

Informatics Institute of Technology

Software development Group Project(SDGP)

5COSC021C

Group Report

Group Members:

<u>Full Name:</u>	<u>UOW number:</u>
Chanithu L. Prathapasinghe(Team Leader)	w1956122
Reshein A. Bastians	w1985736
Nammuni Omila Vihan De Silva	w1985543
Kavishkar Rajendrakumar	w1985533
W.D Rithil Joshua Withanage	w1999430

Declaration Page

I confirm that this project report and all the associated artifacts are my original work and have not been submitted for any degree program before or currently

Full Name of Student : Chanithu Lithmal Prathapasinghe

Uow Number: w1956122

Registration Number: 20221385

Full Name of Student :Nammuni Omila Vihan De Silva

Uow Number:w1985543

Registration Number:20222135

Full Name of Student : Reshein Ahaan Bastians

Uow Number: w1985736

Registration Number: 20222179

Full Name of Student : Kavishkar Rajendrakumar

Uow Number: w1985533

Registration Number: 20222068

Full Name of Student : Rithil Joshua Withanage

Uow Number: w1999430

Registration Number: 20220681

Date: 2024.03.23

Abstract

Consumers find it difficult to make informed choices when selecting cosmetic products due to complex ingredient lists and potential health risk. existing solutions for chemical analysis of the products are limited. Our project aims to develop a mobile app that collates ingredients with advanced chemical analysis algorithms, providing users with instant insights into the ingredients of the cosmetic product. The app seeks to empower consumers to make healthier and more informed choices when it comes to selecting cosmetic products.

Keywords

Cosmetic, chemical analysis mobile applications scanning health consciousness consumer empowerment personalized recommendations transparency ethical consumption beauty industry informed decision-making

Acknowledgement

Firstly, we want to thank God for giving us the strength and bravery to face all the challenges and complete this Project.

Secondly, We want to thank our parents and family members for believing in us and helping us achieve greatness in life by giving us words of wisdom and helping us become better versions of ourselves

We would also like to express our gratitude to our supervisor, Miss Vinodani for her constant support, guidance and oversight of this Project. We are thankful to her for steering us in the correct direction and offering encouragement during challenging moments

Finally, We want to convey our thanks to all our friends and peers who have constantly offered a helping hand when needed.

Table of Contents

Declaration Page	1
Abstract	2
Keywords	2
Table of Contents	4
List of Figures	6
List of Tables	7
Chapter 1: Implementation	1
1.1 Chapter Overview.....	1
1.2 Overview of the prototype.....	1
1.3 Technology selections.....	1
1.4 Implementation of the data science component.....	3
1.5 Implementation of the backend component.....	6
1.6 Implementation of the front end component.....	9
1.7 GIT Repository.....	11
1.8 Deployments/CI-CD Pipeline.....	12
1.9 CRUD operations.....	13
Chapter 2: Testing	43
2.1 Chapter Introduction.....	43
2.2 Testing Criteria.....	43
2.3 Testing functional requirements.....	43
2.4 Testing non-functional requirements.....	44
2.5 Unit testing.....	47
2.6 Performance testing.....	54
2.7 Usability testing.....	56
2.8 Compatibility testing.....	57
2.9 Chapter Summary.....	57
Chapter 3: Evaluation	58
3.1 Chapter Overview.....	58
3.2 Evaluation methods.....	58
3.3 Quantitative evaluation.....	58
3.4 Qualitative evaluation (Feedback from end users, domain experts and industry experts)..	58
3.5 Self evaluation.....	59
3.6 Chapter Summary.....	59
Chapter 4: Conclusion	60
4.1 Chapter Overview.....	60
4.2 Achievements of aims and objectives.....	60
4.3 Limitations of the research.....	62

4.4 Future enhancements.....	62
4.5 Extra work (Competitions, research papers, etc).....	64
4.6 Concluding remarks.....	64
References.....	66
Appendix.....	67

List of Figures

Figure 1.1 - Backend Date Receiver	4
Figure 1.2 - Dataset	4
Figure 1.4 - Python Library	5
Figure 1.5 -Machine Learning process.....	6
Figure 1.6 - Firebase Database.....	6
Figure 1.7- Snapshot of Github Action(CI-CD pipeline).....	12
Figure 1.9-Create Code	13
Figure 1.8-Read Code	20
Figure 1.9-Update Code.....	21
Figure 1.11-Dependencies Code	22
Figure 1.12 - Text Recognition Code.....	25
Figure 1.13 - Results Sheet Code.....	26
Figure 1.14 - Recommendation system screenshot.....	30
Figure 1.15 -Recommendation system screenshot-2.....	32
Figure 1.16 -Result page UI Changes.....	35
Figure 1.23- Import Packages.....	38
Figure 1.24 - Polyline linking to two locations.....	39
Figure 1.25 - Refresh locations.....	40
Figure 1.26 - Dependencies imported and installed.....	41
Figure 1.27 - API keys (Android and IOS).....	42
Figure 1.17- Sentiment Analysis	40
Figure 2.1 - Unit Testing.....	47

List of Tables

Table 1.1 -Functional requirements

Table 1.2 - Non-Functional Requirements

Chapter 1: Implementation

1.1 Chapter Overview

This chapter highlights the steps the team took in the implementation process, offering the rundown on the fully functional prototype. Moreover, illustrating the critical decisions, collaborative efforts, and technological choice that shaped the development process. By providing a detailed account of the development journey, this chapter aims to showcase the meticulous planning and execution that our team underwent for the success of this project.

1.2 Overview of the prototype

The completed prototype effortlessly brings together the user friendly aspects, a strong backend, and a machine learning model. Imagine using your phone's camera or typing in ingredients simple, right? That is all it takes for users to check the safety of the cosmetic products. The application then quickly gives a clear and easy to understand warning screen for each product. The team wanted to keep things straightforward, accurate, and easy to use— just what you'd expect when you are looking into the safety of your favorite cosmetics. With a focus on you, the user, the application promises a smooth and enjoyable for anyone curious about the safety of their cosmetic choices.

1.3 Technology selections

When developing the application, we relied on a variety of technologies to bring our vision to life. Here are the technologies the team used when creating the application;

Python (Programming language):

Think of Python as the language that brings everything together behind the scenes. It's super versatile and known for its simplicity and readability, which made coding a breeze for us. Plus, it has a massive library of tools and frameworks

Flask (server)

Flask supports connecting the backend and Python as different components. Helps to share data using HTTP services as the bridge for the backend and front end to communicate with both parts.

VS Code (IDE):

This was our go-to coding environment, packed with handy features that helped us write and debug code efficiently. From its intuitive interface to its extensive collection of extensions, VSCode made our coding experience smooth and enjoyable.

Dataworld:

Dataworld was our secret weapon when it came to handling all the data crunching. It's a powerful platform that allows us to store, analyze, and visualize data effortlessly. With its user-friendly interface and robust features, Dataworld made it easy for us to work with large datasets and extract valuable insights, ultimately enhancing the accuracy and effectiveness of our app.

Flutter (Frontend):

Flutter was instrumental in creating a smooth and intuitive user interface for our application. It's a cross-platform framework that allows us to develop beautiful and responsive UIs for both Android and iOS devices using a single codebase. With its hot reload feature, we could instantly see changes as we made them, speeding up the development process and ensuring a polished end product. Flutter has been very useful for us in all our frontend development pages especially for the google maps integration. Attained by use of the google_maps_flutter plugin, which may be obtained using the Dart package management (Pub).

Among the functions are:

- Displaying different map types (hybrid, satellite, and conventional)
- Marking locations and pathways with markers, polylines, and polygons
- Reacting to clicks, swipes, and pans made by the user
- The user's current location is determined by using geolocation services

Firebase:

Firebase served as our backend as a service platform, offering a suite of tools and services to support our application's backend needs. The team used Firebase authentication for user authentication, Firebase Cloud Firestore for real-time database storage, and Firebase CloudFunctions for serverless computing. With Firebase, we could quickly set up a scalable backend infrastructure without worrying about managing servers or infrastructure maintenance.

Dart (Programming language):

Dart was used to develop the front-end of the application with flutter. It's a versatile language optimized for building fast and scalable apps across multiple platforms. Dart's robust system and reactive programming capabilities allows us to design a responsive user interface that is also attractive which seamlessly works on both android and iOS.

1.4 Implementation of the data science component

Implementing a data science project played a major role in cosmoscan because this application totally functions with data and it solely depends on the accuracy of data. First of all, the app saves user information such as usernames, email and passwords in the Firebase cloud database. Users can sign up or log in with the use of that data.

The application scans ingredients in cosmetics using the scanner, in the process the scanned query is saved in another Firestore database with a unique identity for the user. After saving Flutter send the data query to Python using Flask HTTP services.

Now the main user of data science heads to the app. Python matches the scanned query using its database that has harmful chemicals. If matches are found it retrieves it back to the app. Then with that data, there is a simple machine learning process that works with data that are about

more information about chemicals such as health hazards, compounds and information. This predicts more info about these matched chemicals.

This is how data science is involved in cosmoscan, as it pivots on trustworthy and accurate data sets.

```

Received query: Titanium dioxide
boron, carbon, mineral oils, olive oil,
Matches found for chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide (airborne, unbound particles of respirable size)
Matches found for chemical name: carbon
Matched chemical name: Carbon black
Matched chemical name: Carbon black (airborne, unbound particles of respirable size)
Matched chemical names: ['Titanium dioxide', 'Titanium dioxide (airborne, unbound particles of respirable size)', 'Carbon black', 'Carbon black (airborne, unbound particles of respirable size)']
192.168.1.4 - - [23/Mar/2024 21:31:38] "POST /api HTTP/1.1" 200 -
Received query: Titanium dioxide
boron, carbon, mineral oils, olive oil,
Matches found for chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide (airborne, unbound particles of respirable size)
Matches found for chemical name: carbon

```

Figure 1.1 - Backend Data Receiver

This is an example of how data is received and matched in the backend Python using databases for this.

Chemical	Health Hazards	Compounds	Additional Information																	
Titanium Dioxide	Respiratory tract irritant	TiO ₂	Commonly used as a pigment and sunscreen ingredient. Linked to potential lung cancer risk in inhalation exposure.																	
Ethyl Acrylate	Irritation of eyes, skin	C ₅ H ₈ O ₂	Used in the production of various plastics and coatings.																	
Carbon Black Extract	Respiratory issues, carcinogenic	Carbon	Used as a pigment and strengthening agent in cosmetics.																	
Retinol	Skin irritation, photosensitivity	C ₂₀ H ₃₀ O ₂	A form of vitamin A, commonly used in skincare products.																	
Coffee	Potential staining, jitteriness	Various compounds	Contains caffeine and various antioxidants. Can have cosmetic applications like reducing puffiness.																	
Silica, Crystalline	Respiratory issues, lung cancer	SiO ₂	Used in cosmetics for its absorbent and anti-caking properties.																	
Cocamide DEA	Skin irritation, potential allergen	Various compounds	Used as a foaming agent and emulsifier in cosmetics.																	
Cocamide Diethanolamine	Skin irritation, potential allergen	Various compounds	Used similarly to Cocamide DEA.																	
Estragole	Potential carcinogen	C ₁₀ H ₁₂ O ₂	Found naturally in various herbs and spices, used as a fragrance in cosmetics.																	
Methyleugenol	Potential carcinogen	C ₁₁ H ₁₄ O ₂	Found naturally in certain herbs and spices, used in fragrances and flavorings.																	
Butylated Hydroxytoluene	Skin irritation, endocrine disruption	C ₁₁ H ₁₆ O ₂	Used as an antioxidant in cosmetics and food. Associated with various health concerns.																	
Benzophenone-3 (UV Filter)	Endocrine disruption, potential allergen	C ₁₄ H ₁₂ O ₃	Commonly used as a UV filter in sunscreens. Concerns over hormone disruption and environmental impact.																	
Acetaldehyde	Respiratory irritant, carcinogenic	C ₂ H ₄ O	Naturally found in ripe fruits and fermented beverages, also produced during combustion.																	
Triethanolamine	Skin irritation, potential allergen	C ₆ H ₁₅ NO ₃	Used as an emulsifier and pH adjuster in cosmetics.																	
Diethanolamine (DEA)	Skin irritation, potential allergen	C ₄ H ₁₁ NO ₂	Used as a foaming agent and pH adjuster in cosmetics.																	

