Product Design

Team 25
Astitva Gupta
Sachin Kumar Danisetty
Trusha Sakharkar
Vivek Pamnani

Design Overview.

Architectural Design

We have the following modules:

1. User - It has the 'login' method where user sign in is done. Also, the searches done by the particular user are stored in the user database via some handler function.
2. UI - The UI framework in NodeJS are coded here. All UI related activities are looked after here.
3. Search-The search module has methods to call database queries, trigger crawling and all inputs related to previous searches.
4. We may add new modules as required. This is for the design as of now. Methods from different modules will be called. Like the output is stored in the database via some handler function to the 'Search' module for database queries.
   Also, the modularity reduces confusion and long written codes. Efficiency increases and error analysis is easy.

System interfaces

User Interface

Initial page will be a user login page. The user must sign-in with the different options provided to enter. There will also be an option for new user registration, which will require the user to set-up his credentials for future logins.

The next page shows a search phrase for the type of 'General Institutions' for example 'School Management Systems' or a Startup working on a domain. The user may enter the type of institution, number of employees or also its geographical location. There will be a search button to begin the search query and, on its completion, the results are displayed in material format.

Each result in the result list can be clicked leading to its profile page for additional information.

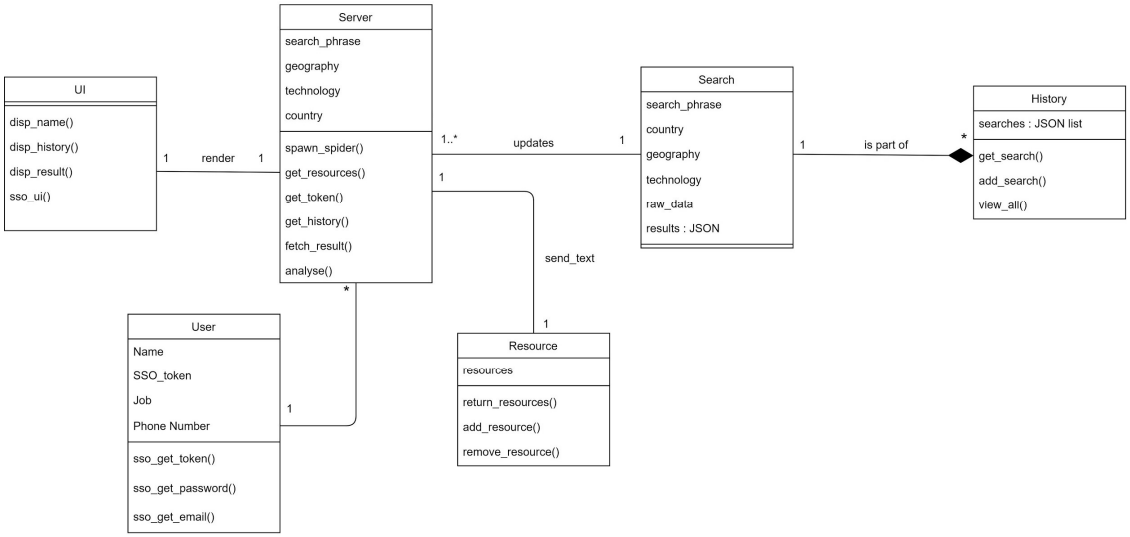
All the past searches can simply be accessed by going to the previous searches page from the navigation bar.

Some searches may not necessarily finish in a short while; hence the client will be notified via email once the query is processed.

APIs

We don’t provide any API to interact with the system.

