Table of Contents

1. GANTT Chart Planning

Full image can be found here:
https://raw.githubusercontent.com/401ChemistryGenealogy/ChemistryGenealogy/master/doc/img/updated_gantt_chart_v3.png

2. Autocomplete Documentation

The `/auto_complete` route accepts a query string with a name parameter and returns a `LIKE` query on the `people` table along with the relevant institutions.

example

```
curl -X GET "localhost:3000/auto_complete?name="
```

```
{
   "people":[
      {
         "id":1,
         "name":"todd lowary",
         "position":"professor",
         "institution_id":1
      },
      {
         "id":2,
         "name":"wei shi",
         "position":"assistant professor",
         "institution_id":3
      },
      {
         "id":3,
         "name":"ole hindsgaul",
         "position":"professor emeritus",
         "institution_id":1
      },
      {
         "id":4,
         "name":"david bundle",
         "position":"professor emeritus",
         "institution_id":1
      },
      {
         "id":5,
         "name":"moreten meldel",
         "position":null,
         "institution_id":null
      },
      {
         "id":6,
         "name":"raymond lemieux",
         "position":null,
         "institution_id":null
      },
      {
         "id":7,
         "name":"james baddiley",
         "position":null,
         "institution_id":null
      },
      {
         "id":8,
         "name":"jun liu",
         "position":null,
         "institution_id":null
      },
      {
         "id":9,
         "name":"clinton ballou",
         "position":null,
         "institution_id":null
      },
      {
         "id":10,
         "name":"harold jennings",
         "position":null,
         "institution_id":null
      }
   ],
   "institutions":[
      {
         "id":1,
         "name":"university of alberta"
      },
      {
         "id":3,
         "name":"university of arkansas"
      }
   ]
}
```

As you can see, leaving a blank name parameter matches against everything in the database. An example using a non-blank querystring:

```
curl -X GET "localhost:3000/auto_complete?name=to"
{
   "people":[
      {
         "id":1,
         "name":"todd lowary",
         "position":"professor",
         "institution_id":1
      },
      {
         "id":9,
         "name":"clinton ballou",
         "position":null,
         "institution_id":null
      }
   ],
   "institutions":[
      {
         "id":1,
         "name":"university of alberta"
      }
   ]
}
```

We can also autocomplete institutions:

```
curl -X GET "localhost:3000/auto_complete?institution="
```

```
{
   "institutions":[
      {
         "id":1,
         "name":"university of alberta"
      },
      {
         "id":2,
         "name":"carlsberg laboratory"
      },
      {
         "id":3,
         "name":"university of arkansas"
      },
      {
         "id":4,
         "name":"johns hopkins university"
      },
      {
         "id":5,
         "name":"university of california, berkeley"
      },
      {
         "id":6,
         "name":"nottingham university"
      },
      {
         "id":7,
         "name":"national research council of canada"
      }
   ]
}
```

```
curl -X GET "localhost:3000/auto_complete?institution=un"
```

```
{
   "institutions":[
      {
         "id":1,
         "name":"university of alberta"
      },
      {
         "id":3,
         "name":"university of arkansas"
      },
      {
         "id":4,
         "name":"johns hopkins university"
      },
      {
         "id":5,
         "name":"university of california, berkeley"
      },
      {
         "id":6,
         "name":"nottingham university"
      },
      {
         "id":7,
         "name":"national research council of canada"
      }
   ]
}
```

See for links to code:
https://github.com/401ChemistryGenealogy/ChemistryGenealogy/wiki/auto-complete-documentation

3. Search Documentation

The `/search` path accepts a query string with a `name` or `id` parameter and will return all the immediate links related to that person. That is: the person's supervisor, mentor, who they supervised and who they mentored.