Figure 1.2 - Dataset

An example database of chemicals and their additional information. below is the harmful chemical names database.

ChemicalName
Titanium dioxide
Ethyl acrylate
Carbon-black extracts
Retinol
Coffee
Silica, crystalline (airborne particles of respirable size)
Cocamide DEA
Cocamide diethanolamine
Estragole
Methyleugenol
Trade Secret
Butylated hydroxyanisole
Benzophenone-3
Acetaldehyde
Triethanolamine
Diethanolamine
Genistein (purified)
1,4-Dioxane
Retinol
Formaldehyde (gas)
Stvrene

Figure 1.3 - Dataset example

```

matched_chemical_names = set()
# Exact string matching
exact_matches = chemical_data[chemical_data['ChemicalName'].str.lower() == input_chemical_name]
matched_chemical_names.update(exact_matches['ChemicalName'].tolist())

# Fuzzy string matching with tokenized names
for name in chemical_data['ChemicalName']:
    if pd.notna(name): # Skip NaN values
        name_tokens = tokenize_chemical_name(name.lower())
        if all(token in name_tokens for token in input_tokens):
            matched_chemical_names.add(name)

# If no exact matches found, use fuzzy matching
if not matched_chemical_names:
    fuzzy_matches = process.extract(input_chemical_name, chemical_data['ChemicalName'], limit=5)
    matched_chemical_names.update([match[0] for match in fuzzy_matches if match[1] >= 95])

```

Figure 1.4 - Python Library

This Python snippet uses a fuzzy-wuzzy library to match data with the database. and the matching accuracy is ≥ 95 because there might be slight changes when scanning text from the scanner

```

# Read data from csv file with explicit data type specification
chemical_data = pd.read_csv("chemicals_in_cosmetics.csv", dtype={'ChemicalName': str})

# Load the dataset for predictions
cos_data = pd.read_csv('chem.csv')

# Separate features (X) and target variables (Y)
X = cos_data['Chemical']
Y = cos_data[['Health Hazards', 'Additional Information', 'Compounds']]

# Vectorize the chemical names using bag-of-words representation
vectorizer = CountVectorizer()
X_vectorized = vectorizer.fit_transform(X)

# Train multi-output classifier with SVM classifiers
classifier = MultiOutputClassifier(SVC())
classifier.fit(X_vectorized, Y)

```

Figure 1.5 -Machine Learning process

This snippet shows the simple machine learning process that divides data into x and y components and trains itself to match according to x (chemical).

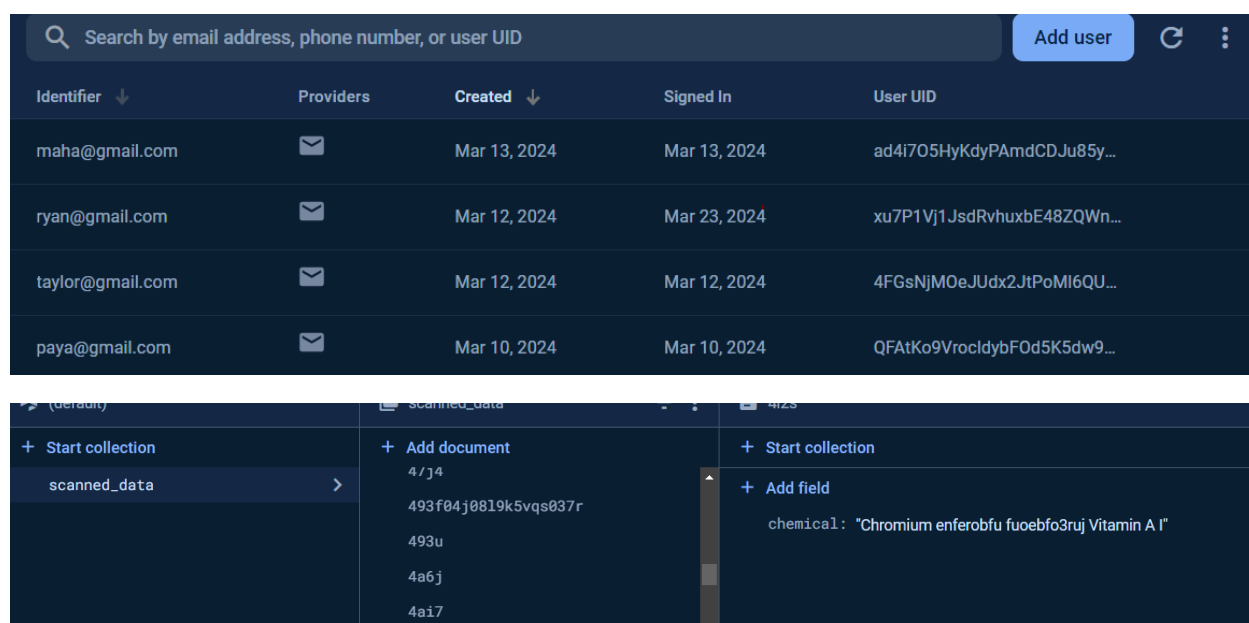


Figure 1.6 - Firebase Database

The above screenshots are related to Firebase databases that store data of users and scanned data of them. This scanned data is useful for future research and to make more accurate databases.

1.5 Implementation of the backend component

The backend in cosmoscan is coded with Python and connected with Flask to support HTTP service and connect between two sides. The team decided to use Python because the app handled more data structures in the project and the app uses a machine learning model for prediction. backend connects with a Flask server to share data between applications. As an HTTP request to the local server (laptop) as a POST request, the python sends back the processed output with the same server to a different endpoint.

```
final url = 'http://192.168.1.5:9000/api';
```

Data is sent from the scanner to Python using this post request to the server and data is received to the predicted endpoint in the same server.

```
final url = 'http://192.168.1.5:9000/predict';
```

Data is simply encoded when sending from flutter to python for quick access and its decoded in the python code.

```
void sendHttpRequest(String text) async {
  print('Text to be sent to server: $text');

  // Encode the text
  String encodedText = jsonEncode({'Query': text});
  print('Encoded text: $encodedText');

  final url = 'http://192.168.1.5:9000/api';
  final response = await http.post(
    Uri.parse(url),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode({'Query': text}),
  );
```

This is how data is encoded using “jsonEncode”

```
@app.route('/api', methods=['POST'])
def receive_data():
    if request.method == 'POST':
        data = request.json
        if data and 'Query' in data:
            query = data['Query']
            decoded_query = unquote(query)

            print('Received query:', decoded_query)
```

Decodes it back in the Python script then it functions the matching and prediction.

As mentioned above python has two main functions

1. matching chemicals from the database
2. Predicting the side effects and more information.

After removing and decoding the data the matching process is divided into 3 parts for accurate and quick matches. Those are exact matching, tocanize matching and fuzzy matching. First, it looks for the exact string matches if matches aren't found it checks for tokenised matching if tokenized data is present in the database, if not it goes to fuzzy matching using the fuzzuwuzzy library. Then search for the matches and retrieve the top matches greater than or equal to 95 in similarity score.

```
Debugger: 192.168.1.4 - - [23/Mar/2024 21:31:38] "POST /api HTTP/1.1" 200 -
Received query: Titanium dioxide
boron, carbon, minaral oils,olive oil,
Matches found for chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide (airborne, unbound particles of respirable size)
Matches found for chemical name: carbon
Matched chemical name: Carbon black
Matched chemical name: Carbon black (airborne, unbound particles of respirable size)
Matched chemical names: ['Titanium dioxide', 'Titanium dioxide (airborne, unbound particles of respirable size)', 'Carbon black', 'Carbon black (airborne, unbound particles of respirable size)']
192.168.1.4 - - [23/Mar/2024 21:31:38] "POST /api HTTP/1.1" 200 -
Received query: Titanium dioxide
boron, carbon, minaral oils,olive oil,
Matches found for chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide
Matched chemical name: Titanium dioxide (airborne, unbound particles of respirable size)
Matches found for chemical name: carbon
```

Output of data after matching.


```

192.168.1.4 - - [24/Mar/2024 15:26:47] "POST /api HTTP/1.1" 200 -
Predicting additional information for matched chemicals: ['Talc (powder)', 'Talc', 'Talc containing asbestiform fibers', 'Cosmetic talc']
Additional information predictions: {'Talc (powder)': {'Health Hazards': 'Respiratory issues, potential carcinogen', 'Additional Information': 'Used in cosmetics as an absorbent and anti-caking agent. Prolonged inhalation of talc particles may pose respiratory risks and is associated with a potential increased risk of certain cancers, particularly ovarian cancer.'}, 'Compounds': 'Various compounds'}, 'Talc': {'Health Hazards': 'Respiratory issues, potential carcinogen', 'Additional Information': 'Used in cosmetics as an absorbent and anti-caking agent. Prolonged inhalation of talc particles may pose respiratory risks and is associated with a potential increased risk of certain cancers, particularly ovarian cancer.'}, 'Compounds': 'Various compounds'}, 'Talc containing asbestiform fibers': {'Health Hazards': 'Respiratory issues, potential carcinogen', 'Additional Information': 'Talc contaminated with asbestiform fibers can pose respiratory risks and is associated with a potential increased risk of certain cancers, particularly mesothelioma.'}, 'Compounds': 'Various compounds'}, 'Cosmetic talc': {'Health Hazards': 'Respiratory issues, potential carcinogen', 'Additional Information': 'Used in cosmetics as an absorbent and anti-caking agent. Prolonged inhalation of talc particles may pose respiratory risks and is associated with a potential increased risk of certain cancers, particularly ovarian cancer.'}, 'Compounds': 'Various compounds'}}
192.168.1.4 - - [24/Mar/2024 15:26:52] "POST /predict HTTP/1.1" 200 -

```

How it predicts and sends back to the front end

1.6 Implementation of the front end component

These are the login pages that are connected to the Firebase and save information about users.

