# Project Track 1 Stage 4 Report

**Group Number: 097**
**Group Members:** Ria Desai, Julie Lima, Nitya Sunkad, Oju Chaudhary

**Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

Although our final project stayed mostly consistent with our project proposal, there were a few necessary changes. Since we wrote the proposal in a way that accurately represented our vision for the project, we did not make any major changes, just minor tweaks.

1. Reviews: In our proposal, we explain that we originally wanted users to be able to write their own reviews. We expected this to be a POST request. Our decision not to include this feature is thoroughly discussed in a later question*. Essentially, we did not have the bandwidth to assess the reviews and ensure that they would be appropriate and genuine. Since it can be difficult to sort through spam reviews in other apps like Amazon, we did not want the same issue to arise in our app. Instead of maintaining this feature, we spent more time on honing the product information page so that users would have as much detail as possible to help them decide whether or not to add the product to their bag.

2. Sign Up: In our mockup in the proposal, we only show 3 text inputs for the sign up feature: Username, Password, and Password confirm. In our actual implementation, we decided to add First Name, Last Name, and Email. We specifically decided to include these features so that we could improve the user's experience. Having their first name allowed us to display personal messages such as "Welcome to Vanity, Oju" or "Oju's Bag". Our Users table stores this information as well.

3. Product Page: Our proposal did not focus much on product information, but this ended up being a large component of our app. When working with the dataset and testing out different queries with our dataset, we realized that we could gather much more information about products than we initially thought. We decided it would be a good idea to add a page for each product that displayed the name, brand, popularity, page views, and reviews of the product. This page also displayed products that other users who bagged this item also bagged, and the results of our recommendation algorithm that suggested similar products to users.

**Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

Our application, Vanity, succeeded at creating an informative, data-driven social media service for beauty and skincare product users.

As a product discovery tool, Vanity enables users to search for and explore new products, view information about those products, read product reviews written by real customers, and interact with other users on the app. Users can add their favorite products to their bag, and based on the product the user is browsing, Vanity also generates helpful product recommendations for the user.

As a social media service, Vanity also allows users to search for friends on the app, view their friends' makeup bags, and learn about the products their friends are using.

In the future, we hope to incorporate a wider array of beauty and skincare products beyond the 8,495 products enumerated in the Kaggle Sephora products dataset. As we scale our app beyond our current dataset, we hope to further improve app usefulness.

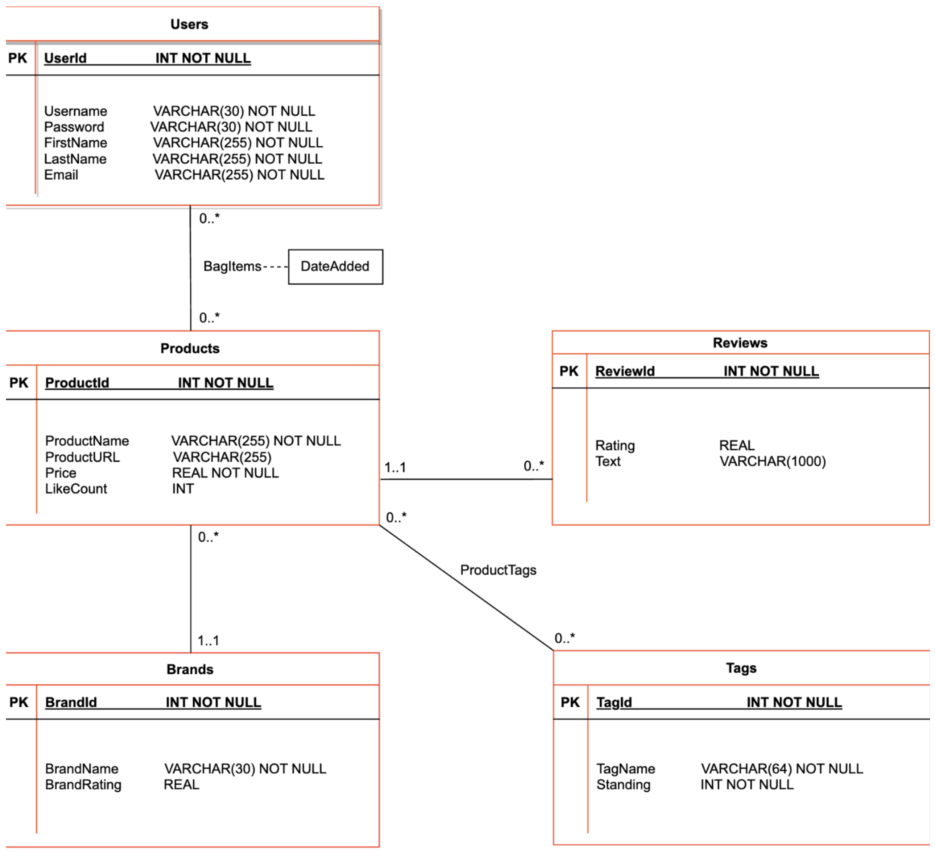
**Discuss if you changed the schema or source of the data for your application**

