SnapShop

Milestone 3: README



Tan Yi Xuan

Chen Shaoheng

1. Introduction **3**
1.1 Team Name: SnapShop 3
1.2 Proposed Level of Achievement: Apollo 11 3
1.3 Project Scope 3
1.4 Motivation 4
1.5 User Stories 5
2. Features **6**
2.1 User Account & Authentication 6
2.1.1 Google sign-in feature 6
2.2 Image Recognition 10
2.3 Search by keyword 16
2.4 Product Search 17
2.5 Price Comparison 21
2.5.1 Load More Button 21
2.5.2 To The Top Button 22
2.6 Review Aggregation 25
2.9 User Recommendations 34
3. Tech Stack **36**
4. Software Engineering Practices **37**
4.1 Version Control 37
4.2 DRY (Don't Repeat Yourself) 38
4.3 Code Refactoring 39
4.4 Documentation 40
4.5 User-Centred Design (UCD) 41
4.6 Security measures 42
4.7 Agile Development Methodology 42
4.8 Separation of Concerns 43
4.9 Model-View-Controller software design 44
4.10 CRUD operations 44
5. Testing **45**
5.1 Unit Testing 45
5.2 Integration Testing 54
5.3 System Testing 60
5.4 User Testing 62

1. Introduction

1.1 Team Name: SnapShop

1.2 Proposed Level of Achievement: Apollo 11

1.3 Project Scope

SnapShop is a fuss-free shopping app that offers efficient visual product search and convenient price & review comparisons to facilitate a seamless shopping experience.

We hope to:

1. Provide a more convenient and efficient product search experience

- The primary aim of the app is to provide users with a seamless and convenient way to search for and discover products online.
- By leveraging image recognition technology, the app aims to streamline the product search process, reducing the time and effort required to find specific items.
- Through visual product search capabilities, users can simply upload images of desired products, eliminating the need for cumbersome text-based search queries.

2. Provide a unified shopping experience with efficient price comparisons

- Another aim of the app is to provide users with a unified shopping experience by integrating with multiple e-commerce platforms such as Shopee and Lazada, to allow users to be acquainted with a more diverse range of products related to their search.
- The app aims to simplify the process of comparing prices across different online stores and platforms.
- By aggregating product listings from various retailers, users can easily compare prices and make informed purchasing decisions without having to visit multiple websites individually.

1.4 Motivation

(A) Lack of visual product search options, resulting in reduced convenience & efficiency

Traditional search relies on keywords, but users might struggle to describe a product accurately, especially for unfamiliar items where they are unsure of its exact name or brand. For example, someone might see a pair of sunglasses they love on a friend but doesn't know the brand or model. This is especially so for items with specific features or characteristics that are difficult to articulate. For example, users may struggle to search for a specific type of plant or a unique piece of furniture using text alone. This limitation often leads to frustration and dissatisfaction among users who struggle to find the products they are looking for. The need to refine keywords based on initial search results can also be a time-consuming and frustrating process as it can lead to browsing irrelevant products and wasted time.

Visual product search addresses this issue by allowing users to simply upload an image of the desired item, bypassing the need for textual descriptions altogether. This streamlined approach enhances convenience and efficiency for users. By simply taking a photo or uploading an image, users can quickly discover similar products available online without the need for tedious and time-consuming manual searching. Furthermore, image recognition can identify specific products or similar styles, leading to more relevant results. This significantly reduces the time and effort required to find specific products.

(B) Disjointed shopping experience & tedious price comparisons

Comparing prices across different online stores and platforms is highly tedious and time-consuming, engendering a disjointed experience as users have to navigate between multiple e-commerce platforms to compare products, prices, and availability. This app aims to streamline the process by aggregating product listings from various sources and presenting them into a unified one-stop platform, saving users time and effort, thus offering a seamless and cohesive shopping experience.

Additionally, Price transparency is crucial for consumers. By offering a feature that compares prices across different platforms, users can make informed decisions and potentially find better deals. Reviews of the products will also be provided to allow customers to better evaluate their options.

1.5 User Stories

User Story #1

As a user who values efficiency, I want to be able to take a photo of a desired item and find relevant products online so that I can browse and select the one I like. Furthermore, I want to be able to filter search results by price and rating, so that I can find the best product quickly.

User Story #2

As a price-conscious consumer who wants to save money and find the best deals available, I want to be able to upload an image of a product or scan the product in the store and see all the matching listings on different platforms like Carousell, Lazada, Shopee, and Amazon, so I can compare prices and make an informed decision of whether I should buy the product in store or online.

User Story #3

As a user who values transparency and authenticity, I want to be able to view aggregated product reviews and ratings from multiple platforms, so I can make an informed decision based on the experiences of other shoppers before deciding if the product is worth buying.

User Story #4

As a shopper who values convenience, I want to be able to create a wishlist of products I'm interested in, so I can easily keep track of them and make purchases when the time is right. I would also want to share my wishlist with friends and family so that they can see what products I'm interested in.

User Story #5

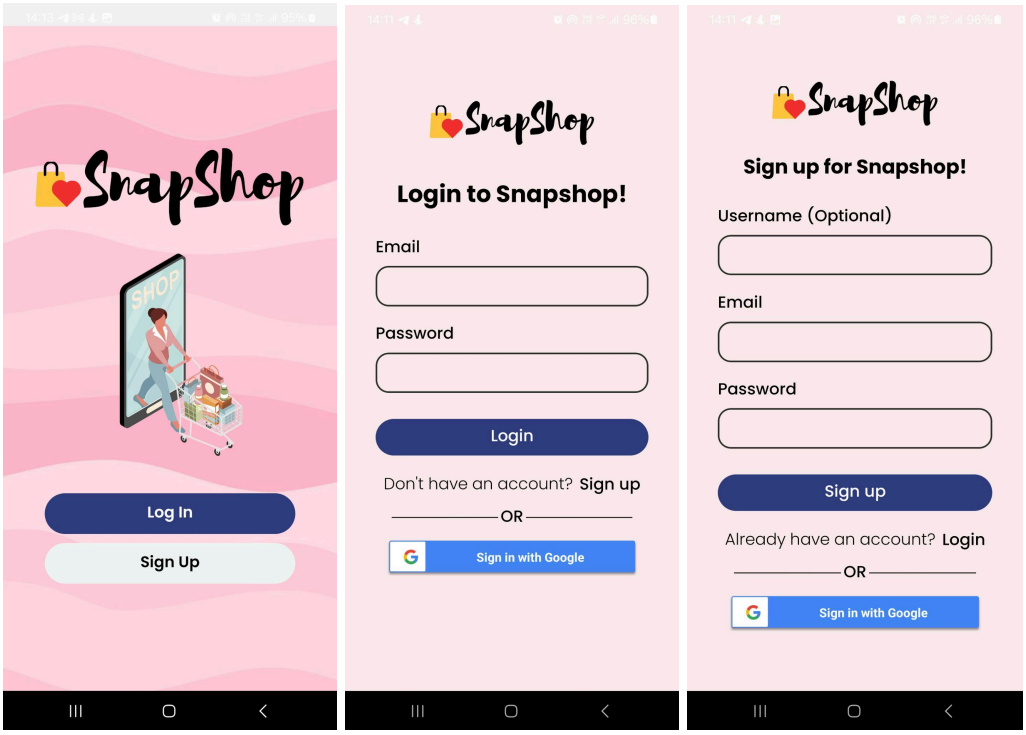
As a user who values data privacy and security, I want to log in securely using my email and password or social media accounts so that I can access my saved data. I would want my account to save my preferences, search history and wishlist in a secure manner.

2. Features

2.1 User Account & Authentication

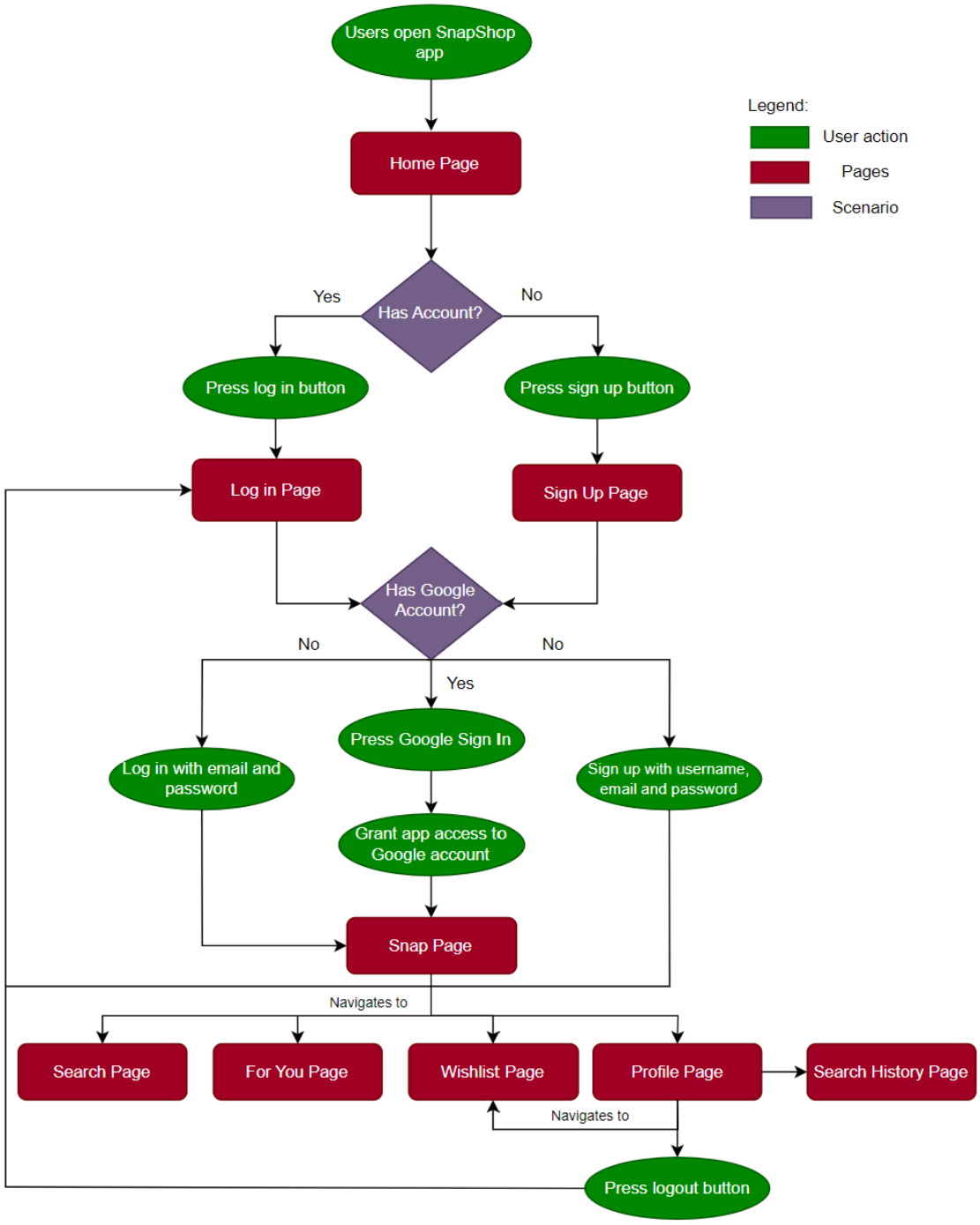
Description

Our app provides a robust user account and authentication system to allow users to have their own personalised account that can keep track of their searches and wishlist. Users without an account can easily sign up by providing a username, email, and password. Those with an existing account can log in using their email and password.



2.1.1 Google sign-in feature

Furthermore, our app incorporates a social media sign in option with google sign in for users to sign in more conveniently using an existing google account with just the click of the “Sign in with Google” button instead of having to create a new account from scratch. This can save users’ time as they need not enter an email and password every time to log in to the app and thus can make it more appealing to users.



Authentication and app flow

Implementation design

This entire process is powered by Firebase Authentication, a cloud-based solution to securely manage user credentials and authentication.

To access the app's features, users must be logged in. If a user is not authenticated, they will be redirected to the login page. This ensures that all interactions within the app are associated with a verified user, enhancing both security and personalised user experience.

Furthermore, using the react-native-google-signin/google-signin library, we were able to integrate Google authentication with Firebase in our React Native application. By calling GoogleSignin.signIn(), we can obtain a user's ID token which is then passed into GoogleAuthProvider.credential() to obtain a Google Credential with the ID token. We can then sign in to Firebase using the Google Credential obtained, thus allowing us to connect Google sign in with Firebase to ensure that all user activities can be stored in and retrieved from Firebase the next time the user logs in to the same account via Google sign in.

```
// Check if your device supports Google Play services.
await GoogleSignin.hasPlayServices({ showPlayServicesUpdateDialog: true });

// Get the user's ID token
const { idToken } = await GoogleSignin.signIn();

// Create a Google credential with the token
const googleCredential = GoogleAuthProvider.credential(idToken);

// Sign-in the user with the credential
await signInWithCredential(authy, googleCredential)
  .then((res) => {
    router.replace('/snap')
  })
```

Limitations

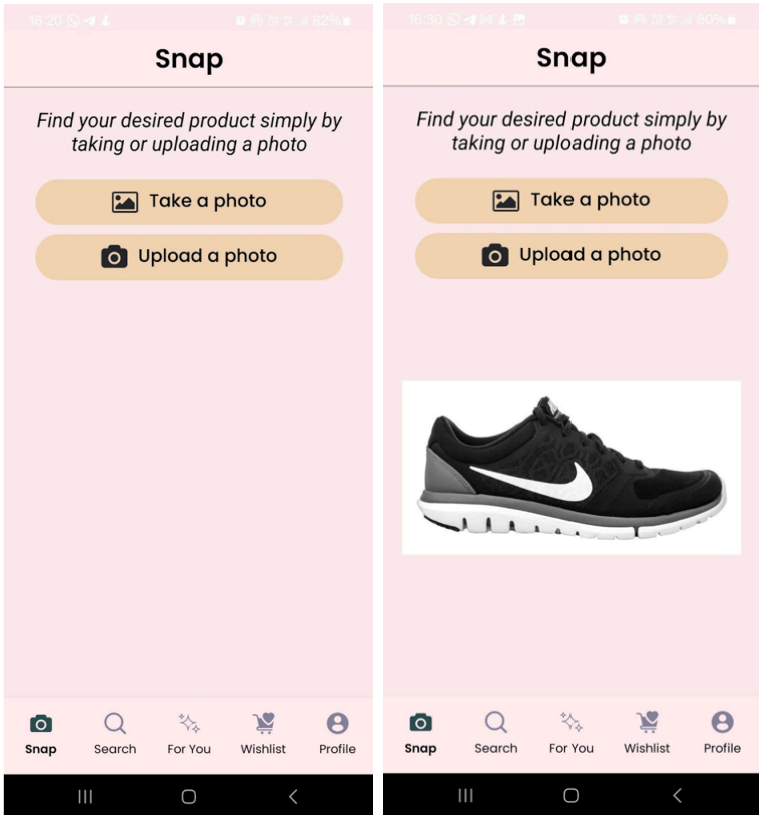
Currently, the only social media sign in feature available is Google sign in. Other common sign in options include Facebook sign in which can provide a greater variety of sign in options for users. However, incorporating Facebook sign in requires business verification using business documentations and is subject to Facebook’s approval. As we are not running a business, we are unable to perform that verification and thus we will not be able to add in a Facebook sign in option. However, having Google sign in should generally be sufficient since Google is one of the

most, if not the most used social media platform so most users should have a Google account and be able to sign in via Google, thus incorporating additional social media sign in options would be good but may not be entirely necessary.