example

```
curl -X GET "localhost:3000/search?name=todd%20lowary"
```

```
{
   "target":{
      "id":1,
      "name":"todd lowary",
      "position":"professor",
      "institution_id":1
   },
   "mentors":[
      {
         "id":4,
         "name":"david bundle",
         "position":"professor emeritus",
         "institution_id":1
      },
      {
         "id":5,
         "name":"moreten meldel",
         "position":null,
         "institution_id":null
      }
   ],
   "mentored":[

   ],
   "supervisors":[
      {
         "id":3,
         "name":"ole hindsgaul",
         "position":"professor emeritus",
         "institution_id":1
      }
   ],
   "supervised":[
      {
         "id":2,
         "name":"wei shi",
         "position":"assistant professor",
         "institution_id":3
      }
   ],
   "institutions":[
      {
         "id":1,
         "name":"university of alberta"
      },
      {
         "id":3,
         "name":"university of arkansas"
      }
   ]
}
```

notice: the `%20` code in place of the space character.
using `curl -X GET "localhost:3000/search?id=1"` will attain the same result.

See for links to code:
https://github.com/401ChemistryGenealogy/ChemistryGenealogy/wiki/Search-documentation

4. Technology Choices

application_tech

- Tier 1

  Our GUI application is a client side browser application. We will be utilizing the Angular JS 1framework. This choice was made primarily because Angular 1 has a proven history and a large community, because of this large community and the implied widespread use we expect that many solutions to technical problems can be solved simply by searching Google. Angular resources

- Tier 2

  In order to separate the concerns of the back and front ends we will deploy two different servers (on the same host). These will be denoted as the frontend and the backend server. The primary role of the frontend server is to serve the Angular application to the user, while the backend handles RESTful requests from the frontend. An implication of this architecture is that if one were to make a change to the frontend code the backend code remains the same. As such, the development process for the application should be simplified.

  The frontend server will be a node.js application using the express framework. We will also use a javascript build tool that will allow us to minify our static resources -- including our Angular application. Furthermore, the frontend server will use gzip to further compress the payload. To be clear, the point of the frontend server is to serve the Single Page Application along with other static resources (CSS, images, etc) to the user.

  Aside: point to consider

  Our backend server will be a RESTful Ruby on Rails (RoR) app. Like angular RoR has a proven track record of success with a very large and active community, and so we can again expect to find many online tutorials and various resources. RoR also has a reputation for being relatively easy to pick up and for fast prototyping. With the latest version there is the addition of a API mode. Using this mode we can limit the scope of the RoR app to performing RESTful services. The API mode is an integration of the rails-api project which has a history reaching back at least 4 years, so the maturity of this feature is present. RoR resources

- Tier 3

  The database layer will be an SQLite server. Again the factors that influenced this decision are primarily related to popularity. A high degree of popularity implies sufficient online resources and tutorials.

implementation_tech

These are choices for technologies required to implement specific requirements of the system.

- *Authentication*

  Requirement: Registration and login
  Technology: Ruby on Rails, bcrypt gem, and jwt gem
  Reason: jwt is still in active development, and both have clear documentation and over 13 million downloads

- *Editing Data*

  Requirement: Editing/curating information about scientists, relations between scientists (supervised, studied with, collaborated on, …), ideas, …
  Technology: Angular.js and d3.js will be employed in order to make editing relationships between these types of properties editable, and these changes will be pushed to the server. The technology is difficult to describe because we have to build it first.
  Reason: There is no pre-built library that offers such specific functionality.

- *Audit Trail*

  Requirement: Maintaining an audit trail of edits
  Technology: Ruby on Rails, paper trail
  Reason: This project is under active development, has thorough documentation, and 3755 stars on GitHub.

- *Visualizations*

  Requirement: Visualization
  Technology: D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3 emphasizes web standards and combines powerful visualization components with a data-driven approach to DOM manipulation, giving you the full capabilities of modern browsers without tying yourself to a proprietary framework. The library used specifically for this project is dagre/dagre-d3.
  Reason: There are a plethora of examples and a number of books written about this library. Also the GitHub repository has 45k+ stars.

See link for resources:
https://github.com/401ChemistryGenealogy/ChemistryGenealogy/wiki/Technology-choices-(and-pointers)

5. Use Cases

A) Register for website

Use Case Name	Register for website
Participating Actors	Not logged in user
Goal	Obtain a new personal account for this particular user.
Trigger	User chooses to register for a new account from the login dialog box.
Precondition	User does not have an account and desires a new one.
Postcondition	User's information for a new account is sent to an administrator for approval.

