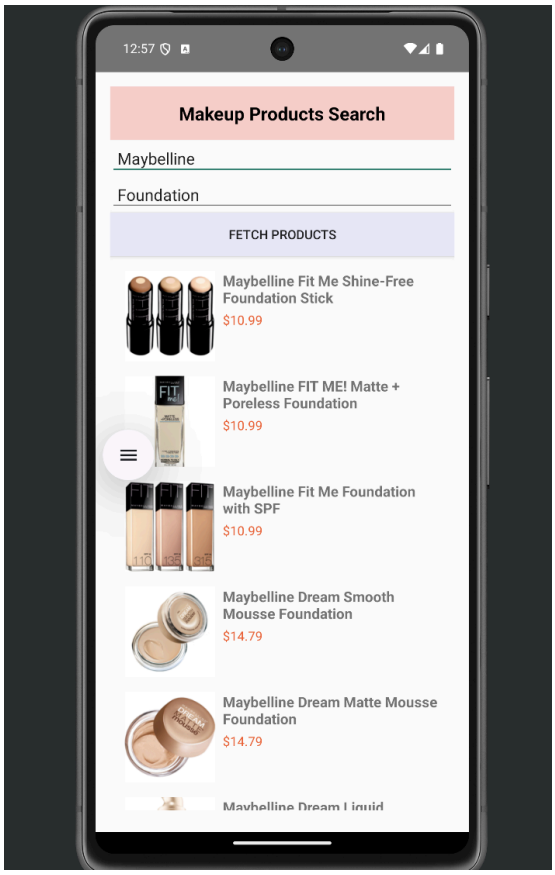
Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

Project 4 Task 2

Makeup Products Search app

Task 2: Distributed Application and Dashboard

App Overview :
This Makeup Products Search app allows users to search for makeup products by entering a brand name and/or product type. It fetches data from a makeup API and displays a list of relevant products, including their names, prices, and images, price from lowest to highest, providing users with an easy way to explore and compare cosmetics



Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

Requirement 1: Android Application

a. Has at least three different kinds of Views in your Layout

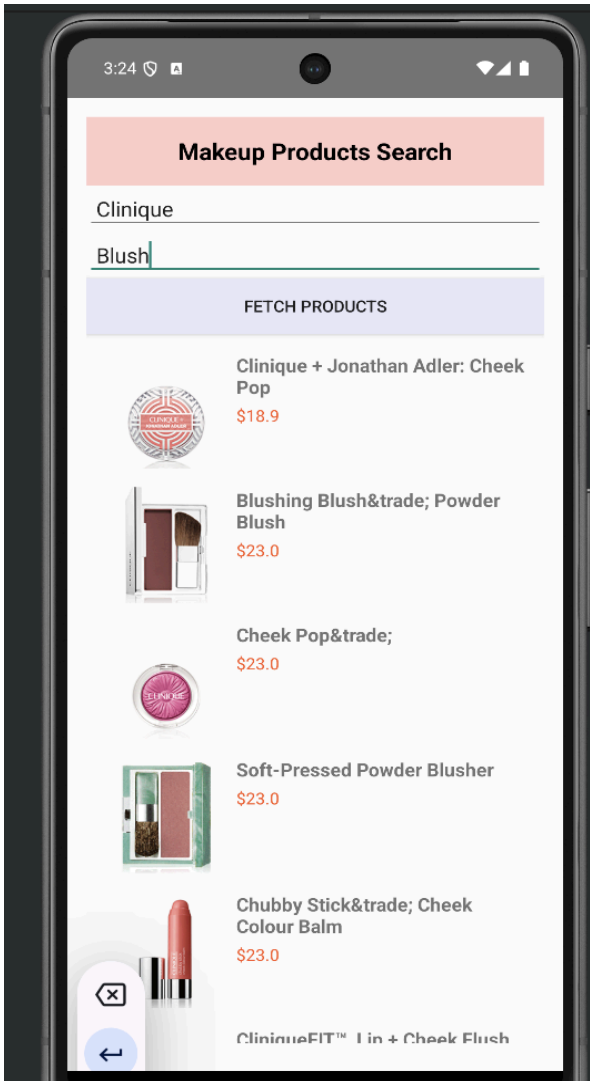
The application includes the following Views:
EditText: Allows users to input the brand name and product type.