2.2 Image Recognition

Description

The image recognition is a feature used to make product search a lot simpler by allowing users to take a picture or upload an existing picture of their desired product to the app and the app will display listings of all relevant products from Amazon, Lazada and eBay related to the uploaded picture. For example, if a user wishes to buy a shoe they came across online, they can simply save the image to their device and upload it to SnapShop.



Our app leverages Microsoft Azure's AI Vision to perform comprehensive image analysis on user-uploaded photos. When a user uploads an image, the app sends the image data to Azure's Computer Vision API. This API employs an advanced object detection model, based on deep learning algorithms and convolutional neural networks (CNNs), to analyse the image and returns the results of the image analysis in the form of JavaScript Object Notation (JSON).

Implementation design

In order to utilise the Microsoft Azure AI Vision APIs, we first created a Microsoft Azure account which allows us to obtain a key and URL endpoint. When calling the API, authentication is done

by adding the HTTP request header Ocp-Apim-Subscription-Key and setting it to our vision key. The call is made to the URL

https://snapshop.cognitiveservices.azure.com/computervision/imageanalysis:analyze?api-version=2024-02-01

, where https://snapshop.cognitiveservices.azure.com is our unique computer vision endpoint URL. To analyse a local image, we need to post the binary image data in the HTTP request body. That is done by converting the image jpeg file uploaded to a BLOB type and setting the Content-Type in the request header to “application/octet-stream”.

```
// Convert image uri to blob
const blob : BlobPart = await new Promise((resolve, reject) => {
  const xhr = new XMLHttpRequest();
  xhr.onload = function () {
    resolve(xhr.response);
  };
  xhr.onerror = function (e) {
    reject(new TypeError("Network request failed"));
  };
  xhr.responseType = "blob";
  //console.warn(image)
  xhr.open("GET", imageUri, true);
  xhr.send(null);
});

const imageFile = new File([blob], "image.jpg", { type: "image/jpeg" })
```

Subsequently, the API call is made, utilising the brand detection and image captioning features of the Azure AI Vision. Azure AI Vision API contains many image analysis features such as detecting common objects in images, extracting common tags from images, detecting sensitive content in images, adding captions and dense captions to images, recognising products on shelves and many more. After careful testing of the features, we found that the captioning feature is most precise and relevant for our users in terms of generating text that best describes the product that users will be uploading.

```
//Image captioning
fetch('https://snapshop.cognitiveservices.azure.com/computervision/imageanalysis:analyze?api-version=2024-02-01&features=caption', {
  method: 'post',
  headers: { "Ocp-Apim-Subscription-Key": key,
             "Content-Type": "application/octet-stream"
   },
  body: blob
})
```

From the JSON data returned by the API, the relevant text can be extracted by parsing the string for the contents of the "captionResult" section.

```
{modelVersion: '2023-10-01', captionResult: {…}, metadata: {…}}
  captionResult: {text: 'a black and white shoe', confidence: 0.7874889969825745}
  metadata: {width: 702, height: 360}
  modelVersion: "2023-10-01"
  [[Prototype]]: Object
```


Additionally, we have leveraged Azure AI Vision's brand detection API to identify and recognize brands in user-uploaded photos. Utilising advanced neural networks trained on a vast database of thousands of global logos, Azure's brand detection can accurately identify commercial brands in images or videos. If a brand is detected by the brand detection API, our app augments the detected item with the brand name to provide a more precise description of the detected product. This enhancement improves the accuracy of our image recognition, thus helping users to find related products and information more efficiently.

```
//Brand detection
fetch('https://snapshop.cognitiveservices.azure.com/vision/v3.2/analyze?visualFeatures=Brands', {
  method: 'post',
  headers: { "Ocp-Apim-Subscription-Key": key,
             "Content-Type": "application/octet-stream"
   },
  body: blob
})
```

From the JSON data returned by the API, the brand with the highest confidence score can be extracted by retrieving the first item in the brands array. In this case, that will be "Nike". Results from both the image captioning and brand detection API will be put together via string concatenation where the brand is concatenated to the end of the image caption to obtain the final search query "a black and white shoe Nike".


```
brands: Array(1)
  0: {name: 'Nike', confidence: 0.933, rectangle: {…}}
    length: 1
  [[Prototype]]: Array(0)
metadata: {height: 360, width: 702, format: 'Jpeg'}
  modelVersion: "2021-05-01"
  requestId: "704d039e-4f7a-46b5-929c-63ce1cb0a27b"
[[Prototype]]: Object
```


The image analysis process involves several key steps:

Preprocessing: The uploaded image is preprocessed to normalise it, resize it to the required dimensions, and optimise it for analysis.

Feature Extraction: The model extracts key features from the image, such as edges, textures, shapes, and colours, by applying multiple convolutional layers. These features are critical for recognizing and differentiating objects.

Object Identification: Using the extracted features, the model classifies and identifies the objects in the image by comparing them against its extensive training data. The model can recognize a wide range of objects, from everyday items to specific products.

Bounding Boxes and Labels: For each detected object, the model generates a bounding box that highlights the object's location in the image. It also assigns a label and confidence score, indicating the type of object and the model's confidence in its identification.

The results, including the identified objects, their locations, and confidence scores, are then returned to our app. This detailed information allows the app to accurately identify the item found in user-uploaded photos, enabling a seamless user experience in recognizing and managing their items.

Implementation challenges

We started off using a simpler image recognition model known as Ultralytics YOLOv8 as the API was simpler to implement. However, upon vigorous testing, we realised that this model is severely limited in its object detection capabilities as it was unable to return a correct analysis of the image on many occasions. For example, after uploading an image of a Nike shoe, it will say that a shoe is a person. Thus, we decided that we needed a more accurate and well-trained object detection model and eventually found Microsoft Azure AI vision image analysis model to be highly suitable for our app.

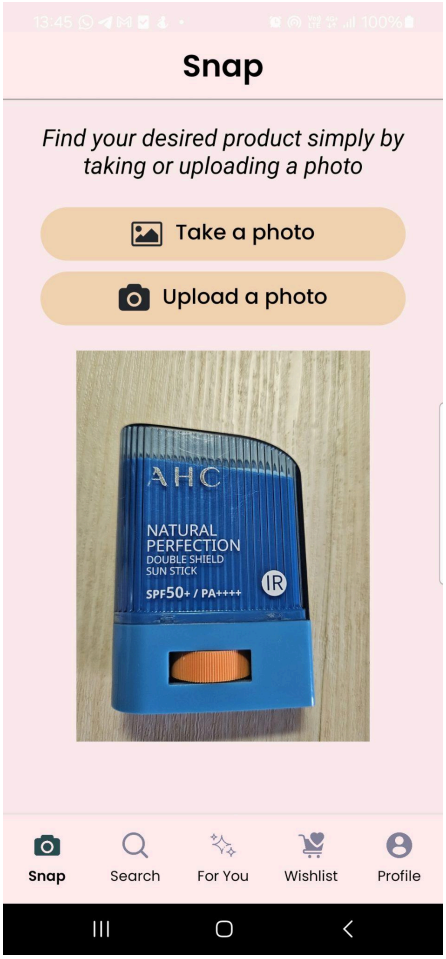
As we first started to incorporate the Microsoft Azure AI vision image analysis API, we faced difficulties in getting it to work as we often faced error: 400, bad request when trying to send image data to the API. We subsequently realised after reading the documentation that we needed to convert the uploaded image file to a Binary file format as specified in the documentation before sending it.

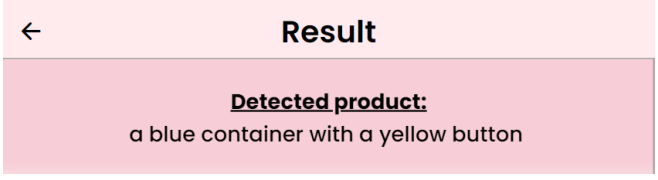
After getting the image captioning feature of Microsoft Azure AI vision image analysis API to work, we realised that this feature alone does not provide useful product information to the users as it is unable to recognise prominent product brands and therefore more can be done to

enhance the image recognition capabilities of our app. From there, we decided to incorporate the brand detection feature of Microsoft Azure AI vision image analysis version 3.2 so that prominent brands like Nike, Adidas of products in photos can be added to the search query.

Limitations

The Microsoft Azure AI vision image analysis is not all powerful. If the product is not a common object, or is in a weird shape, it will not be able to identify the product itself. For example, given a sunstick, it may not be able to recognise that it is a sunstick, but rather it will only provide a vague description of the object, for example, a blue container with a yellow button.



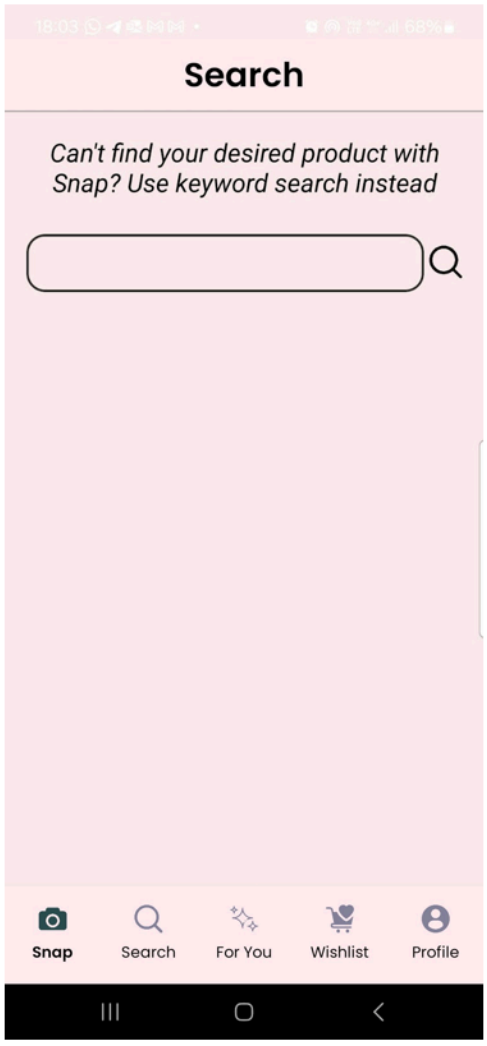


This is also due to the fact that the Microsoft Azure AI vision image analysis feature used here is image captioning which generates a suitable caption that best describes the image, rather than solely focusing on identifying the product itself. We could overcome this by adding a call to the image text extraction feature of the API to allow any text on the product to be extracted so that in the above case, at least it will be able to identify the product as a double shield sun stick. However, doing so could result in the search query being very long if the product has a lot of text. Also, if the product has gibberish text written on it that is not related to the product, then using a text extraction feature will only cause the product search to become unreliable. In such cases, the use of image recognition may not be suitable and a manual text search of the product using the search feature will be more reliable for users to find the product they want.

2.3 Search by keyword

Description

If the image recognition is not precise enough or if users prefer using search by keyword instead, a search bar is present for users to key in the product that they want to search for and the relevant products will then be displayed based on whatever was typed into the search bar.



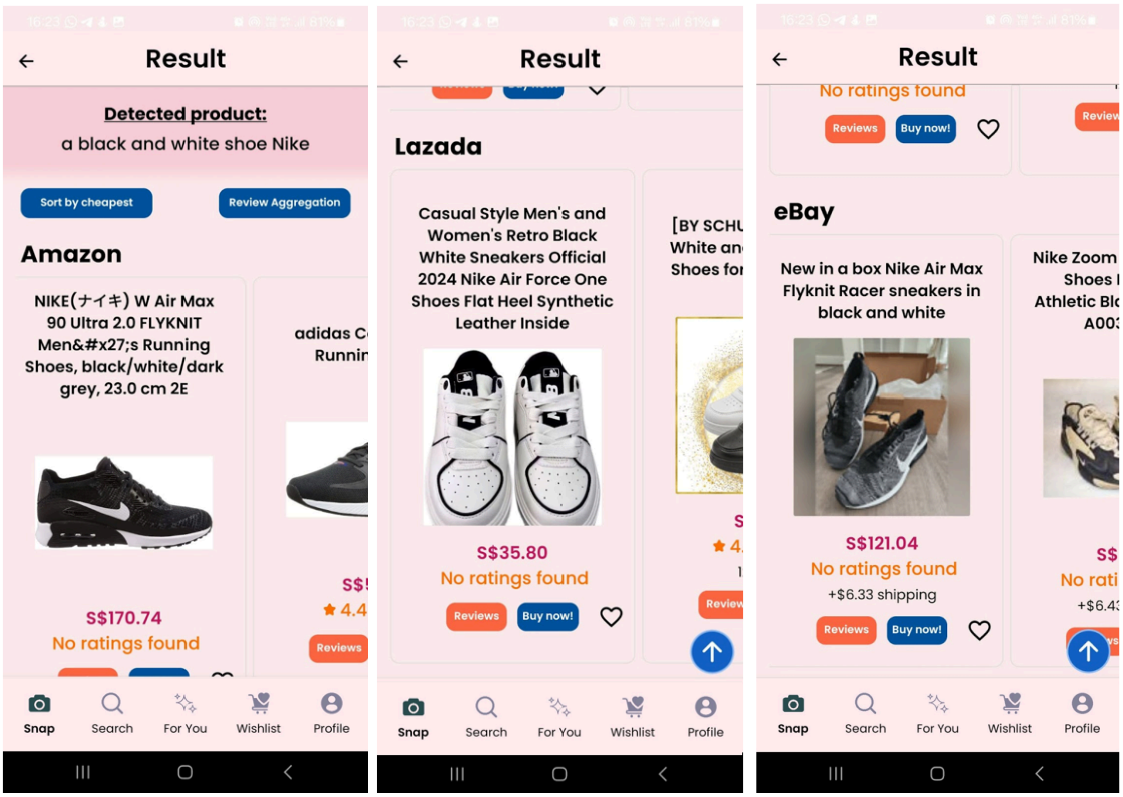
Implementation design

Keyword search is implemented by keeping track of user input using state and setState and subsequently passing the user input as a string directly to the ecommerce APIs to retrieve the relevant product information based on what the user has entered.