3:31 30%

←

SignUp

Username

Email

Password

SignUp

Already have an account ? [Login](#)

IIT-CS-91

11:24 50%

←

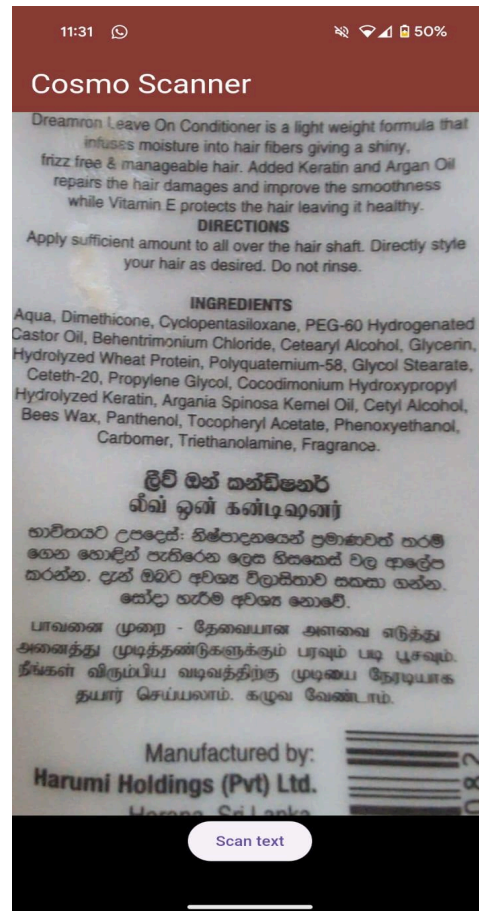
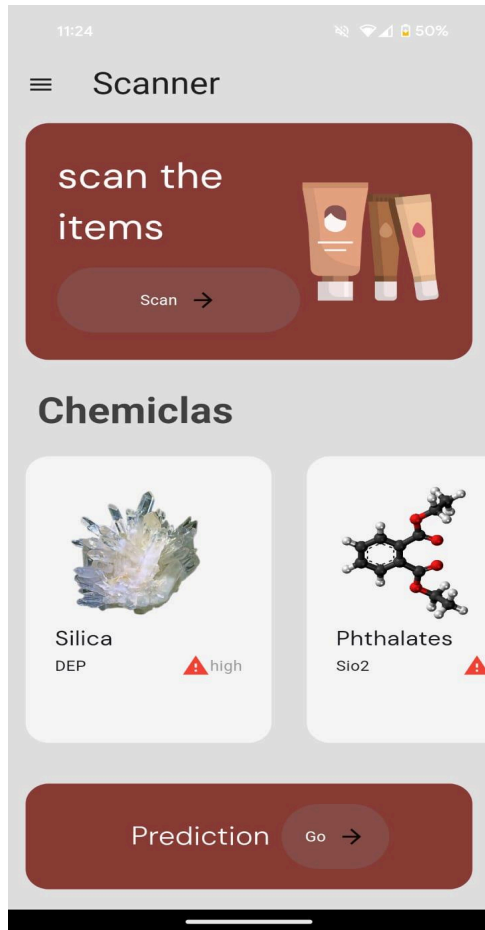
Login

Email

Password

Login

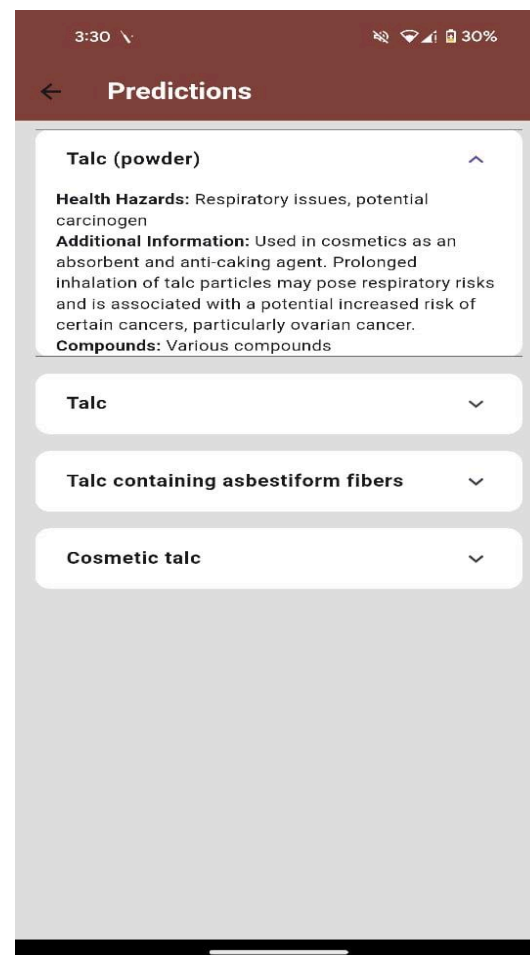
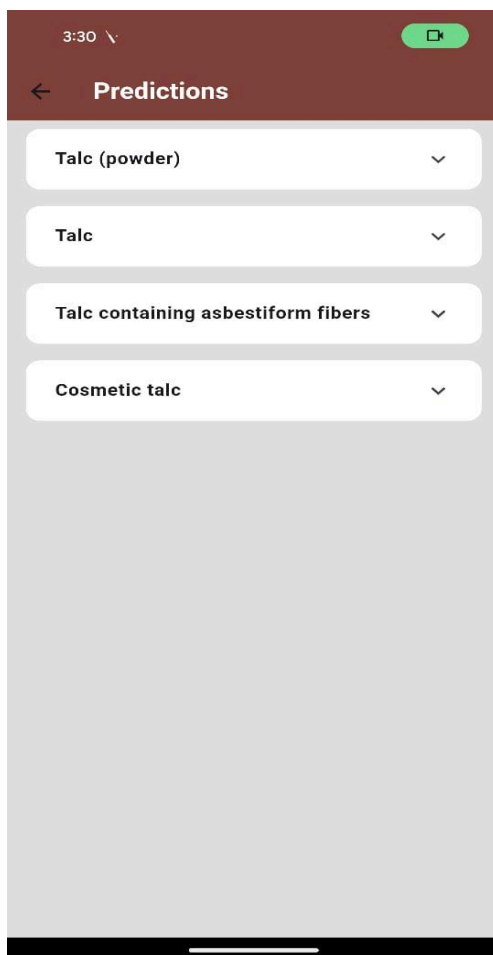
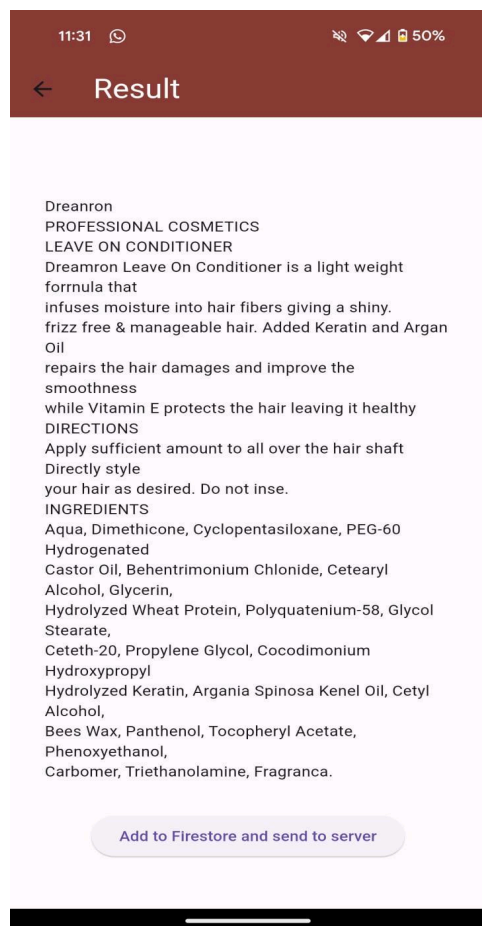
Don't have an account? [Sign Up](#)



The first screenshot shows the main page of the application then you can navigate to the scanner part to scan the cosmetic products. Another screenshot is the scanner itself and how it shows demo products

The scanner scans the data and gets to the rescue pages as the below screenshot then when the add to restore button is clicked it sends to the server.

After sending it takes a bit of time to get the result back to the application and shows in the prediction page as below with the predicted and matched chemicals.



Once press on the matched chemical users can see the additional information about the chemicals

1.7 GIT Repository

<https://github.com/omila112/SGDP.git>

1.8 Deployments/CI-CD Pipeline

The GitHub Actions CI/CD pipeline builds and tests code from the GitHub repository.

```
...      ...      @@ -0,0 +1,28 @@
1 + name: Flutter CI
2 +
3 + on:
4 +   push:
5 +     branches:
6 +       - main
7 +
8 + jobs:
9 +   build:
10 +     runs-on: ubuntu-latest
11 +
12 +     steps:
13 +       - name: Checkout repository
14 +         uses: actions/checkout@v2
15 +
16 +       - name: Setup Flutter
17 +         uses: subosito/flutter-action@v2
18 +         with:
19 +           flutter-version: '2.x'
20 +
21 +       - name: Get dependencies
22 +         run: flutter pub get
23 +
24 +       - name: Run tests
25 +         run: flutter test
26 +
27 +       - name: Build APK
28 +         run: flutter build apk
```

Figure 1.7- Snapstop of Github Action(CI-CD pipeline)

<https://authenticator-6fb97.web.app>

Deployed web application link

1.9 CRUD operations

Rithil Joshua - 20220681 CRUD

In create, read, update and delete operations as a member I have been involved in every aspect but not in delete. As a cosmetic scanning app, there is no need to log out or delete an account. As a result, delete isn't involved in this application.

Create

In create aspect there are several parts involved with frontend and backend. The main creation can be categorized as user creation in the app. Sign-in is the main user creation and with the support of signing users can log in whenever they want into the app. This part is done by using flutter as frontend and firebase as backend.

Figure 1.8-Create Code

```
import 'package:app/global/toast.dart';
import 'package:app/pages/ex.dart';
import 'package:app/pages/form_container_widget.dart';
import 'package:app/user_auth/firebase_auth_ser.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class Space extends StatefulWidget {
  const Space({Key? key}) : super(key: key);

  @override
  State<Space> createState() => _SpaceState();
```

```

}

class _SpageState extends State<Spage> {
  bool _isSignInUp = false;

  final FirebaseAuthService _auth = FirebaseAuthService();

  TextEditingController _usernameController = TextEditingController();
  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _usernameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[300],
      appBar: AppBar(
        backgroundColor: Colors.grey[300],
      ),
      body: Padding(
        padding: EdgeInsets.symmetric(horizontal: 15),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text(
              "SignUp",
              style: TextStyle(
                fontSize: 43,
                fontWeight: FontWeight.bold,

```

```

        color: Color.fromARGB(255, 22, 3, 2)),
    ),
    const SizedBox(
      height: 30,
    ),
    //username.....
    FormContainerWidget(
      controller: _usernameController,
      hintText: "Username",
      isPasswordField: false,
      helperText: '',
    ),
    const SizedBox(
      height: 10,
    ),
    //email.....
    FormContainerWidget(
      controller: _emailController,
      hintText: "Email",
      isPasswordField: false,
      helperText: '',
    ),
    const SizedBox(
      height: 10,
    ),
    //password.....
    FormContainerWidget(
      controller: _passwordController,
      hintText: "Password",
      isPasswordField: true,
      helperText: '',
    ),
    const SizedBox(
      height: 30,
    ),
    GestureDetector(

```

```

onTap: () {
  _signup();
  showToast(message: "user is succesfully created");
},
child: Container(
  width: double.infinity,
  height: 55,
  decoration: BoxDecoration(
    color: const Color.fromARGB(255, 138, 60, 55),
    borderRadius: BorderRadius.circular(10),
  ),
  child: Center(
    child: _isSigninUp
      ? const CircularProgressIndicator(
          color: Colors.white,
        )
      : const Text(
          "SignUp",
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.bold,
            fontSize: 20),
        ),
    ),
  ),
),
const SizedBox(
  height: 20,
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      "Already have an account ?",
      style: TextStyle(fontSize: 20, fontWeight:
FontWeight.w400),

```



```

        ),
        const SizedBox(
          width: 5,
        ),
        GestureDetector(
          onTap: () {
            Navigator.push(context,
              MaterialPageRoute(builder: (context) => const
page())));
          ),
        child: const Text(
          "Login",
          style: TextStyle(
            color: Color.fromARGB(255, 130, 36, 36),
            fontWeight: FontWeight.bold,
            fontSize: 20),
        ),
      ),
    ],
  ),
],
),
),
);
}

void _signup() async {
  setState(() {
    _isSignInUp = true;
  });

  // ignore: unused_local_variable
  String username = _usernameController.text;
  String email = _emailController.text;
  String password = _passwordController.text;

```

```

User? user = await _auth.signInWithEmailAndPassword(email, password);

setState(() {
  _isSignInUp = false;
});

if (user != null) {
  showToast(message: "User is successfully created");
  Navigator.pushNamed(context, "/menupage");
} else {
  showToast(message: "Some error happened");
}
}
}

```

Using firebase auth services i have created the authentication that saves data in the firebase.

