StyleSwipe ReadMe

NUS Orbital 2024

Milestone 1

FinBros

Jonathan Ngien

Jaeho (Jose) Kim

Table of Contents

Introduction

Team name:

FinBros

Application name:

StyleSwipe

Proposed level of achievement:

Apollo 11

Summarized Project Scope/Aim

Our project aims to provide a Tinder-style shopping application where users swipe on items to denote their preferences.

Motivation

In the digital era, the fashion industry has increasingly turned to technology to enhance customer experiences. The sheer volume of clothing options available online can be overwhelming for consumers, making it difficult to find items that match their style. Recognising this, we aim to harness machine learning to streamline this process, offering a personalised shopping experience that adapts to user preferences over time, much like a digital stylist. Also, we noticed that thrifting online can be a hassle - with customers waiting hours for a drop, and competing with each other in a fastest-fingers-first race to get their hands on a piece of clothing through social media platforms like Telegram and Instagram, which don't provide a smooth customer experience.

Project Scope/Aim

Our project is to develop an AI-powered shopping application named "StyleSwipe", which operates like a clothing version of Tinder. Users will swipe left or right on clothing items, allowing the app to learn from these interactions and refine its suggestions accordingly. This will make shopping not only more efficient but also more enjoyable, transforming how users discover and select fashion items.

User Stories

- As a **fashion enthusiast** who enjoys discovering new trends, I want an app that learns my style preferences and continuously adapts its recommendations, so that I can find items that I love faster.
- As a **busy professional** with little time to shop, I want a quick and easy way to browse through potential purchases with help from a personalised recommendation system.
- As a **college student** with a tight budget and a limited clothes collection, I want a fun way to find cheap and stylish clothes to diversify my clothing.
- As a **thrift shop owner** with a lot of products to manage but no way to efficiently manage and advertise those products, I want a platform that can help me do just that.
- As a **social media enthusiast**, I want to share my unique clothing collection with my friends and also see what interesting collections others have going on
- As an **eco-conscious shopper**, I want to receive suggestions that align with sustainable fashion choices to support environmentally friendly practices.

Features

Our current goal is to implement 6 features listed as below, 4 of which are core functionalities of the app and 2 of which are extensions for making the app more helpful for users.

We plan to implement features 1 and 2 for milestone 1, features 3, 4, and 5 for milestone 2, and features 6 and 7 for milestone 3 if time permits.

Feature 1 (core): User Interaction with Clothing Items

Objective: To allow users to interact with a curated selection of fashion items, swiping right to like and left to dislike.

The main feature of our application is to let users go through a catalogue of clothes in the main page and swipe right on the ones that they like and left on the ones that they don't like, similar to Tinder. We hope this makes for a fun experience for the users to view our clothes and denote their preferences.

Feature 2 (core): Personal Collection and Wishlist

Objective: Allow users to save liked items to a personal collection and create wishlists.

As users browse through our collection of clothes, they may want to "save" similar clothes or pair ones that look good together. This feature lets users do just that by letting them create their personal collection and wishlist, which can be viewed in their "collections" page.

Feature 3 (core): Dynamic Personalization Algorithm

Objective: To refine clothing recommendations based on user interactions.

Using our machine learning model, after users have denoted their preference by swiping on several clothes, we offer suggestions on new clothes that the user will like. These suggestions can be seen in the "explore" page where users can browse through new items based on the items they liked previously.

Feature 4 (core): Catalog extension from external vendors

Objective: To allow external vendors to add their pre-existing catalogue to the app's catalogue of items and advertise their products

We plan to let users add on to our existing catalogue of clothes by allowing them to sell clothes that they don't want anymore on our app. Our app will have a dedicated page for users to create their own listings and manage the listings that they've created. User-made listings will be seen in the home page whilst the users are swiping through items. It will also be seen in the explore page which has a tab containing all the user-made listings.

Feature 5 (core): Adding items to cart and purchasing

Objective: Allow users to add items into their cart for purchase

As users browse through our collection of clothes, they may want to purchase specific pieces of clothing they come across or save it to their cart. This feature lets users do that by allowing them to add any items to their cart, and then letting them purchase those items once they navigate to their "cart" page.

Feature 6 (extension): Social Sharing and Community Engagement

Objective: Enable users to share their collections with friends or the public and receive feedback.

Users may find it more helpful and engaging if they could see collections made from other people that show them a new way to group clothes together. On the explore page of our app, we plan to display collections made from other users should they choose to share it. Users will be able to copy and save public collections to their own private collection so that they can view it later and modify it to their needs.