2.4 Product Search

Description

Based on the identified product, our app will search for relevant products on Lazada, Amazon and eBay by sending a “GET” request to the Lazda, Amazon and eBay APIs to retrieve data related to the search query on these ecommerce platforms. Our app will then display these product listings on an user-friendly interface, with clear images, titles, prices, ratings and brief descriptions. This ensures that users can easily browse and select items of interest. For instance, after uploading an image of the desired shoe, our image recognition feature detects that the product is “a black and white shoe Nike” and that query is sent to the ecommerce APIs. The detected product is also displayed at the top of the app to let the users know what the image recognition detected and the products listed will be based on that line of text.



Products listed are categorised by ecommerce platforms. Users can scroll through the products horizontally for each ecommerce platform to identify items of interest if they have a preference for products from a particular ecommerce platform, like Amazon for example. Users can see the price of the products, the ratings based on user reviews and how many sold at a glance to help

them decide whether they want the product. Clicking on “Buy now!” will redirect users to the product website directly where they can browse the product in greater details or purchase the product. Clicking the heart icon will save the product to a wishlist for users to look at them again in the future.

Implementation design

The ecommerce APIs are obtained from rapidAPIs and a “GET” request is made to the APIs using the text obtained from Microsoft Azure AI vision image analysis API, where the item variable contains the text string that is obtained from the image recognition step, and by substituting that into the API endpoint, the API call for the desired product can be made.

```
function getAmazonProducts() {
  return fetch(`https://real-time-amazon-data.p.rapidapi.com/search?query=${item}&country=SG`, {
    method: 'GET',
    headers: {
      'X-RapidAPI-Key': api_key,
      'X-RapidAPI-Host': 'real-time-amazon-data.p.rapidapi.com'
    },
  })
```

```
function getLazadaProducts() {
  return fetch(`https://lazada-api.p.rapidapi.com/lazada/search/items?keywords=${item}&site=sg&page=1`, {
    method: 'GET',
    headers: {
      'X-RapidAPI-Key': api_key,
      'X-RapidAPI-Host': 'lazada-api.p.rapidapi.com'
    },
  })
```

```
function getEbayProducts() {
  return fetch(`https://ebay-search-result.p.rapidapi.com/search/${item}`, {
    method: 'GET',
    headers: {
      'X-RapidAPI-Key': api_key,
      'X-RapidAPI-Host': 'ebay-search-result.p.rapidapi.com'
    },
  })
```

The data obtained is in JSON format and can be parsed to obtain the relevant information.

```
▼ 0:
    delivery: "S$19.94 delivery"
    image: "https://m.media-amazon.com/images/I/61XrkhmZAbL._AC_UL960_FMwebp_QL65_.jpg"
    platform: "amazon"
    price: "260.66"
    rating: "4.5"
    sales: ""
    title: "Nike Wmns Free RN Flyknit 2018, Women&#x27;s Running Shoes"
    url: "https://www.amazon.sg/dp/B01M7PEJWT"
  ▶ [[Prototype]]: Object
▼ 1:
    delivery: "S$21.11 delivery"
    image: "https://m.media-amazon.com/images/I/51ANCj+1OlL._AC_UL960_FMwebp_QL65_.jpg"
    platform: "amazon"
    price: "172.05"
    rating: "5"
    sales: ""
    title: "Nike Air Max Dawn Women&#x27;s Running Shoes (White/Total Orange/Black DH5131-101, US Footwear Size S
    url: "https://www.amazon.sg/dp/B0B5QK8TB8"
  ▶ [[Prototype]]: Object
▼ 2:
    delivery: "S$20.78 delivery"
    image: "https://m.media-amazon.com/images/I/612FKS4WsGL._AC_UL960_FMwebp_QL65_.jpg"
    platform: "amazon"
    price: "151.39"
    rating: "4.5"
    sales: ""
    title: "Nike Wearallday, Women&#x27;s Running Shoes"
    url: "https://www.amazon.sg/dp/B0838KQFC5"
```


Edge cases where the product data contains strange values are accounted for by iterating through the array of products and replacing any invalid price, reviews and sold count by “invalid price”, “invalid reviews” and “null” respectively. The product picture is displayed by linking the image url in the JSON. Upon clicking the “Buy now!” button, users will be redirected to the url of the product page given in the JSON.

Challenges

One main challenge we faced with the implementation of the product search at the start was that we first wrote the API code in Python and that required starting up another server in order to access these API call functions. This caused an issue when trying to run the app on an emulator or on the Expo Go app on the phone because these devices do not have access to the new server and thus a network error was raised and the API retrieval failed. In the end we worked around it by porting the code to TypeScript so that all the code is consolidated on one page and the API calls can all be done on the same IP address, thus allowing the API calls to go through.

Limitations

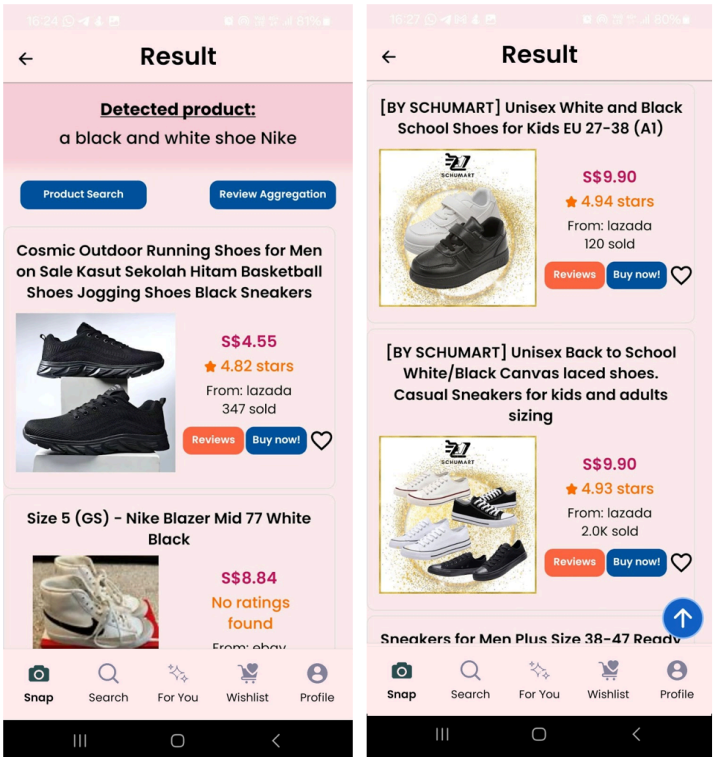
There's a limit to the number of API calls that can be made for our free model. For Amazon it's 100 calls per month and 50 calls per month for Lazada. Thus, we need to be stringent in the number of usages of the API calls if we do not want to exceed the limit and risk not being able to call the API once the limit is exceeded. Thus, when testing the product search feature, we use makeshift data on toothbrushes in the same JSON format to avoid having to call the ecommerce APIs too many times.

Furthermore, we are unable to incorporate the APIs of other popular ecommerce platforms such as Shopee, Carousell and Taobao due to different reasons. Shopee's API is not free to use, Taobao's API requires chinese input to be more reliable, while Carousell's API only provides 20 API calls per month for the free version which is way too limiting to incorporate. Our inability to use these APIs results in a lack of variety of popular ecommerce platforms that most Singaporeans use. However, this is a limitation that we will not be able to overcome unless we are given extra monetary resources to use the paid APIs which can then allow us to incorporate the APIs of more ecommerce platforms with greater allowance of API calls per month.

2.5 Price Comparison

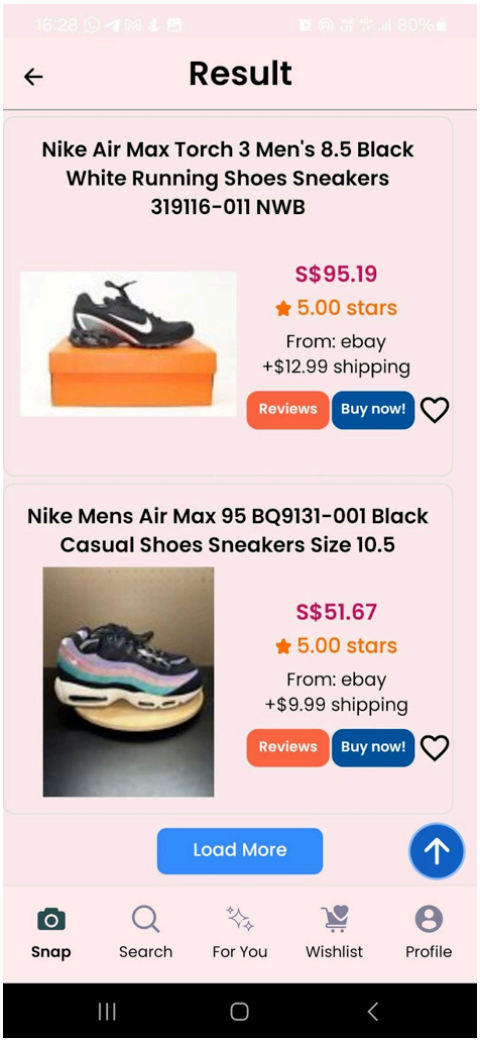
Description

By aggregating pricing data from multiple platforms, our app allows users to efficiently compare prices of the identified products across different platforms, thus enabling users to find the best deals available. Users can also choose to sort the products by price in ascending order to find the cheapest deals at a glance. This is done by clicking on the “Sort by cheapest” button on the top left of the product search page. Users can then scroll through the products vertically to find their desired product.



2.5.1 Load More Button

Additionally, as only the first 15 products are displayed at the start, once users reach the bottom of the page, they can press the “Load More” button to load another 15 products and this process continues until there are no longer any more products to display. This allows users to view products at their own pace without feeling overwhelmed by the sheer number of products aggregated from the 3 ecommerce platforms.



2.5.2 To The Top Button

As the number of products to display may be large and the page may be very long causing it to be very time consuming for users to manually scroll back to the top once they have reached the bottom, a To The Top Button in the shape of an upward arrow enclosed in a blue circle is implemented to allow users to instantly head back to the top of the page with ease just by clicking on the button, no matter how far down they are, thus bringing great levels of convenience.

Implementation design

From the JSON data obtained from each ecommerce API, a setState call is made to amazonResult, lazadaResult and ebayResult variables respectively to store the data. The data is then cleaned by iterating through each array to see if the price data is null or empty string and setting the price to the string “Invalid price” if so. Furthermore, as the products from eBay are given in USD instead of SGD, the prices in the JSON retrieved from the eBay API are converted to SGD by multiplying them by 1.36. Subsequently, items with invalid prices are filtered out and the arrays of products from Amazon, Lazada and eBay are concatenated to create a combined price array. Using this combined array, the items are sorted by their prices after using a comparator function that compares the prices after converting the price strings to float values to order the items from cheapest to most expensive. The first 15 items are then displayed, which represents the cheapest 15 items.

To implement the Load More Button, a separate array is used to store the full list of all items across all 3 platforms sorted by their prices from cheapest to most expensive. A global variable numPrice is used to keep track of the number of items to display at the current juncture, which is set to 15 at the start to display only the first 15 items. Every time the Load More Button is pressed, numPrice is incremented by 15 and the array that is used to render the display will take the items from the array containing the full list of items and slice it by the number of items to display based on numPrice. Once numPrice exceeds the length of the full list array, the load more button will no longer be displayed as all items are already displayed.

To implement the To The Top Button, the createRef hook from React is used to keep track of the current position of the user in the scroll view by creating a variable scrollRef using createRef() and linking scrollRef to the ScrollView. And when the button is pressed, the app will scroll the user up from the current position back to the top.

```
const scrollRef : any = createRef();
const onPressTouch = () => {
  scrollRef.current?.scrollTo({
    y: 0,
    animated: true,
  });
}
```

For better design purposes, the button will only appear once the user is about one phone screen length down. This is achieved by setting a number to indicate the vertical threshold where the

button will appear after scrolling down beyond that threshold. And by updating the position of the user every time the user scrolls, once the position of the user exceeds the specified threshold, the button will be displayed, and will be hidden once again when the user scrolls up and goes back below the threshold.

