Spark Tok – Project Report

He Yingzhi (A0237328B) Huang Ziyu (A0241579W)

Github Repository:
https://github.com/HappyPointer/IT5007_Project_Spark-Tok

Architecture and Implementation:

For the front end, we designed an interface that can interact with users and recommend articles according to their characteristics. For a user who has not logged in, our system can make random recommendations, and for users who have logged in, our system can recommend articles according to their preferences. The title of the article, the author, the abstract, and the preferences of the logged-in user are displayed in the front end. A logged-in user can choose to like an article, then click on the next one, and all the information on the page is updated accordingly. In addition, in the design of the front UI, our buttons have unique performance, which can display animation when the mouse is close, and can turn dark when clicked. 'Spark Tok' in the upper left corner of the page is also decorated with an artistic font. Also, the background of the page is a dynamic pinball. The main information is displayed in the bottom purple text box, there are two parts, the tab color and the height can change during switching between these two parts, so as to have a certain dynamic effect.

For the backend, the website is implemented with React framework, GraphQL and MongoDB. For better modularization, static web code and GraphQL-MongoDB code have been separated as two separate servers running on port 8000 (static web part) and port 3000 (GraphQL-MongoDB part). When a user is accessing our website, the browser will first request html/JavaScript files from port 8000, where react will generate the web content dynamically and send the html/JavaScript files to the user. In order to display the full content of the web, detailed data (for example, detailed paper information or login user information) is still needed. And the browser will access MongoDB with the provided data APIs on port 3000.

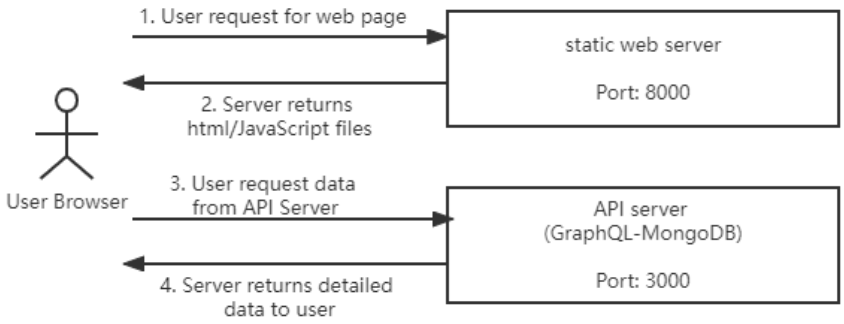

Figure 1. Basic workflow of the web server

To support user authentication and personalized recommendation, the core APIs provided by server are as follows:

addUser: Add a new user account to the system. Used for new user registration

checkLogin: Check whether the provided password corresponds to the user email. Used for login authentication

likePaper: Indicate the user's preference of the papers, which will influence the future recommendation results. Called when the user pressed the "Like" button on the page.

recommendPaper: Get a recommended paper with personal preference. Used when the user has logged in and his interest data (generated by "**likePaper**" API) exists in the system.

randomPaper: Get a random paper information. Used when the user has not logged into the system so the recommendation logic does not apply here.

UI Design:

The front-end design of this project hopes to be concise, clear and highlight the key points, so the main content is purple, occupying the largest area of the page, while the dynamic background can keep users happy and attract their attention to the page all the time. The login and registration interface on the top uses buttons to open or close, saving the page area, and after login or registration, the button will disappear, so as to make the page as simple as possible. Buttons and text boxes throughout the page are purple, and are designed with similar click animations to achieve a unified UI design. The UI design is shown below.

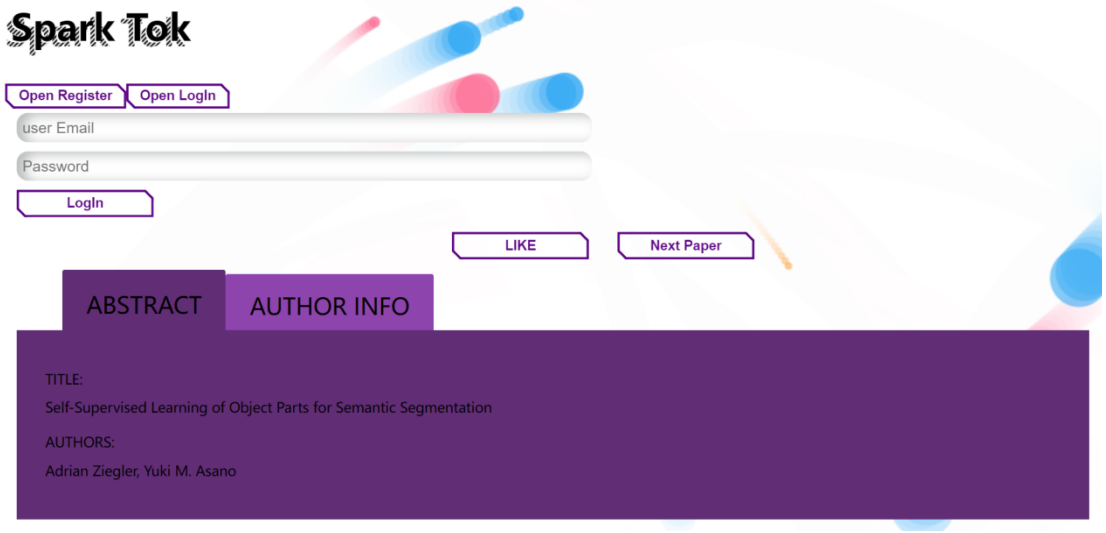

Figure.2 UI design page

This design uses a lot of buttons and text box effects. One button effects from http://www.bootstrapmb.com/, and using the dynamic background particlebg, from making a third-party libraries.