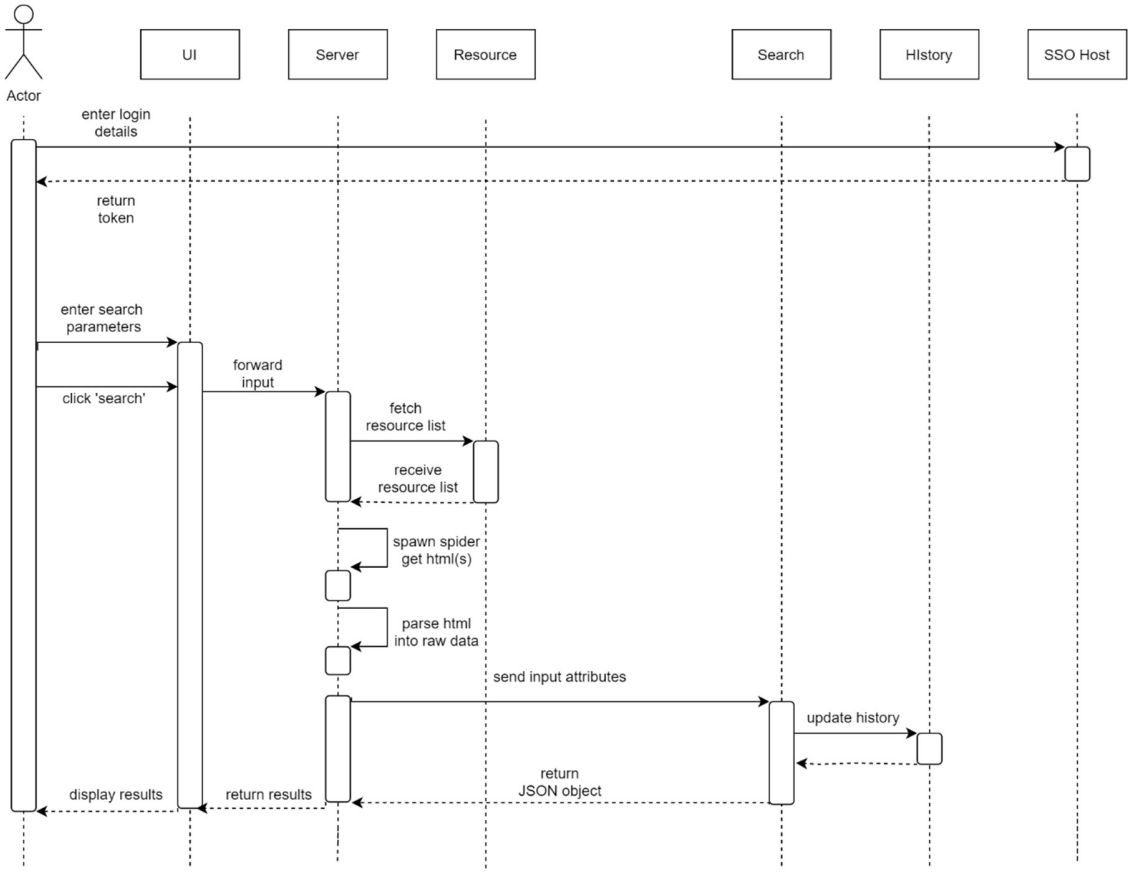
Model



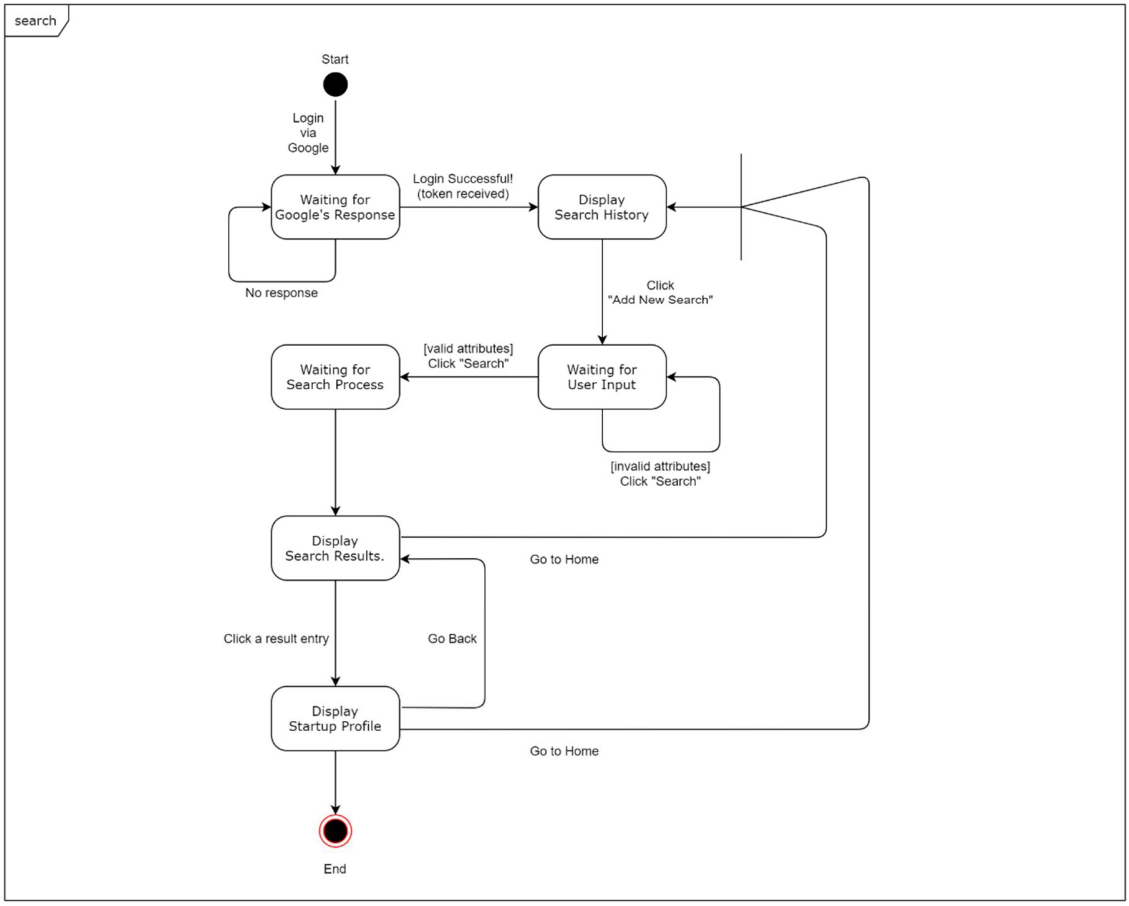
History	Class state: ● This class has all the previous searches, all the users have done stored in a database. Class behavior: ● A search handler method exists here. ● For each unique user, a method will return all the previous searches made by that user. ● All the database queries are called from here.
User	Class state: ● All the users that have registered with the software are stored in a database. Class behavior: ● A search handler method exists here. ● User profile details of each user are stored in the database and a method in this class returns all these values. ● All the database queries are called from here.
Search	Class state ● The data fetching part is handled here via the means of web scraping and web crawling and the result is subsequently stored in the PreviousSearches_Database. Class behavior ● This will load up the crawler and the scraper to find the related query on web and add this to the PreviousSearches_Database. ● A method to add to the database is called from here.

UI and HTML	Class state ● This class handles the front-end part of the application. Class behavior ● The front-end files are present in this class. ● Various static and dynamic tasks are handled from here.
Server	Class state ● This module handles the back-end part of the application. Class behavior ● The methods here implement the structure of the application and provide server support.

Sequence Diagram (for use cases: Login and Search)



State Diagram (for use case: Search)



Design Rationale

Initially, the software system would display all the details on the results page itself. Therefore, the startup specific information was moved to a separate profile page.

Loading animation was introduced to make the system feel more responsive (since, a search might take a long time).