ResearchMate

Software Design

CSCI-P465/565 (Software Engineering I)

<u>Project Team</u>

Gulshan Madhwani

Shubham Basu

Jayendra Khandare

Xinquan Wu

---

1. Introduction

This document is a high-level description of the system that we are building – ResearchMate. ResearchMate is a web application that let PhD researchers connect to other PhD students and advisors. This document provides explanation of the system design, design approach, system architecture and component design of ResearchMate.

1.1 System Description

ResearchMate is a web application, a collaborative platform where researchers can connect with other researchers that share a common interest. It is an easy to use, user friendly way of posting doubts, conducting a group discussion, having live chat. It also provides a bullet-in board where user can get latest updates, trends in their area of interest. This application targets the users who find it difficult to connect with people of his/her interest. Having a group discussion with people having different perspective can help them conduct their research smoothly and efficiently by considering all possible scenarios.

1.2 Design Evolution

1.2.1 Design Issues

Sprint 1 -

Due to lack of experience in MEAN stack, we thought of proceeding with .NET but we don’t have place to host .NET application on silo machine. So we stick to MEAN stack and started learning.
Issues we faced in the past two weeks:

- Whether to program front end using AngularJS or jQuery. We decided to go with AngularJS since it is good for single page application development.
- We are also trying to make the application responsive using Bootstrap. Considering Bootstrap is new to most of the members, it is taking time to make a good and robust design.
- MEAN stack being a new technologies to learn, we were not comfortable with making service requests from AngularJS to NodeJS.
- NodeJS provides lots of packages to use that eases the development process. Being new to NodeJS, we had a hard time searching for an appropriate module. For example, nodemailer for sending emails is most widely used but is not very secure.
- For Duo authentication, we are generating a random key which is sent to the user through email (as a link). Once user clicks the link, we authenticate the user on backend.

Sprint 2 –

- We started developing user profile when we realized keeping username as primary key is not secure enough. Any user can hit a post request with username and get/set user’s data.
- This basic design issue was overcome using a session key generated when user login each time. This session string is stored in user table and each time user wants to perform get/set on profile info, we get the user info by mapping session string to user id and then allowing to perform any action. This session string is 16 characters log random alpha-numeric string. To reduce the probability of collision and increase security, this can be increased to 32 character long.
- One of the module in our project is that a user can follow other users, join other groups, etc. To avoid redundancy of records in database, we normalized the tables and decided to create one more table that stores the mapping between user Id and either followee ID or Group ID respectively.

- We are using mongoose (MongoDB) as our database. It does not support auto-increment of ID for records. We used a library called mongoose-auto-increment but it wasn’t of any help. We could not integrate with our existing database structure.

Sprint 3 –

- The main issue faced during this sprint was how to fetch data from multiple tables for Search functionality since MongoDB calls such as find(), findOne(), findMany(), etc. are asynchronous in nature i.e. if there is anything outside find() functions such as sending response, response is sent first even if find() hasn’t completed its execution.
- How to display search results on front end – User interface, characteristics to display, etc

1.2.2 Candidate Design Solutions

Sprint 1 -

With some research, MEAN stack came out to be a perfect stack of technologies for our project. Our application mainly consists of CRUD operations. MEAN provides all the required technologies in one stack – MongoDB for database, AngularJS for front end, NodeJS for backend, ExpressJS facilitates communication between AngularJS and NodeJS.

- We are currently exploring more appropriate and efficient NodeJS modules for our application useful in various parts of the application. This can help us in future and design a system that is scalable and easily manageable.
- We are also exploring how we can perform Duo authentication using Google Duo, similar to CAS login. Currently we are performing Duo authentication using random key generation and authenticating using email address. We are currently exploring a NodeJS module called “passport” for this task.

Sprint 2 –

- To increase the security, we are generating session string which is 16 bytes long each time user logs in. There was one more possible solution to store username and hash of password and send both each time.
- For auto-increment of ids, search for some other stable and compatible library.
- Create auto-increment module by own.
- Re-design database and check if any table can be normalized to avoid redundancy of records.