In addition, Lighthouse was used to analyze the performance. The analysis results were as follows: Performance: 93, accessibility: 71, best practice: 100, SEO: 78. The report of the performance is shown below.

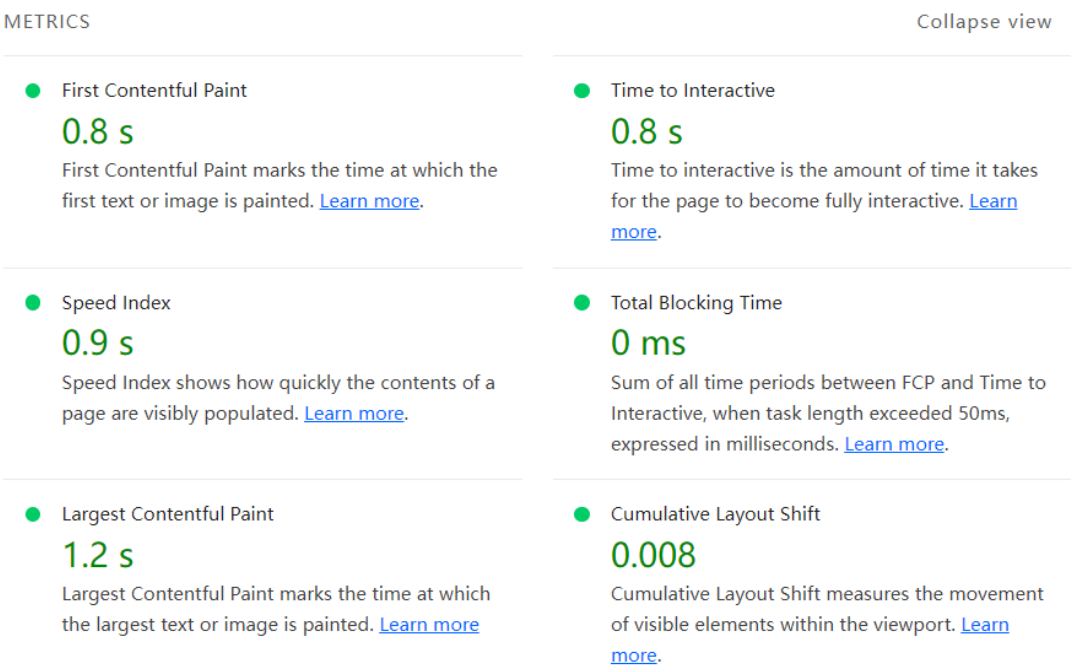

Figure.3 Light house report for performance

Feature:

1. User Preference Feature

In order to support the personalized recommendation, we need the user's preference feature to be the evidence for certain recommendation results. Therefore, for our system, it is necessary to save some data that represents the user's interest.

Representing a user's interest can be quite a complex topic, and in our case, we can simply achieve this with the help of our "Like" button. In our web page, for each paper shown on the screen, there is a "Like" button and by clicking it, the user is telling us that he likes this paper. The server will then record this action and update this user's "interest data". Here in our database, the "interest data" is represented by two lists of the same length. One list indicates the labels that the user likes and the other indicates the numbers of likes corresponding to the labels. Each paper recorded in the database also holds a list storing its labels. When a user likes a paper A, the labels of A will then be added into the user's "interest data".

2. Personalized Recommendation

One of the motivations of our project is that, instead of just listing all papers, our website gives personalized recommendation and save you from spending time browsing a huge amount of papers to find one you are interested in. So the recommendation system is actually vital to our project. Implementing a recommendation system itself can be very complex with the latest deep learning methods and obviously, we do not have the time and knowledge to do that in this course. But we also do not want our recommendation system to throw garbage data. In this project, we implemented a simple weighted random recommendation that can be implemented through a few JavaScript functions and is indeed working (not giving meaningless results).

In the previous section, we mentioned that for each user, his interested labels and corresponding numbers of likes are recorded. Based on that, the user will be recommended with a paper containing a certain label under the weighted possibility. For example, a user is recorded to like 5 papers with label “NLP”, 3 papers with label “CV” and 2 papers with label “GNN”. Then he will have 50% chance to be recommended with a paper containing label “NLP”. The chance to be recommended with “CV” and “GNN” are correspondingly 30% and 20%.

We also like to mention that, even though the recommendation logic in our system is simple for now, if we take the recommendation part as a black-box function, the input and output of our recommendation are the same as complex ones. So in the future, if we want to get a more advanced deep-learning-based recommendation system, we can easily replace this module by removing the current recommendation functions and calling the deep-learning model instead.