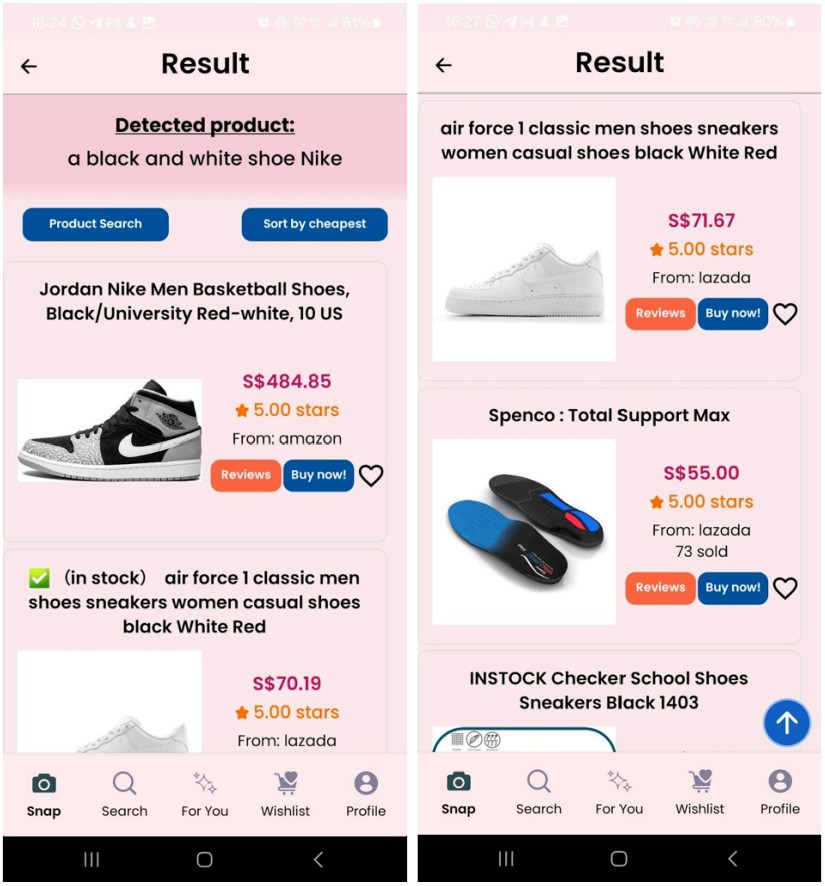
Challenges

The main challenge faced was trying to standardise the prices for sorting since the different ecommerce APIs return prices in different formats. For instance, Amazon prices are given with a “S$” in front so the price data needs to be adjusted by slicing from the second index of the string to obtain the numerical value. Likewise, eBay prices are given with a “$” in front so the data needs to be sliced from the first index of the string. We also faced issues with invalid prices as sometimes the data from ebay can contain 2 prices which will mess up the sorting algorithm so we need to iterate through all the data and identify price data that deviates from the norm and filter them out to allow the price sorting algorithm to work without error.

2.6 Review Aggregation

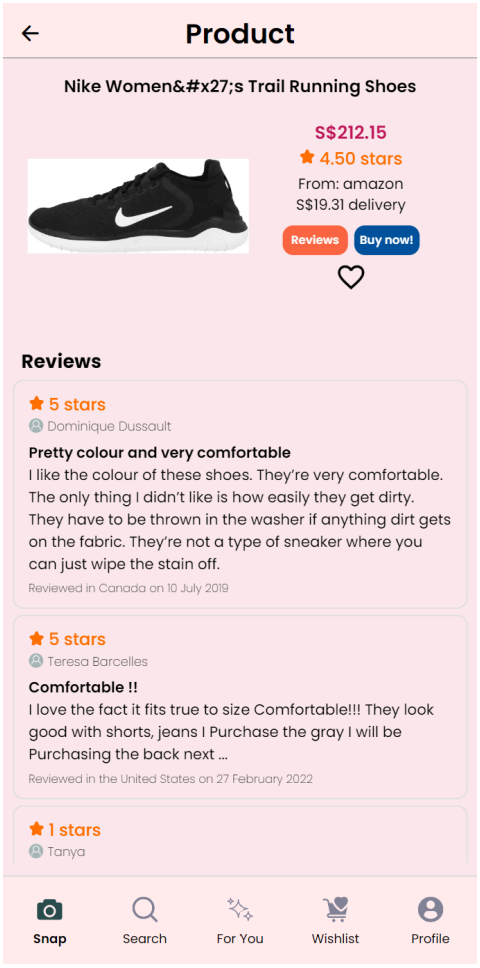
Description

Our app allows for comparison of product ratings across the different ecommerce platforms to provide a comprehensive overview of a product's reception by sorting the products from most highly rated to least rated, which will be highly effective in assisting users to make informed purchasing decisions. Users can simply click on the “Review Aggregation” button on the top right corner of the product search page and the products will be sorted in order from highest number of stars. Similar to price comparison, users can scroll down vertically to browse the products. Once they have reached the end of the page, they can press the load more button to view the next 15 products and press the Upward Arrow button to return to the top of the page.



2.6.1 Reviews Page

Additionally, we have integrated a convenient "Reviews" button for each product listing. When users click on this button, they are seamlessly redirected to a dedicated reviews page. This page is designed to display reviews from other users specifically for the selected product, offering valuable insights and firsthand experiences.



Implementation design

By iterating through the array of items obtained from the API calls, If the rating data obtained in the JSON from the API calls is an empty string or null, it will be replaced by “No ratings found” to standardise the display for the case when there are no reviews for the product yet. The arrays for products from Amazon, Lazada and eBay are then concatenated to create a combined reviews array and the products with no ratings found will be filtered. Subsequently, the products are sorted using a comparator function that converts the rating score to float values and sorting

the products from highest rated to lowest rated. The first 15 items from the resultant array representing the highest rated 15 items are then displayed.

To implement the reviews page, a new route /result/[id] was created, allowing a unique page to be rendered based on the id of a product. Upon clicking the “Reviews” button, users will be redirected to this dedicated reviews page where another API is called to retrieve user reviews of the products. Information such as author, review title, review star rating and review date for each review obtained from the API are then displayed on the page.

```
return fetch(`https://real-time-amazon-data.p.rapidapi.com/product-reviews?asin=${id}&country=SG`, {
  method: 'GET',
  headers: {
    'X-RapidAPI-Key': api_key,
    'X-RapidAPI-Host': 'real-time-amazon-data.p.rapidapi.com'
  },
})
  .then(res => {
    if (!res.ok) {
      console.error("Error fetching review data")
    }
    console.log("Fetch review success")
    return res.json()
  })
  .then(data => {
    console.log(data)
    let reviewData: any[] = data["data"]["reviews"]
    reviewData = reviewData.map(review => ({
      id: review["review_id"],
      title: review["review_title"],
      comment: review["review_comment"],
      rating: review["review_star_rating"],
      author: review["review_author"],
      author_avatar: review["review_author_avatar"],
      date: review["review_date"]
    }))
    setReviews(reviewData)
  })
  .catch(error => console.error("Failed to fetch reviews" + error))
```

Challenges

A slight challenge for this feature was dealing with product data that contained no ratings as it cannot be sorted if there’s no numerical value to the rating so we had to filter out these products to allow the sorting algorithm to work without error.

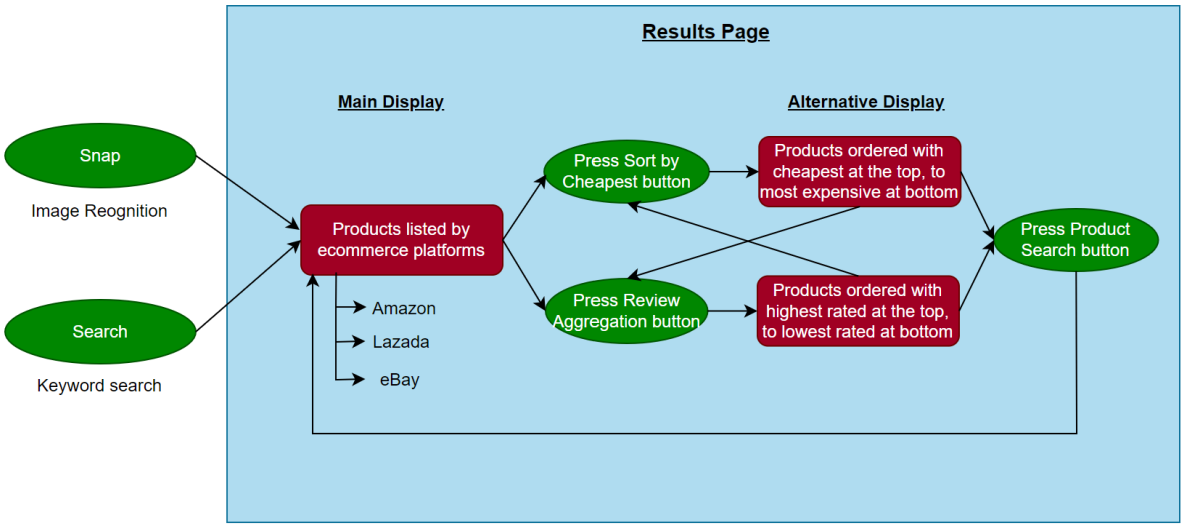
Another challenge when implementing the reviews page was to pass product data between different pages. In React Native, passing large amounts of data between pages using parameters is not feasible due to limitations in data handling and performance constraints. Consequently, I had to adopt a more efficient approach by storing the product data in a Firestore

database. This way, when a user navigates to the reviews page, the app can fetch the relevant product data directly from Firestore. This method ensures that data transfer remains efficient and seamless, while also providing a more scalable and maintainable solution for managing and accessing large datasets within the app.

Limitations

The API to retrieve reviews for a product after pressing the “Reviews” button only works for Amazon products and not Lazada nor eBay products as only the Amazon API offers such a functionality. Thus, reviews for Lazada or eBay products will not be displayable. However, users can still view reviews of the products by clicking on the “Buy now!” button and viewing the reviews on the ecommerce website itself.

Summary



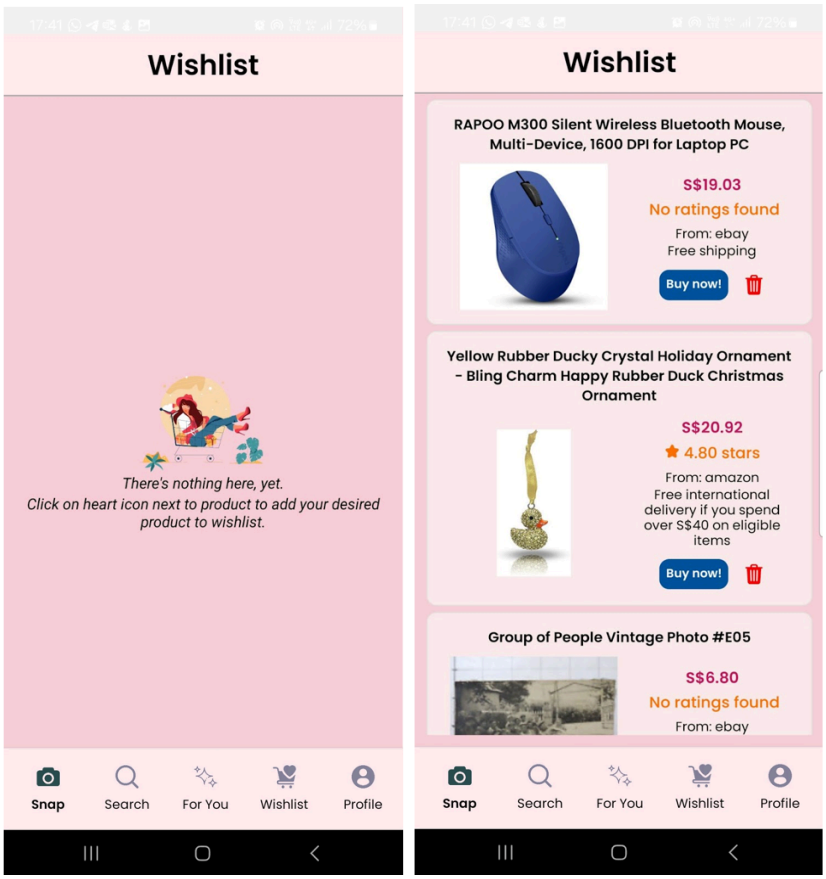
Results page navigation flow

The results page can be classified into a main display which is the default that is rendered after users either upload a photo or search for a product by keyword, and two alternative displays which are rendered upon pressing “Sort by cheapest” and “Review Aggregation buttons” at the top of the pages. Pressing these buttons again or pressing the “Product Search” button can allow users to toggle between the different displays based on user preferences.

2.7 Wishlist

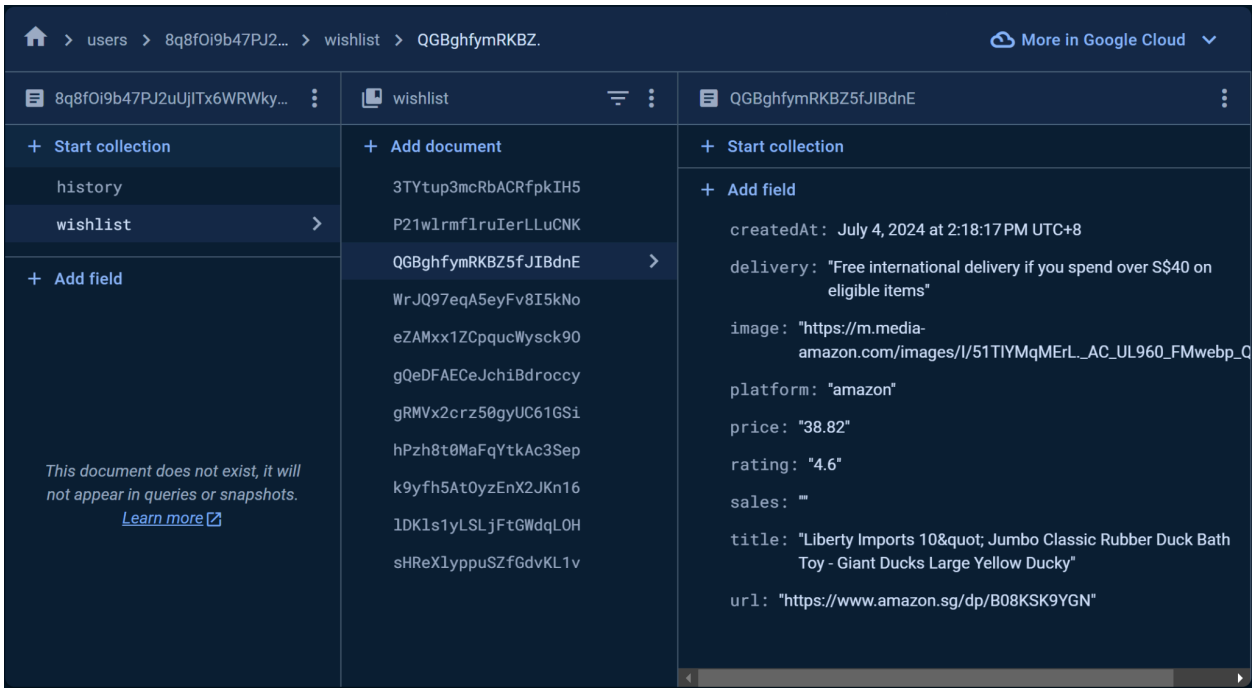
Description

Our app has a wishlist that allows users to keep track of all the products they are interested in so that they can easily come back to them again in the future if they wish to purchase the products. When users press on the heart icon for a product that they are interested in, it will be saved to the wishlist and can be retrieved again easily by going to the wishlist tab. If there is nothing in the wishlist, a basic interface is rendered to let the user know that there is nothing in the wishlist currently and that they can add items to the wishlist by clicking on the heart icon next to a product.



