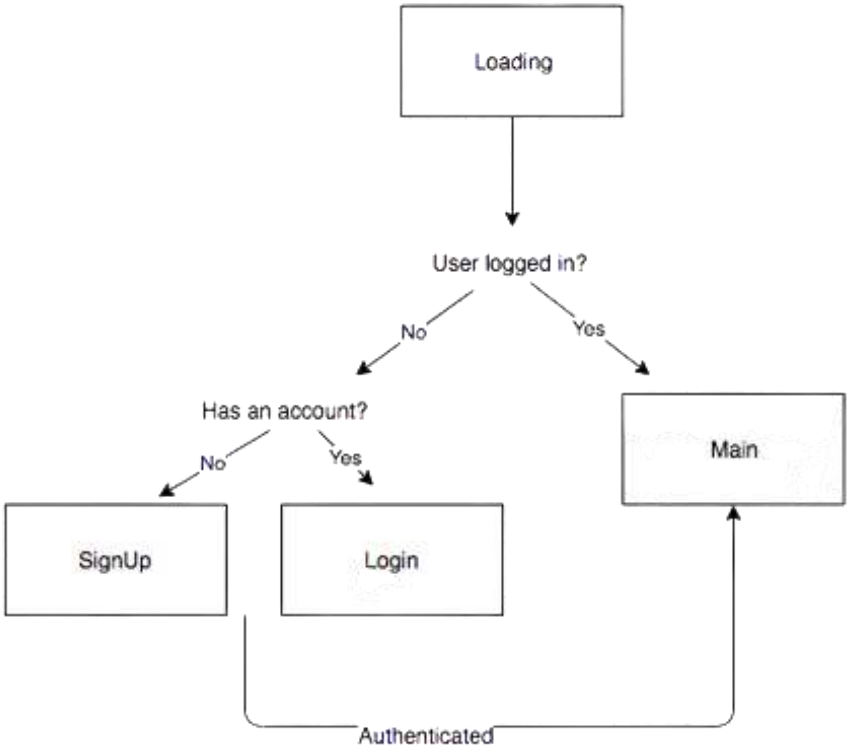
Project Weekly Progress Report Agile – Scrum

Semester	**W2023, SEM-2**
Course Code	**AML-2404**
Section	**Section 2**
Group Name	**D**
Student names/Student IDs	**Jash Vaghasiya - C0884733** **Nivedini Kathagonda - C0872720** **Keval Parmar - C0882386** **Monil Rupawala - C0882370** **Sai Divya Madhuri Guntupalli - C0882360**
Reporting Week	**10**
Team Lead for the reporting week	**Jash Vaghasiya**

1. Progress Made in Reporting Week:

I worked this week on creating the frontend and integrating crucial Firebase user authentication functionality for the skincare recommender system. The ability for users to create accounts and log in securely was one of the project's main components. To accomplish this, I added Firebase Authentication to the frontend, enabling users to sign up using their email addresses and passwords or login in using their already-established credentials. I ensured user data was secure and encrypted by using Firebase's strong authentication system.



I added client-side form validation to ensure that users supply accurate and correctly formed inputs during the sign-up and login processes to provide a seamless user experience. This required handling form submissions using JavaScript and validating information before communicating with Firebase's authentication services. I also included error handling to give users clear, illuminating notifications if there were any problems with the authentication procedure. The sign-up and login pages were created with user-friendliness and responsiveness in mind, offering an easy-to-use interface for users to interact with the system. Users can now safely register profiles and log in to obtain personalized skincare recommendations based on their interests and skin types as a consequence of these efforts. Firebase has played a crucial role in streamlining the authentication procedure and guaranteeing user data security, improving the skincare recommender system's overall performance and dependability.

2. Difficulties Encountered in Reporting Week:

The most significant difficulty encountered during this week's development was getting null replies from the Flask application's backend while making requests from the UI. The Flask app's '/API/advice' endpoint, where the backend analyses incoming data to give customers personalized skincare recommendations, was where the problem was located. Further investigation revealed that the backend was not sending the anticipated array of data, including the skincare recommendations, but instead was returning a 500 error number, which denotes an internal server fault.

The technical examination identified the backend code as a likely source of the issue. Notably, a problem with the data processing processes prevented the Flask app's 'get_recommendations' function from producing the required recommendations. First, the backend function used request. json['text'] to extract the user's description from the incoming request. The preprocessed data in 'df_processed['both']' was then converted into numerical vectors using the 'TfidfVectorizer' from the'scikit-learn' library. The 'df_processed' DataFrame appears empty or incorrectly loaded with the required data, which could cause vectorization issues. As a result, null replies occur from the backend function's inability to determine the cosine similarity scores between the user's description and the preprocessed data.

Further debugging and logging were implemented to evaluate the data transmitted from the front end to the backend to fix this problem. The 'df_processed' DataFrame was empty, preventing the subsequent vectorization and similarity calculations from succeeding. To fix this, the method for importing data from the backend was changed to ensure the DataFrame had all the essential information before the suggestion generation process began. Additionally, systems for handling errors were set up to identify potential exceptions during data processing and give the front-end helpful error messages instead of a generic 500 error. By making these changes, the backend can now analyze incoming requests, determine cosine similarity scores, sort the outcomes, and send an array of data to the frontend containing skincare recommendations while resolving the null answer problem. As a result of the updated backend code, the '/api/recommendations' endpoint is now operational and can offer consumers helpful, personalized skincare recommendations based on their input descriptions.