Sprint 3 –
Possible solutions on how to synchronize database calls for multiple tables:

- The most popular libraries that we found to synchronize calls between functions were async.js and series.js.
- In async.js, we used async.waterfall, async.series, and async.parallel but none of the functions satisfied our needs to pass the result between many functions.
- To display the search results on front end, we can either show in accordions, sections, create modules for each search result like user, group, skill, etc.

1.2.3 Design Solution Rationale

Sprint 1 -

- We are currently using ‘nodemailer’ module in NodeJS for emailing. It is the most widely used and has good amount of online documentation. Though not very secure as compared to other modules, using this module reduces a good amount of time of development and serves an initial purpose of implementing Duo authentication.
- Implementing Duo authentication using Google services is time consuming and requires some initial monetary investment as well. We are currently using authentication through email which serves the purpose of verifying if the user is authentic or not. In future, if time permits, Google Duo can be implemented.

Sprint 2 –

- To make the application access secure, we are generating session string since storing username and hash of password will waste memory and is less secure. This session string can also help to implement “Remember Login” feature if required in future.
- To reduce the data redundancy, we are creating mapping tables. For example, UserGroupMapping for users who are in particular group and UserFollowerMapping for users who is following who.
- For auto-increment of ids, we are fetching all records, sorting them and getting max id and generate new id by incrementing it by 1.

Sprint 3 –

- Since none of the libraries we found didn’t properly handled and passed result data between functions, we defined a synchronous data flow and synchronized each function from scratch. Once we find a result for that section, we call in the next function is process and pass the data.

- Passing the data is very important because if it breaks in between function calls, no response will be sent to front end.
- To display search results, we are creating different sections for each search result. Users found with the given search string will be shown in a different section. Similarly, groups will be show in different section. Skills are searched with respective to user – when the given string matches with the skill name, we search for the users having those skills and return the users.

1.3 Design Approach

1.3.1 Methods

Sprint 1 -

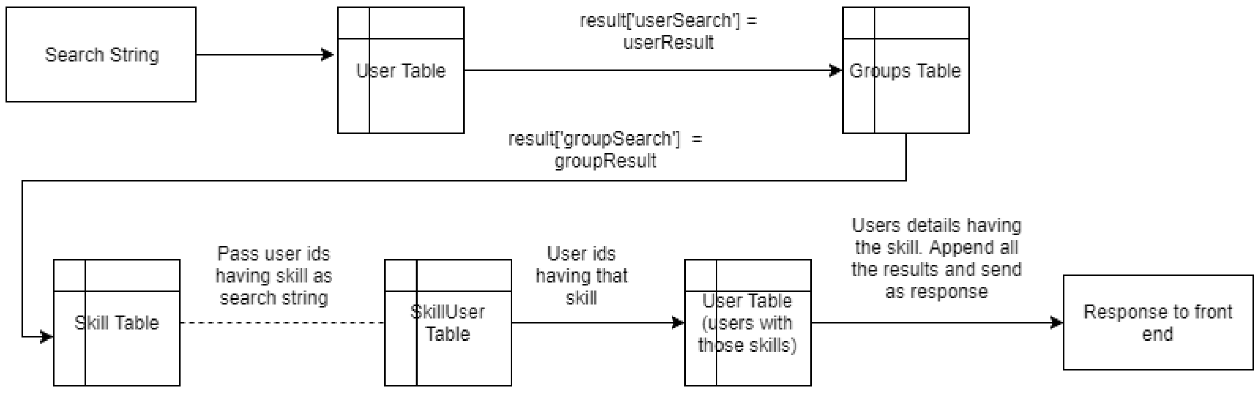
The primary feature of MEAN stack is abstraction that it provides. Each technologies provides a layer of abstraction. Service calls are made from AngularJS using Express which in turn makes the actual call to NodeJS. NodeJS provides interfaces to connect to the database and performs actual CRUD operations. Each layer works independently which allows different members work independently.
We are also using various node modules that makes the application efficient, secure, and scalable.