Flow:

1	Website prompts for the user's first and last name, their email, and a password.
2	User enters their information
3	User submits their information.
4	System creates a new user with the given information.
5	System sends the information to an administrator for approval.

Exceptions:

2	Email is already in use
2.1	System displays a notification to the user informing them that the email is already taken.
2.2	System returns to step 1.

Tests:

1	Upon arriving at the login screen, confirm that the prompt is displayed.
2	Test that the user can input text in (1) first name field (2) last name field (3) email field (4) password field.
3	Test that the user can successfully submit their information to the System.
4	[u] Test that the database (1) makes a new user (2) that the new user has the information that the User submitted (3) that this account CAN be accessed with the email and password tied to it.
5	Test that the information is sent to an administrator's notifications.

B) Log into website

Use Case Name	Log into website
Participating Actors	Not logged in user

Goal	Obtain logged in status on website with a registered account.
Trigger	User opens the login dialog box.
Precondition	User is not logged into the website.
Postcondition	User is logged into the website.

Flow:

1	System Prompts for email and password entry.
2	User supplies the email and password.
3	User confirms submission.
4	System logs in the User with the credentials provided.

Exceptions:

3	Incorrect email and password combination.
3.1	System displays a notification to the user stating that they entered incorrect email or password.
3.2	System returns to step 1.

Tests:

1	Test that the System displays the prompt for email and password entry.
2	Test that the User can input their email and password.
3	Test that the User can submit their entered email and password to the System.
4	[u] Test that the System successfully logs in the user with the supplied credentials.

C) Ability to Log Out

Use Case Name	Ability to Log Out
Participating Actors	Logged in Users
Goal	User is logged out of the System.
Trigger	User desires to log out of the System.
Precondition	User is logged into the System.
Postcondition	User is logged out of the System.

Flow:

1	User hits the logout button.
2	System logs the User out.

3	System returns User to the default entry page of the website.

Tests:

1	Test that Users are logged into the System.
2	Test that the System displays a method to the User to log out of the System.
3	Test that the method displayed to log out of the System is functional.
4	Test that the method, once pressed, will inform the server to log the User out of the system.
5	Test that the User is properly logged out (such as reloading the page won’t log them in).
6	Test that upon being logged out that the User is brought to the main entry page of the System.

D) Visualization

Use Case Name	Visualization
Participating Actors	Users
Goal	Display a fantastic visual representation of the database data.
Trigger	User desires to see a visual representation of the website’s data.
Precondition	The user doesn’t see a visual representation of the data.
Postcondition	The user now sees a visual representation of the data.

Flow:

1	User searches for a person in the search bar from the homepage.
2	System searches for the person and displays the results.
3	User has the information displayed for a particular person where visualizations of relationships that the person has, represented by the node-vertex relationship, are presented on the page the user is currently on.

Tests:

1	Test if the method can be interacted with by the User.
2	Test if the User is taken to the Visualization page.
3	Test if the Visualizations and search options are displayed.

E) Search

Use Case Name	Search
Participating Actors	Users

Goal	To search for a particular person on the website and obtain the results.
Trigger	User wants to find someone through the search page.
Precondition	User does not see the person through the search.
Postcondition	User is presented with the data of the search.

Flow:

1	User navigates to the home page where the search option is in.
2	User types in the person they want to search for into the box and initiates the search.
3	The System takes in the supplied entry and then searches the database for related results.
4	The System presents the results from the query to the User.

Exceptions:

4	The field desired to be searched for does not exist.
4.1	Proceed to the same final search page (step 4) however (refer to 4.2):
4.2	System displays a notification displaying to the User their search query had no results.

Tests:

1	Test that the User has a method to navigate to the page.
2	Test that the User can select search fields.
3	Test that the User can type input data into the search tool.
4	[u] Test that the System searches the database for the related results.
5	[u] Test that if the query has no results, the User is informed of it.
6	Test that incorrect field and query information can still work and proceeds as if normal.