When building our app, we found it necessary to refine our schema and modify how we planned to use our sources. Our schema changes are discussed in more detail in the next question**. To briefly go over them, here are the changes we made to our schema:

1. Added a ProductClusters table: The recommendations algorithm needed a table to store the clustering that it created. Our ProductClusters table's primary key is the ClusterID.
2. Adjusted the Products table's attributes: We removed the ProductURL attribute and added the Size and ViewCount attributes. We calculate the ViewCount dynamically as users interact with the application.
3. Adjusted the Reviews table's attributes: We added the DateAdded attribute to help organize the reviews and display the most recent, relevant ones.

Additionally, while the kind of data we use did not change, we did end up using a different product reviews dataset than we had originally selected in our project proposal. This was simply due to the variety of information included in this other dataset, which we found more useful for our application, along with a more seamless integration with the product dataset. While we used a different dataset, we still used it for the same purpose of displaying useful product reviews by real customers to our application's users.

****Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

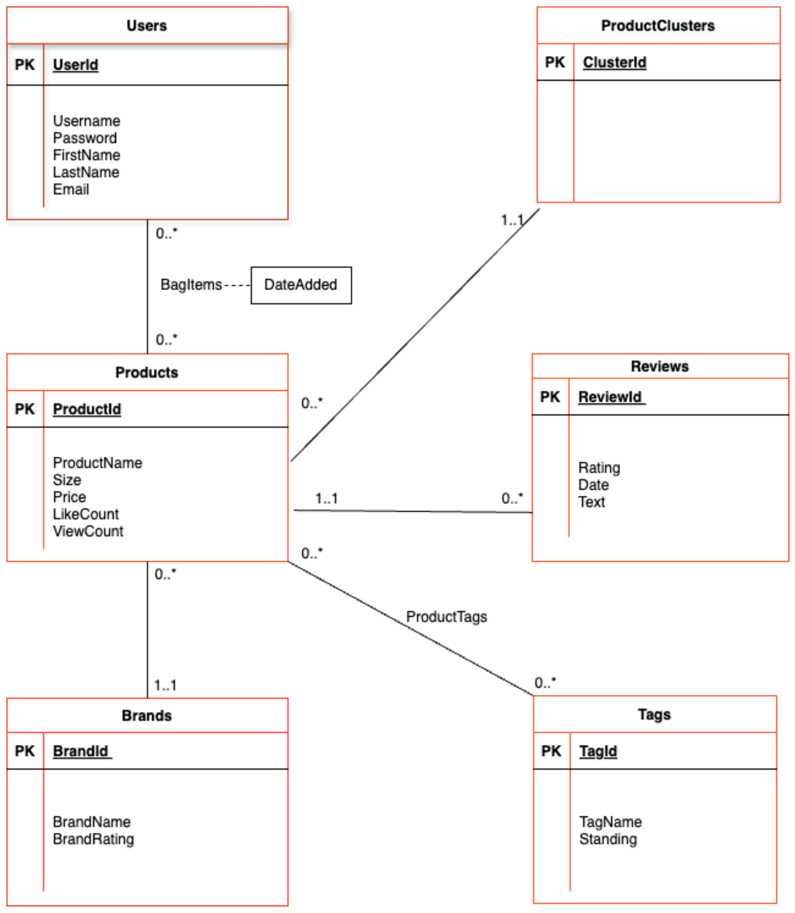


Our first draft of the UML diagram, pictured above, included five entities — Users, Products, Reviews, Tags, and Brands, with many-to-many relationship tables.

As we began entering our dataset information into GCP, we noticed that we needed to add and remove a few attributes from our initial tables. For example, we removed the ProductURL attribute from the Products table, as it didn't exist in the dataset. We also added the Size attribute to the Products table, as we figured this product info could be useful to the user, and we had it in the dataset.

Eventually, wanting to add a product recommendation feature to our app, we developed a feature to calculate product similarity using KMeans clustering. To support this feature in our database, we created a sixth entity called ProductClusters to store information about products and the similarity clusters each was associated with. We decided to make ProductClusters a new entity as opposed to a new attribute in the Products table, because it would be useful for future development of our recommendation algorithm. For example, if we decided to incorporate KNN clustering in future, we could add the new attribute KNNCluster to the ProductClusters table.

We believe that our final UML design is more suitable for our application because it is compatible with our dataset constraints and supports useful and necessary app features.



