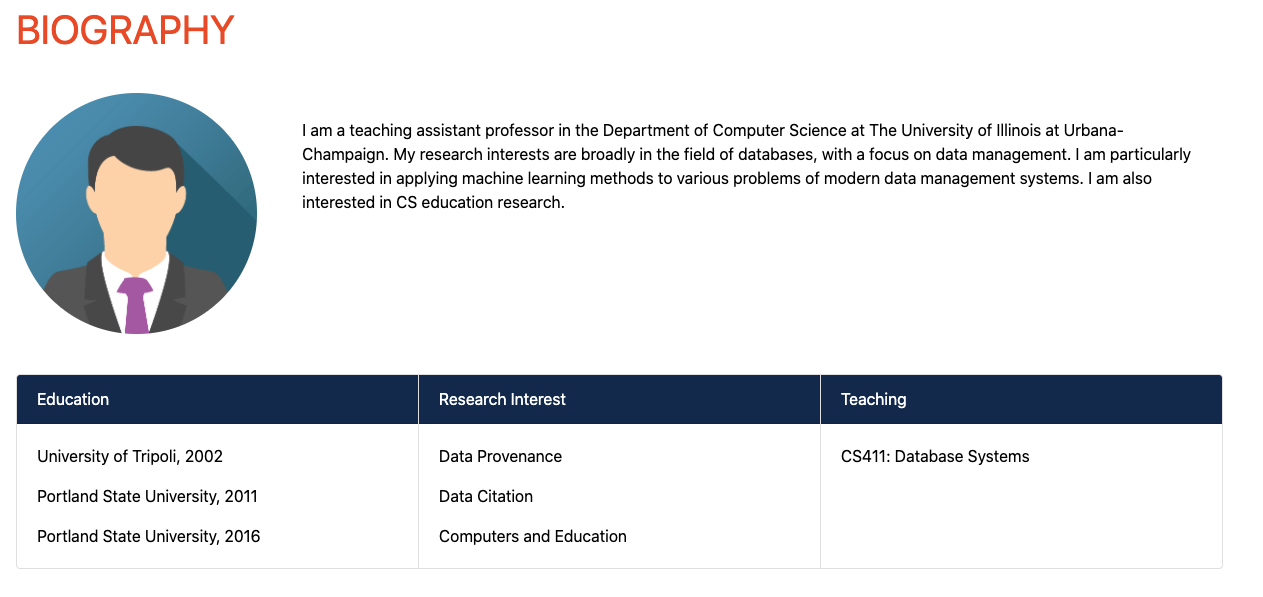


Figure 6. Biography Section of Scholar’s Page

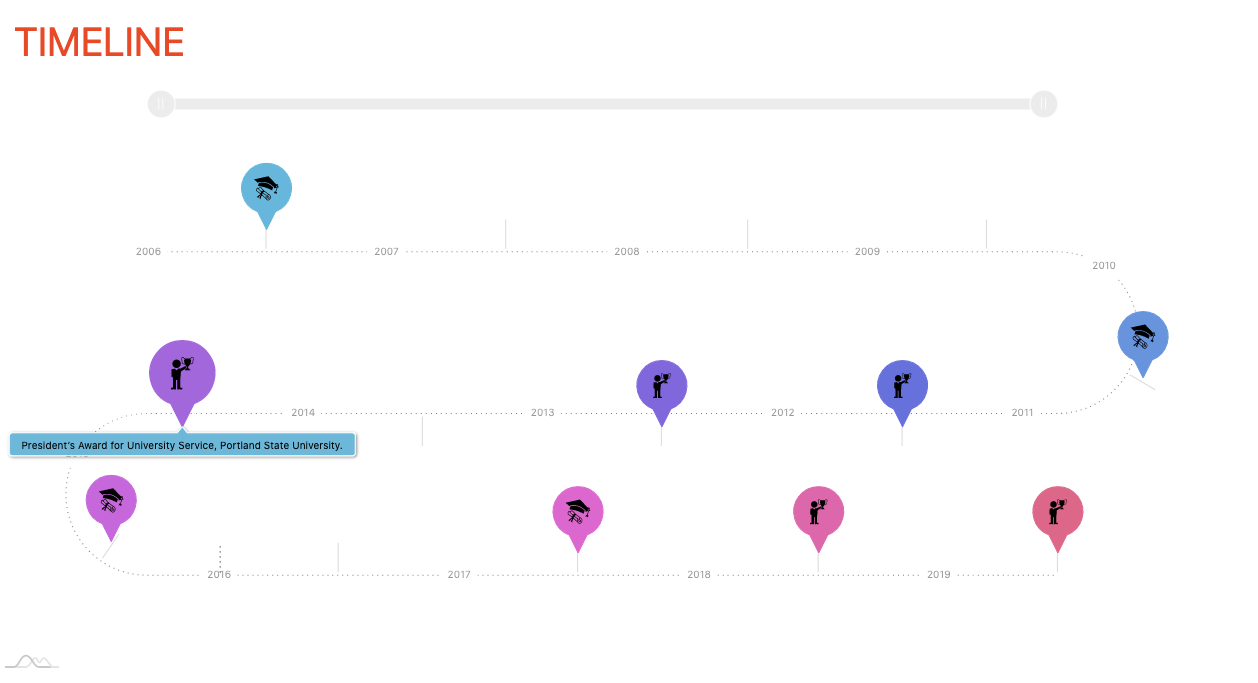


Figure 7. Timeline Section of Scholar’s Page