F) Detail Information

Use Case Name	Detail Information
Participating Actors	Users
Goal	To obtain information of a particular person.
Trigger	User desires to learn more about a particular person.
Precondition	User does not know information about the other person.
Postcondition	Information of the person is displayed to the User.

Flow:

1	User navigates through the search results on the side of the visualization box and chooses a person to view more information on.
2	The System presents the User with the person’s information.

Tests:

1	Test that there is a method for Users to select to view the information of a person.
2	Test that the method is something a User can successfully interact with. (Is it functional?)
3	Test that upon utilizing the method, that the User is successfully shown the information.
4	[u] Test that the User is viewing the correct information.

G) Submit information

Use Case Name	Submit Information
Participating Actors	Logged in user
Goal	User submits new information.
Trigger	User would like to submit information.
Precondition	User knows what information they would like to submit.
Postcondition	Submitted information is sent to an administrator for approval.

Flow:

1	User chooses the option to submit information.
2	System displays the page for submitting new information about a person.
3	User enters the new information.
4	User confirms the new information they would like to add.
5	System creates a new save object of the data.
6	System sends the new information to an administrator for approval.

Tests:

1	Test that the submit button is present on the page.
2	Test that the submit button is functional.
3	Test that the page presents to the User the submit information page.
4	Test that the User can successfully confirm the way they wish to submit the page’s information.
5	Test that the User has a method to input the data.

6	[u] Test that the User has a method to confirm the information, and that this commits them.
7	[u] Test that the System creates a new save object of data.
8	Test that the information is sent to an administrator’s notifications.

H) Edit information

Use Case Name	Edit Information
Participating Actors	Logged in user
Goal	User chooses to edit information.
Trigger	User desires to edit information.
Precondition	User knows what information they would like to edit.
Postcondition	Information is changed in the system.

Flow:

1	User is viewing the details of a person and chooses to edit the entry.
2	System displays the pre-existing information about the person in fields that can be edited.
3	User makes changes to the person’s information.
4	User confirms that they want these changes made to the page.
5	System creates a new save object of the edited data and stores it for audits.

Tests:

1	Test that data is present on the page, as is the edit button.
2	Test that the edit button is functional.
3	Test that the page presents to the User the ways to edit the data on the page.
4	Test that the User can successfully confirm the way they wish to edit the page’s information.
5	Test that the User has a method to input the data modifications
6	[u] Test that the User has a method to confirm their modifications, and that this commits them.
7	[u] Test that when the User confirms their commit that the System accepts the modifications.
8	[u] Test that the System accepts the changes that the System creates a new save object of data.
9	[u] Test that the System successfully maintains this new save object of data in the database.

I) View Audit Trail of Edits

Use Case Name	View Audit Trail of Edits
Participating Actors	Administrator
Goal	Administrator can see the edits made to the information.
Trigger	Administrator would like to view edits made.
Precondition	Edited information exists in the system.
Postcondition	System displays a log of edits to the Administrator.

Flow:

1	Administrator navigates to the admin panel.
2	Administrator clicks on the view edits button.
2	System displays the log of edits for the information.

Tests:

1	Test that there is a method for Administrators to select to view the log of information about a person.
2	Test that the method is something the Administrator can successfully interact with.
3	Test that the Administrator is shown the correct information.

J) Notifications

Use Case Name	Notifications
Participating Actors	Administrator
Goal	Administrator views notifications for new user and data information.
Trigger	Administrator would like to view their notifications.
Precondition	A new user or data entry is submitted to the system.
Postcondition	The System displays the Administrator's new notifications.

Flow:

1	Administrator navigates to the admin panel page.
2	System displays the notifications that require approval.

Tests:

1	[u] Test that all Administrator accounts receive a notification from the System.
2	Test that the Administrators can view these messages.

K) Information Verification

Use Case Name	Information Verification
Participating Actors	Administrator
Goal	Administrator approves new information coming into the System.
Trigger	Administrator would like to approve or disapprove new information while checking their notifications.
Precondition	A new user or data entry is submitted to the system and the Administrators are notified.
Postcondition	The System is modified depending on the Administrator's decision.