Sprint 2 –

- For auto-increment, we are sorting the available values, getting the max and incrementing it by 1 to generate the new ID.
- For session management, we are generating random session string that is stored in user table. It is generated each time when user logs in. Though there was a library already existing, called Passport, but to have customization on our end, we decided to manage session by our code instead of any third party library.

Sprint 3 –

- To synchronize function calls for search functionality, we created a data flow diagram – how the data will flow between various functions that searches in each of the table.


Search String
User Table
result['userSearch'] = userResult
Groups Table
result['groupSearch'] = groupResult
Skill Table
Pass user ids having skill as search string
SkillUser Table
User ids having that skill
User Table (users with those skills)
Users details having the skill. Append all the results and send as response
Response to front end

- Each table creates its own result object and passes onto next function. When the last function is called (in our case getting users having a skill as search string), after execution completes, it calls sendResponse() function that sends response to front end.
- The advantage of having this kind of architecture is that we can scale the search to search for any other attribute just by including a function call between the flow.

1.3.2 Standards

- We are using Bootstrap for UI development since it is efficient, user friendly, and responsive.
- We are using AngularJS for front end development because it’s good for single page application development.
- For back end development, we are using NodeJS and express.

1.3.3 Tools

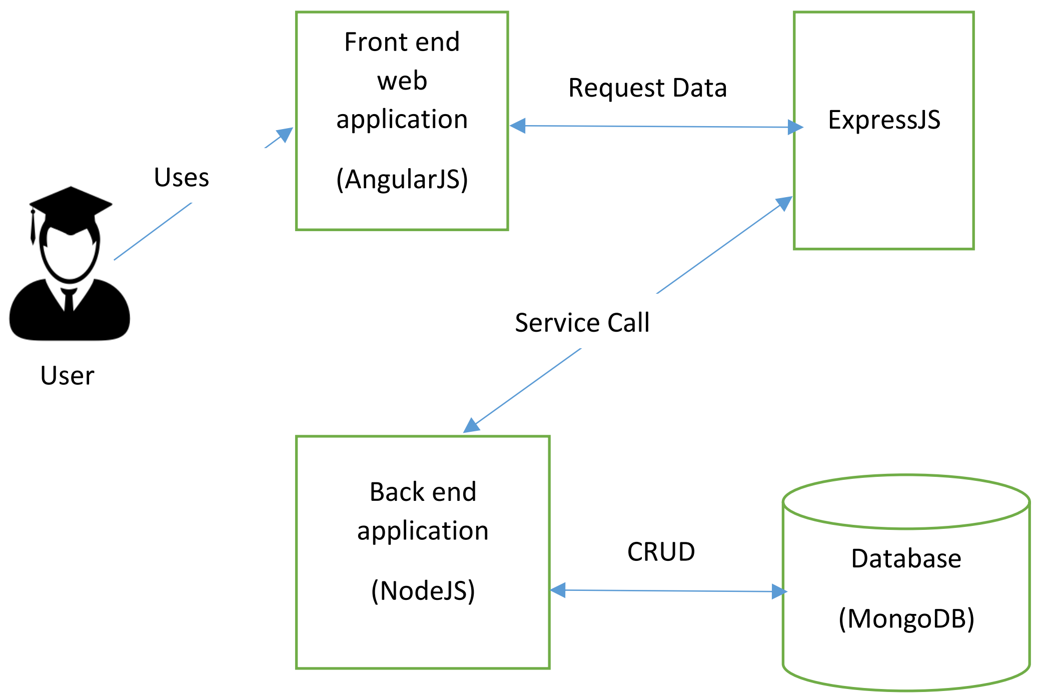
Front end:
- Programming: AngularJS
- IDE: Atom/Visual code

Back end:
- Programming: NodeJS
- IDE: Atom

Build: Build using webpack. It will allow to create a single JS file for production and eliminates the need to managing many JS files.
Database: MongoDB
Hosting: silo.indiana.edu
Source Control: GitHub
Coordination: JIRA