Button: A "Fetch Products" button initiates the search query.

RecyclerView: Displays the results retrieved from the web service, including product names, images, and prices.

```
<!-- Product Type Input: EditText for user to enter the product type -->
<EditText
   android:id="@+id/etProductType"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Enter Product Type"
   android:padding="8dp" />
<!-- Fetch Button: Button to trigger the product search -->
<Button
   android:id="@+id/btnFetch"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Fetch Products"
   android:background="#E6E6FA"
   />
<!-- RecyclerView: Displays the list of fetched products -->
<androidx.recyclerview.widget.RecyclerView
   android:id="@+id/rvProducts"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:padding="8dp" />
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu



b. Requires input from the user

- Users must input a brand name and/or product type into the respective **EditText** fields.

c. Makes an HTTP request to the web service

- The app sends an HTTP GET request to the `/api/products` endpoint of the deployed web service.

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
/**
* Fetches makeup products based on brand and product type.
* Sends a GET request to /makeupSearch/api/products.
*/

public interface ApiService {
   @GET("/makeupSearch/api/products")
   Call<List<Product>> getProducts(
           @Query("brand") String brand,
           @Query("product_type") String productType
   );
}
```

d. Receives and parses an XML or JSON-formatted reply

- The web service returns a JSON-formatted response, which is parsed in the Android app using Retrofit and Gson libraries.

```
/**

* Function to fetch products from the web service based on user input.

* If valid inputs (brand or product type) are provided, it makes an HTTP
request to the server,

* processes the response, and updates the UI accordingly.

*/

private void fetchProducts() {

   String brand = etBrand.getText().toString().trim();

   String productType = etProductType.getText().toString().trim();



   // Showing error if both inputs are empty

   if (brand.isEmpty() && productType.isEmpty()) {

       Toast.makeText(this, "Please enter at least a brand or product type",
Toast.LENGTH_SHORT).show();
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
        return;

    }



    progressBar.setVisibility(View.VISIBLE);



    Retrofit retrofit = new Retrofit.Builder()


.baseUrl("https://bug-free-space-umbrella-wrr6r7ww765jcgw6r-8080.app.github.dev
")

            .addConverterFactory(GsonConverterFactory.create())

            .build();



    ApiService apiService = retrofit.create(ApiService.class);



    // Call the API with parameters

    Call<List<Product>> call = apiService.getProducts(brand, productType);



    call.enqueue(new Callback<List<Product>>() {

        @Override

        public void onResponse(Call<List<Product>> call, Response<List<Product>>
response) {

            progressBar.setVisibility(View.GONE);



            if (response.isSuccessful() && response.body() != null &&
!response.body().isEmpty()) {

                // Populate the RecyclerView with products
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
                List<Product> products = response.body();

                productAdapter = new ProductAdapter(products);

                rvProducts.setAdapter(productAdapter);

            } else {

                // Clear the RecyclerView and show a meaningful error message

                rvProducts.setAdapter(null); // Clear the list

                Toast.makeText(MainActivity.this, "No products found. Please try
a different search.", Toast.LENGTH_SHORT).show();

            }

        }
```