Feature 7 (extension): Item sorting, searching, and filtering

Objective: Provide insights into sustainable fashion choices and eco-friendly brands.

Users may want to sort and filter through a list of items in their collections or in their explore page, or search the entire item catalog to quickly find what they want. This feature aims to help the user do that by adding a search bar on top of the collection and explore pages as well as the option to sort and filter out items.

Feature 8 (extension): Image based similarity recommendation of products

Objective: Provide similar recommendations to products currently viewed by users.

A key objective of a clothing application is ensuring that users are able to find the item that they are looking for, and sellers to increase the likelihood of selling their items. We believe that this feature is a crucial component that meets both these criterias. This feature allows users to seamlessly browse through clothes that are similar to the one that they are interested in.

Milestone 2

Feature 1 (core): User Interaction with Clothing Items

Once a user is logged into our app, they're automatically taken to the home tab where they're shown the uppermost card of a deck of cards made from a sample catalog of items stored in our app's database.

The card shows the image of a clothing item and the name of the item below the image, and the card can either be swiped left or right if the user doesn't like the item or likes it, respectively.

The image of the home screen users are taken to after logging in.

Swiping the card in either direction shows the next card from the deck, and swiping the card right automatically saves the item in the user's "Liked Items" collection.

The user can also double tap the card to get more information about the item displayed on the card.

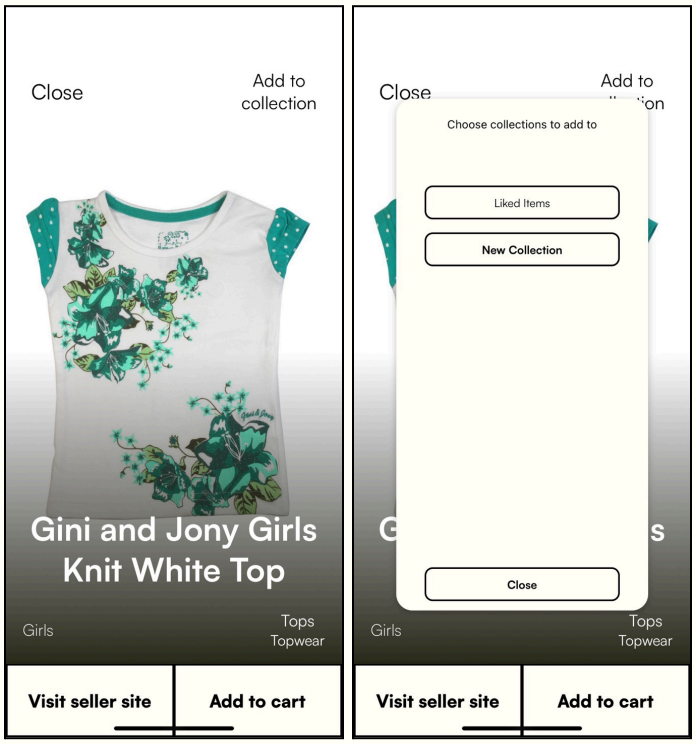
Double tapping a card opens a page containing additional information about the item displayed on the card, such as the intended gender and product type and category of the item.

From this page, users can then add the item to collections other than the “Liked Items” collection by tapping on “Add to Collections” text at the top right of the card and choosing which collection the item should be stored into.

There are 2 buttons at the bottom as well which allows for further interaction with the item.

Users can now add the item to their cart as well by pressing the button at the bottom right.

Users can also press the button at the bottom left to either chat with the seller of the item through Telegram if the item is a user made listing, or go to the item vendor’s website such as Nike or Adidas if the item is sold by a brand instead.



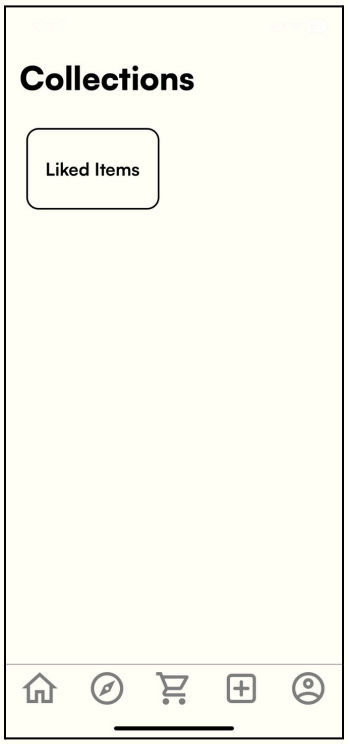
The image of the item details page screen users are taken to after double tapping the screen. Users can select collections to save the item into by pressing the “add to collection” button on the top right and then pressing the collections they want to save the item in.