Implementation design

Our app features a comprehensive wishlist functionality powered by Cloud Firestore. Each user has a dedicated wishlist collection within the database, structured hierarchically as follows: users → userId → wishlist → wishlist item.



Wishlist items stored in Firestore

Users can easily add products to their wishlist by clicking on the like button associated with each product. This action creates a new document in the user's wishlist collection containing detailed information about the item, such as the title, price, image, rating, URL, sales volume, delivery info, and platform.

The app includes a dedicated wishlist page that displays all the products a user has added to their wishlist. When the wishlist page is rendered, it queries for the wishlist collection of the user in Cloud Firestore, and also sorts the items in descending order by the time which they were added to the wishlist, thus allowing the latest items to be displayed first.

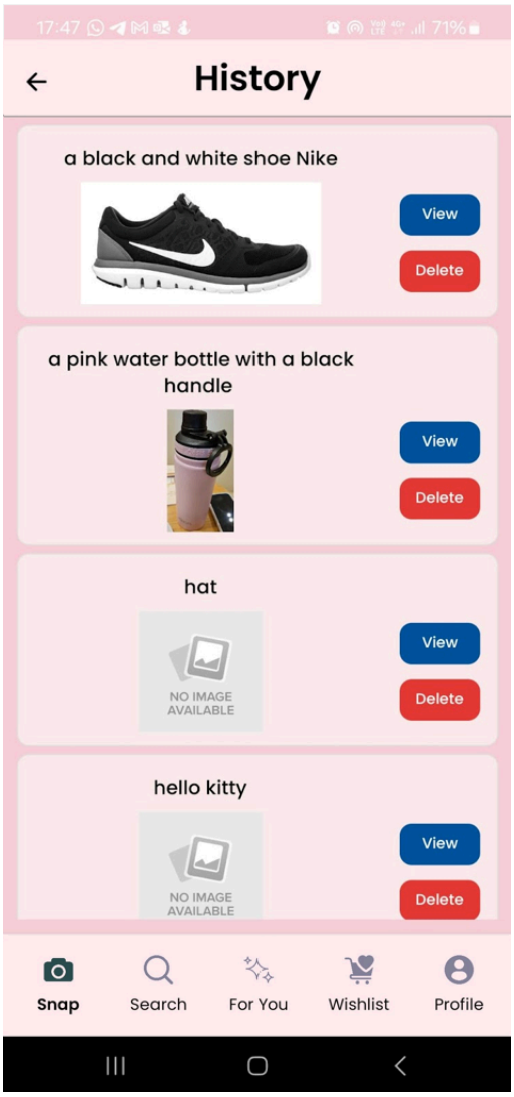
Additionally, each item in the wishlist has a delete button, allowing users to remove items from their wishlist easily. This is achieved by removing the document associated with the wishlist item from the database. When an item is deleted, the wishlist page automatically refreshes to show the updated wishlist, ensuring that users always have an up-to-date view of their favourite items

Overall, this feature enhances the shopping experience by allowing users to keep track of items they are interested in, facilitating easy access and management of their potential purchases.

2.8 Search history

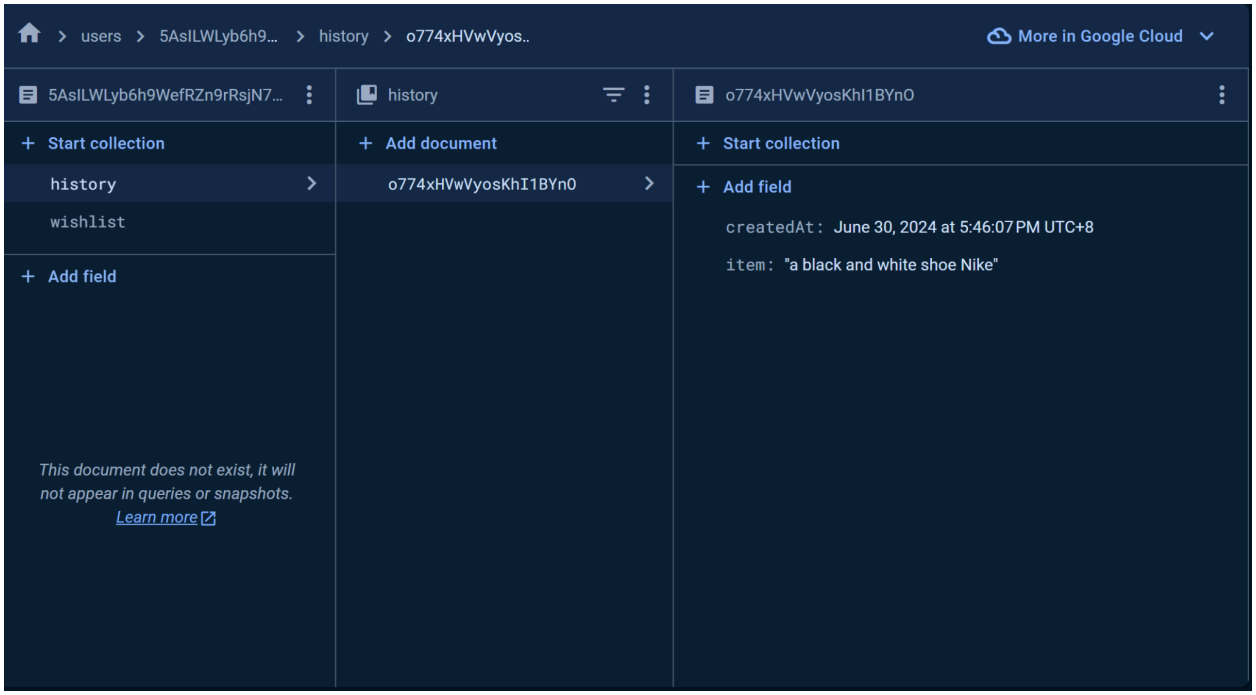
Description

SnapShop keeps a record of user searches under the search history segment of the profile tab so that users can easily track their past searches. This can be highly useful in the event that there is a product that they were interested in but forgot to add to their wishlist or if they simply just want to have a look at the most recent products that they have browsed.



Implementation design

Our app includes a robust search history feature, leveraging Cloud Firestore to keep track of users' search activities. Each user has a dedicated history collection within the database, structured as follows: users → userId → history → searched item.



Item added to user search history in Firestore

When a user uploads an image to search for relevant products, the detected item name is stored in a new document within their history collection. Additionally, the user-uploaded image is uploaded to Google Cloud Storage with a filename formatted as userId/docId, where docId refers to the corresponding document in the history collection. This ensures that each image is uniquely identified and easily retrievable. In the event where the user searched for a product by keyword instead of uploading a photo, the default “No image available” photo is used instead.

Furthermore, the app features a search history page that displays all previously searched items, including their names and associated images. The item name is obtained by querying for the history collection in the Cloud Firestore, while the item image is obtained by downloading the corresponding image file from Google Cloud Storage. The page also sorts the items in descending order by the time which they were added to the history, thus allowing the latest items that were searched to be displayed first.

Each item in the search history also has a delete button, allowing users to remove items from their history easily. This is achieved by removing the document and image associated with the item from the database. When an item is deleted, the history page automatically refreshes to show the updated history, ensuring that users have an up-to-date view of their history. A view button is also included to allow users to view their previous search result

This functionality allows users to conveniently review their past searches, helping them quickly revisit and manage items they have searched for, thus enhancing the overall user experience.

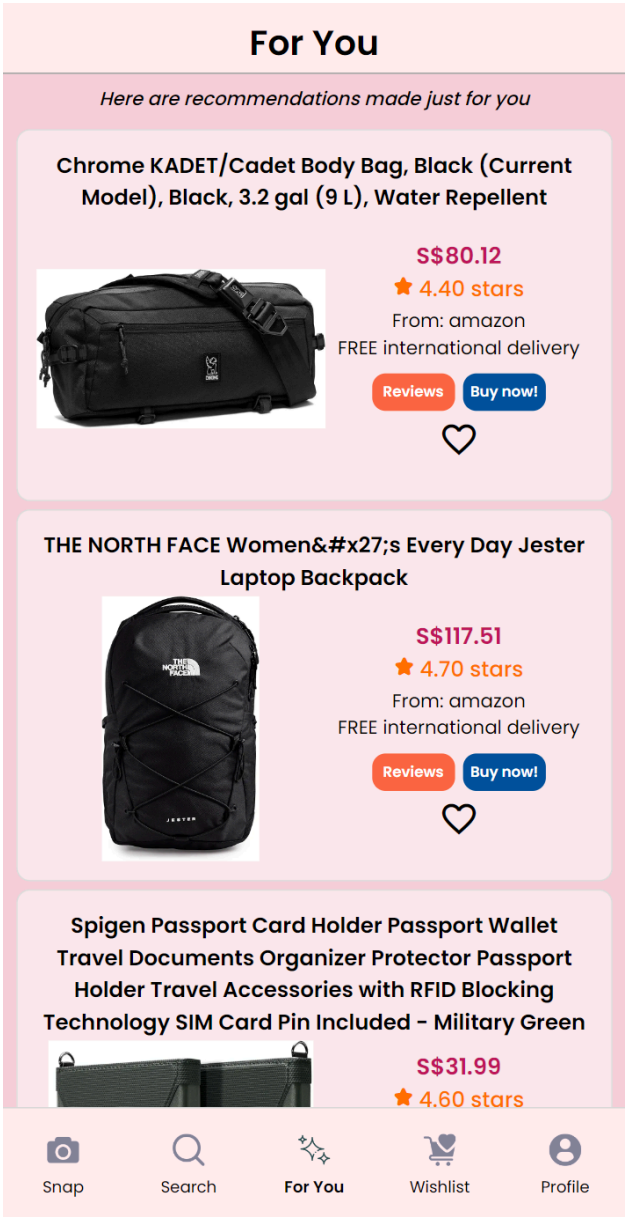
Challenges

We were unable to store images in Cloud Firestore. Thus, we had to create a Google Cloud storage to store our image and tag them to the corresponding document in Cloud Firestore. As such, the image and item name are being stored in separate databases.

2.9 User Recommendations

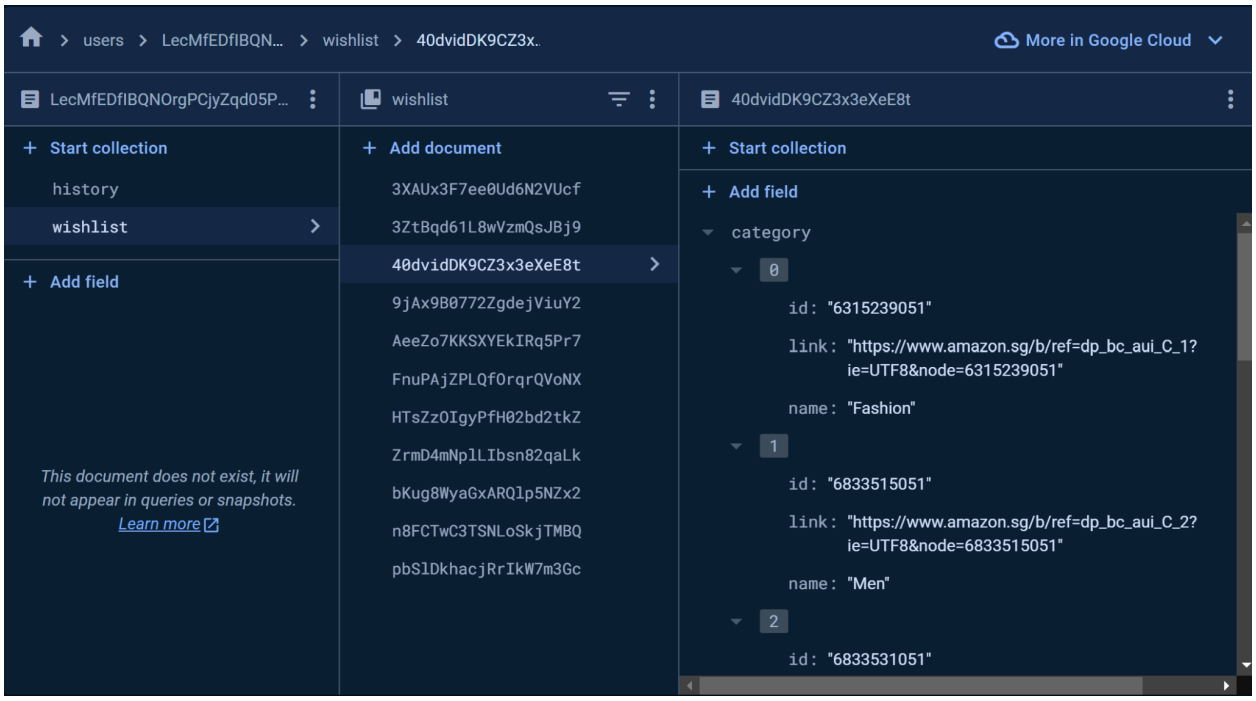
Description

SnapShop comes with a For You page to display product recommendations based on the items that they added to their wishlist, helping users find similar products that may be of interest to them. Similar to price comparison and review aggregation, users can scroll down horizontally to browse products, press the load more button to load more products and press the upward arrow button to go back to the top of the page.