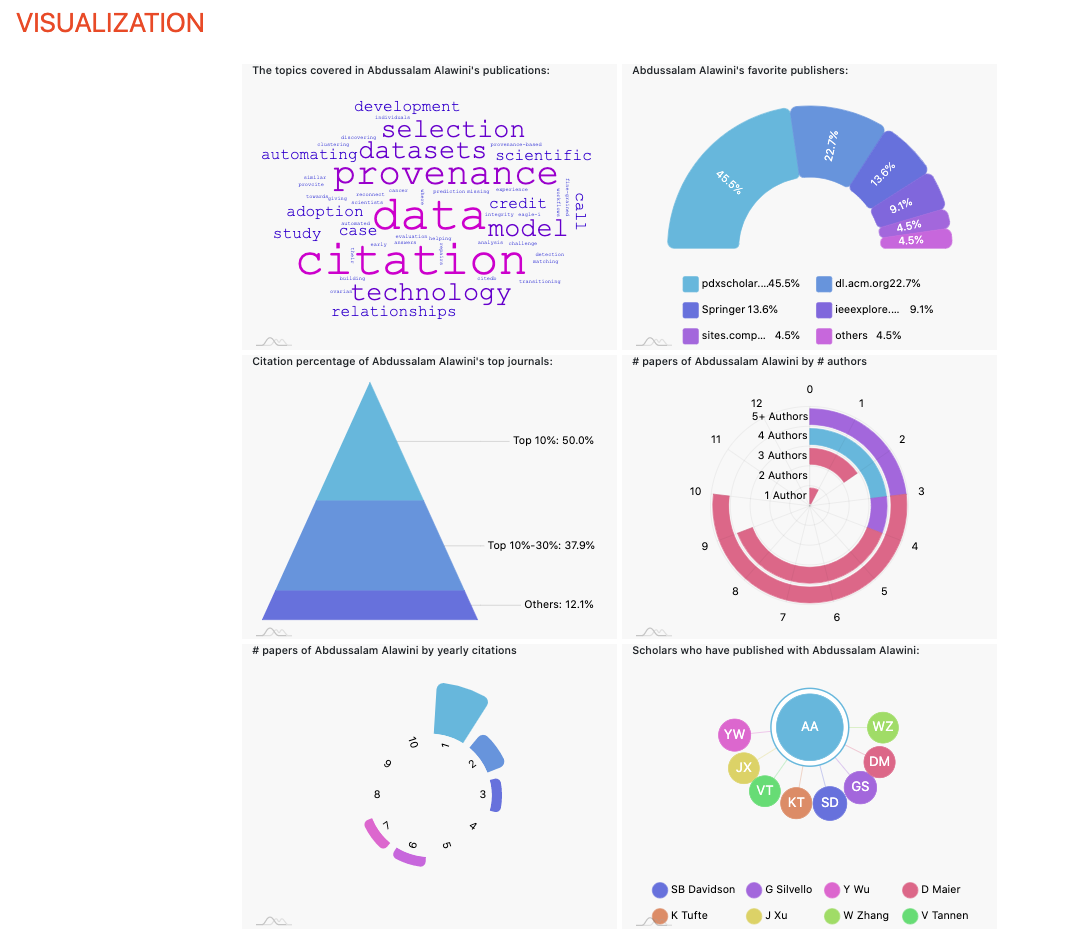


Figure 8. Visualization Section of Scholar’s Page

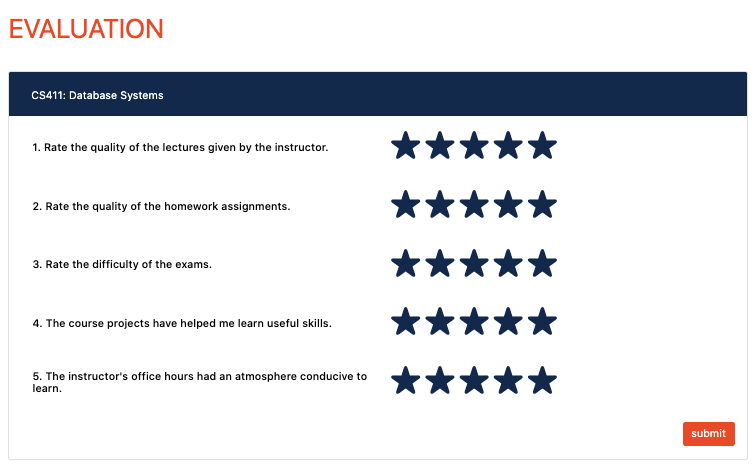


Figure 9. Evaluation Section of Scholar’s Page