This saves users' usernames and passwords in a Firestore database.

```

import 'package:app/global/toast.dart';
import 'package:firebase_auth/firebase_auth.dart';

class FirebaseAuthService {
  FirebaseAuth _auth = FirebaseAuth.instance;

  Future<User?> signUpWithEmailAndPassword(
    String email, String password) async {
    try {
      UserCredential credential = await
_auth.createUserWithEmailAndPassword(
        email: email, password: password);
      return credential.user;
    } on FirebaseAuthException catch (e) {
      if (e.code == 'email-already-in-user') {
        showToast(message: 'An error occurred:${e.code}');
      } else {
        showToast(message: '"The email address is already in use."');
      }
    }
  }
}





```

```

    }
}
return null;
}

Future<User?> signInWithEmailAndPassword(
    String email, String password) async {
    try {
        UserCredential credential = await _auth.signInWithEmailAndPassword(
            email: email, password: password);
        return credential.user;
    } on FirebaseAuthException catch (e) {
        if (e.code == 'user-not-found' || e.code == 'wrong-password') {
            showToast(message: 'An error occurred: ${e.code}');
        } else {
            showToast(message: 'Invalid email or password.');
```

This handles the errors in the signup and login with the use of firebase saved data.

chanithu@gmail.com		Mar 24, 2024	Mar 24, 2024	UDkJH2k3i0QFjswKe8vWpW...
mareesha@gmail.com		Mar 24, 2024	Mar 24, 2024	cLVmR5UgusY5R3VLiB9LBrTR...
chanithulithmal@gmail....		Mar 24, 2024	Mar 24, 2024	sZVfreQgMaaG40o0RAJYL1CL...
maha@gmail.com		Mar 13, 2024	Mar 13, 2024	ad4i7O5HyKdyPAmdCDJu85y...
ryan@gmail.com		Mar 12, 2024	Mar 24, 2024	xu7P1Vj1JsdRvhuxbE48ZQWn...

This is how data is saved in the database

Read

Read is another main aspect that works as the backbone of the code there are many read parts involved in the cosmoscan application but as a member, I mainly focused on reading the scanned data and getting it to python and reading it again. For this, I've used a Flask HTTP service to read data in the Python backend. After data is scanned the the data is sent as an encoded text to Python and decoded and read it was one of the most challenging parts of this project.

Figure 1.8-Read Code

```
// Method to send HTTP request to server
void sendHttpRequest(String text) async {
    print('Text to be sent to server: $text');

    // Encode the text
    String encodedText = jsonEncode({'Query': text});
    print('Encoded text: $encodedText');

    final url = 'http://192.168.1.5:9000/api';
    final response = await http.post(
        Uri.parse(url),
        headers: <String, String>[
            'Content-Type': 'application/json; charset=UTF-8',
        ],
        body: jsonEncode({'Query': text}),
    );
}
```

Scanned data is sent to the server using this snippet. The data is decoded and read by the below snippet

```

@app.route('/api', methods=['POST']) |
def receive_data():
    if request.method == 'POST':
        data = request.json
        if data and 'Query' in data:
            query = data['Query']
            decoded_query = unquote(query)

            print('Received query:', decoded_query)
            matched_results_info = {} # Dictionary to store matched chemicals and their predictions
            matched_chemicals_list = [] # List to store matched chemical names
            for input_chemical_name in re.split(r'[\n,]', decoded_query):
                input_chemical_name = input_chemical_name.strip()
                if input_chemical_name:
                    # Check if the result is cached
                    if input_chemical_name in query_cache:
                        matched_chemicals, returned_input_chemical_name = query_cache[input_chemical_name]
                    else:
                        matched_chemicals, returned_input_chemical_name = match_chemical_name(input_chemical_name)
                    if matched_chemicals:
                        matched_chemicals_list.extend(matched_chemicals)
                        print("Matches found for chemical name:", input_chemical_name)
                        for matched_chemical in matched_chemicals:
                            print("Matched chemical name:", matched_chemical)

```

Update

Data is updated in two aspects as the matched chemicals and the prediction. In both components data is updated by a specific database to the application. If matched chemicals are found data is updated from this code snippet by using a HTTP endpoint

Figure 1.9-Update Code

```

# Sending matched chemicals to Flutter
if matched_chemicals_list:
    matched_results_info['MatchedChemicalNames'] = matched_chemicals_list
    print("Matched chemical names:", matched_chemicals_list)
    return jsonify(matched_results_info), 200

# No matches found for the provided chemical names
else:
    return jsonify({"message": "No matches found for the provided chemical names."}), 404
else:
    return jsonify({"message": "Invalid data format."}), 400
else:
    return jsonify({"message": "Only POST requests are allowed."}), 405

```

It shows in the frontend and with another code the prediction is updated to the chemicals that matched.

```

@app.route('/predict', methods=['POST'])
def predict_additional_information():
    data = request.json
    if data and 'MatchedChemicalNames' in data and data['MatchedChemicalNames']:
        matched_chemicals = data['MatchedChemicalNames']
        print("Predicting additional information for matched chemicals:", matched_chemicals)
        additional_info = {}
        for chemical_name in matched_chemicals:
            predictions = predict(chemical_name)
            # Update the dictionary with formatted data for each chemical
            additional_info[chemical_name] = {
                'Health Hazards': predictions['Health Hazards'],
                'Additional Information': predictions['Additional Information'],
                'Compounds': predictions['Compounds']
            }
        print("Additional information predictions:", additional_info)
        return jsonify(additional_info), 200
    else:
        return jsonify({"message": "Invalid data format or no matched chemicals provided."}), 400

```

The below screenshots show how the data is updated appropriately in the fronted screens.

Omila De Silva- 20222135 CRUD

This Code below shows the Dependencies and the The main() function initializes the Flutter application by running the App widget.

Figure 1.11-Dependencies Code

```

import 'dart:io';

import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:google_mlkit_text_recognition/google_mlkit_text_recognition.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:text_recognition_flutter/result_screen.dart';

```

Figure 1.11-Main Code Scanner

```

void main() {
  runApp(const App());
}

class App extends StatelessWidget {
  const App({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Text Recognition Flutter',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: const MainScreen(),
    ); // MaterialApp
  }
}

class MainScreen extends StatefulWidget {
  const MainScreen({super.key});

  @override
  State<MainScreen> createState() => _MainScreenState();
}

```

Camera Handling:

To handle the camera, the code requests camera permissions asynchronously using the `_requestCameraPermission()` method. After that, it initializes the camera controller and displays the camera preview using the `CameraPreview` widget.

Text Recognition:

The code implements text recognition functionality through the `_scanImage()` method. When the user presses the "Scan text" button, the code captures a picture using the camera. After that, it processes the captured image using ML Kit's text recognition, extracting text from the image. Finally, it displays the recognized text in a separate `ResultScreen`.

Figure 1.12 - Text Recognition Code

```

class _MainScreenState extends State<MainScreen> with WidgetsBindingObserver {
  bool _isPermissionGranted = false;

  late final Future<void> _future;
  CameraController? _cameraController;

  final textRecognizer = TextRecognizer();

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);

    _future = _requestCameraPermission();
  }

  @override
  void dispose() {
    WidgetsBinding.instance.removeObserver(this);
    _stopCamera();
    textRecognizer.close();
    super.dispose();
  }
}

```

```

@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (_cameraController == null || !_cameraController!.value.isInitialized) {
    return;
  }

  if (state == AppLifecycleState.inactive) {
    _stopCamera();
  } else if (state == AppLifecycleState.resumed &&
    _cameraController != null &&
    !_cameraController!.value.isInitialized) {
    _startCamera();
  }
}

@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: _future,
    builder: (context, snapshot) {
      return Stack(
        children: [
          if (_isPermissionGranted)
            FutureBuilder<List<CameraDescription>>(
              future: availableCameras(),
              builder: (context, snapshot) {
                if (snapshot.hasData) {
                  _initCameraController(snapshot.data!);
                }
              },
            ),
          Center(child: CameraPreview(_cameraController!));
        ],
      );
    },
  );
}

```



```

Future<void> _scanImage() async {
  if (_cameraController == null) return;

  final navigator = Navigator.of(context);

  try {
    final pictureFile = await _cameraController!.takePicture();

    final file = File(pictureFile.path);

    final inputImage = InputImage.fromFile(file);
    final recognizedText = await textRecognizer.processImage(inputImage);

    await navigator.push(
      MaterialPageRoute(
        builder: (BuildContext context) =>
          ResultScreen(text: recognizedText.text),
      ), // MaterialPageRoute
    );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('An error occurred when scanning text'),
      ), // SnackBar
    );
  }
}

```

The following code creates a Flutter widget called `ResultScreen` that displays the recognized text. It has only one dependency, `flutter/material.dart`. The `ResultScreen` widget is a stateless widget that requires a parameter called `text` which contains the recognized text to be displayed. It consists of a `Scaffold` that has an `AppBar` and a `SingleChildScrollView`. Inside the `SingleChildScrollView`, there is a `Container` that holds the recognized text. The `AppBar` displays the title "Result" and has a blue background color. The recognized text is displayed inside a `Text` widget that is wrapped in a `Container`. Padding is applied to the container to create space. The `Text` widget is wrapped with a `SingleChildScrollView` to allow the text to be scrollable if it exceeds the available screen space

```
import 'package:flutter/material.dart';

class ResultScreen extends StatelessWidget {
  final String text;

  const ResultScreen({Key? key, required this.text}) : super(key: key);

  @override
  Widget build(BuildContext context) => Scaffold(
    appBar: AppBar(
      title: const Text('Result'),
      backgroundColor: Color(0xFF42A5F5),
    ), // AppBar
    body: SingleChildScrollView(
      child: Container(
        padding: const EdgeInsets.all(30.0),
        child: Text(text),
      ), // Container
    ), // SingleChildScrollView
  ); // Scaffold
}
```

Figure 1.13 - Results Sheet Code

Reshein Ahaan Bastians - 20222179 CRUD

I was in charge of creating the recommendation system for the application, which suggested alternate cosmetic product options. However, it was later removed by the team and I was not able to find a proper dataset for it, so we had to let go of that idea. Moreover, I did some UI changes for the result page, giving it a more user friendly look and also was given to do the sentiment analysis, which was not added to the application as there was a problem while adding it to the application.