2. System Architecture

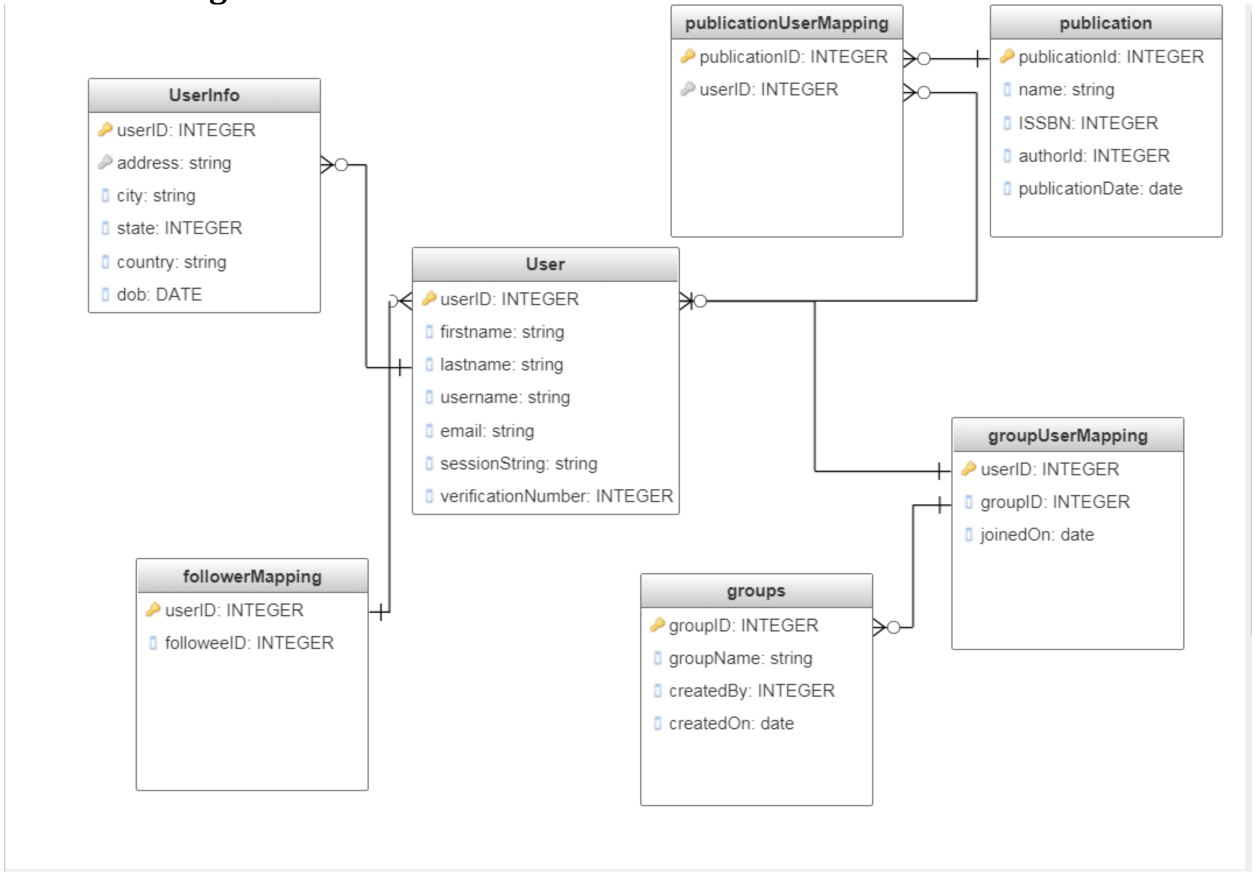
2.1 System Design



User: User is any person having a profile on our web application. User can be a researcher, advisor, admit, etc.

Front end: Front end application developed using AngularJS, Bootstrap, HTML, and CSS provides a basic user interface where user can login, sign up, view bullet-in, participate in discussion, etc.
Back end: Backend is developed using NodeJS. This component provides an interface between database and front end and facilitates to perform CRUD operations on the database.
Database: This is where all the data is stored. We are using MongoDB for database.