**Discuss what functionalities you added or removed. Why?**

Initially, we aimed to allow users to search for products by scanning the barcode for that specific item to get information and reviews about the product. This posed both technical and logistical challenges that rendered this feature infeasible for our application. On the technical side, implementing a barcode scanning feature required a camera, which implied an application that would be compatible on a mobile device. We decided this would not be possible given our existing skill sets and the time constraints of the project, as we wanted to prioritize the successful application of database concepts and not "fancy" technical features that would take time and research to implement. A logistical challenge was also presented, because while there were multiple online databases that contained item information by UPC codes, these databases were not easily accessible in a .csv format, and had no attributes that could match items in the UPC database with items in the Sephora dataset we had planned to use.

*We also planned to allow users to write their own reviews on products to add to the reviews page. We decided to leave this feature out to maintain the integrity of our application. The reviews in our dataset came from verified customers of the products they corresponded to, which is confirmed on the Sephora website. Our application had no way of checking whether new reviewers had actually purchased the product, which would potentially diminish the quality of the reviews on our application and defeat the purpose of providing users with quality feedback on makeup products.

**Explain how you think your advanced database programs complement your application.**

For the friends lookup functionality, we implemented two advanced queries that give you greater insight into the bags of other users on the app that you may want to view. The search results list 15 users ordered by the Levenshtein distance of their username, first name, or last name from the search string entered. With each listed user, we also display the number of products in their bag and the similarity level (on a scale of 1-5) of the products in their bag with the products in your bag, each of these being an advanced query. When looking for friends, the number of items in a user's bag can help determine legitimacy and trust in a user (similar to follower count on Instagram), and the similarity level can help determine if this friend's bag would be a good place to look for products that you are likely to enjoy, as well. Therefore, both of these advanced queries complement the purpose of our application.

For the product information functionality, we implemented two advanced queries: product popularity and "Users who bagged this product also bagged…". The latter is a collection of products that you may be interested in if you enjoy the current product you're viewing, since other users who like the current product like those other items, as well. The product popularity measure is a very important standard in the makeup and beauty industry, since it can help show the legitimacy of a product and/or brand. Therefore, both of these advanced queries help users find trustworthy products that they would like to try, which complements the purpose of our application.

**Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or were to maintain your project.**

**Julie -** One technical challenge we faced was integrating the backend Flask API with the React frontend. None of our team members had done work with integrating the two before, so we had to take it step-by-step and learn from the beginning. One thing that we found very helpful was using Postman to make test calls to our Flask endpoints, allowing us to debug the backend and frontend separately instead of struggling with error logging to figure out where the issue was. Once we knew the endpoints were working correctly, we could focus on the integration with the frontend, using try-catch blocks to handle exceptions and logging them to help with debugging. When debugging the frontend, we used the inspect and console features of our browser to view what we logged. Throughout the entire project, we also frequently referred to the Flask API Documentation, to learn more about syntax and appropriate code design.

**Nitya -** One technical challenge we faced with creating our creative component — a recommendation algorithm — was learning how to build a recommendation algorithm with no prior knowledge or experience in the area. The most important questions we needed to ask ourselves here were:

1) Which algorithm is most well-suited to our use case?

2) What are the inputs and outputs to this algorithm?

Using some background knowledge from linear algebra and probability & statistics classes, we decided that a clustering scheme would be appropriate for our use case — determining which products were similar to which other products. Using tutorials and articles online, we learned how to use sklearn's KMeans clustering algorithm and understand its inputs. We learned that Kmeans needed standardized vector data as input to work properly, so we developed SQL queries and Python code to retrieve our product data, process it, vectorize it, and standardize it, so we could pass it to Kmeans as valid input. Finally, we needed somewhere to store this new classification information, so we created a new table on GCP called ProductClusters. Now, the backend could query recommendation information directly from the database.

**Ria** - An additional technical challenge was posed when figuring out how to store a user's information after they logged in or signed up. We quickly realized that, when a user signed up or logged in, although a request to the backend could be made to put their information in the database or check against existing information, this didn't guarantee that their profile was signed in to when navigating to the "My Bag" page or clicking "Add to Bag" from the product page. We needed to store a user's information upon log in or sign up, so the rest of the app's features were still in the context of that unique user for the time that they were using the app. After some research, we implemented "User Context" in the front end to allow us to store key information about the app's current user that we could call upon whenever necessary to make requests to the backend, so users only had to input their information once upon login and the rest of the features were presented unique to their user id.

**Oju -** We faced a couple of technical challenges related to making API calls in the frontend to call the functions in the backend.