Feature 2 (core): Personal Collection and Wishlist

Users can navigate to their profile page, and then click on the “view my collections” button to go to the collections page.

We originally wanted to have a separate navigation button for going into the collections page inside the navigation bar. However, as we already had 5 navigation buttons, we thought that putting 6 buttons on there may look cluttered. Furthermore, we thought that people would be naturally used to finding their personal collections in their profile tab, as other popular apps like Instagram also have users go to their profile tab to see their archived posts.

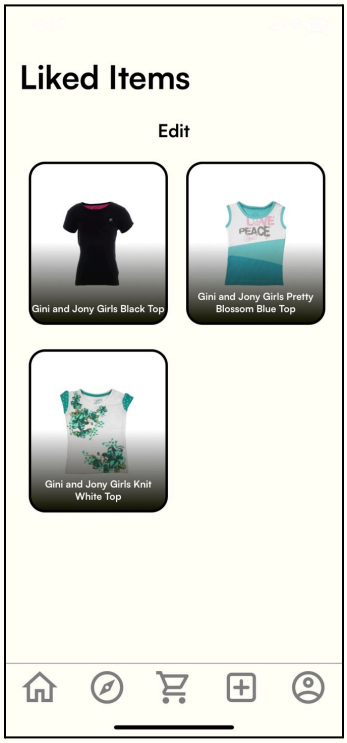
The collections page displays a list of collections the user has created so far, starting from the “Liked Items” collection which is created automatically for every user.



The collections page, currently displaying only the “Liked Items” collection.

Tapping on any of the collection names from the collections tab takes the user into a single collection page where items from that specific collection are displayed as a list.

Users can edit this list of items and remove items from a collection by tapping the edit button on the top of the single collection page.



The "Liked Items" collection page.

Feature 3 (core): Dynamic Personalization Algorithm

Our algorithm is based of a peer to peer recommendation system which collects data from users based on the items they swiped right on, their personal data such as age, gender and clothing preferences. The model then predicts what the user would like based on all these information. However this model should also take account of diversity, in order to prevent a recommendation system that may be discriminatory. We are at the deployment phase of this model, but running into some bottleneck issues with the retrieval of data from numpy files. Since the recommendations would have to be seamless to its all its users, the model should either take account its new swipes and make recommendations in the background.

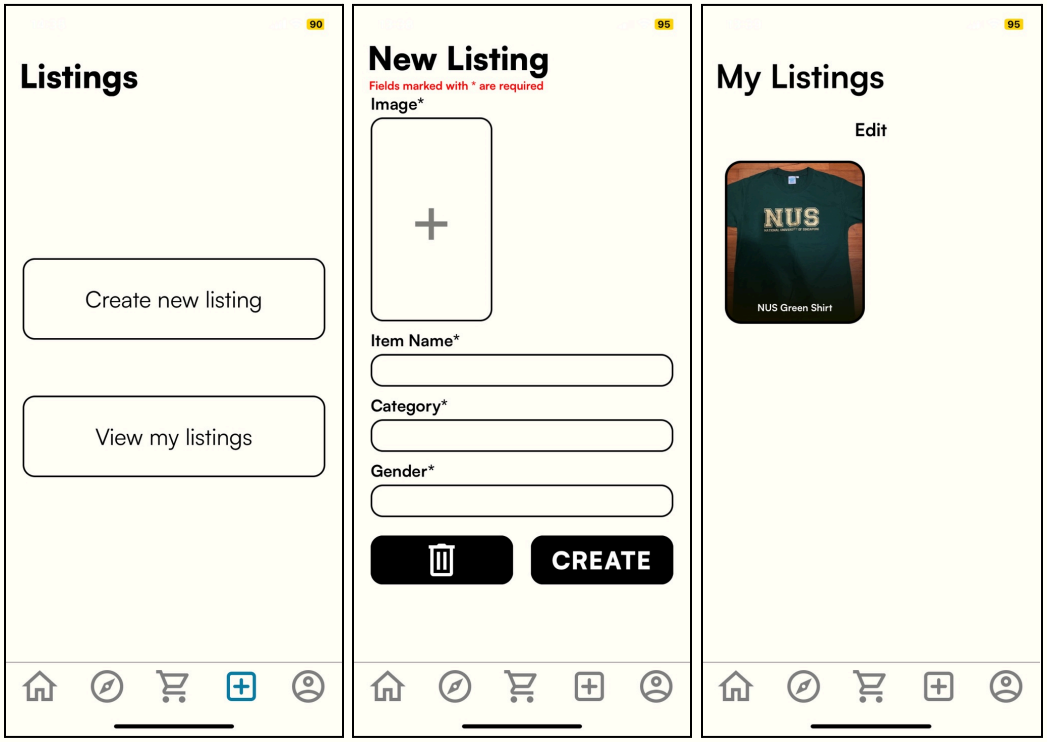
Our deployment phase utilises Amazon Web Services (AWS) such as the S3 bucket to store csv files and user information. Our ML model, which has been containerised into a DockerFile will then load multiple files which they can then run the predict function

on. However, the loading step takes quite a while and we are in the process of asynchronizing the process to make it quick while making it run in the background automatically to ensure that users have a seamless experience.