Database Diagram:



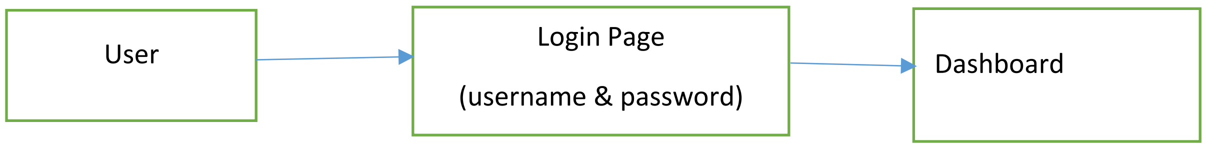
2.2 External Interfaces

External interfaces for our application are users. These users are classified as PhD students/researchers, advisors. Researchers can post on discussions page, join group, view updates on bullet-in board, and share their profile. Advisors can create groups for students to join and can approve their requests.

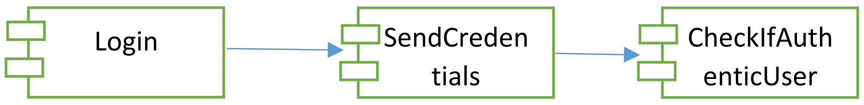
3. Component Design

Component 1:

- **Component Name**
  Login Component
- **Component Description**
  This component allows the user to login by providing username and password.
- **Responsible Development Team Member**
  Shubham, Jayendra, Gulshan.
- **Component Diagram**



- **Component User Interface**
  For login, user is supposed to provide username and password.
- **Component Objects**
  classes: Login, SendCredentials, CheckIfAuthenticUser, GenerateSessionString



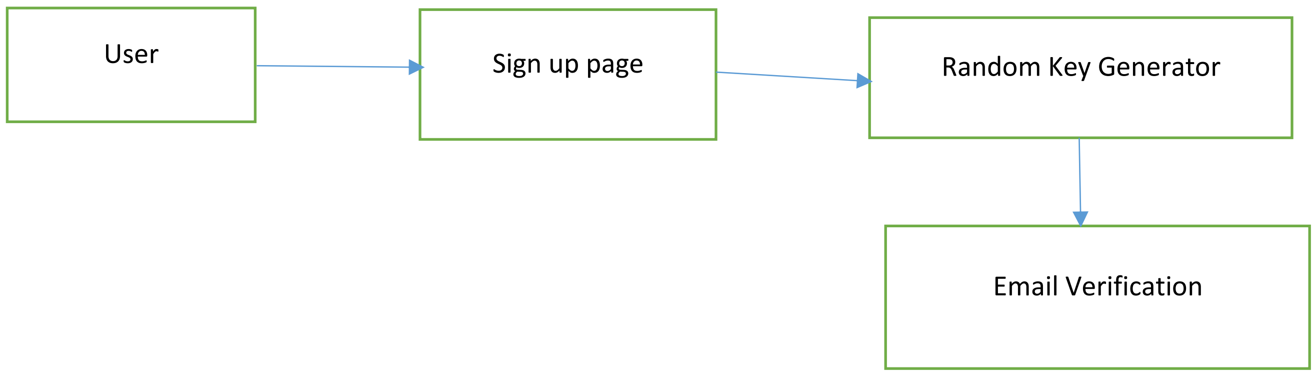
- **Component Interfaces (internal and external)**
  For login, user provides username and password using the text boxes provided on the login page. These credentials are then passed to the server side where the user is authenticated and checked if the profile already exist. If user record is found, we verify if user has verified his/her account using the email provided.

If account is verified, we generate a 16 byte long session string and pass it to the front end for future use.

- **Component Error Handling**
  Username and password is checked for blanks and valid email address. User account is checked if verified or not.

Component 2:

- **Component Name**
  Sign up
- **Component Description**
  This component allows the user to sign up. User needs to provide email address, first name, last name, and password. User is authenticated using random key generation and email verification.
- **Responsible Development Team Member**
  Gulshan, Jayendra, Xinquan.
- **Component Diagram**



- **Component User Interface**
  For sign up, user needs to provide first name, last name, email address, and password using the text boxes provided on the sign up page.
- **Component Objects**
  GetUserInput, CheckValidInput, GenerateRandomKey, EmailKey, VerifyUser
- **Component Interfaces (internal and external)**
  GetUserInput takes the input from user form.