Implementation design

Every time a product is added to the wishlist, the category associated with the product is saved to Firestore as well.



Product category saved to Firestore

When users navigate to the For You page, the product categories of items in the wishlist are retrieved from Firestore and the most popular category is determined based on the number of occurrences of that category amongst all items. The most popular category is then sent to Amazon API to retrieve products of similar category, which are then displayed on the For You page.

Limitations

The For You page algorithm will only consider Amazon products and recommends products only from Amazon as the API functionality to obtain product category and search products by category are only offered by the Amazon API. Thus, if users added Lazada or eBay products to their wishlist, those products will not be counted by the algorithm to generate the list of recommended products in the For You page. As a result, the For You page may not be fully representative of users' product interests.

3. Tech Stack

Frontend mobile app: React Native + Expo

Backend Server: FastAPI (for API queries) + Firebase (For authentication & database)

Image recognition & Object Detection: Microsoft Azure AI vision

E-commerce platforms API: Amazon, Lazada, Ebay

Version Control System: Github

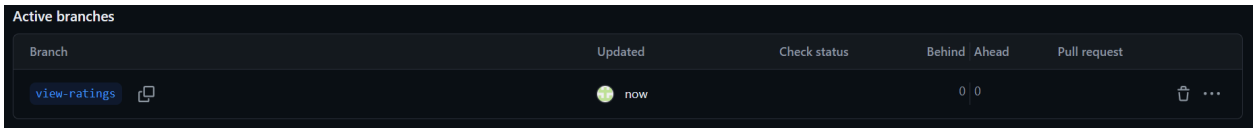
Development Tools: Visual Studio Code

Programming languages: JavaScript, TypeScript, Python

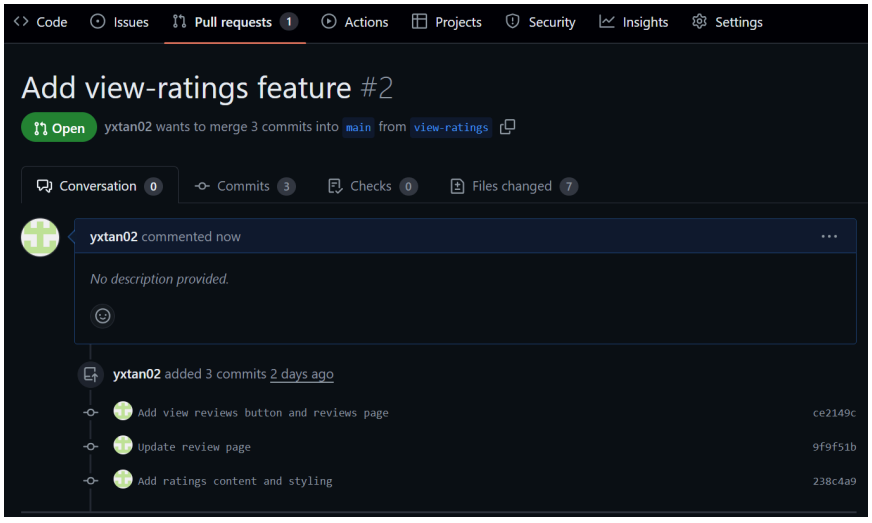
4. Software Engineering Practices

4.1 Version Control

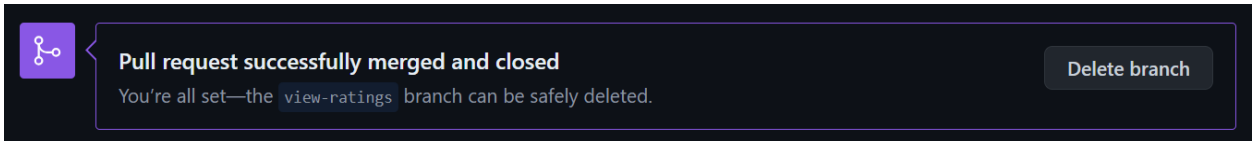
We use Git for version control and GitHub for collaboration, enabling us to track changes to the codebase, work together efficiently, and revert to previous versions when necessary. To prevent the cross-contamination of code, our codebase is managed using separate branches: a main branch that contains the working version of the app and individual feature branches. To develop new features, we pull from the master branch and create a new feature branch. Upon completion of the new feature, a git commit is made to the side feature branch, and a pull request is created to merge the new changes into the master branch.



A separate branch is created when working on a new feature



A pull request is made after the feature has been completed

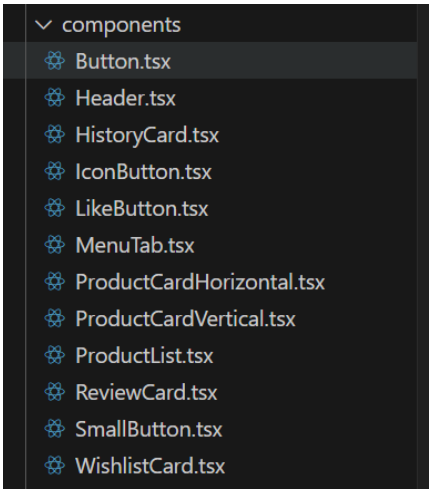


New changes are successfully merged into the main branch

4.2 DRY (Don't Repeat Yourself)

This principle involves extracting common functionalities into reusable functions, classes, or modules to promote code reusability, reduce redundancy, and facilitate easier maintenance. One of the most effective ways to apply the DRY principle is by creating reusable components in React. Thus, we abstracted many common functionalities, such as buttons, input fields, headers and cards, into individual components that can be used across multiple parts of our app, thus reducing code duplication and ensuring consistency.

For instance, a button component can be defined once and then utilised wherever needed by importing the custom component. This not only minimises redundancy but also simplifies updates, as any change to the component's logic or styling needs to be made in a single location. Consequently, the codebase remains cleaner, more modular, and easier to manage, exemplifying the efficiency and maintainability that the DRY principle aims to achieve.

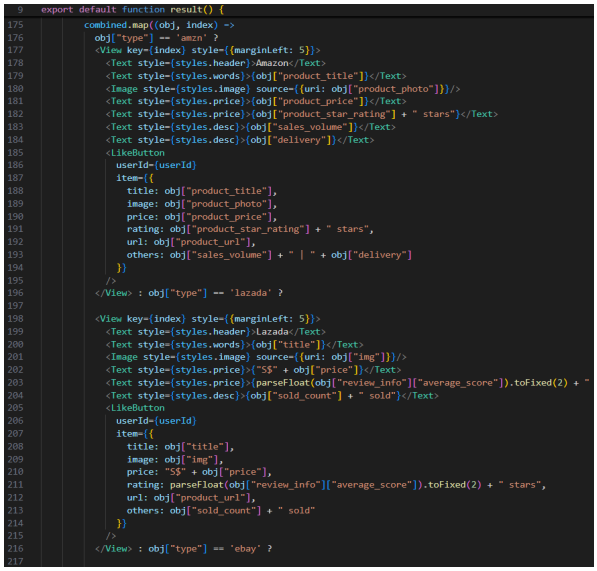


A list of reusable React components created for the app

4.3 Code Refactoring

Code refactoring is a critical practice in my project aimed at improving the internal structure of the code without altering its external behaviour. By systematically revising and optimising the codebase, refactoring enhances readability, maintainability, and performance. This process involves simplifying complex code, removing redundancies, and ensuring adherence to coding standards and best practices. Refactoring also helps in identifying and fixing potential bugs, thereby increasing the overall stability and robustness of the application. Additionally, it makes the code more modular, facilitating easier integration of new features and modifications. Ultimately, regular code refactoring ensures that the project remains scalable, efficient, and adaptable to future requirements and technological advancements.

An example of code refactoring



```
export default function result() {
        combined.map((obj, index) =>
          obj["type"] == 'amzn' ?
          <View key={index} style={{marginLeft: 5}}>
            <Text style={styles.header}>Amazon</Text>
            <Text style={styles.words}>{obj["product_title"]}</Text>
            <Image style={styles.image} source={{uri: obj["product_photo"]}}/>
            <Text style={styles.price}>{obj["product_price"]}</Text>
            <Text style={styles.price}>{obj["product_star_rating"] + " stars"}</Text>
            <Text style={styles.desc}>{obj["sales_volume"]}</Text>
            <Text style={styles.desc}>{obj["delivery"]}</Text>
            <LikeButton
              userId={userId}
              item={{
                title: obj["product_title"],
                image: obj["product_photo"],
                price: obj["product_price"],
                rating: obj["product_star_rating"] + " stars",
                url: obj["product_url"],
                others: obj["sales_volume"] + " | " + obj["delivery"]
              }}
            />
          </View> : obj["type"] == 'lazada' ?

          <View key={index} style={{marginLeft: 5}}>
            <Text style={styles.header}>Lazada</Text>
            <Text style={styles.words}>{obj["title"]}</Text>
            <Image style={styles.image} source={{uri: obj["img"]}}/>
            <Text style={styles.price}>{"S$" + obj["price"]}</Text>
            <Text style={styles.price}>{parseFloat(obj["review_info"]["average_score"]).toFixed(2) + "
            <Text style={styles.desc}>{obj["sold_count"] + " sold"}</Text>
            <LikeButton
              userId={userId}
              item={{
                title: obj["title"],
                image: obj["img"],
                price: "S$" + obj["price"],
                rating: parseFloat(obj["review_info"]["average_score"]).toFixed(2) + " stars",
                url: obj["product_url"],
                others: obj["sold_count"] + " sold"
              }}
            />
          </View> : obj["type"] == 'ebay' ?
```



```
<View style={styles.normalResultContainer}>
  <ProductList title="Amazon" data={amazon} userId={userId} />
  <ProductList title="Lazada" data={lazada} userId={userId} />
  <ProductList title="eBay" data={ebay} userId={userId} />
</View>
```

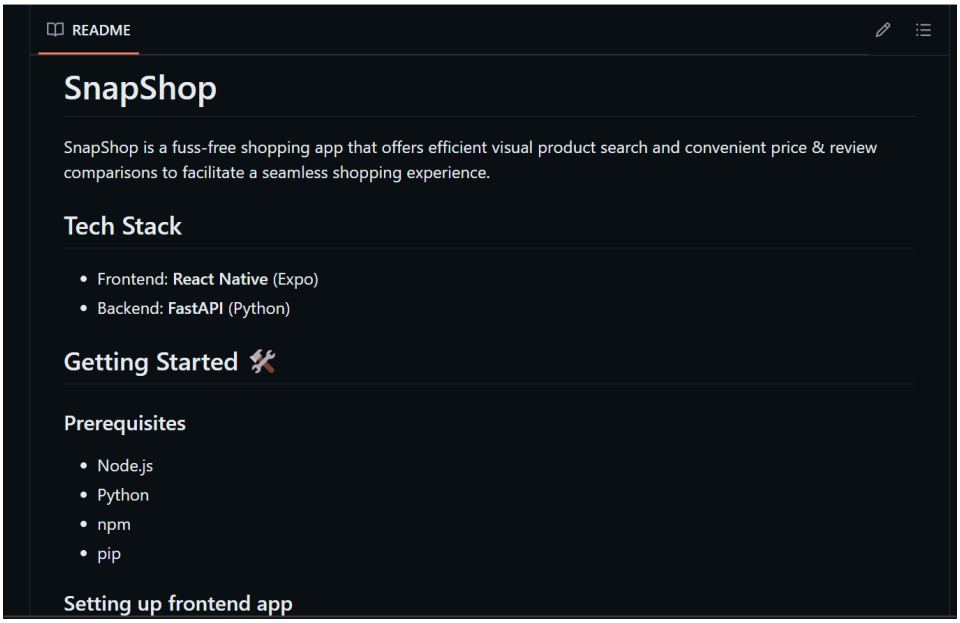
Before refactoring

After refactoring

4.4 Documentation

We will maintain thorough documentation for the project, including technical documentation for the codebase and system architecture, as well as user documentation for the app. This serves as an effective avenue to ensure clarity and ease of use for both developers and end-users.

The README file provides a concise overview, including the project's purpose, installation steps, and basic usage instructions, making it easy for anyone to get started quickly. For those needing more comprehensive information, a detailed README word document is available. This document delves into technical specifics, offering in-depth explanations of the project's architecture, detailed user guides, API references, and implementation details. Together, these documentation resources create a robust framework for understanding and effectively utilising the project, facilitating smoother onboarding, development, and maintenance processes.

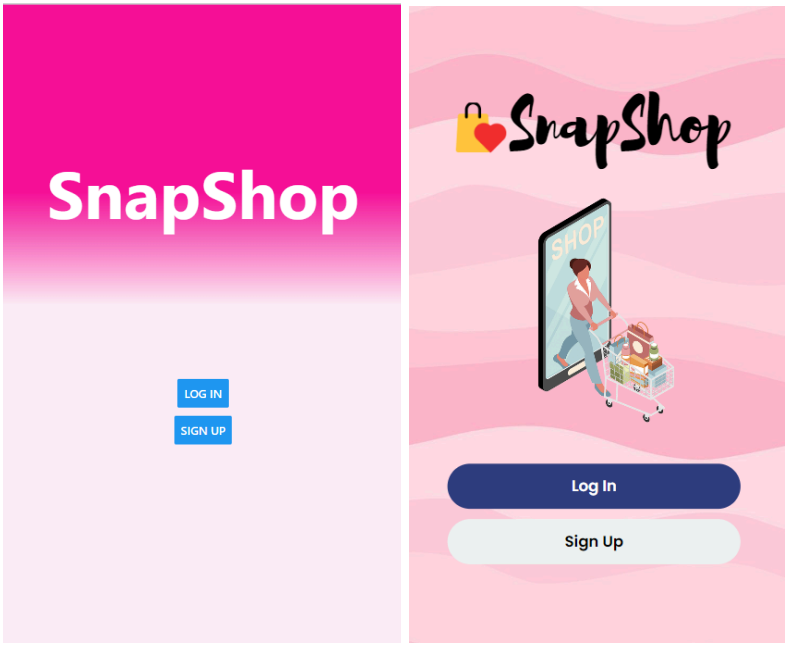


Screenshot of our GitHub README

4.5 User-Centred Design (UCD)

We proactively adopt a User-Centred Design (UCD) approach to ensure the app's user interface and user experience align with the needs and preferences of our target users. UCD involves understanding user needs through research and feedback, and designing the app to meet those needs effectively. For example, we redesigned the home page after user feedback indicated it was unappealing and the buttons were too small. By enhancing the visual appeal with a more modern design and increasing the size of the buttons for better accessibility, we created a more engaging and user-friendly interface. This iterative process of gathering insights, designing, testing, and refining will continue throughout the development cycle, ensuring that the app remains aligned with user expectations and delivers a satisfying user experience.