Feature 4 (core): Catalog extension from external vendors

Users can navigate to the listings page to create their own listings for others to see.

The listings page has two sub-pages, one for creating a new listing, and one for viewing all the listings the user has created.



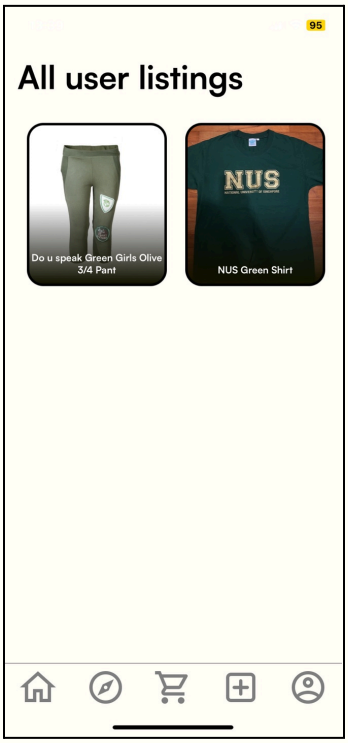
The listings page found inside the profile page, along with the "create new listing" page and "view my listings" page.

To create a new listing, users can press the button that says "create new listing", which takes them to a form-like page where users can input information regarding the item they want to sell.

To view and take down listings they made, users can press the button that says "view my listings", which shows them the list of their own listings that can be edited similarly to the collections page.

These listings can be seen by other users as they're swiping through items in the home page.

Additionally, the explore page has a dedicated "all user listings" section where users can see only the listings other users created and none of the non-user listed items.



The "all user listings" page found inside the explore page.

As mentioned in the description for feature 1, if other users are interested in a user listed item, they can contact the owner of the listing through Telegram. Because of this, users now need to input their Telegram username when signing up for an account.

We originally wanted to make a dedicated chatting feature for our app using Supabase, which was already being used for the backend. However, most of the documentation online for creating a real-time chat application with Supabase had other frameworks in place such as NextJS, which is inapplicable in a mobile development environment. We plan to make this happen for milestone 3 if there is enough time.

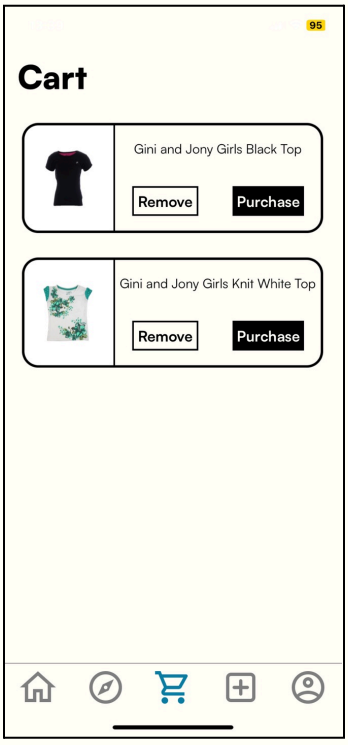
Feature 5 (core): Adding items to cart and purchasing

An important feature for any shopping app is allowing the users to add items to their cart and let them purchase it.

As mentioned in the description for feature 1, users can now add items into their cart from the detailed item page by pressing the button at the bottom right.

Users can then view items stored in their cart by going to the cart page, where they can either choose to remove items from the cart, or purchase the items.

Pressing remove on all the items simply removes the item from the cart, while pressing purchase on a user made listing takes the user to a Telegram chat for price negotiation, while pressing purchase on a non-user made alerts the user that they made a purchase for now.



The cart page containing two items.