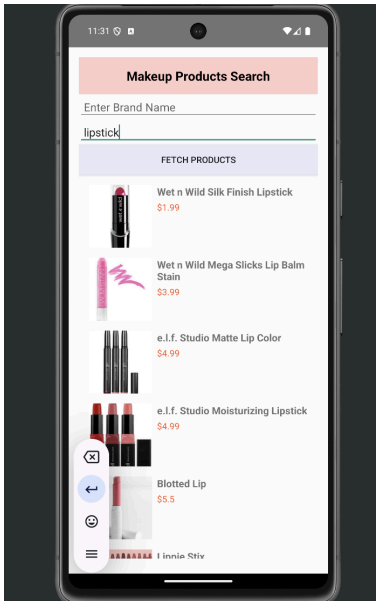
e. Displays new information to the user

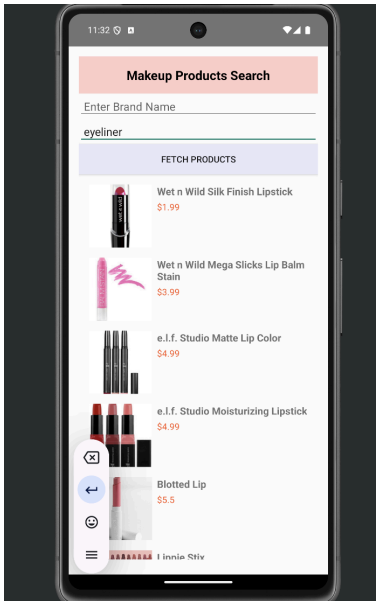
- The parsed response is displayed in a **RecyclerView**, including product names, prices, and images.

f. Is repeatable

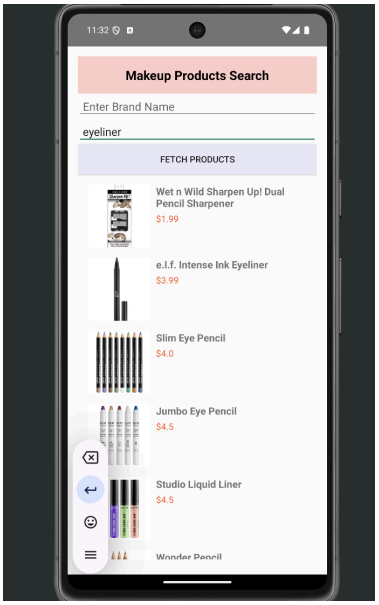
- The application allows users to perform multiple searches without restarting the app. Each new search updates the displayed results.

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu





Search for another product



After clicking fetch

Requirement 2: Web Service

a. Implement a simple API

- The web service provides a simple endpoint `/api/products` that accepts `brand` and `product_type` as query parameters.

```
@WebServlet("/api/products")
public class ProductSearchServlet extends HttpServlet {


   private MongoDBLogger logger = new MongoDBLogger();
   @Override
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
   // Handles GET requests to search for products based on brand and product
type

   protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

       // Extract query parameters

       String brand = request.getParameter("brand");

       String productType = request.getParameter("product_type");

       long startTime = System.currentTimeMillis();
```

b. Receives an HTTP request from the native Android application

- The web service receives HTTP GET requests sent by the Android application.

c. Executes business logic appropriate to the application

- The web service:
    1. Validates input parameters (`brand` and `product_type`).
    2. Fetches data from the Makeup API using the provided parameters.
    3. Filters and processes the response to include only the necessary fields: product name, price, image link, and product
    4. Sorts by price

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

   // Extract query parameters

   String brand = request.getParameter("brand");

   String productType = request.getParameter("product_type");

   long startTime = System.currentTimeMillis();
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
    // Construct the Makeup API URL
    String apiUrl = "http://makeup-api.herokuapp.com/api/v1/products.json";
    List<String> queryParams = new ArrayList<>();
    if (brand != null && !brand.isEmpty()) {
        queryParams.add("brand=" + brand);
    }
    if (productType != null && !productType.isEmpty()) {
        queryParams.add("product_type=" + productType);
    }
    if (!queryParams.isEmpty()) {
        apiUrl += "?" + String.join("&", queryParams);
    }
    try {
        long apiStart = System.currentTimeMillis();
        // Fetch data from the Makeup API
        String apiResponse = fetchMakeupApiData(apiUrl);
        long apiLatency = System.currentTimeMillis() - apiStart;

