# Implementation

Implementation section defines the interaction between all of the components and how data is processed in to a meaningful format to display to the user. Following the 3-layer architecture as discussed in the design section, Figure 6.1 Implementation Workflow shows the implementation and workflow of the application. User interaction is recorded in the database rather than being communicated to the backend first. Thanks to the real-time serverless connection just the updates that need retrieving are fetched via the flask server. On logging in to the application the recommendation is fetched from the server for the specific user. Similarly, on adding and or removing something from the like list will fire up an API that will fill up the recommendation for the signed in user. All the necessary processing and data fetching is done between these API calls. The server retrieves the data and sends it to the 2 functions i.e. content based filtering and collaborative based filtering respectively. These 2 API calls based on their specific features which were discussed in section 5.5.2 Feature Engineering execute the models which then generate the recommendations of products in the form of product IDs and send it back to Firebase in the form of an integer array/list. The data for content-based recommendation is processed through a ***Tf-idf*** vector and their cosine similarity whereas, collaborative based recommendations generate results with the help of a sparse matrix that calculates the ***correlation*** between the ratings of products by the user and return similarly rated products. The ***decision tree*** on the other hand takes in to account the users and their unique characteristics depending on which products are recommended to them. The details for the models, the data that they process, and the operations that are carried out have been discussed in the text that follows.