Extra features

Explore page

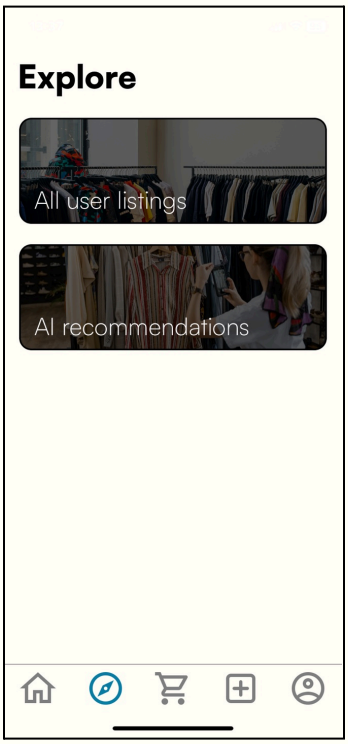
A new explore page has been created for milestone 2.

Currently the explore page has two main tabs - all user listings tab and AI recommended products tab.

Pressing on the “all user listings” tab takes the user to view a collection of only the items/listings posted by other users of the app.

Pressing on any of the listings opens up a detailed item page identical to the one shown in the home screen when a user double taps the item card, enabling users to see more details about the item, add it to their cart, and contact the seller or visit the brand site.

We plan to populate the explore page with more tabs, as well as with collections created by users in milestone 3.



The explore page currently displaying two different navigation options

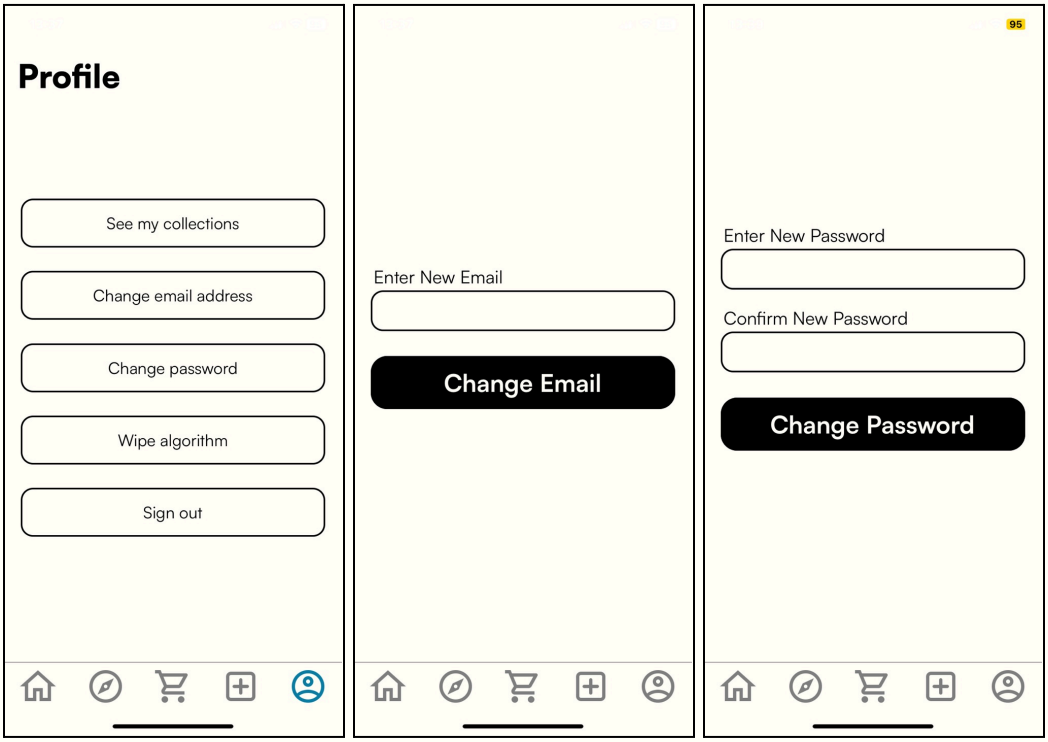
Profile page

A new profile page has been created for milestone 2.

Besides the user's collections page, the profile page also contains pages for users to change their email, password, reset their personalized recommendation algorithm, and sign out.

Changing users' email and password does not log them out of the app, but just requires the users to sign in with their updated email/password next time they need to sign in.

Pressing on the sign out button takes the users back to the sign in page.



The profile page, along with the "change email" page and "change password" page.

Future plans (Milestone 3)

Feature 6 (extension): Social sharing and community engagement

As mentioned previously, we plan to implement feature 6 through the explore page. Inside the explore page, there will be a navigation button with the label “View community-made collections”. Tapping on that button will take the user to a page filled with a list of collection names from collections made from other users, sorted in the order of likes, descending. Tapping on the collection name will then take the user to a page of the list of items in that collection. The users can choose to import and save the entire collection to their own collections, and they can press “like” on the collection.

From their own collections page, users can choose to share their collections or not by toggling the public/private option on the collections page. They’ll also be able to see how many likes their collection got if it was set to public.