CheckValidInput checks if the entered input is valid and all the required fields are provided.

GenerateRandomKey generates a random key for email authentication.

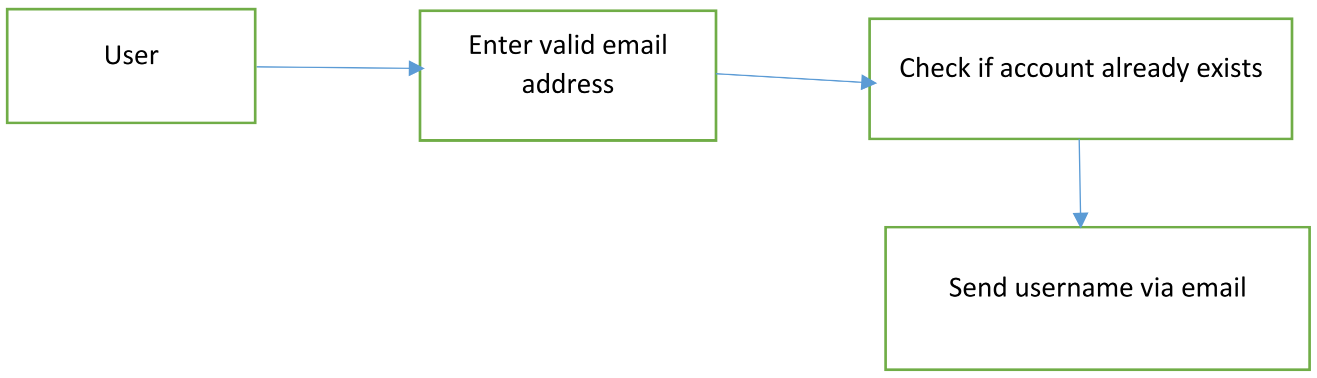
EmailKey is a node module that email the randomly generated key to the user.

VerifyUser is a module that checks if the key provided for verification is same as the key generated for that user.

- **Component Error Handling**
  Check for all the required fields: First name, last name, email address and password.
- Email address verification: checking the format and authenticating it using random key generation.

Component 3:

- **Component Name**
  Forget username
- **Component Description**
  This component allows the user to get the username on email if (s)he forgets.
- **Responsible Development Team Member**
  Gulshan, Jayendra.
- **Component Diagram**

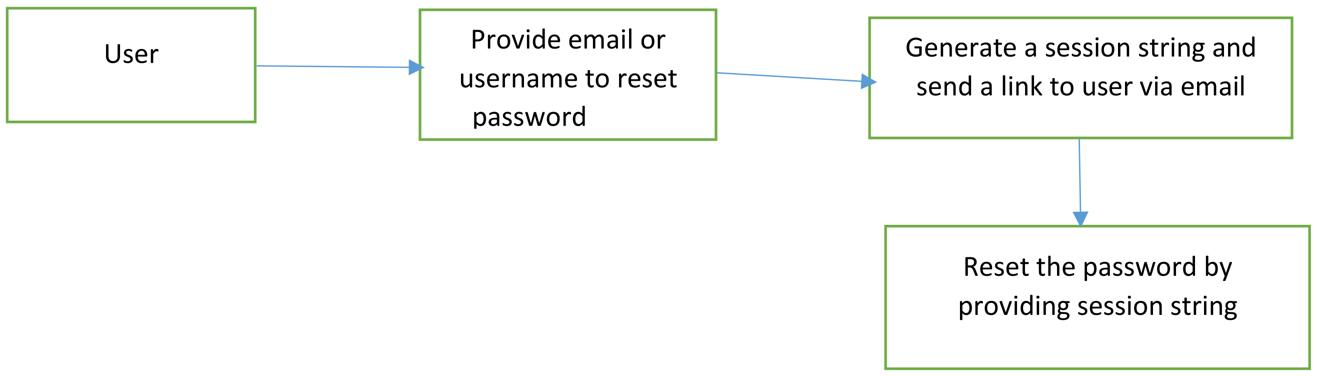


- **Component User Interface**
  User needs to provide his/her email address entered while sign up.
- **Component Objects**
  SendUserEmail, VerifyAccount, SendEmail
- **Component Interfaces (internal and external)**
  SendUserEmail sends the entered user email to the back end.
  VerifyAccount checks if the account is already created for the entered email address.
  SendEmail sends the username to the specified email address.