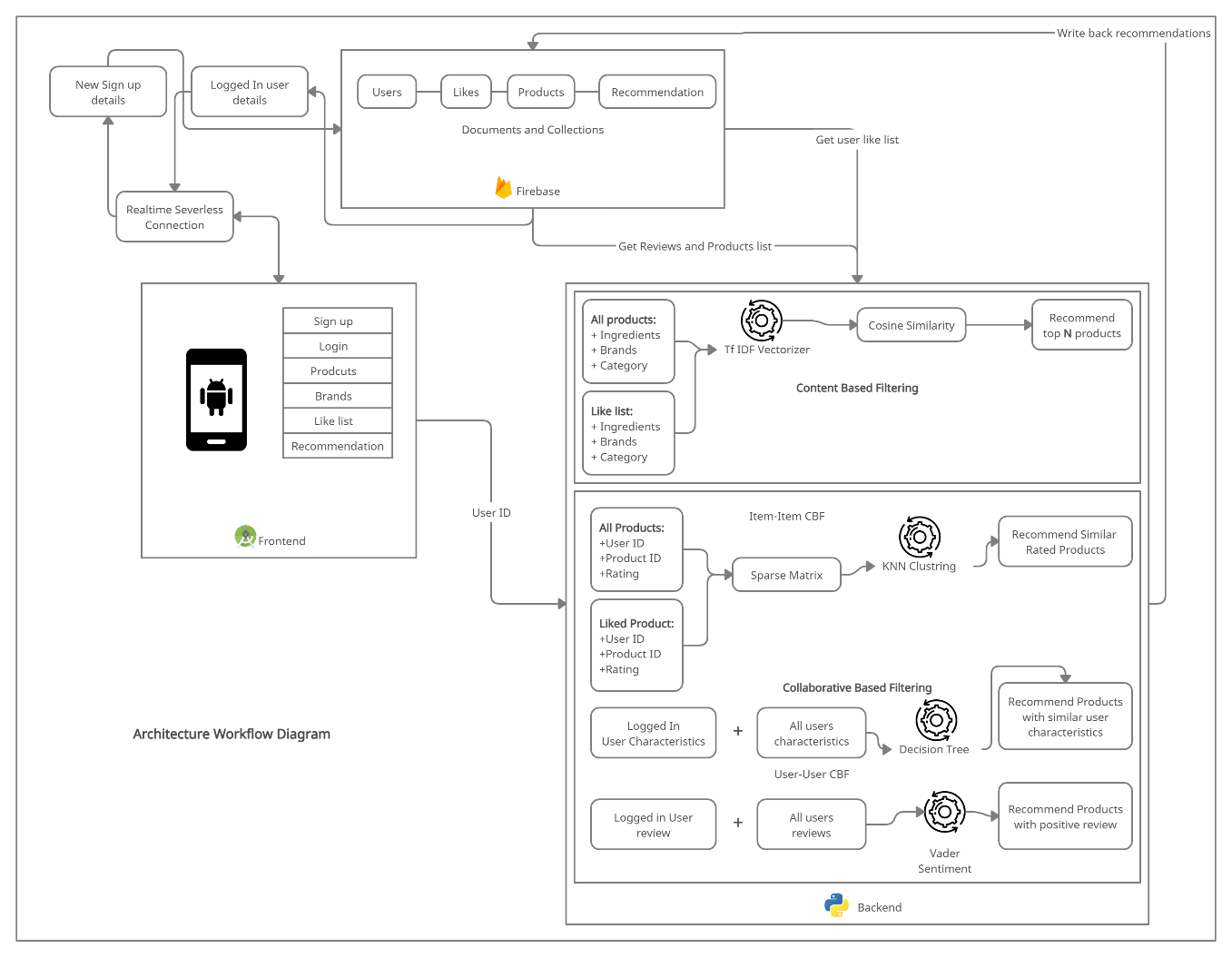


Figure 6.1 Implementation Workflow

## Backend and API

### Data Processing

Given that not much was available for processing as far as skincare data is concerned, data processing was a big part of the project. Following the design of the data discussed in section Data Pre-processing the data obtained from the source is unbalanced and represents ***skewness*** for multiple product IDs. This was overcome by duplicating the data, this makes the pattern occur more often and makes the algorithm recognize changes more accurately. Doing this also helps with ***stochasticity*** in the data, and any random occurrence of rows can be explained more logically now. Typically, any structured dataset includes multiple columns, a combination of numerical as well as categorical variables. Hence, another step that occurs in the processing part is ***label encoding*** where textual values are assigned a unique integer values which makes it easier for the algorithm to process. ***Scikit*** ***learn*** in Python provides a way of doing so based on alphabetical orders of the characteristics/features. Another form of doing this is called ***One hot encoding*** which produces binary vector of values so that the algorithm doesn’t process 2 characteristics as a dependency on one another, but One-Hot Encoding results in a ***Dummy Variable Trap*** as the outcome of one variable can easily be predicted with the help of the remaining variables. The Dummy Variable Trap leads to the problem known as ***multicollinearity*** hence I go forward with label encoding in this scenario. Besides this, the number of characteristics in our case is enough to avoid the reason for implementing one hot coding. Figure 6.2 shows processing of the user reviews that require their comments to be ***tokenized*** for the purpose of natural language processing, given this a separate column was created called ‘reviews cleaned’. The data was also cleaned by extracting a new data frame from the original with just the required features with a ***train test split*** of 70 and 30 which is usually the ideal amount for scenarios such as this. ***Cross validation*** was also applied to these algorithms with 5 folds.



Figure 6.2 Tokenizing Reviews

These scripts were first prepared with a prototype of the project using ***Jupyter Notebooks***. These required connection with the firebase admin, nltk corpus for python and the SQL server database. Important imports for this purpose can be observed below.



Figure 6.3 Important imports

#### Scrapping Data

As discussed, the scrapping part of the project was an important aspect. This concerned with using the link of the websites provided in the dataset to collect further information such as stars, reviews, username, and title for more products. Figure 6.4 below shows a part of the code that was implemented for this purpose.



Figure 6.4 Python Scrappy Framework Script

### Database Construction

Firestore database was created to get the API token to connect to the database. This token provides an Auth key to set up connection with the desired application as can be observed in Figure 6.5. In this scenario there were 2 application set up for access to the serverless architecture, the frontend application and the backend API. Figure 6.3 shows the firebase admin as one of the important imports for setting up the application.

Before integrating the database to the main server code base the data collected from the data engineering process carried out in section 5.3.3 needed to be inserted to the Firestore real-time database which was to be displayed to the user on login. Along with this a few users were to be created in the database for testing and verification. For this purpose, a script was created which uploaded all the 2000+ brands and 12000+ user reviews to the database.



Figure 6.5 Database Connectivity

Each time a new user is created the collaborative function fetches the new information from the database to incorporate the changes into the model for continuous learning. Figure 6.6 shows the query to fetch the new user data. The user IDs which have been specifically removed were integrated for testing purposes and have been discarded.