        // Parse and filter the data (simplify the response)
        List<Product> products = parseAndFilterResponse(apiResponse);

        // Logs API interaction details (parameters, latency, results count,
etc.) to MongoDB
        Document logEntry = new Document()
                .append("timestamp", System.currentTimeMillis())
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
                .append("phone_model", request.getHeader("User-Agent"))

                .append("parameters", new Document("brand",
brand).append("product_type", productType))

                .append("third_party_latency", apiLatency)

                .append("results_count", products.size())

                .append("status", "success");




        logger.log(logEntry);



        // Set response headers

        response.setContentType("application/json");

        response.setCharacterEncoding("UTF-8");



        // Convert filtered products to JSON and send response

        ObjectMapper mapper = new ObjectMapper();

        OutputStream out = response.getOutputStream();

        mapper.writeValue(out, products);

        out.flush();

    } catch (Exception e) {

        logger.log(new Document("error", e.getMessage()));

        response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "Failed
to fetch data");

    }
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
}

// Fetches raw data from the Makeup API

private String fetchMakeupApiData(String apiUrl) throws IOException {

   URL url = new URL(apiUrl);

   HttpURLConnection connection = (HttpURLConnection) url.openConnection();

   connection.setRequestMethod("GET");



   // Read response from the Makeup API

   BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));

   return reader.lines().collect(Collectors.joining());

}

// Parses and filters the Makeup API response

private List<Product> parseAndFilterResponse(String apiResponse) throws
IOException {

   // Parse JSON response using Jackson

   ObjectMapper mapper = new ObjectMapper();

   Product[] allProducts = mapper.readValue(apiResponse, Product[].class);

   // Simplify the data to return only relevant fields

   List<Product> filteredProducts = new ArrayList<>();

   for (Product product : allProducts) {

       // Only add products with valid numeric prices greater than $0

       if (product.getPrice() != null && !product.getPrice().isEmpty()) {

           try {

               double price = Double.parseDouble(product.getPrice());
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
                if (price > 0) {
                    filteredProducts.add(new Product(
                            product.getName(),
                            product.getPrice(),
                            product.getImageLink(),
                            product.getProductLink()
                    ));
                }
            } catch (NumberFormatException e) {
                System.err.println("Invalid price format for product: " +
product.getName());
            }
        }
    }


    // Sorting products by price (ascending order)
    filteredProducts.sort((p1, p2) -> {
        double price1 = Double.parseDouble(p1.getPrice());
        double price2 = Double.parseDouble(p2.getPrice());
        return Double.compare(price1, price2);
    });


    return filteredProducts;
}
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

---

d. Replies with an XML or JSON formatted response

The processed data is returned to the Android application in JSON format.

The web service uses Jackson's `ObjectMapper` to convert the list of `Product` objects into a JSON string, ensuring that the data is properly formatted for transmission. The serialized JSON is written to the `OutputStream` of the `HttpServletResponse`, allowing the Android application to receive and process the response seamlessly.

Requirement 3: Handle Error Conditions

While this is not required to be documented, the following error conditions are handled gracefully:

1. **Invalid mobile app input**
    - If both input fields are empty, the app displays an error message to the user.

```
// Showing error if both inputs are empty

if (brand.isEmpty() && productType.isEmpty()) {

   Toast.makeText(this, "Please enter at least a brand or product
type", Toast.LENGTH_SHORT).show();

   return;

}
```

2. **Invalid server-side input**
    - If the web service receives invalid input, it returns an appropriate HTTP status code and an error message.

```
catch (Exception e) {

   logger.log(new Document("error", e.getMessage()));

   response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Failed to fetch data");
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
}
```

3. **Mobile app network failure**
    - If the app cannot connect to the server, it displays a "Failed to fetch data" message.

```
    // Handle API call failure (network issues)
```

```
   public void onFailure(Call<List<Product>> call, Throwable t) {

       progressBar.setVisibility(View.GONE);

       Toast.makeText(MainActivity.this, "Failed to fetch data: " +
t.getMessage(), Toast.LENGTH_SHORT).show();

   }

});
```