Below are some screenshots on the recommendation system ,the UI Changes and the sentiment analysis:

Recommendation system:

```

1  import pandas as pd
2  from sklearn.feature_extraction.text import TfidfVectorizer
3  from sklearn.metrics.pairwise import linear_kernel
4
5  # Read data from CSV file
6  chemical_data = pd.read_csv("chemicals_in_cosmetics.csv")
7
8  # Combine relevant text features into one string for vectorization
9  chemical_data['combined_features'] = chemical_data['ProductName'].fillna('') + ' ' + chemical_data['ChemicalName'].fillna('')
10
11 # Vectorize the text features using TF-IDF
12 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
13 tfidf_matrix = tfidf_vectorizer.fit_transform(chemical_data['combined_features'])
14
15 # Function to get alternative products based on subcategory similarity
16 def get_alternative_products(matched_chemical_index, df):
17     # Get subcategory of the matched chemical
18     matched_subcategory = df.iloc[matched_chemical_index]['SubCategory']
19
20     # Find chemicals with similar subcategories
21     alternative_chemicals = [
22         dict(df.iloc[idx]) for idx in range(len(df)) if
23         idx != matched_chemical_index and df.iloc[idx]['SubCategory'] == matched_subcategory
24     ]
25
26     return alternative_chemicals
27
28 # Function to match input product name with database and get alternative products
29 def match_product_name_with_alternatives(input_product_name):
30     # Exact string matching
31     exact_matches = chemical_data[chemical_data['ProductName'].fillna('').str.lower() == input_product_name.lower()]
32

```

Figure 1.14- Recommendation system screenshot

```

# Get alternative products based on subcategory similarity
alternative_chemicals = get_alternative_products(matched_chemical_index, chemical_data)

return alternative_chemicals

Example usage
input_product_names = input("Enter the product names separated by commas: ").split(',')

for input_product_name in input_product_names:
    alternative_chemicals = match_product_name_with_alternatives(input_product_name.strip())

    if alternative_chemicals:
        print("\nAlternative Products for product name", input_product_name.strip() + ":")

        # Ensure at least 5 distinct alternative products
        distinct_alternatives = set()
        for alternative_chemical in alternative_chemicals:
            if len(distinct_alternatives) >= 5:
                break
            distinct_alternatives.add(alternative_chemical["ProductName"])
            print("ProductName:", alternative_chemical["ProductName"])
            print("CompanyId:", alternative_chemical["CompanyId"])
            print("CompanyName:", alternative_chemical["CompanyName"])
            print("BrandName:", alternative_chemical["BrandName"])
            print("PrimaryCategoryId:", alternative_chemical["PrimaryCategoryId"])
            print() # Add a newline between alternative products
    else:
        print("\nNo alternative products found for product name", input_product_name.strip() + ". This product may not be in our database."

```

Figure 1.15-Recommendation system screenshot-2

This code snippet represents a simplified implementation of a recommendation system for cosmetics products based on chemical similarity. Initially, the chemical data is loaded from a CSV file, and relevant text features such as product names and chemical names are combined and vectorized using TF-IDF. When a user inputs a product name, the system searches for exact matches in the database and proceeds to find alternative products based on the subcategory similarity of the matched chemical. The system then presents these alternative products, ensuring at least five distinct options, and displays relevant information such as product name, company details, brand name, and category. This approach aims to provide users with diverse choices that share similar chemical characteristics, thereby enhancing their experience and decision-making process when selecting cosmetics products.

Results page UI changes:

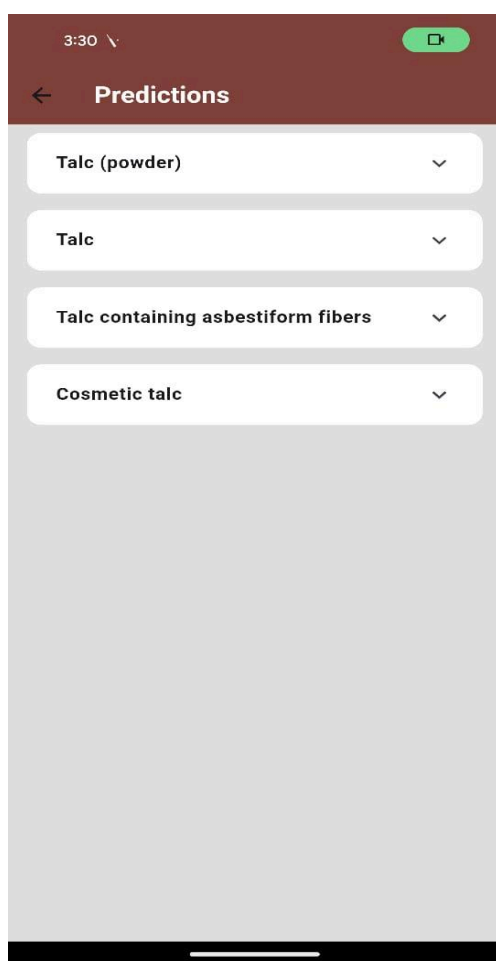


Figure 1.16 Result page UI Changes

Sentiment Analysis:

```

6 def get_sentiment(text):
7     analysis = TextBlob(text)
8     if analysis.sentiment.polarity < 0:
9         return 'Negative'
10    elif analysis.sentiment.polarity == 0:
11        return 'Neutral'
12    else:
13        return 'Positive'
14
15 def analyze_sentiment_for_chemical(chemical_name):
16     chemical_data = data[data['Chemical'] == chemical_name].copy() # Create a copy explicitly
17     if len(chemical_data) == 0:
18         print(f"No data found for chemical: {chemical_name}")
19         return
20     chemical_data.loc[:, 'sentiment'] = chemical_data['Health Hazards'].apply(get_sentiment)
21     sentiment_counts = chemical_data['sentiment'].value_counts()
22     print("Sentiment analysis for chemical", chemical_name + ":")
23     print(sentiment_counts.to_string(index=False))
24
25     # Calculate the risk level based on the count of sentiment analysis
26     risk_level = ''
27     if 'Negative' in sentiment_counts and 'Positive' in sentiment_counts:
28         if sentiment_counts['Negative'] > sentiment_counts['Positive']:
29             risk_level = 'Higher'
30         elif sentiment_counts['Negative'] < sentiment_counts['Positive']:
31             risk_level = 'Lower'
32         else:
33             risk_level = 'Equal'
34     else:
35         risk_level = 'N/A'
36
37     print(f"higher the number, greater the risk")
38
39 chemical_name = input("Enter the name of the chemical: ")
40 analyze_sentiment_for_chemical(chemical_name)
41

```

Figure 1.17- Sentiment Analysis

This Python script above performs sentiment analysis on health hazard data associated with chemicals stored in a CSV file. It uses the Pandas library for data manipulation and the TextBlob library for sentiment analysis. The script prompts the user to enter a chemical name, then analyzes sentiment for that chemical's health hazards. It calculates sentiment counts ('Negative', 'Neutral', 'Positive') and determines the risk level ('Higher', 'Lower', 'Equal') based on sentiment counts. Finally, it prints the sentiment analysis results and risk level.(Taboada 2016)

Chanithu Prathapasinghe- 20221385 CRUD

```

import 'package:example/sections/chat.dart';
import 'package:example/sections/chat_stream.dart';
import 'package:example/sections/embed_batch_contents.dart';
import 'package:example/sections/embed_content.dart';
import 'package:example/sections/response_widget_stream.dart';
import 'package:example/sections/stream.dart';
import 'package:example/sections/text_and_image.dart';
import 'package:example/sections/text_only.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gemini/flutter_gemini.dart';

```

Figure 1.18- packages

```

Run | Debug | Profile
void main() async {
  Gemini.init(
    apiKey: 'AIzaSyCHN7aRNfVa7-Ip5bEnNHx-Sgs-f2yd9Ww', enableDebugging: true);
  runApp(const MyApp());
}

```

Figure 1.19- Api key

The first picture above are the packages being imported to the file. Then in the main function the Gemini object is being initialized with the API key.

```

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Gemini',
      themeMode: ThemeMode.dark,
      debugShowCheckedModeBanner: false,
      darkTheme: ThemeData.dark().copyWith(
        colorScheme: ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 236, 8, 8)),
        cardTheme: CardTheme(color: Color.fromARGB(255, 250, 7, 7)),
        home: const MyHomePage(),
      ); // MaterialApp
    }
}

class SectionItem {

```

Figure 1.20- root of chatbot

The code represents the root of the chat application. It configures the material app and sets it with a theme.

```

1  import 'package:flutter/material.dart';
2
3  class ChatInputBox extends StatelessWidget {
4    final TextEditingController? controller;
5    final VoidCallback? onSend, onClickCamera;
6
7    const ChatInputBox({
8      super.key,
9      this.controller,
10     this.onSend,
11     this.onClickCamera,
12   });
13
14   @override
15   Widget build(BuildContext context) {
16     return Card(
17       margin: const EdgeInsets.all(8),
18       child: Row(
19         crossAxisAlignment: CrossAxisAlignment.end,
20         children: [
21           if (onClickCamera != null)
22             Padding(
23               padding: const EdgeInsets.all(4.0),
24               child: IconButton(
25                 onPressed: onClickCamera,
26                 color: Theme.of(context).colorScheme.onSecondary,
27                 icon: const Icon(Icons.file_copy_rounded)), // IconButton
28           ), // Padding
29           Expanded(
30             child: TextField(
31               controller: controller,
32               minLines: 1,
33               maxLines: 6,
34               cursorColor: Theme.of(context).colorScheme.inversePrimary,
35               textInputAction: TextInputAction.newline,
36               keyboardType: TextInputType.multiline,
37               decoration: const InputDecoration(

```



```

        contentPadding: EdgeInsets.symmetric(vertical: 10, horizontal: 4),
        hintText: 'Message',
        border: InputBorder.none,
      ), // InputDecoration
      onTapOutside: (event) =>
        FocusManager.instance.primaryFocus?.unfocus(),
    )), // TextField // Expanded
    Padding(
      padding: const EdgeInsets.all(4),
      child: FloatingActionButton.small(
        onPressed: onSend,
        child: const Icon(Icons.send_rounded),
      ), // FloatingActionButton.small
    ) // Padding
  ],
), // Row
); // Card
}
}

```

Figure 1.21- Interface of chatbot

The code above is creating an interface for the users to input and send messages in the chatbot. The TextField widget allows users to input messages and the send button is a FAB(floating action button) that triggers the onSend when pressed. If the user presses outside the keyboard then it automatically closes it.

```

import 'package:example/widgets/chat_input_box.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gemini/flutter_gemini.dart';
import 'package:flutter_markdown/flutter_markdown.dart';
import 'package:lottie/lottie.dart';

class SectionTextInput extends StatefulWidget {
  const SectionTextInput({super.key});

  @override
  State<SectionTextInput> createState() => _SectionTextInputState();
}

class _SectionTextInputState extends State<SectionTextInput> {
  final controller = TextEditingController();
  final gemini = Gemini.instance;

```

```

String? searchedText, result;
bool _loading = false;

bool get loading => _loading;

set loading(bool set) => setState(() => _loading = set);

@override
Widget build(BuildContext context) {
  return Column(
    children: [
      if (searchedText != null)
        MaterialButton(
          color: Color.fromARGB(255, 238, 11, 11),
          onPressed: () {
            setState(() {
              searchedText = null;
              result = null;
            });
          },
          child: Text('search: $searchedText')),
      Expanded(
        child: loading
          ? Lottie.asset('assets/lottie/ai.json')
          : result != null
            ? Padding(
                padding: const EdgeInsets.all(8.0),
                child: Markdown(data: result!),
              )
            : const Center(child: Text('Search something!')),
      ChatInputBox(
        controller: controller,
        onSend: () {
          if (controller.text.isNotEmpty) {
            searchedText = controller.text;
            controller.clear();
          }
        },
      ),
    ],
  );
}

```

```

        loading = true;

        gemini.text(searchedText!).then((value) {
            result = value?.content?.parts?.last.text;
            loading = false;
        });
    },
    ],
);
}
}

```

This provides a user with an interface for searching content and then displays the results.