An example of user-centred design



Before *After*

4.6 Security measures

To strengthen the security of our app, we store our private API keys in a .env file instead of hard-coding these keys into our source code. This .env file is included in .gitignore to ensure it is not uploaded to our remote GitHub repository during commits. This prevents unauthorised access to sensitive API keys, thus reducing the risk of exploitation by malicious actors

4.7 Agile Development Methodology

We will follow an Agile approach, breaking down the development process into small, manageable iterations. This will allow us to continuously deliver working software and iterate based on feedback. The project will be divided into multiple 2-week sprints. At the start of each sprint, we will meet to plan and prioritise tasks, ensuring they are manageable within the two-week timeframe. We will then work collaboratively on these tasks, focusing on delivering functional components of the software. At the end of the two weeks, we will reconvene to review our progress, demonstrate the completed work, and gather feedback. This retrospective meeting will allow us to assess what went well, what challenges we encountered, and what improvements can be made. Based on this review, we will make necessary adjustments to our plans and processes for the next iteration. This cyclical process ensures that we remain flexible and responsive to changes, continuously improving our product and processes based on regular feedback and reflection.

4.8 Separation of Concerns

Separation of concerns involves dividing the application into distinct layers or modules, each responsible for a specific aspect of functionality. Separate concerns such as presentation (frontend), business logic (backend), and data access (database) to improve code readability, maintainability, and testability.

In our project, this principle is demonstrated through a clear division of responsibilities between different components of the system.

The React Native frontend is dedicated to the presentation layer, handling user interactions and displaying data in a user-friendly manner. This separation ensures that the user interface remains responsive and easy to update independently of the backend logic.

The FastAPI backend serves as the business logic layer, processing requests from the frontend, performing necessary computations, and making queries to external APIs. This modularity allows for efficient handling of business rules and logic without cluttering the frontend code, thus maintaining a clean separation between presentation and logic. Additionally, it simplifies the process of updating and scaling individual components without impacting the overall system.

Finally, the Firebase database is used for data access, storing crucial information about products and users. By isolating data storage and retrieval in the database layer, we ensure that the data access logic is decoupled from both the frontend and backend, enhancing maintainability and testability. This structured approach, dividing the application into frontend, backend, and database layers, exemplifies the principle of separation of concerns, resulting in a well-organised, efficient, and scalable system.

4.9 Model-View-Controller software design

Model (M) - Data Management

- Define models to represent the data entities in the app, such as User, Product, Wishlist, etc.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to manage the data effectively.

View (V) - User Interface:

- Design user interfaces (UI) for the app based on user-centred design principles.
- Implement views to display data to users and capture user inputs.

Controller (C) - Application Logic:

- Implement controllers to handle user requests and business logic.
- In controllers, interact with models to perform CRUD operations and return appropriate responses to the views.

4.10 CRUD operations

- **Create:** Implement functionality to create new records (e.g., add a new product to the database, create a new wishlist item, create a new user account).
- **Read:** Implement functionality to retrieve and display data (e.g., fetch product details, display wishlist items, fetch user id).
- **Update:** Implement functionality to update existing records (e.g., edit product information, update wishlist items).
- **Delete:** Implement functionality to delete records (e.g., remove a product from the database, delete a wishlist item).

5. Testing

5.1 Unit Testing

Our unit testing focuses on the individual components of our app to ensure that they work correctly in isolation.

Feature 1: User Account & Authentication				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Sign-up	Verify that users can successfully sign up by providing a username, email, and password	1. Navigate to the sign-up page. 2. Enter a valid username, email, and password. 3. Click the "Sign Up" button.	- The user is successfully signed up and redirected to the app's login page. - A new user entry is created in Firebase. - The user can access all app features after signing up.	Pass
Login with email and password	Ensure that existing users can log in using their email and password	1. Navigate to the login page. 2. Enter a valid email and password. 3. Click the "Log In" button.	- The user is successfully logged in and redirected to the app's main page. - The user can access all app features after logging in.	Pass
Login with google account from Login page	Verify that users can sign in using Google third party sign in from Login page	1. Navigate to login page. 2. Click on “Sign in with Google Button”. 3. Select a google account	-The user is successfully logged in and redirected to the app’s main page - The user can access all app features after logging in. - The user’s name and email displayed on the profile page are in accordance with their google email and name.	Pass
Login with google account from Sign-up page	Verify that users can sign in using Google third party sign in from Sign-up page	1. Navigate to Sign Up page. 2. Click on “Sign in with Google Button”. 3. Select a google account	-The user is successfully logged in and redirected to the app’s main page - The user can access all app features after logging in. - The user’s name and email displayed on the profile page are in accordance with their	Pass

			google email and name.	
Invalid Login	Verify that users cannot log in with invalid credentials	1. Navigate to the login page. 2. Enter an invalid email and/or password. 3. Click the "Log In" button.	- The user is not logged in. - An error message is displayed, indicating an incorrect email or password.	Pass
Access Control	Ensure that unauthenticated users are redirected to the login page when attempting to access the app	Attempt to access the app's main page or other features without being logged in.	- The user is redirected to the login page. - The user cannot access any app features without logging in.	Pass
Logout	Ensure that users can log out successfully	1. Log in with valid credentials. 3. Click the "Log Out" button.	- The user is logged out and redirected to the login page. - The user cannot access app features without logging in again.	Pass

Feature 2: Image recognition				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Image capture	Verify that users can take a photo using the app	Click on the “Take a photo” button and use the device’s camera to take a photo	The app captures the photo successfully and displays it for further actions.	Pass
Image upload	Verify that users can upload a photo from their device's gallery	Click on the “Upload a photo” button. Select a photo from the gallery confirm the selection	The app opens the device's gallery, allowing the user to select a photo. The selected photo is uploaded to the app	Pass
Image editing	Verify that users can crop the photo taken	After taking or uploading a photo, use the cropping tool to select a specific area of the photo.	The cropping tool allows precise selection and the cropped image is saved for analysis.	Pass
Object detection &	Test that the app can accurately	Upload an image of a running shoe	The app accurately identifies the item as a	Pass

recognition	recognize and identify objects in user-uploaded photos.		running shoe	
Brand detection & recognition	Test that the app can detect and recognize brands in user-uploaded photos	Upload an image of a Nike running shoe	The app accurately identifies the brand as Nike	Pass

Feature 3: Search by keyword				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Keyword search	Verify that search input is correctly passed to ecommerce APIs	Typed “black Nike running shoe” in the text box.	- Results page shows that the product searched for is indeed “black Nike running shoe” at the top of the page. - Products displayed are black running shoes	Pass

Feature 4: Product Search				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Amazon search	Verify that the app can search for products on Amazon and retrieve relevant results	1. Upload an image of a specific product (e.g., "Nike running shoes") 2. Send a search query to the Amazon API	The app sends a search query to Amazon API and retrieves a list of relevant products. Each product includes the title, price, image, rating, and URL	Pass
Lazada search	Verify that the app can search for products on Lazada and retrieve relevant results	1. Upload an image of a specific product (e.g., "Nike running shoes") 2. Send a search query to the Lazada API	The app sends a search query to Lazada API and retrieves a list of relevant products. Each product from Lazada includes the title, price, image, rating, and UR	Pass
eBay search	Verify that the app can search for products on eBay and retrieve relevant results	1. Upload an image of a specific product (e.g., "Nike running shoes") 2. Send a search query to the eBay API	The app sends a search query to eBay and retrieves a list of relevant products. Each product from eBay includes the title, price, image, rating, and URL.	Pass
Buy now button	Test that the “Buy now!” button is working	Click on the "Buy Now" button for a specific product	The user is redirected to the correct product purchase page.	Pass
Reviews button	Test that the “Reviews” button is working	Click on the “Reviews” button for an Amazon product	The user is redirected to the reviews page where the reviews of the product are displayed.	Pass
Error handling	Verify that the app renders correctly when no search results are found	1. Disable queries to ecommerce API 2. View the results page	The results page displays “No results found”.	Pass

Feature 5: Price Comparison				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Data Accuracy	Verify that prices obtained via ecommerce APIs are accurate	Manually compare prices returned by the API and the actual prices displayed on the website	The prices in the app should match the prices listed on the ecommerce site	Pass
Price sorting	Test that the products can be sorted by price in ascending offer	Click on the “Sort by cheapest” button	The app sorts the products by price in ascending order, displaying the cheapest items first.	Pass
Error handling	Verify that missing or null price data are handled appropriately	Perform a product search where some items have missing or null price data	Items with missing or null prices are labelled as "Invalid price" and subsequently filtered out from the final displayed list	Pass

Feature 6: Review Aggregation				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Data Accuracy	Verify that ratings obtained via ecommerce APIs are accurate	Manually compare ratings returned by the API and the actual ratings displayed on the website	The ratings in the app should match the ratings listed on the ecommerce site	Pass
Review ratings sorting	Test that the products can be sorted by rating in descending offer	Click on the “Review Aggregation” button	The app sorts the products by rating in descending order, displaying the highest rated items first.	Pass
Error handling	Verify that missing or null rating data are handled appropriately	Perform a product search where some items have missing or null rating data	Items with missing or null ratings are labelled as "No ratings found" and subsequently filtered out from the final displayed list	Pass

Feature 7: Wishlist				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Add to wishlist	Verify that users can add a product to their wishlist	1. Click the heart icon on a product listing to add it to the wishlist	- The product is added to the user's wishlist collection in Cloud Firestore. - A new document is created in the user's wishlist collection with the product's details	Pass
Undo adding to wishlist	Verify that users can undo their adding of a product to wishlist	1. Click the heart icon on a product listing to add it to the wishlist 2. Click on the heart icon on the same product 3. Navigate to wishlist page and check if the product is no longer in wishlist	- The product is removed from the user's wishlist collection in Cloud Firestore. - The wishlist page automatically refreshes to show the updated list without the deleted product.	pass
View wishlist	Ensure that users can view all items in their wishlist	1. Navigate to the wishlist tab.	- The wishlist tab displays all products added to the wishlist. - The products are displayed in descending order by the time they were added, with the latest items shown first.	Pass
Empty wishlist	Verify that a design is rendered when there are no items in wishlist	1. Navigate to the wishlist tab 2. Ensure no items in wishlist	-The wishlist tab displays an image of a women in a trolley, with the words “Nothing here, yet. Click on heart icon next to product to add your desired product to wishlist”	Pass
Remove from wishlist	Verify that users can remove a product from their wishlist	1. Navigate to the wishlist tab. 2. Click the delete button on a product in the wishlist.	- The product is removed from the user's wishlist collection in Cloud Firestore. - The wishlist page automatically refreshes to show the updated list	Pass

			without the deleted product.	
Auto-refresh	Ensure that the wishlist page automatically refreshes after an item is added	1. Add a product to the wishlist. 2. Navigate to the wishlist tab to view the added product.	- The wishlist page refreshes automatically after adding an item. - The added product appears immediately after being added	Pass
Auto-refresh	Ensure that the wishlist page automatically refreshes after an item is removed	1. Remove the product from the wishlist.	- The wishlist page refreshes automatically after removing an item. - The removed product disappears immediately after being removed.	Pass

Feature 8: Search History				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Add to search history	Verify that a user's search is correctly added to their search history	1. Upload an image and perform a product search. 2. Navigate to the search history page.	- The searched item is added to the user's history collection in Cloud Firestore. - A new document is created in the user's history collection with the detected item name and timestamp. - The uploaded image is stored in Google Cloud Storage with the correct filename format.	Pass
View search history	Ensure that users can view all items in their search history	1. Navigate to the search history page.	- The search history page displays all previously searched items, including names and associated images. - The items are displayed in descending order by the time they were added, with the latest items shown first.	Pass
Remove from search	Verify that users can remove a	1. Navigate to the search history	- The search item is removed from the user's	Pass

history	search item from their history.	page. 2. Click the delete button on a search item.	history collection in Cloud Firestore. - The corresponding image is deleted from Google Cloud Storage. - The search history page automatically refreshes to show the updated list without the deleted item.	
View button	Ensure that users can view each individual search history item	1. Navigate to the search history page. 2.Click the view button on a search item.	- The app navigates to the search results page, displaying the results of the previous search. - The user can review the product details associated with the past search.	Pass

Feature 9: Load More button				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Press load more button	Ensure more items are displayed after load more button is pressed	1. Upload photo of a nike running shoe and navigate to results page 2. Press “sort by cheapest 3. Scroll down all the way 4. Press load more	More items are displayed without removing the items that are initially on display	Pass
Hiding of load more button	Ensure that the load more button is no longer displayed once all items are shown	1. Upload photo of a nike running shoe and navigate to results page 2. Press “sort by cheapest 3. Scroll down all the way 4. Press load more 5. Repeat steps 3 and 4 until no more new products	Load More button disappears once no more new items are shown	Pass

Feature 10: To the top button				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Press To the top button	Ensure that the screen scrolls back to the top upon clicking the button	1. Upload photo of a nike running shoe and navigate to results page 2. Press “sort by cheapest 3. Scroll down all the way 4. Press To the top button	The screen scrolls back to the top of the page	Pass
Hiding of To the top button	Ensure that the To the top button is hidden from view when user is already at the top of the screen	1. Upload photo of a nike running shoe and navigate to results page 2. Scroll down and back up	The To the top button is not shown at the start. Upon scrolling down, the button appears. After scrolling back up again, the button disappears.	Pass

Feature 11: Product Recommendations				
Component / Function	**Testing Objective**	**Steps taken**	**Expected Results**	**Pass /Fail**
Recommendation Algorithm	Verify that the recommendation algorithm returns the correct list of recommended products based on a sample wishlist.	1. Add a product (Nike shoe) into the wishlist 2. Check and verify recommendations displayed on the for you page	Recommendations related to sports and footwear should be displayed	Pass
Error handling	Verify that the recommendation algorithm handles an empty wishlist correctly.	1. Create a new account 2. Click on the for you page	No recommendations would be displayed as the wishlist is empty	Pass

5.2 Integration Testing

Our integration testing checks the interaction between different components to ensure that they work together as expected.