We believe this feature would be beneficial as users looking to buy clothes may look for good pairings for the items they’re interested in, and users may look for a unique grouping of items such as summertime clothes or clothes to wear to a rock concert. Enabling the users to share their collection of items can help other users with this process and make for a more enjoyable shopping experience.

Feature 7 (extension): Item sorting, searching, and filtering

We plan to implement feature 7 on the explore page and the user’s collections page. Inside each “sub-page” for the explore page and the collections page, we plan to put a search bar on the top that lets users first search for an item on that sub page. We also plan to have a navigation button at either the top or the bottom of the explore page that lets users look and search through all items, regardless of if the user swiped left or right on an item.

We also plan to put a sort and filter button on the top of those screens as well to let users sort and filter items based on their needs. The current options we thought of for filtering items are by their color, category, gender, price, and brand. To do this, we would need to tag each item with a price and a brand as well and require users to input those fields in as well during listing creation. The current options we thought of for sorting items are by their price and name, both ascending and descending.

Whether or not filtering should be added into the main swipe page is still being discussed amongst the team members.
We believe this feature would be beneficial as sometimes, users may be looking for a specific piece of items rather than browse through the given collections and slowly build up their preference. Enabling the users to quickly sift through items based on their urgent needs will give them a satisfactory shopping experience.

Feature 8 (extension): Image based similarity recommendation of products

As users swipe through clothes,, users can double click on a specific item to find out more about it. At the bottom of the page, we provide useful information such as similar items that the user may be interested in, in the form of a sliding pane that shows a quick preview of the similar items that they might be interested in. Users are able to quickly swipe and glance through items personally recommended by our image-recognising Machine Learning model. If a user is interested in an item,, they can simply click on the item to find more information about it.

This promotes relevant alternatives for users, thereby driving sales for sellers and increases the chances of users being able to find the items they desire.

This feature is both intuitive and a quick way for fashion enthusiasts and casual shoppers to find the item that they are looking for with ease.