Figure 6.6 Fetching user data

### Routes and Server

The application takes in to consideration 2 routes one for content based and one for collaborative based filtering respectively. The input parameters, return statement and design are somewhat similar except the model and algorithm that are being called with.

Data frames and variables have been declared global as can be observed in Figure 6.8, to make them available in the application throughout. These are the data frames that will be processed by the algorithms.



Figure 6.7 Global Variable Declaration

Along with some functions that are required for the process, such as the item and recommendation function as seen in Figure 6.9 that log and print user recommendations to the console log and also return the specific product ID to the API.

The item function is used to get the item name from the data frame for logging purposes, whilst the recommendation function is set to return the top N stated recommendation product IDs to the API and for the logs.



Figure 6.8 Global Functions Declaration

Figure 6.10 describes the recommendation writing function from the content-based perspective, it returns the top best suggestion from the recommendation matrix generated by the model’s algorithm.



Figure 6.9 Collaborative Recommendation Writing Function

The server is executed when it encounters Flask’s main function. Figure 6.11 shows the application named app being executed by the Flask server. By default, the debug server is started at port 5000 however, this can be adjusted.



Figure 6.10 Server Start up

### API Implementation

API or Application Programming Interface provides a point of communication for the server with the frontend of the application. Any information sent or received through here. All of the functionality and models have been implemented in these routes. Figure 6.12 shows the route implementation for the collaborative based filtering.



Figure 6.11 Collaborative Filtering Route

## Machine Learning Model

### Content Based Filtering

Content based filtering refers to making recommendations or suggestions on the basis of the characteristics of the product itself regardless of the user. This is the most common approach used throughout in recommendation engines. Content based filtering in this approach included multiple algorithms that were implemented and tested. These included *KNN clustering, Decision tress with pruning and gini indexing, Naïve Bayes and ensemble learning*.

### Tf-idf Vectorization

Tf-idf vector work on the basis of calculating similarity between occurrences of a word throughout documents. Tf-idf now is the right measure to evaluate how important a word is to a document in a collection or corpus.

Equation 1 Tf-idf Vectorization

Tf – is frequency counter for a term t in document d

Df – is the count of occurrences of term t in the document

Idf – is the inverse of the document frequency which measures the informativeness of term t

Certain terms, such as “is”, “of”, and “that” or any other stop-words the like may appear a lot of times but have little importance. Hence, to counter this I need to weigh down the frequent term while scale up the rare ones. This is achieved by IDF aka ***Inverse Document Frequency***. It diminishes the weight of the terms that occur too frequently and increase the weight of the terms that occur rarely in throughout the document set the data for the Tf-idf vector balanced. Figure 6.13 shows the Tf-idf vector generation.



Figure 6.12 Tf-idf Vectorization

Now as I have all the product names in the form of a vector based on their occurrence I can use a measuring index to find out the similarity between what the user has entered/liked and suggest the top matching to them. For this I’ll be using ***cosine similarity*** index. It is a measure of similarity between two non-zero vectors. Non-zero, hence making it ideal for a sparse matrix where zero values are ignored. The resulting similarity ranges from −1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.

Equation 2 Cosine Similarity

A combined feature of brand, ingredients and product category is passed through the Tf-idf vectorization model and the top 2 recommendations with the highest results for the liked list items are populated into the recommendations list for that specific user. Figure 6.14 below is the sample code that represents the implementation for cosine similarity between each of the entries after which a recommendation function is used to return the closest matching value from the matrix.



Figure 6.13 Tf-idf Matrix and Cosine Similarity

### Collaborative Based Filtering

However, how relatable content based filtering may seem this can cause rather ambiguous suggestions since not always will be a correlation between the user and the product as trends are not monitored by this approach and if for example a user used to watch horror movies and now seems more interested in drama scripts then anyone who is being recommended what this user like or dislikes not necessarily follows the same trend, this is where collaborative filtering comes in.