First, we faced a challenge when trying to make GET requests. When we tested the API calls on PostMan, these GET requests were successful, but when we actually made the API call and used the *fetch* function on the frontend, we consistently got error messages. These error messages specifically told us that GET requests do not support JSON bodies. This ended up being a quick fix: changing the GET requests to accept parameters in the URL itself. For example, we changed the /bag-items endpoint to accept the user_id as an input in the url in this manner: http://localhost:8000/bag-items?username=${user.userName}

Second, we faced issues when trying to make multiple API calls in one page. Although we were able to successfully implement the /bag-items, /create-bag-item, and /delete-bag-item all on our MyBagPage, we were not able to do the same for our ProductPage. Our ProductPage supports the /product-info and /create-bag-item endpoints, but it does not support /get-product-reviews on this page itself. We ended up directing the user to click on a button that leads them to a ProductReviewsPage. Since the /product-info endpoint had two advanced queries implemented in it, we predicted that there was some complexity that caused the /get-product-reviews API call to get messed up. When making the /get-product-reviews API call on the ProductPage itself, the other API calls stopped working. There would be error messages in the console and the UI would not get populated.

**Are there other things that changed comparing the final application with the original proposal?**

In the final application, our MyBagPage looks and functions slightly differently than we planned in the original proposal.

Design-wise, we originally drew out the shape of an actual bag that we would have users populate. Having to display all of the items within a bag shape would be difficult to implement because of the dynamic size of the BagItems entries. So instead, we displayed the products as an array on the screen. This was a more typical format to implement in React.js and CSS, especially when using the React.js map. To do this, we took the JSON results of the /bag-items{user_name} query, stored them in a list, then converted this list to a map.

We also originally planned for the MyBagPage to consist of products that users actually own in real life and want to virtually track and organize. Being able to digitally view one's personal inventory of beauty products would be extremely helpful when choosing what products to use everyday and when remembering what products one owns. This concept was difficult to implement without a barcode scanning feature, so we kept it to a more simple functionality by allowing the users to search for and add a product through key word search.

**Describe future work that you think, other than the interface, that the application can improve on**

One function we would like to add to our application in the future is the ability to "friend" people. That is, on top of being able to search for other users and look at their bags, being able to "add" them to a list of friends, similar to how you can "follow" people on Instagram. In this way, you can refer back to your list of friends to find the same people easily, and you can also find who your friends' friends are, and perhaps find some new products in their bag! We think this would really contribute to the "social media" aspect of our beauty-oriented application and help people make connections with those who share their interests.

We would also like to add additional information about individual products, such as its ingredients, along with images. Ingredients can play a very important role when it comes to picking products that you trust to put on your skin, since there are many bad ingredients out there. Furthermore, it would be great if we could provide guidance on our application as to which ingredients are known to be harmful, since this information isn't very easily accessible in the beauty industry. Both ingredient information and product images would require us to find new databases that have this information and that integrate with the product database that we're currently using.

**Describe the final division of labor and how well you managed teamwork.**

Overall, teamwork was very well-managed throughout the project because of our commitment as a team to put in a consistent level of work each week on the project, with regular check-ins and open communication regarding the project all throughout the semester. At the commencement of the project, our team met as a group to bring initial ideas to the table. We were able to decide on a scope and tech stack for the project relatively quickly, and chose a theme for our project that we felt we were all personally invested in and passionate about. We all had a mutual respect for each other's ideas and made sure to consider different angles brought up, so we all contributed to the final idea for the app. We set a standing time to meet and/or check-in each week, and made a group chat to maintain communication. These initial measures set a strong foundation for the rest of our work on the project.

We decided to divide labor initially by assigning Julie and Nitya to work on the back-end implementation of the project, and Ria and Oju to work on the front-end implementation. However we quickly realized that for stages 1-3, collaboration among all 4 of us was required to fully flesh out the technicalities of the project before splitting up for implementation. For stages 1-3, we evenly split up the work using the bulleted requirements for each stage as a guide for where to split the work. For example, we evenly split the task of writing up each component of the project proposal in Stage 1. In Stage 2, we also evenly split the UML diagram, logical schema writeup, assumptions, and normalization process. In stage 3, we each took charge of 1 advanced query. Although the physical work was split, our standing meetings ensured we were all on the same page about the entirety of the project stage we were working on at the time and we could bounce ideas off of each other and help each other. We also made sure to look over everyone's work at the end and ask questions about concepts we were unsure of.

In stage 4, after we had all worked together to flesh out the technical specifications of the application, we split into our predetermined front-end and back-end pairs. As a team, we listed out all the back-end requirements with their corresponding front-end requirements, with the ability for us to check off items as we finished them. This helped both pairs stay accountable for their tasks in the implementation, and also made it easy to track the progress of the other pair. We maintained regular communication at this point through our group chat and through comments and notes made on the central document. Conflicts and bugs were easily resolved in a timely manner, as we all made sure to regularly check and respond in our chat. Implementing and utilizing these communication channels and accountability tools ultimately led to the successful completion of our project.