The variables ‘searched text’, ‘result’, ‘loading’ are used to track the text being searched, the searched result and loading state.

When the send button is pressed in the ‘ChatInputBox’, the text input is retrieved from the controller, if it's not empty then the searched text is set and controller is cleared and then loading is changed to true. The Gemini library then is used to perform a text search using the input.

I have made the logo for the app and then implemented it in the “final app” code and helped with the implementation of the scanner.



```
"name": "app",
"short_name": "app",
"start_url": ".",
"display": "standalone",
"background_color": "#0175C2",
"theme_color": "#0175C2",
"description": "A new Flutter project.",
"orientation": "portrait-primary",
"prefer_related_applications": false,
"icons": [
  {
    "src": "icons/Cosmo.png",
    "sizes": "192x192",
    "type": "image/png"
  },
  {
    "src": "icons/Cosmo.png",
    "sizes": "512x512",
    "type": "image/png"
  },
  {
    "src": "icons/Cosmo.png",
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "maskable"
  },
  {
    "src": "icons/Cosmo.png",
    "sizes": "512x512",
    "type": "image/png",
    "purpose": "maskable"
  }
]
```

Figure 1.22- icon

Kavishkar Rajendrakumar- 20222068 CRUD

Creating a map application that helps users find local skin care centers and hospitals based on where they are right now is my task. Offering consumers an easy means of finding nearby healthcare services is the main purpose of the program.

The program will make use of geolocation services to precisely ascertain the user's present position in order to accomplish this. The locations of surrounding hospitals and skin clinics will subsequently be indicated on the map using markers or symbols created using this information.

The map interface will allow users to engage by enabling them to pan and zoom in and out of the map to explore different locations. The program displays further information about a hospital or skin clinic, including its name, address, contact information, and perhaps user ratings or reviews, when the user selects one of the markers or icons representing the institution.

Through simple identification and access to surrounding medical institutions, the map application seeks to enable users to make educated decisions regarding their healthcare requirements. Through the provision of this useful information, the application seeks to improve user ease and accessibility to critical healthcare services.

Code explanations

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:flutter_polyline_points/flutter_polyline_points.dart';
import 'package:google_maps_yt/consts.dart';
import 'package:location/location.dart';
```

*Figure 1.23- Import Packages***Import packages information**

- `dart:async`: Handles asynchronous operations.
- `flutter/material.dart`: Provides UI components for Material Design.
- `google_maps_flutter`: Integrates Google Maps into Flutter apps.
- `flutter_polyline_points`: Draws polylines on Google Maps.
- `google_maps_yt/consts.dart`: Imports custom constants for Google Maps.
- `location`: Accesses device location information.

Figure 1.24-Polyline linking to two locations

```

class MapPage extends StatefulWidget {
  const MapPage({super.key});

  @override
  State<MapPage> createState() => _MapPageState();
}

class _MapPageState extends State<MapPage> {
  Location locationController = new Location();

  final Completer<GoogleMapController> _mapController =
    Completer<GoogleMapController>();

  static const LatLng _pGooglePlex = LatLng(6.9271, 79.8612);
  static const LatLng _pApplePark = LatLng(6.9271, 79.8612);
  LatLng? currentP = null;

  Map<PolylineId, Polyline> polylines = {};

  @override
  void initState() {
    super.initState();
    getLocationUpdates().then(
      (_) => {
        getPolylinePoints().then((coordinates) => {
          generatePolyLineFromPoints(coordinates),
        }),
      ),
    );
  }
}

```

This code initializes the map with polylines linking two predetermined sites (GooglePlex and Apple Park) and puts up a map page with location-related features.

Figure 1.25-Refresh location

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: _currentP == null
      ? const Center(
        child: Text("Loading..."),
      ) // Center
      : GoogleMap(
        onMapCreated: ((GoogleMapController controller) =>
          _mapController.complete(controller)),
        initialCameraPosition: CameraPosition(
          target: pGooglePlex,
          zoom: 13,
        ), // CameraPosition
        markers: {
          Marker(
            markerId: MarkerId("_currentLocation"),
            icon: BitmapDescriptor.defaultMarker,
            position: _currentP!,
          ), // Marker
          Marker(
            markerId: MarkerId("_sourceLocation"),
            icon: BitmapDescriptor.defaultMarker,
            position: pGooglePlex), // Marker
          Marker(
            markerId: MarkerId("_destinationLocation"),
            icon: BitmapDescriptor.defaultMarker,
            position: pApplePark) // Marker
        },
        polylines: Set<Polyline>.of(polylines.values),
      ), // GoogleMap
  ); // Scaffold
}

```

This sample of code defines the MapPage widget's construct() function. A GoogleMap widget is contained in a Scaffold widget that is returned. A "Loading..." notice appears in the middle if _currentP is null; otherwise, the Google Map is displayed, with markers for Apple Park,

GooglePlex, which is in this case, Dehiwala to Kollupitiya in Colombo and the current position. It also uses the coordinates saved in the polylines map to create polylines on the map.

Figure 1.26- Dependencies imported and installed

```
name: google_maps_yt
description: A new Flutter project.

publish_to: 'none' # Remove this line if you wish to publish to pub.dev

version: 1.0.0+1

environment:
  sdk: '>=3.1.1 <4.0.0'

dependencies:
  flutter:
    sdk: flutter

  google_maps_flutter: ^2.2.8
  location: ^5.0.0
  flutter_polyline_points: ^2.0.0

  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^2.0.0

flutter:

  uses-material-design: true
```

These are the dependencies and the sdk versions the system needed to get google maps up and running. Google map API key is imported as well

Figure 1.27- API keys (Android & IOS)

```
<meta-data android:name="com.google.android.geo.API_KEY"  
    android:value="AIzaSyCLIYmO2ULfnq8Dqj3m8eUZTMztOwjheF4"/>
```

Android API key

```
GMSServices.provideAPIKey("AIzaSyCLIYmO2ULfnq8Dqj3m8eUZTMztOwjheF4")  
GeneratedPluginRegistrant.register(with: self)
```

IOS API key

Chapter 2: Testing

In the previous chapter, we discussed the prototype implementation required for our research. We have completed the implementation and now it's time to evaluate its effectiveness by conducting tests. Effective testing is crucial not only to determine the project's success or failure but also to identify any shortcomings and improve them. This chapter will cover the testing methods and strategies used along with the test results obtained. We invite you to join us on this journey to discover the effectiveness of our prototype. Let's explore the results together.

2.1 Chapter Introduction

Testing is an important aspect of software development. It is critical to guarantee the app's quality, dependability and performance. Testing helps detect and correct defects, inconsistencies, And performance bottlenecks, thereby improving the overall user experience. In this chapter, we will go deeply into the world of testing discussing its relevance and numerous approaches. We want to have a thorough grasp of testing methodologies, including unit testing, integration testing and end-to-end testing. Each of these testing methodologies provides distinct advantages and insights, which contribute to the overall strength of our testing system. Effective testing fosters confidence among developers and users leading to higher user satisfaction and engagement.

2.2 Testing Criteria

As the testing criteria members chose to test the authentication, HTTP requests both posts and get methods, data matching and prediction. The scanner can be involved as a separate testing part in the cosmoscan application. These criteria can be tested to see the success of the app implementation.

2.3 Testing functional requirements

Number	Requirements	Description	Priority	Status
--------	--------------	-------------	----------	--------

FR 1	User Credentials	Logging in with the user's credentials	High	passed
FR 2	Photo of Cosmetic image	Taking a picture of the list	High	passed
FR 3	Analysis of chemicals	The data in the list is compared with the dataset and if there is a match, it will display	High	passed
FR4	Prediction of the data	The matched data predicted for harmfulness and more information	High	passed
FR 5	Clinics	Display nearby skin clinics for the user to see	High	passed
FR 6	Chatbot	The user can chat with the bot about any questions they have regarding chemicals and health issues	Medium	passed

Table 1.1 -functional requirements

2.4 Testing non-functional requirements

Number	Requirements	Description	Priority	Status
--------	--------------	-------------	----------	--------

NFR 1	Performance	The overall performance of the cosmoscan application	High	Performance is stable as a flutter application, there can be little lags but overall performance is in a high position
NFR 2	Usability	The usability of the application, usage and how it helps as a software solution.	High	The app helps to identify healthy products for cosmetic usage either skin, hair, nails body anything an app can identify what is best and what is not
NFR 3	Security	Overall security of the application	Medium	This application doesn't dig deep into user information but it takes user emails for the authentication. Basic firebase protection is applied to that data
NFR 4	Accuracy	The accuracy totally depends on the databases that are used in the app	High	Accuracy is reliable for the app but more data is better for an app like this because more data means the matching is more

				accurate and it is diverse into the harmful chemicals.
NFR 5	compatibility	The compatibility with various devices.	medium	The application has been designed to be an Android application but this app is compatible with the web there should be a scanner to get full use of the app.