4. **Third-party API unavailable**
    - If the Makeup API is unavailable, the web service responds with an appropriate error message.
5. **Third-party API invalid data**
    - If the Makeup API returns unexpected or invalid data, the web service logs the error and sends a user-friendly message back to the app.

```
 if (product.getPrice() != null && !product.getPrice().isEmpty()) {
       try {
           double price = Double.parseDouble(product.getPrice());
           if (price > 0) {
               filteredProducts.add(new Product(
                  //rest of the code   ));
           }
       } catch (NumberFormatException e) {
           System.err.println("Invalid price format for product: " +
product.getName());
       }
   }
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

Requirement 4: Log Useful Information

- The web service logs the following 6 (or more) pieces of information for each request/reply interaction with the mobile app:

  Mobile Request Information:

  - Brand: The brand name passed from the mobile app.
  - Product Type: The product type passed from the mobile app.

  Third-Party API Request Information:

  - Timestamp: When the API call to the third-party service was made.
  - Latency: Time taken to receive a response from the third-party API.

  Third-Party API Response Information:

  - Status: The status of the third-party API response (success or error).
  - Results Count: The number of products returned by the third-party API.

  Reply to Mobile Application:

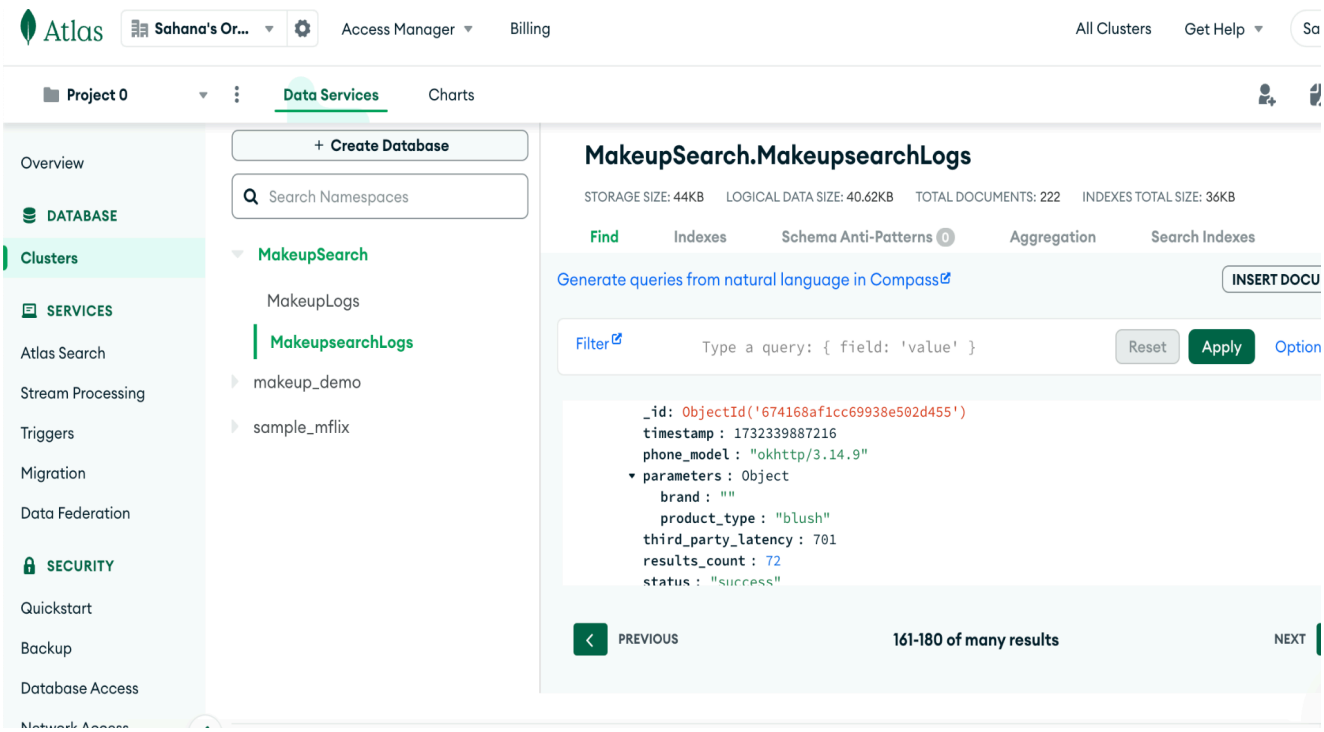
  - Response Status: Success or error while replying to the mobile app.
  - Phone Model: The User-Agent string from the mobile phone's request.