Feature 1: User Account and Authentication			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Verify that the user **sign-up** and **sign-in** components work together seamlessly.	1. Sign up with a new user account. 2. Log out (if automatically logged in). 3. Log in with the newly created account credentials.	1. Successful account creation. 2. Successful logout. 3. Successful login with the new account credential	Pass
Integration between **account creation** and **access control**: Ensure that only authenticated users can access the app's features.	1. Attempt to access app features without logging in. 2. Sign up and log in with a new account. 3. Attempt to access app features after logging in.	1. Redirection to the login page when trying to access features without logging in. 2. Successful account creation and login. 3. Access to app features after logging in.	Pass
Integration between **log-out** function and **access control**	1. Log in with valid credentials. 2. Click on the logout button. 3. Attempt to access a restricted feature after logging out.	1. Successful login 2. Successful logout and redirect to the login page. 3. Redirection to the login page when attempting to access restricted features after logging out.	Pass

Feature 2: Image recognition			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Test that the app can accurately **identify both the object and brand** in tandem	1. Upload an image with a recognisable brand (eg. Nike running shoes)	- Successful detection of object -> running shoes - Successful detection of brand -> Nike	Pass
Verify that a **photo taken** by a user that is **cropped** is **accurately recognised**	1. Capture a photo using the camera and crop the photo 2. Check the detected item returned by the app	1. Successful photo capture and photo is sent to the API 2. Correct recognition of the item in the photo.	Pass
Verify that a **photo uploaded** by a user that is **cropped** is **accurately recognised**	1. Upload an photo and crop the photo 3. Check the detected item returned by the app	1. Successful photo upload and photo is sent to the API 2. Correct recognition of the item in the photo.	Pass

Feature 3: Keyword Search			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure **seamless transition** from snap result to keyword search	1. Upload a photo of a black Nike running shoe and ensure that the results are shown on the results page. 2. Navigate to the “Search” page. 3. Type “Hello Kitty” in the search box and press the search icon.	- Results page shows that the product searched for is indeed “Hello Kitty” at the top of the page. - Products displayed are Hello Kitty related products.	Pass

Feature 4: Product Search			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Test that the app is able to fetch data from all 3 platforms simultaneously	1. Upload an image of a specific product (e.g., "Nike running shoes") 2. Check the product listings returned	1. Successful item recognition. 2. Search results retrieved from Amazon, Lazada and eBay	Pass
Verify that the app's user interface (UI) can display product details accurately and in a user-friendly manner.	1. Upload an image to perform a visual product search 2. View the results	The app's UI displays a list of search results with each product's details clearly visible, including the title, price, image, rating, and URL. The product details are well-aligned and presented in a consistent layout,	Pass

Feature 5: Price Comparison			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Verify that the app can handle a mix of valid and invalid prices retrieved from ecommerce APIs	1. Upload an image to perform a visual product search 2. View the results in the price comparison tab	- Valid prices are displayed correctly in their respective currency format. - Invalid prices are correctly filtered out	Pass

Feature 6: Review Aggregation			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Verify that the app can handle a mix of valid and invalid ratings retrieved from ecommerce APIs	1. Upload an image to perform a visual product search 2. View the results in the review aggregation tab	- Valid ratings are displayed correctly - Invalid ratings are correctly filtered out	Pass

Feature 7: Wishlist			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure that wishlist data is persisted across sessions.	1. Log in to the app. 2. Add a product to the wishlist. 3. Log out of the app. 4. Log in again. 5. Navigate to the wishlist tab.	- The previously added product is still present in the wishlist. - The data persists across sessions and is correctly retrieved from Cloud Firestore.	Pass
Ensure that only authenticated users can access and modify their wishlist	1. Attempt to access the wishlist without logging in. 2. Log in and access the wishlist.	Users must log in to access and modify their wishlist. Unauthorised access attempts are redirected to the login page.	Pass

Feature 8: Search History			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure that search history data is persisted across sessions.	1. Log in to the app. 2. Search for a product 3. Log out of the app. 4. Log in again. 5. Navigate to the search history tab.	- The previously searched product is still present in the history. - The data persists across sessions and is correctly retrieved from Cloud Firestore.	Pass
Ensure that only authenticated users can access and modify their search history	Attempt to access the search history without logging in. Log in and access the search history.	Users must log in to access and modify their search history. Unauthorised access attempts are redirected to the login page.	Pass

Feature 9: Load More button			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure that the Load More button loads additional items correctly and disappears once all items are displayed for all relevant pages: price comparison, review aggregation and For You page.	1. Upload a photo, wait for the results page 2. Press Sort by cheapest 3. Scroll down all the way 4. Press Load more 5. Repeat steps 3 and 4 until no more items 6. Press Review Aggregation 7. Repeat steps 3 and 4 until no more items 8. Navigate to For You page 9. Repeat steps 3 and 4 until no more items	For all 3 pages: -Initial set of items is displayed correctly. -Clicking the "Load More" button loads additional items and appends them to the list. -The "Load More" button continues to load additional items until all items are displayed. -Once all items are loaded, the "Load More" button disappears.	Pass

Feature 10: To the top button			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure that the To the top button scrolls the user back to the top of the page on click and only appears after users have scrolled down by about 1 screen length for the following pages: product search, price comparison, review aggregation and For You page.	1. Upload a photo, wait for the results page 2. Scroll down and up 3. Press Sort by cheapest 4. Scroll down and up 5. Press Review Aggregation 6. Scroll down and up 7. Navigate to For You page 8. Scroll down and up	For all 4 pages: -To the top button is not visible at the start -To the top button appears after scrolling about 1 screen length down -Pressing the button brings users back up to the top of the page	Pass

Feature 11: For You Page			
Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Check that recommendations are updated after a user adds a new item to the wishlist	1. Initialise the app with a user having a predefined wishlist. 2. Navigate to the recommendations page and fetch initial recommendations based on the current wishlist. 3. Add a new item to the user's wishlist. 4. Re-fetch the recommendations after the new item has been added. 5. Verify that the updated list of recommendations reflects the addition of the new item to the wishlist.	- Initially, the recommendations page should display a list of products based on the current wishlist. - After adding a new item to the wishlist, the recommendations should be updated to include new relevant products influenced by the addition.	Pass

5.3 System Testing

Our system testing validates if our app as a whole is able to meet the specified requirements, by mimicking real-world usage and covering end-to-end scenarios.

Testing Objective	**Steps taken**	**Expected Results**	**Pass/Fail**
Ensure that user **search history and wishlist is tied to the user account** for users that are logged in via email or google sign in	1. Log in to an account with email and password 2. Perform some searches by uploading a photo or searching by keyword 3. Add some products to wishlist by clicking on the heart icon next to product 4. Log out of account and log back in 5. Repeat steps 2-5 after signing in with Google	-User search history and wishlist remain intact after logging out and logging back in -User actions performed for an account are tied to that account	Pass
Ensure that **image recognition, keyword search** and **product search** works seamlessly together as intended	1. Upload a photo of a product. 2. Verify that the app recognizes the item and performs a product search across multiple platforms 3. Perform a keyword search. 4. Check that the product details are correctly displayed.	1. The photo is uploaded and recognized correctly. 2. The app performs a search across multiple platforms. 3. The product details are accurately displayed in the app.	Pass
Verify that the user can access product listings, price comparisons, and review aggregation all from the search results page	1. Perform product search by uploading an image 2. View the search results page	- The search results page should display a list of relevant products, with price comparison and review aggregation options available - Users should be able to view all this information directly from the search results page without navigating to other sections.	Pass
Ensure that user can access reviews	1. Perform product search	-Every product card in the search results page should	Pass

page, product website and save to wishlist for each product card in the product listings for product search, price comparisons and review aggregation	2. View the search results page 3. Click on “Reviews” button, “Buy now!” button and heart button for Amazon item 4. Click on Sort by cheapest 5. Repeat step 3 6. Click on Review Aggregation 7. Repeat step 3	contain a “Buy now!”, “Reviews” and heart button - Users should be able to click on each of the buttons and access their functionalities without navigating to other sections	
Test that a logged-in user can access all features of the app	1. Log in to the app 2. Perform product search by uploading an image 3. View the results 4. Add a product to wishlist and view wishlist page 6. View search history page	1. User should be able to log in successfully. 2. Image recognition and product search should function correctly, displaying relevant results. 3. The search results should include price comparisons and review aggregation. 4. The user should be able to add products to the wishlist and view them on the wishlist page. 5. The search history should be updated with recent searches and displayed correctly on the search history page.	Pass

5.4 User Testing

User testing was conducted by having end users interact with the app and provide feedback on its features and functionalities. This ensures that our app can meet the users' needs and expectations

User Story	Task	User Action	Feedback
As a user who values data privacy and security, I want to log in securely using my email and password or social media accounts so that I can access my saved data. I would want my account to save my preferences, search history and wishlist in a secure manner.	1. Guide the user to the sign up page. 2. Once signed in, guide the user to the wishlist tab, For You page and search history page.	1. User saw the "Sign in with Google" button and immediately clicked on it, ignoring the sign up with email and password text boxes. 2. User clicked on the wishlist page, profile page and search history tab from the profile tab.	-Convenient user sign in process with the Google Sign in button, saves time from not having to enter an email and come up with a password to create an account. (Good) -Good indication of name and email on profile page to make the account feel highly personalised. (Good) -Maybe can add option to customise profile photo. (Can be improved) -Account feels secure with Google third party authentication, user can trust that malicious users will not be able to access their account data. (Good)
As a user who values efficiency, I want to be able to take a photo of a desired item and find relevant products online so that I can browse and select the one I like. Furthermore, I want to be able to filter search results by price and rating, so that I can find the best product quickly.	1. Guide the user to take a photo of any item near them. 2. Once results page is loaded, guide the user to scroll through the products horizontally for each ecommerce platform 3. Guide user to click on Sort by cheapest 4. Guide user to click on Review Aggregation	1. User took a photo of a waterbottle on the table 2. User quickly scrolled through the products from each ecommerce platform 3. User clicked on Sort by cheapest and viewed the top few products. 4. User clicked on Review Aggregation and viewed the top few products.	-Nice, intuitive and clean user interface for product listings for each ecommerce platform, making it easy to scroll through the products. (Good) -Very easy to find the cheapest products by clicking on "Sort by cheapest" and highest rated products by clicking on "Review Aggregation". (Good) -Convenient To the top button that allows user to head back to the top of the page to see the cheapest

			product again after scrolling very far down. (Good) -Product listing becomes a bit laggy and harder to scroll after clicking on Load more button. (Can be improved) -Many irrelevant products that are not even close to the product in the photo uploaded. (Can be improved) -Popular ecommerce platforms like Shopee and Carousell are missing. (Can be improved)
As a user who values transparency and authenticity, I want to be able to view aggregated product reviews and ratings from multiple platforms, so I can make an informed decision based on the experiences of other shoppers before deciding if the product is worth buying.	1. Guide the user to click on the “Reviews” button next to an Amazon product 2. Guide the user to click on “Buy now!” button of any product	1. User clicked on the reviews button and scrolled through the reviews 2. User clicked on the “Buy now!” button and was redirected to the product website.	-Clean, neat and easy to read interface for product reviews all consolidated in one place so that user does not need to navigate outside the app to see reviews of the product and decide whether to purchase it. (Good) -Very useful redirecting to product website so that user can purchase the product or see more relevant information pertaining to the product that is not displayed on the app. (Good)
As a shopper who values convenience, I want to be able to create a wishlist of products I'm interested in, so I can easily keep track of them and make purchases when the time is right.	1. Guide user to click on heart button next to any product to add item to wishlist 2. Guide user to view products in wishlist	1. User clicked on the heart button of multiple products that piqued their interest. 2. Subsequently, user clicked on the wishlist tab to view the products in the wishlist.	-Easy and intuitive way of adding items to wishlist. (Good) -Very useful wishlist that sorts items from most recently added at the top so user can easily see their most recent preferences and can easily click on the “Buy now!” button to purchase the product. (Good)

As a user who values convenience and efficiency, I want to be able to view and manage my search history, so that I can quickly revisit previously searched items and refine my searches without starting from scratch.	1. Guide user to click on profile tab and search history	1. User clicked on search history and briefly browsed through the items. 2. User clicked on an item and was brought back to the results page of the item that was searched for recently.	-Convenient way to access past searches via search history tab so that user does not need to upload the same photo again and can be a lifesaver in the event where the user does not have access to the past photo. (Good) -Nice and handy to be able to keep track of past searches. (Good) -Can easily delete past searches with just a click. (Good)
As a user who values personalized shopping experiences, I want to see a product recommendations page based on my browsing and purchase history, so that I can discover new products that match my interests and preferences.	1. Guide user to click on For You tab	1. User clicked on For You tab and scrolled through all the products on the For You page	-Good idea of providing product recommendations so that user can easily find products that interest them. (Good) -Many irrelevant products that are not related to user past searches or wishlist. (Can be improved)

Link to GiHub Repo (for testing purposes)

https://github.com/yxtan02/snapshop

Project Video

https://drive.google.com/file/d/1kIs7_73rGlKJBKE0b_rxvcPXOOUeGOMJ/view?usp=drive_link

Project Log

https://docs.google.com/spreadsheets/d/1QApkTQxfbvRvXJmZoZg_c10PY6wDhpFavSEv-rTvQbI/edit?usp=drive_link

Project Poster

https://drive.google.com/file/d/1HJhedJ1gh4i5_QPf-POcYmOE5065xRkb/view?usp=drive_link

Link to android APK

https://expo.dev/artifacts/eas/p2vaKJz1kcrXqXEY4sBgrY.apk