Diagrams

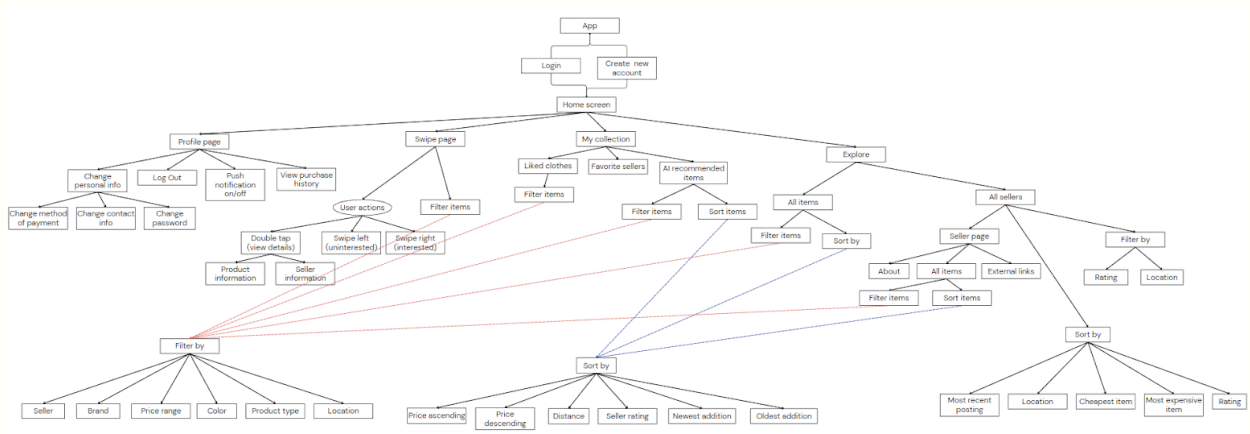


Diagram representing the proposed features of the completed application

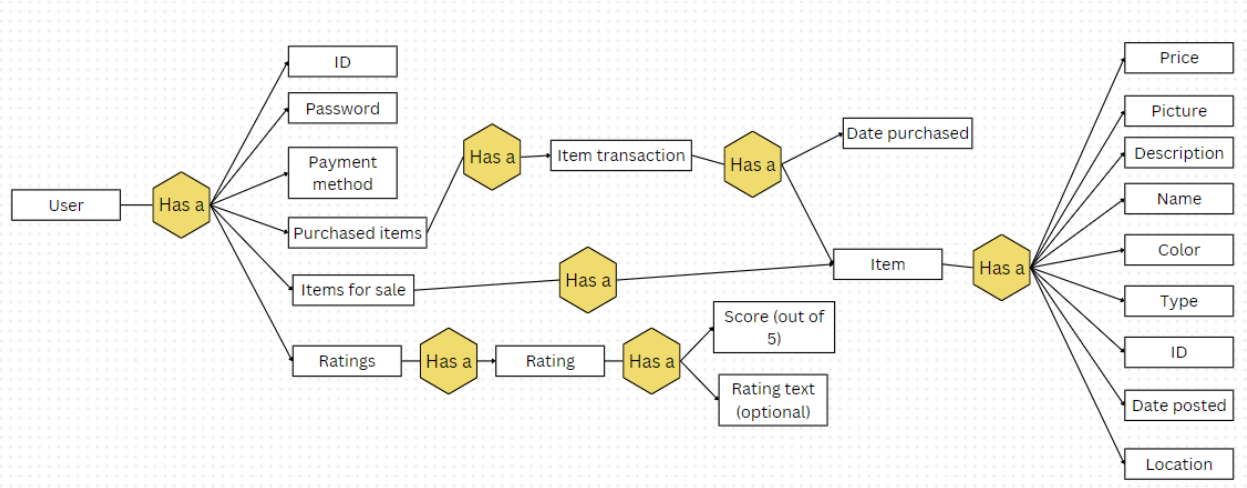


Diagram representing the object relationship model diagram of the completed application

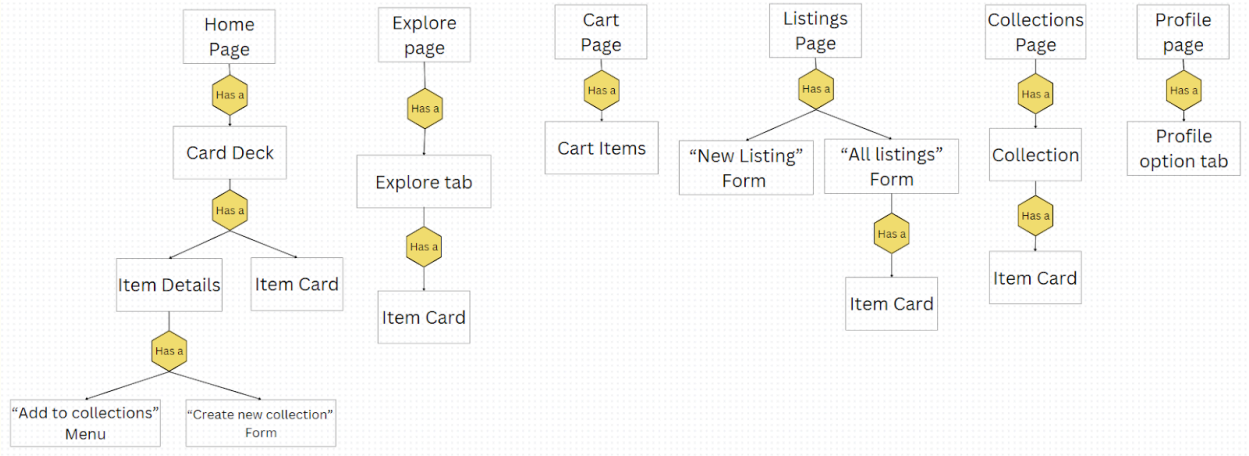


Diagram representing the composition of React components of the current prototype

Tech Stack

- React Native

React Native is a front-end framework for mobile applications similar to what React is to web applications. We chose React Native to develop the front-end of our application as it allows for easy cross-platform development and it provides various tools and components to test, debug, and develop our app. Everything the user can interact with in our application is made with React Native.

- Expo

Expo is an open-source framework for apps that run natively on mobile devices. We chose Expo as it significantly reduces the extra work needed to compile React Native code into applications that can work natively on either Android or Apple, and it provides a platform to host our prototype code online.

- Supabase

Supabase markets themselves as an open-source alternative to Firebase, one of the most popular databases used in mobile application development. We chose Supabase as unlike Firebase, Supabase provides a relational database. Using a relational database makes it easier and faster to work with the data needed for our application, which is stored in rows and columns containing specific attributes. The current sample data, as well as user information such as their email, password, collections, items stored in their cart, and their listings, are stored inside Supabase.

- Python

Python serves as the core programming language for our Machine Learning models due to the availability of extensive Machine Learning and data analytic libraries such as Numpy, ResNet, Pandas and TensorFlow. Python is also a modern programming language that is widely supported by many platforms used within deployment such as Amazon Web Services (AWS).