```
 // Logs API interaction details (parameters, latency, results count, etc.) to
MongoDB
Document logEntry = new Document()
       .append("timestamp", System.currentTimeMillis())
       .append("phone_model", request.getHeader("User-Agent"))
       .append("parameters", new Document("brand",
brand).append("product_type", productType))
       .append("third_party_latency", apiLatency)
       .append("results_count", products.size())
       .append("status", status);
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

Requirement 5: Store the Log Information in a Database

- **Details:**
  - The web service connects to a MongoDB database
  - Logs are stored in a structured format as documents in a MongoDB collection.
  - Each log entry includes all the pieces of information mentioned above.
  - The web service retrieves this logged data for analysis on the operations dashboard.




```
/**
* MongoDBLogger is responsible for handling interactions with a MongoDB database.
* It provides functionality for logging data and retrieving logs for analysis.
*/

public class MongoDBLogger {

   private static final String CONNECTION_STRING =
"mongodb://sahanabaggaon1999:VdF0a6MqwqLSUQzo@cluster0-shard-00-00.c4osx.mongod
b.net:27017,cluster0-shard-00-01.c4osx.mongodb.net:27017,cluster0-shard-00-02.c
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

```
4osx.mongodb.net:27017/MakeupSearch?w=majority&retryWrites=true&tls=true&authMe
chanism=SCRAM-SHA-1";
   private static final String DATABASE_NAME = "MakeupSearch";
   private static final String COLLECTION_NAME = "MakeupsearchLogs";

   private MongoCollection<Document> collection;


   /**
    * Constructor initializes the connection to the MongoDB database.
    * It sets up access to the specified collection.
    */
   public MongoDBLogger() {
       try {
           MongoClient client = MongoClients.create(CONNECTION_STRING);
           MongoDatabase database = client.getDatabase(DATABASE_NAME);
           this.collection = database.getCollection(COLLECTION_NAME);
       } catch (Exception e) {
           System.err.println("Failed to connect to MongoDB: " +
e.getMessage());
           e.printStackTrace();
       }
   }

   /**
    * Logs a document entry to the MongoDB collection.
    *
    * @param logEntry A BSON Document containing the data to log.
    */
   public void log(Document logEntry) {
       collection.insertOne(logEntry);
   }