- **Component Error Handling**
  Check if email address is not blank and a record is already present in database for entered email address.

Component 4:

- **Component Name**
  Forget Password and Update Password
- **Component Description**
  This component allows the user to reset the password.
- **Responsible Development Team Member**
  Gulshan, Jayendra.
- **Component Diagram**


User
Provide email or username to reset password
Generate a session string and send a link to user via email
Reset the password by providing session string

- **Component User Interface**
  Enter either username or email for the account to reset the password.
- **Component Objects**
  SendInput, GenerateSessionString, SendEmail, UpdatePassword
- **Component Interfaces (internal and external)**
  SendUserEmail sends the entered user email or username to the back end.
  GenerateSessionString is used to generate session string which is sent to user via email in the form of the link. While updating password, this session string is compared with string in the database.
  SendEmail sends the password reset link to the specified email address.
  UpdatePassword is used to update the password. It accepts session string and password.

- **Component Error Handling**
  Check if user already exists for specified username and email address
  Verify if session string matches the generated one
  Check if password is provided

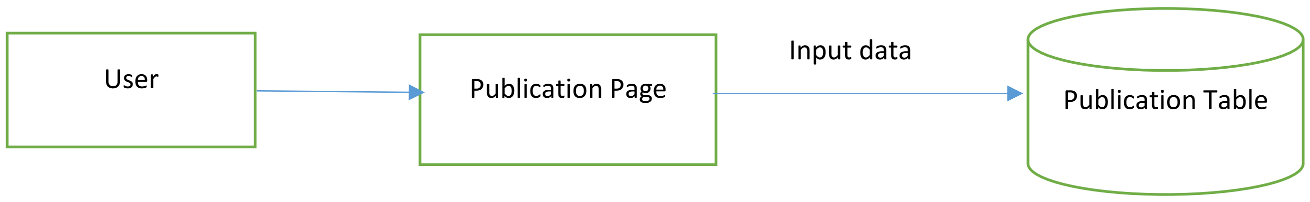
Component 5:

- **Component Name**
  User profile
- **Component Description**
  This is a user interface where user can view or edit his/her profile information which includes profile picture, name, location, city, state, country, summary, skills/interest, publication, etc.
- **Responsible Development Team Member**
  Gulshan, Jayendra, Xinquan.
- **Component User Interface**
  Display or edit of user info – profile picture, name, location, following, groups, publications.
- **Component Objects**
  GetUserInfo, SetUserInfo
- **Component Interfaces (internal and external)**
  GetUserInfo is used to get all the user info for profile page.
  SetUserInfo is used to set all the edited values of user.

- **Component Error Handling**
  Check if all the values are provided while editing.
  Check if session string matches with the one present in the database while editing.

Component 6:

- **Component Name**
  Publication
- **Component Description**
  This component allows the user to add or update publications. Fields – Publication name, ISSN, Abstract, Journal published in, date of publication, co-authors, PDF file of paper.
- **Responsible Development Team Member**
  Gulshan, Jayendra.
- **Component Diagram**



- **Component User Interface**
  On the publication page, user needs to provide publication name, ISSN, abstract, date of publication, journal, pdf file, and co-authors, if any.
- **Component Objects**
  GetPublicationData, ValidateData, SavePublicationData
- **Component Interfaces (internal and external)**
  GetPublicationData – Get all the required input from user.