Flow:

1	Administrator chooses the information they would like to approve or disapprove.
2	If the Administrator decides to approve the modification.
2.1	Administrator confirms the data modification.
2.2	The System makes the changes to the database and the webpage.
3	If the Administrator decides to disapprove the modification.
3.1	Administrator denies the data modification.
3.2	The System drops any modifications that could have been made to the System.

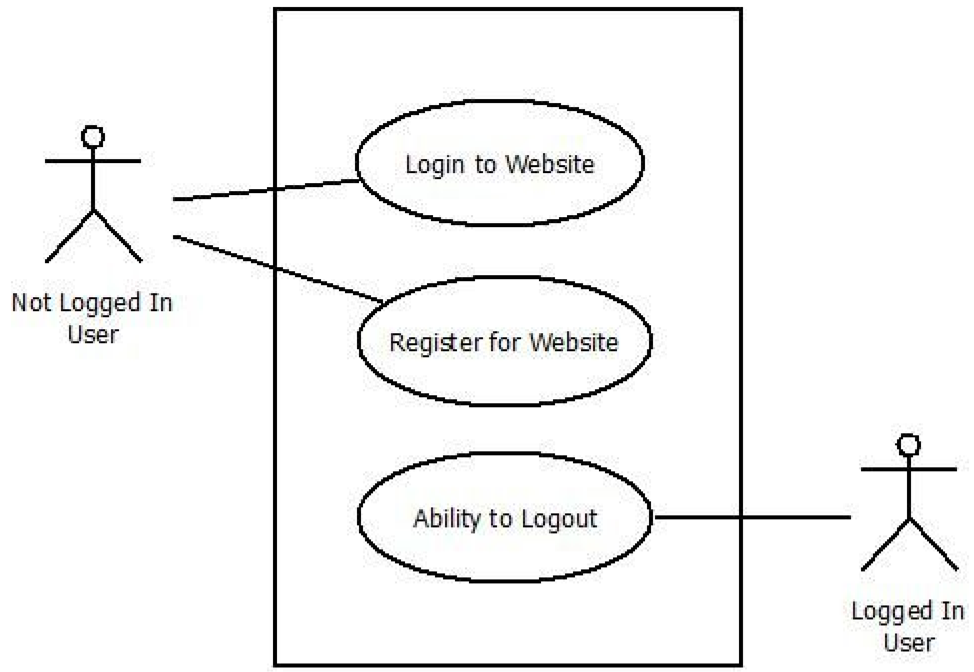
Tests:

1	Test that the Administrators have a functional method to confirm or deny changes.
2	[u] Test that the confirmation or denial of changes is committed by the System.

6. Use Case Diagram

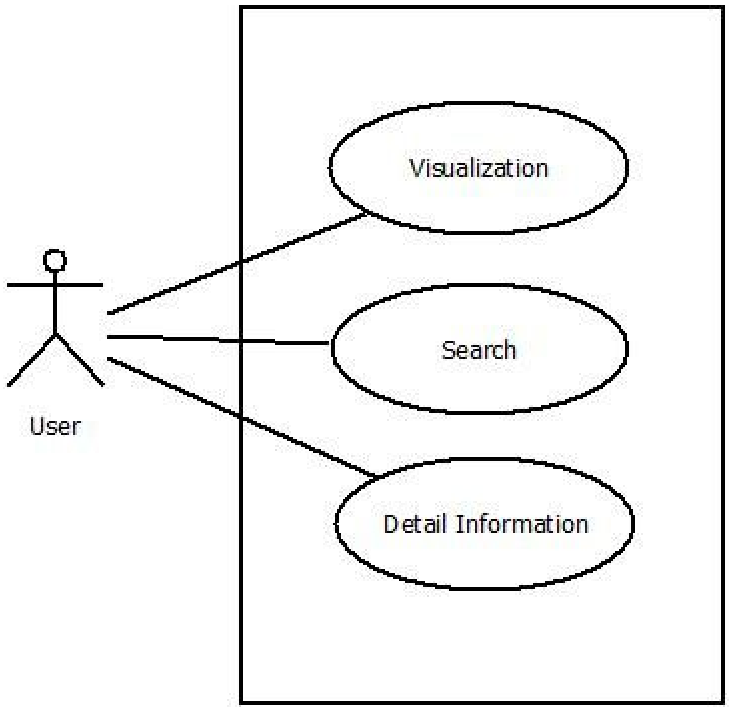
A) Registration and Login

This covers the use cases concerning registration, logging in, and logging out.