   /**
    * Retrieves all logs from the MongoDB collection.
    *
    * @return A list of documents representing the logged data.
    */
   public List<Document> getLogs() {
       try {
           return collection.find().into(new ArrayList<>());
       } catch (Exception e) {
           System.err.println("Error fetching logs from MongoDB: " +
e.getMessage());
           e.printStackTrace();
           return new ArrayList<>();
```

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu

Requirement 6: Display Operations Analytics and Full Logs on a Web-Based Dashboard

- a. **Unique URL:**
  - The operations dashboard is accessible at a unique URL (e.g., `/dashboard`).

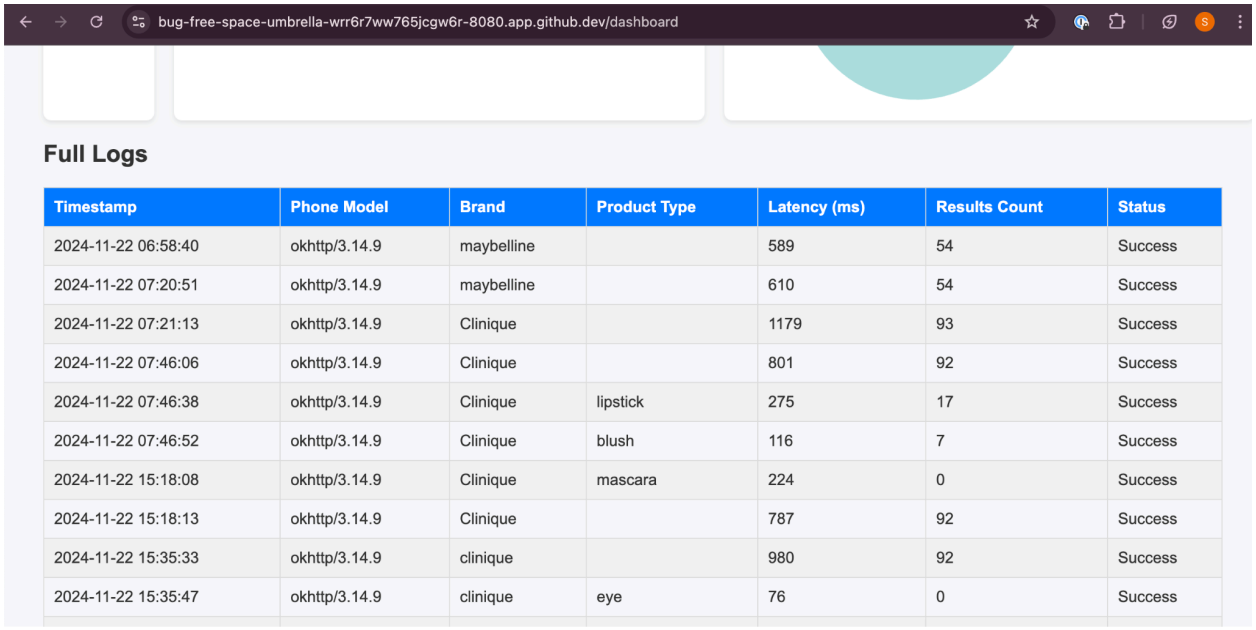
b. **Operations Analytics:**

  - At least 3 meaningful analytics are displayed on the dashboard:
    1. **Top Brands:** Counts of the most frequently searched brands.
    2. **Status Distribution:** A pie chart visualizing the distribution of API response statuses.
    3. **Average Latency:** The average time taken to fetch data from the third-party API.
  - Charts are dynamically rendered using Chart.js for visualization.

c. **Full Logs:**

  - A table displays all logged information in a readable format.
  - Each entry includes request details, third-party API interaction data, and the response sent to the mobile app.

Name : Sahana Baggaon Gajendra
AndrewID : sbaggaon
Email : sbaggaon@andrew.cmu.edu



Full Logs

Timestamp	Phone Model	Brand	Product Type	Latency (ms)	Results Count	Status
2024-11-22 06:58:40	okhttp/3.14.9	maybelline		589	54	Success
2024-11-22 07:20:51	okhttp/3.14.9	maybelline		610	54	Success
2024-11-22 07:21:13	okhttp/3.14.9	Clinique		1179	93	Success
2024-11-22 07:46:06	okhttp/3.14.9	Clinique		801	92	Success
2024-11-22 07:46:38	okhttp/3.14.9	Clinique	lipstick	275	17	Success
2024-11-22 07:46:52	okhttp/3.14.9	Clinique	blush	116	7	Success
2024-11-22 15:18:08	okhttp/3.14.9	Clinique	mascara	224	0	Success
2024-11-22 15:18:13	okhttp/3.14.9	Clinique		787	92	Success
2024-11-22 15:35:33	okhttp/3.14.9	clinique		980	92	Success
2024-11-22 15:35:47	okhttp/3.14.9	clinique	eye	76	0	Success

DEPLOYEMENT :