  ValidateData – Check if all the required data is provided

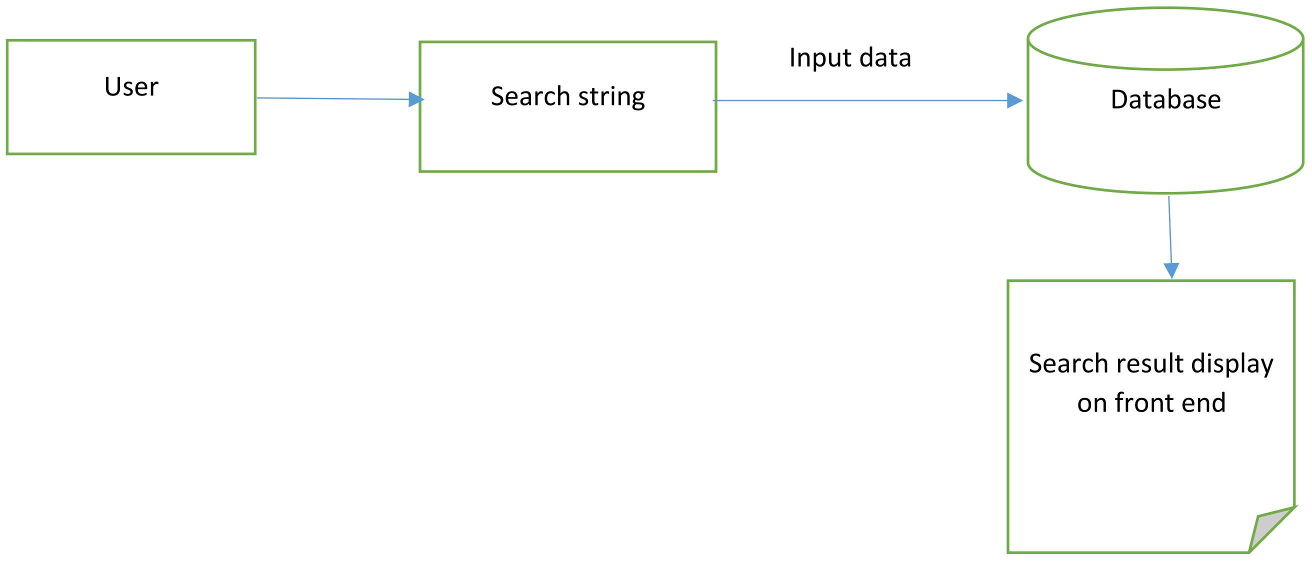
  SavePublication – gather all the data and send it to server to save in the database.

- **Component Error Handling**
  - Validated if all the data are provided to be saved in the database

- Check for duplication of ISSN
- Saving PDF file on the backend with appropriate name.

Component 7:

- **Component Name**
  Search functionality
- **Component Description**
  This functionality allows the user to search the database for various attributes – username, first name, last name, group name, group description, users having a given skill, etc.
- **Responsible Development Team Member**
  Gulshan, Jayendra.
- **Component Diagram**



- **Component User Interface**
  User enter any search query in the textbox provided. The search string is passed to the user and once response is received, the result is displayed in different sections for respective search queries – users, groups, skills, etc.
- **Component Objects**
  SearchForInput, SearchUsers, SearchGroups, SearchSkills, SendResponse
- **Component Interfaces (internal and external)**
  SearchForInput – validate input and send input string to back end for search

SearchUsers – search the string provided in user table and extract results if found

SearchGroups – search the string provided in groups table and extract the results if string matches with group name or description.

SearchSkills – search the string provided in skills table and extracts all the users having that skill

SendResponse – Once all the results are extracted, send the result to front end for display

- **Component Error Handling**
  - Check if valid search string is provided
  - Check in all the databases if value is present for search string
  - Check for SQL injection attempt

Revision History

Revision	**Date**	**Change Description**
Sprint 1	10/01/2017	Module: Login & Sign up using Duo authentication.
Sprint 2	10/15/2017	Login and sign up modules updated. Forget username, forget password and update password added. User profile added
Sprint 3	10/29/2017	Search functionality description added and how search functions, how various parts are handled.

Page Author/Creator: Adeel Bhutta
Last Modified: 8/23/2016