Table 1.2 - Non-Functional Requirements

2.5 Unit testing

```

test_matchall.py > TestMatchAllAPI > setUp
1  import unittest
2  import json
3  from matchall import app
4
5  class TestMatchAllAPI(unittest.TestCase):
6
7      def setUp(self):
8          self.app = app.test_client()
9          self.app.testing = True
10
11     def test_receive_data(self):
12         query = {'Query': 'Chemical1, Chemical2, Chemical3'}
13         response = self.app.post('/api', json=query)
14         data = json.loads(response.data)
15         self.assertEqual(response.status_code, 200)
16         self.assertIn('MatchedChemicalNames', data)
17         self.assertIsInstance(data['MatchedChemicalNames'], list)
18
19     def test_predict_additional_information(self):
20         data = {'MatchedChemicalNames': ['Chemical1', 'Chemical2']}
21         response = self.app.post('/predict', json=data)
22         self.assertEqual(response.status_code, 200)
23         predictions = json.loads(response.data)
24         self.assertIsInstance(predictions, dict)
25         self.assertEqual(len(predictions), 2)
26         self.assertIn('Chemical1', predictions)
27         self.assertIn('Chemical2', predictions)
28         chemical1_predictions = predictions['Chemical1']
29         self.assertIn('Health Hazards', chemical1_predictions)
30         self.assertIn('Additional Information', chemical1_predictions)
31         self.assertIn('Compounds', chemical1_predictions)
32
33     if __name__ == '__main__':
34         unittest.main()
35

```

Figure 2.1 - Unit Testing

```

test.py > TestChemicalPrediction > test_predict_additional_information
1 import unittest
2 import requests
3
4 class TestChemicalPrediction(unittest.TestCase):
5     def setUp(self):
6
7         self.url = 'http://127.0.0.1:9000/predict'
8
9     def test_predict_additional_information(self):
10
11         data = {
12             'MatchedChemicalNames': ['Chemical A', 'Chemical B']
13         }
14
15         # Send POST request to Flask app
16         response = requests.post(self.url, json=data)
17
18         # Check if response status code is 200 (OK)
19         self.assertEqual(response.status_code, 200)
20
21
22         predictions = response.json()
23         self.assertIn('Chemical A', predictions)
24         self.assertIn('Chemical B', predictions)
25         self.assertIn('Health Hazards', predictions['Chemical A'])
26         self.assertIn('Additional Information', predictions['Chemical A'])
27         self.assertIn('Compounds', predictions['Chemical A'])
28         self.assertIn('Health Hazards', predictions['Chemical B'])
29         self.assertIn('Additional Information', predictions['Chemical B'])
30         self.assertIn('Compounds', predictions['Chemical B'])
31
32 if __name__ == '__main__':
33     unittest.main()
34

```

Did manual unit testing for the machine learning and data matching part in the code. Tests were about getting and sending data from HTTP services. And the test came out as the functions are properly working.


```
PS C:\Users\Win 10\Documents\GitHub\SGDP\fl> & "C:/Program Files/Python311/python.exe" "c:/Users/win 10/Documents/GitHub/SGDP/fl/test_matchall.py"
Predicting additional information for matched chemicals: ['Chemical1', 'Chemical2']
Additional information predictions: {'Chemical1': {'Health Hazards': 'Skin irritation, potential allergen', 'Additional Information': 'contains caffeine and various antioxidants. Can have cosmetic applications like reducing puffiness.', 'Compounds': 'Various compounds'}, 'Chemical2': {'Health Hazards': 'Skin irritation, potential allergen', 'Additional Information': 'contains caffeine and various antioxidants. Can have cosmetic applications like reducing puffiness.', 'Compounds': 'Various compounds'}}
Received query: Chemical1, Chemical2, Chemical3
[]
```

```
Ran 1 test in 0.018s
```

```
OK
```

```
PS C:\Users\Win 10\Documents\GitHub\SGDP\fl> []
```

Successful unit test results with mock data, below is the unit test for the front-end flutter application.

Description	Expected Output	Actual Output	Status
User Registration and Login navigation Page			
The user able to navigate to the sign-up page using the button that is present in user registration and login navigation page	The user clicks on the signup button and navigates to the sign-up page of the application.	The user clicks on the sign-up button and navigates to the Sign-up page successfully.	PASS

Users are able to navigate to the login page from sign-up if they already have an account.	The user clicks on the login text and navigates to the login page	The user clicks on the login text and navigates to the login page successfully	PASS
Sign up page			
User inserts the sign up details in the text boxes username, email and password	The user enters the relevant information required for the sign up of the application and pressing sign up and navigating to the next page of the application	The user enters the relevant information and successfully signs up with the application.	PASS
The user enters the wrong format of data to the signup page	There will be an error message about the wrong email or password and to redo it.	The user will display and error message successfully	PASS

The user enters a weak password that consists of less than 6 characters along with other information and presses the sign-up button	There will be an error message showing the user entered password must consist of more than 6 characters and application will not proceed to the next page	The user will be displayed with an error message regarding the Weak password with less characters that has entered.	PASS
Login Page			
The user enters information such as email, and password and presses on the login button.	login will be successful and will proceed to the menu page of the application and will grant full access to the application.	Successfully logs into the account of the user.	PASS
The user enters an invalid email along with a password and presses the login button.	There will be an error message showing the user has entered the wrong format of the email and will not proceed to the next page.	The user will be displayed with an error message regarding the invalid email format that has been entered.	PASS
Scanning page			

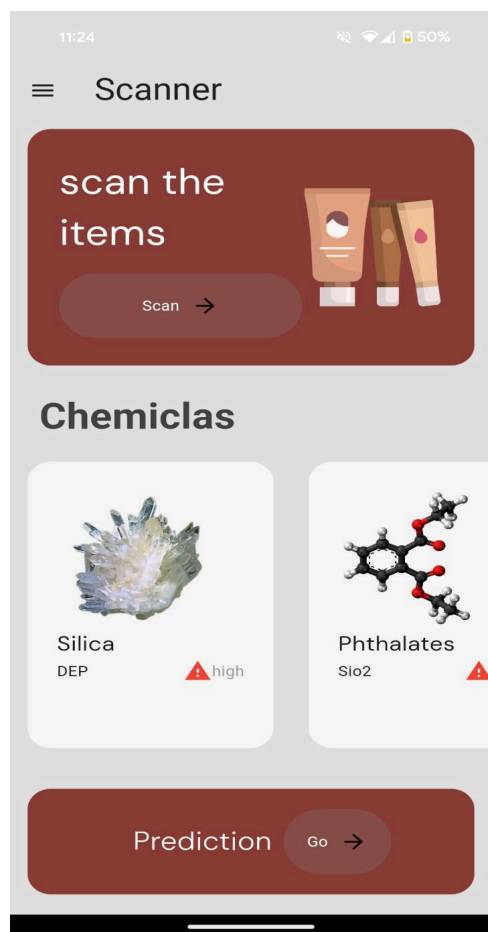
The user needs to navigate the app to the scanner by pressing the scan button	There will be a scan button and the user will be able to navigate to the scanner.	The user successfully navigates to the scanner	PASS
The user can scan whatever text they need to scan by pressing the scan text button	The user will be able to scan any text after pressing the button	User successfully scans the text	PASS
If the scanner had errors about the clarity of the image it shows an error message to scan again	The user will be updated by an error to scan the text again	The user will successfully get the error message	PASS
Result page			
After scanning, the user will see the result of the text that they scanned.	The user will then scan the scanned text result in the result page.	User will successfully see the scanned text	PASS
Harmful chemicals page			

After seeing the scanned text there is a button to harmful chemicals out of the scanned text	The user will see a button to get the chemicals out of the ingredients	The user will successfully see the button for harmful chemical	PASS
The user can navigate to the harmful chemicals screen that has the data about harmful chemicals	The user will be able to see the harmful chemicals out of the chemicals that they scanned	User will successfully see the result of harmful chemicals	PASS
If matches aren't found the app shows an error message that there are no matched chemicals	The user sees an error pop up that there are no matched chemicals.	The user will successfully get the error message.	PASS
More info about chemicals page/prediction			
There will be a button to get advanced information about the matched chemicals.	The button to get the advance info about the matched chemicals.	The user will get the button successfully	PASS
Prediction page			

The matched chemicals will be available with a drop-down menu to get more information	There will be a screen to see more information about chemicals with a drop-down menu for each matched chemical	User successfully gets to the prediction page and get the prediction	PASS
Map			
The system first asks for the users' current location	The user can choose in between whether to allow or not	If the user allows to share the location, System fetches the current location and shows through a marker	PASS

2.6 Performance testing

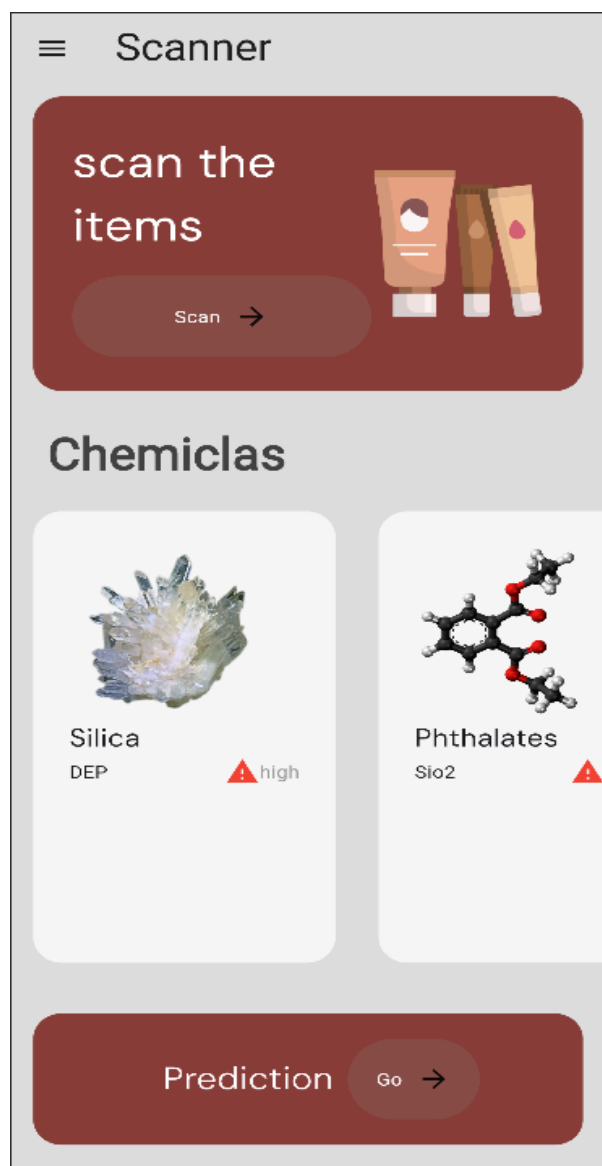
To ensure that the mobile application works properly on various Android smartphones, it was tested on a generic emulator. We tested the application launch speed and camera usage on several smartphones including Huawei, Samsung, and Xiaomi to ensure optimal performance.



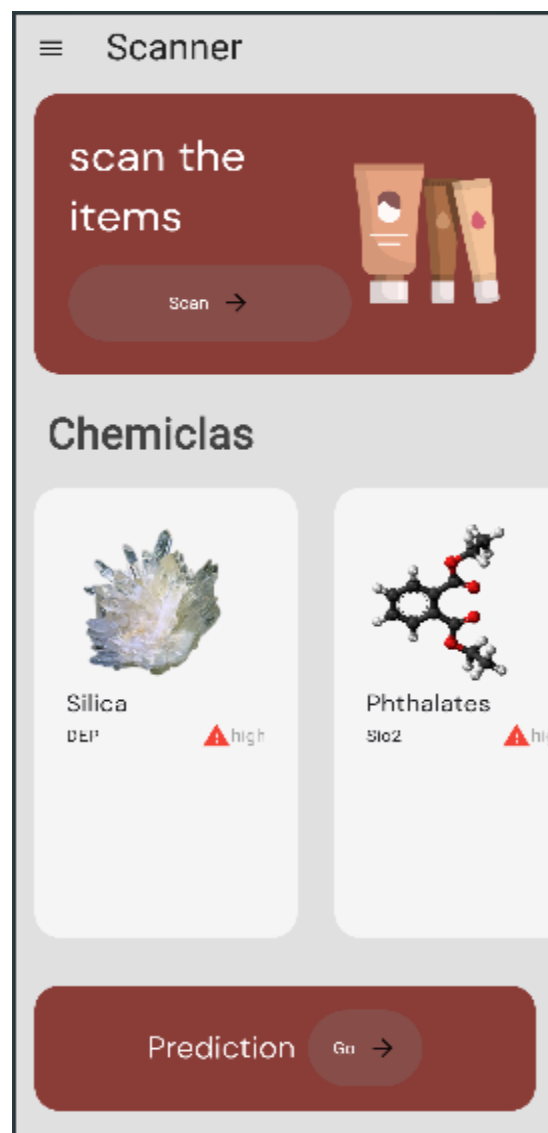
Pixel 8



pixel 7



Samsung s24 ultra



iPhone 14 pro max

2.7 Usability testing

Usability testing is a process that evaluates how easy an application is to use, as well as how well the user interface and experience are managed. The mobile application was designed using user experience standards, to make it user-friendly and accessible across all devices.