#### Correlation Clustering (Item-Item based CBF)

Clustering the process of grouping liked values together. In the process of ordering values with respect to the correlation between them on the basis of liked items, a matrix was prepared that had user ID on the y-axis and product ID on the x-axis. The user entered value column is extracted from the matrix and correlation is calculated with any other user in the list. Any other item with a similar rating is suggested to the user. To make the training process more viable ***cross validation*** was implemented for catching all the product IDs in the training sets. Exactly for this reason users who had bought only a single item and products that were reviews only once were dropped. This type of filtering is ***item based*** since there is no relation whatsoever with user characteristics. Figure 6.14 shows the implementation for training and testing the KNN model.



Figure 6.14 KNN Training and Testing

After the best fit for the value of K i.e. the number of items in a cluster has been identified it can be used to predict on the basis of the new value as shown below.



Figure 6.15 KNN Predictions

#### Decision Tree (User-User based CBF)

Decision trees and simple probabilistic models that compare the change of occurrence of each scenario with every outcome in a tree layout structure. There are many types of decision trees and multiple methodologies available. In this project I have followed a decision tree with ***gini index*** and ***pruning***. Pruning is a data compression technique in machine learning and search algorithms that reduces the size of decision trees by removing sections of the tree that are non-critical and redundant to classify instances. That is, it reduces the complexity of the algorithm whilst making better prediction results. The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

Equation 3 Gini Index

The model is trained to classify product IDs based on user characteristics. It takes into the account the binary encoded values of hair colour, skin tone, skin type etc. as the X values and the product IDs that those respective users have bought based on their characteristics as the Y values. The results obtained were a mixed bag of values rather than perfectly accurate since the data collected is not so continuous to help the algorithm generate the right structure. Ensemble technique was also used in this case to try to overcome the lack of continuity of the data where a few users had only bought a handful of items. ***Bagging*** was introduced in this case by duplicating the data with a ***K-fold cross validation*** of 5.

The implementation of the decision tree itself was divided into different function calls which provides better re-usability for both content and collaborative based filtering. Figure 6.17 shows the split dataset function which when passed with the no of column (4 in case of content based and 5 in case of collaborative) for the matrix return with a split of X and Y respectively. The second function return the hit rate of the algorithm while the third function is for training the decision tree based on the Gini index.



Figure 6.16 Decision Tree Training, Testing and Split



Figure 6.17 Decision Tree Prediction

### Natural Language Processing

Making conclusions is an extensive part of human analysis. Natural Language Processing or ***NLP*** is the study of making sematic analysis from textual inputs in the field of computer science. Python introduces 2 methods of making NLP analysis from within the library held with the ***PyPI*** (Python Package Index), *Textblob and Vader Sentiment analysis*.

In this approach I have utilized the positivity score from the Vader sentiment analyser. User reviews have been summed on the basis of the product ID and their average recorded into a data frame. Figure 6.16 shows a sample of the prepared product ID and their respective user review scores. Vader suits better to our needs it has a better library built for semantically analysing an E-commerce website reviews and recommending user products with similar reviews.

#### **Vader Sentiment Analysis**

***Vader*** uses a list of lexical features (e.g. word) which are labelled as positive or negative according to their semantic orientation to calculate the text sentiment. Vader sentiment returns the probability of a given input sentence to be positive, negative, and neutral. It is optimized for social media data and can yield good results when used with data from Twitter, Facebook, etc. Vader sentiment analyser returns 4 values to summarize the contents of the input sentence these are

* Positive ratio in percentage format ranging from 0 to 1
* Negative ratio in percentage format ranging from 0 to 1
* Neutral ratio in percentage format ranging from 0 to 1
* Compound that lies between [-1,1] just like polarity in Textblob

The sum of positive, neutral, and negative will sum to 100%



Figure 6.18 Product Review Scores

These scores are generated on run time hence making this a ***model-based approach*** rather the usual memory-based approach. Figure 6.17 highlight a bit of the code that was used to generate product recommendations based on an item-item collaborative filtering. The most similar score is then return from the list of which is what the next product the user might be interested in.



Figure 6.19 Vader Sentiment Analysis

Predicting based on the user review:



Figure 6.20 Prediction

## Frontend – Development

Frontend of the application was developed in Java with the wireframes as the base concept of the application. However, there were many changes that occurred during the implementation process, these consisted of visual changes as well as functionality changes which later either became vague or out of scope. The following section of the document shows the step by step process opted to achieve the completion of the application as design.