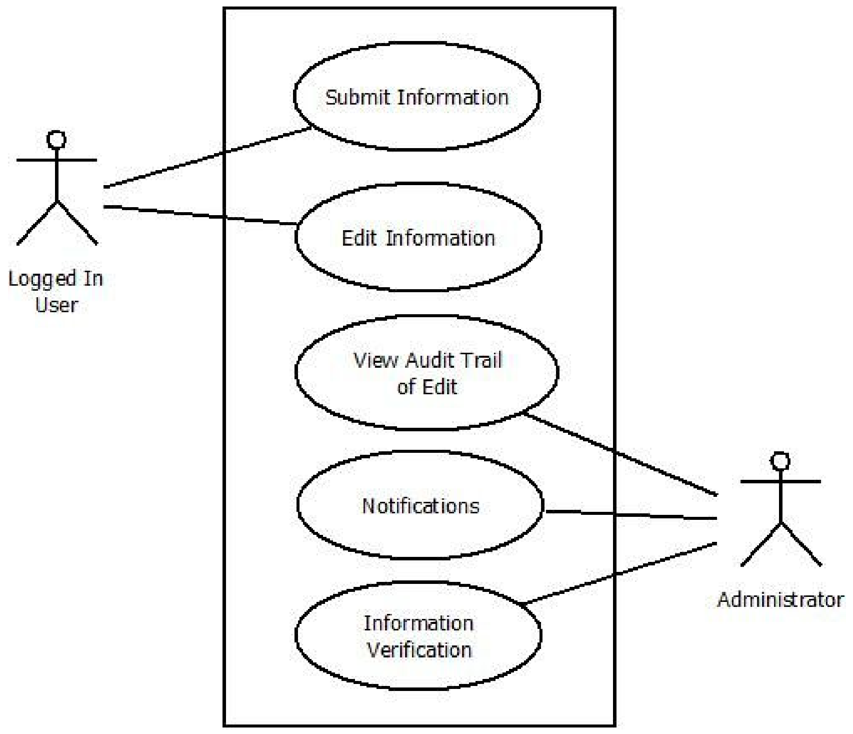


B) General User Functions

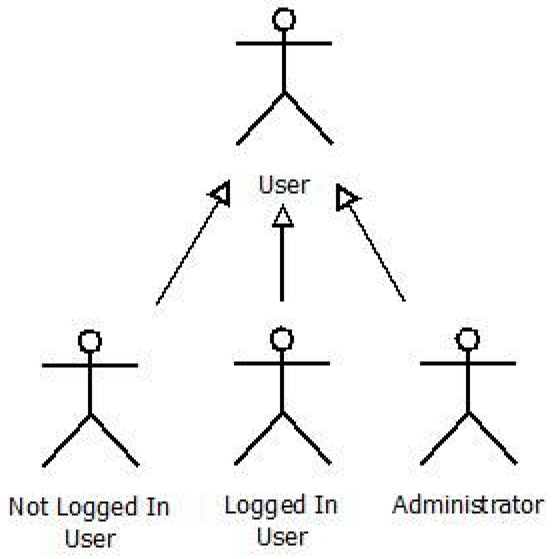
This covers use cases that can apply to any user.



C) Information Handling
This covers use cases that applies to users that are logged in and how information can be handled.



D) Actor Generalization
This shows how the actors are connected. Not Logged In User, Logged In User, and Administrator have different use cases, but any of them can use the same use cases as the User.


User
Not Logged In User
Logged In User
Administrator