Cosmoscan is a very user-friendly application that runs as a step-by-step process. Once the user is signed users just have to follow the steps to get the final result of the products they scan.

2.8 Compatibility testing

With the release of Flutter 2, the mobile application can now be easily converted into a web or desktop application, using the same codebase. This automation ensures that all applications have the same features and there is no need to code for compatibility between the applications. This is made possible by the new Flutter engine.

As a result, the team hosted the application using a Firebase hosting service.

<https://authenticator-6fb97.web.app>

But it was built to be a mobile application so there might be some differences in the UI

2.9 Chapter Summary

The main objective of this chapter was to assess both the functional and non-functional aspects of the prototype. During this stage, numerous minor issues arose that required immediate attention. Furthermore, this chapter includes information about additional testing that was conducted, such as compatibility testing. The evaluation of the final system will be discussed in the subsequent chapter.

Chapter 3: Evaluation

3.1 Chapter Overview

The Evaluation chapter presents a thorough analysis of the app developed, covering both quantitative and qualitative aspects. This section explains the methods used for evaluation, the reasons behind their selection, and the results obtained through quantitative and qualitative assessments. It also includes self-evaluations by all group members to reflect on their contributions to the project.

3.2 Evaluation methods

In this section, we describe the methods used to evaluate the app's effectiveness, usability, and performance. These methods include user testing, surveys, expert reviews, and performance analysis.

3.3 Quantitative evaluation

Quantitative evaluation was conducted to assess the accuracy and performance of the system using statistical analysis. For detailed results and testing of predictions, please refer to Chapter 2.

3.4 Qualitative evaluation (Feedback from end users, domain experts and industry experts)

During the prototype demonstration, a group of domain experts and end users were given the opportunity to examine it and provide feedback for future modifications and improvements. The reviewers had a good understanding of the software's flow and essential functionalities. Further explanations were given to them as to why certain functionalities were implemented at the request of the reviewers. Mr. William Silva, a quality assurance engineer, provided feedback on the application. He appreciated the use of Flutter as it can be used across multiple platforms. He also praised the use of machine learning, which he believes will be a trending topic in the future. Overall, he was pleased with the application's appearance and functionality.

3.5 Self evaluation

The navigation between pages is swift, taking only 0 to 1.3 seconds. Text detection is flawlessly executed, identifying the Text instantly when the user triggers the scan button. The system then processes and detects the text, which takes approximately 0.2 seconds. Once the text is detected, the system returns the results of the chemical, which takes around 1.5 seconds

3.6 Chapter Summary

This chapter provides an overview of the project's evaluations and the methodology used for the assessment. It goes into further detail about the information obtained through qualitative and quantitative assessments, as well as a self-evaluation to determine the project's level of success. The conclusion chapter includes the project's goals and objectives, as well as legal, social, ethical, and professional considerations, and outlines the limitations of the research conducted.

Chapter 4: Conclusion

4.1 Chapter Overview

In this chapter, the team will be addressing the journey throughout this project, from the beginning to the final product. Highlights the success of our aims and objectives, discusses the limitations encountered during the research process, outlines future achievements for this project, and provides additional work undertaken during the whole project.

4.2 Achievements of aims and objectives

The project was able to accomplish the stated goal within the given timeline. The completed objectives are listed below:

1. The first project proposal provided an overview of the project, including the problem context, goals and objectives, prototype scope and functionalities, and a feature comparison chart.
2. The literature review (LR) outlined the existing research activities in related fields, as well as the research gap and a full explanation of their strengths and weaknesses.
3. The Methodology chapter briefly explained the methodologies used in analyzing, developing, designing, evaluating, and managing the application. It emphasized the hazards involved and the system's mitigation plan. It also included structures for teamwork breakdown structure. The chapter thoroughly examined the project's use of project management and collaboration technologies.
4. The system requirements definition included the Onion model, selection of requirement elicitation techniques/methods with functional and non-functional needs, as well as their priority levels, use case diagram, and detailed use case descriptions. It is found in Chapter 4.

5. The Social, Legal, Ethical, and Professional Concerns chapter provided a brief description of how social, legal, ethical, and professional issues are relevant to the study endeavor and how they are mitigated. It was part of Chapter 5.

6. The System Architecture & Design chapter examined the high-level and low-level designs necessary to build the application, as well as illustrations that help users understand the system. It included a high-level architectural diagram, a class diagram, an activity diagram, sequence diagrams, a UI design prototype with wireframes, and a description of the system process flow. These were discussed in Chapter 6.

7. This chapter addressed the feature implementation of the Cosmo application. It covered the frameworks and libraries utilized in the development of this application, as well as how each feature was implemented, with explanations provided through code fragments and screenshots. It was discussed in Chapter 1 Implementation.

8. The primary goal of this chapter was to evaluate the prototype's functionalities and non-functionalities. This chapter also detailed all the extra testing carried out at this phase, such as compatibility testing. It was discussed in Chapter 2 Testing.

9. The chapter on overall evaluation and assessment methodology utilized delved deeper into the information received through qualitative and quantitative assessments, as well as a self-evaluation to determine how successfully the project was done. This was discussed in Chapter 3 Evaluation.

10. A CI/CD pipeline was set up to automate the process of building, testing and deploying the application. This improved the development cycle, ensuring that changes were swiftly and easily merged into the production environment while keeping the program stable and reliable.

11. Integrating Feedback : Collected user input and recommendations for future enhancements. This was received and used to improve the application's functionality, usability and overall user experience

4.3 Limitations of the research

We encountered some challenges with the project that had an impact on our research. Obtaining adequate data was one major problem. Finding thorough and up-to-date data regarding the ingredients and safety of cosmetic products proved to be difficult. This made it challenging to develop robust models and delve deeply into our investigation. We were also limited by time and resources, so we were unable to thoroughly investigate every research suggestion. Together, these drawbacks highlight how critical it is to develop more efficient data collection strategies and to keep ethical issues in mind when doing research in the future.

4.4 Future enhancements

Even though we faced challenges, there are still plenty of ways our project can grow in the future. One big area is improving how we gather data. By working with industry partners and using advanced technology like machine learning, we can get better data and build more accurate models. This could help us predict safety issues with cosmetics products more effectively. We also need to keep an eye on changes in regulations and industry standards to make sure our research stays relevant. By focusing on these areas, we can make our research more impactful and useful for consumers in the cosmetics industry.

Gathering more relevant data is one of the most important aspects of this system because the more data gives more matches and predictions to the system. Furthermore, in the future, the application should use a more complex matching system to match data with results. Currently, it takes a bit more time to give the matched data. Although adding information would be nice in the system even with pictures of the chemicals itself would be nice to the system.

Unrealized Potential: Flutter-Powered Google Maps Nearby Places

Although we've experimented with the capabilities of adjacent locations and incorporated Google Maps, there's a wealth of unrealized potential when it comes to SDKs and API keys. Below is a summary of the options that, owing to time limits, we were unable to investigate in full:

Advanced Functionalities:

- Granular Search: Utilize custom filters based on user preferences (price range, cuisine type) to dive deeper than basic categories (restaurants, ATMs).
- Updates in real time: Provide real-time traffic information or updates on company hours to provide users a more engaging experience.
- Particularized Suggestions: Utilize user preferences and location history to recommend relevant, nearby locations that they would find enjoyable.
- Improving the User Experience
- Rich Location Details: Use the Places API to provide ratings, reviews, and images for neighboring places in place of merely icons.
- Enable autocomplete for search phrases to facilitate intuitive search, which will help users find what they're seeking for more quickly and easily.
- Customizable Map Overlays: Make unique overlays that emphasize key locations or provide information unique to your app.
- Actions dependent on location:
-
- Optimized Routing: Determine the best path to a selected local location by combining traffic information with directions.
- Proximity Triggers: These are events or alerts that are set off by a user's proximity to particular places (for example, reminding consumers of deals nearby when they're about to enter a store).

The options are endless, and these are just a few instances. We can discover a whole new level of functionality and user experience for locating nearby locations within our Flutter app by digging deeper into Google Maps APIs and SDKs.

4.5 Extra work (Competitions, research papers, etc)

No, our team did not take part in any hackathons, conferences, or contests throughout the SDGP (Software Development Group Project) period. During that time, we just paid attention to the advancement and completion of our project. We gave teamwork, problem-solving, and project management top priority in order to successfully complete our software project. We choose to focus all of our resources and attention on finishing our SDGP project, even though we understand the benefits of attending outside events for networking and skill development. This made it possible for us to give the project needs our whole focus, fulfill deadlines, and provide a high-caliber software solution. We thus didn't take part in any extracurricular activities that weren't related to our SDGP project.

4.6 Concluding remarks

The project has reached a critical turning point in its development by effectively completing its primary goals. With careful preparation, careful execution, and teamwork, the team has produced a product that satisfies the core objectives stated at the beginning of the project. These goals may have been the creation of important features, reaching performance targets, guaranteeing platform compatibility, or any other requirements thought necessary for success.

However, it is understood that there is always opportunity for development due to the constant nature of technology and consumer expectations. Although the project's main goals have been met, it is acknowledged that continuous improvement and modification will be required to maximize the final outcome. This includes optimizing functionality, improving usability, resolving any unexpected problems that may surface during testing, and taking stakeholder comments into account.

It is confirmed that the product is at a place where it can be deployed securely without jeopardizing the privacy of its customers, notwithstanding these possible areas for development. This claim highlights the close attention that privacy issues have received throughout the development process. To protect user data and guarantee reliability, protocols including data encryption, privacy-by-design, and adherence to pertinent laws have been scrupulously put into place.

In conclusion, the project has successfully accomplished its main goals, providing a strong basis for a fruitful product launch. Stakeholders should feel confident in the product's deployment readiness even though more improvements could be required to further improve it, since user privacy has always been given top priority."

References

Rahman, A. (2020). Deploying a Flutter Web App to Firebase Hosting. Medium.

<https://medium.com/flutter-community/deploying-a-flutter-web-app-to-firebase-hosting-8749d3bf535f>

Taboada, M., 2016. Sentiment analysis: An overview from linguistics. Annual Review of Linguistics, 2, pp.325-347.

<https://www.annualreviews.org/content/journals/10.1146/annurev-linguistics-011415-040518>

Appendix

A video about the app

<https://youtu.be/3KjcqTmwb9U?si=LVuk95TXq5XZ8-E->