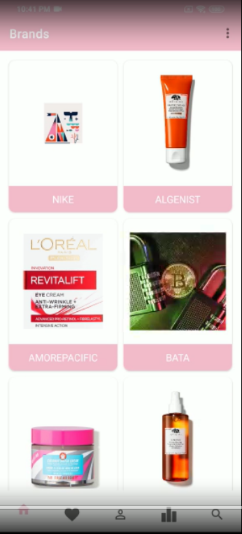
### Sign in Screen

The layout of the application started out with the sign in screen. A pink theme has been implemented throughout. User email and password is requested upon login and if it does not exist then the user can choose to create a new login password. By clicking on the sign-up button, the user is taken to the sign-up screen. The code snippet shown below gives an overlook of the implementation of firebase authentication for user sign in.



Figure 6.21 Firebase Connection and Authentication

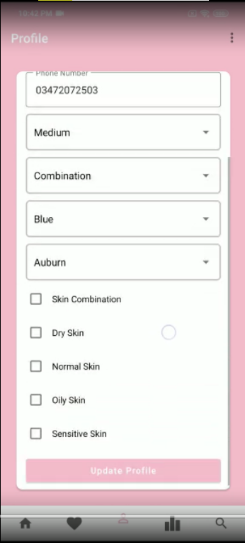
### Home Screen

The default or home screen has been chosen as the brands page which displays all the brands in the application. It delivers a thumbnail view of the brands list and which can be clicked on to redirect the user to the products list screen for that specific brand.

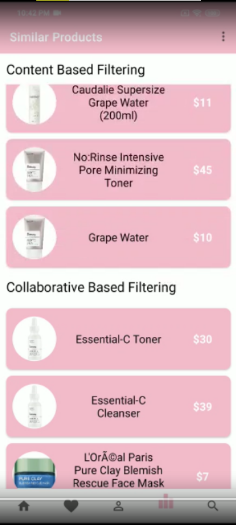
### Like List

The like list has been implemented as originally planned out. The list has all the item that the user has reacted to. The product IDs on this screen are what used to generate suggestion for the currently signed in user.

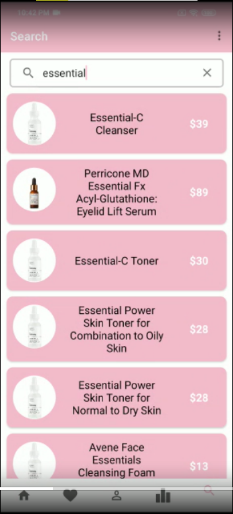
### Profile Screen

Instead of the original idea of having a hamburger menu for user profile change the three-dot menu on the top right shows the choice or logging out of the application and the user profile section has been moved to the bottom traversing bar. The profile update form allows user to update their credentials as well as the characteristics they originally entered whilst signing up for the application.

### Recommendation List

Following the similar from the wireframe, the recommendation screen hosts 2 tabs, one for content based and collaborative suggestions except this time they are vertically aligned instead of horizontally. The two tabs can be scrolled independently. While loading this screen the user might observe some delay due to the reason that product IDs to be fetched here are being generated by the model at the back via the API call.

### Search Screen

The search is where the user can look up from the extensive list of products a specific item that matches the words entered into the search/input bar at the top of the screen. Search screen has been kept a bit simple for an easy implementation since it only limits the search to products list and not brands list. Allowing brands list to be traversable would require a new schema to be prepared for the firebase database.

### Item Description

The item description page has been limited than what was discussed in the wireframe section. This page now displays only the current user entered comments and rating instead of the overall feedback. This restriction was introduced to reduce the complexity of the frontend of the application. Since requiring all the feedback would have to loop through multiple documents and collections. The description of many of the product may seem incomplete this is due to the scrapping process carried out which captured only partial text from the paragraphs entered on the website. Here the user can add the item to their like list, review the product, and rate the item from a total of five stars. Whenever data is updated here the models are executed to adjust to the dynamic nature of the application where continuous learning is the key.