- Amazon Web Services (AWS)

AWS serves as the backbone of our Machine Learning (ML) functionality and deployment processes. AWS serves as a one-stop solution for integrating ML models within our app due to the wide variety of applications available to use within the AWS ecosystem that works together seamlessly. AWS also reduces the need of managing servers, allowing us to focus more on the development of our implementations, with many low-level abstractions managed under the hood by AWS. Common applications that helps to integrate our ML model within our ExpoGo app include AWS Lambda, EC2, API Endpoint, ECR and S3. We also chose to use AWS because students are given a generous 1 year trial of using AWS services which greatly reduces any potential costs that we may have to bear in order to use ML capabilities within our app.

- Docker

When deploying ML models, microservices such as Docker allows us to containerise our ML models in a modularized architecture. When creating ML models, it is crucial that the required libraries and environment remains constant especially during deployment. By containerising ML models within a dockerfile, it ensures that the model works fine when uploading on AWS Lambda for example, which can be executed when API calls are made from the Expo Go application.

Software Engineering Practices and Principles

Single Responsibility Principle (SRP) and File Organization

All of the components of our application are kept as small as possible so that they only have one responsibility associated with it.

Our application folder structure is divided into subfolders containing components for each page, and those subfolders are further divided into singular components that are responsible for performing a singular task in that page. This helps in decoupling our code so that it's easier to read, debug, and maintain.

Common components that are used in multiple pages are kept in a common components subfolder instead of being included in every single subfolder that requires that component.

Common	Auth
	SignIn
	SignUp
	DetailedItemCard
	ItemCard
	LoadingScreen
Home	CardDeck
	CollectionMenu
	Home
	ItemDetailsPage
	NewCollectionForm
Cart	Cart
	CartItem
Explore	Explore
	ExploreLink
	ExplorePage
Listings	Listings
	MyListings
	NewListingForm
Profile	NewEmailTab
	NewPasswordTab
	SignOutTab
	Profile
Collections	CollectionItem
	Collections
	EditableSingleCollection

This principle is followed for our API functions as well, where we have functions for getting collections from a user, creating a new collection for a user, saving an item to a collection for a user, and so on. Each function is responsible for only one job only for that specific subfolder.

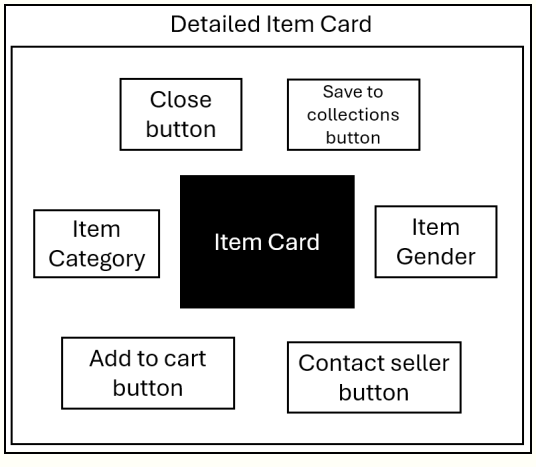
Common	getItemObjById
	getItemsFromIdArray
	getSession
	getUserId
	redirect
	redirectToSeller
	supabase
Home	getCards
	getStartIndex
	incrementStartIndex
	updateStartIndex
Listings	createListing
	getListings
	getNewItemId
	pickImage
	removeitemFromListing
	uploadImage
Profile	changeEmail
	changePassword
	signOutUser
Collections	addItemToCollection
	createCollection
	createCollectionAndAdd
	getCollections
	getSingleCollection
	removeItemFromCollection
Cart	addItemToCart
	getCartData
	getItemsFromCart
	removeItemFromCart

Functional Components

There are two main types of components in React - functional components and class components.
Functional components are Javascript functions that accept properties for the component as an argument and outputs a React element containing those properties.
Class components are classes that extend the React.Component class that have properties and states stored inside the class, and return React elements through its render() method.
We chose to use only functional components for our application due to many advantages that functional components provide.
Functional components are much more concise to write and require little to no boilerplate code, reducing in a less bug-prone code.
They also take up less memory than class components, which results in a faster performance for the app. Functional components can handle state and side effects regarding how the React element is rendered using hooks, which is more intuitive than managing class components. Lastly, functional components are pure functions, which allow us to test our code more easily and focus more on composition rather than inheritance, which helps us better follow the Open/Closed Principle.

Open/Closed Principle (OCP)

As mentioned above, we followed the open/closed principle when coding our app. This helps in creating less lines of code as existing components can be reused without modification to existing code, and as a result, reduces bugs in our program. It also helps with scaling our code with additional functionalities.

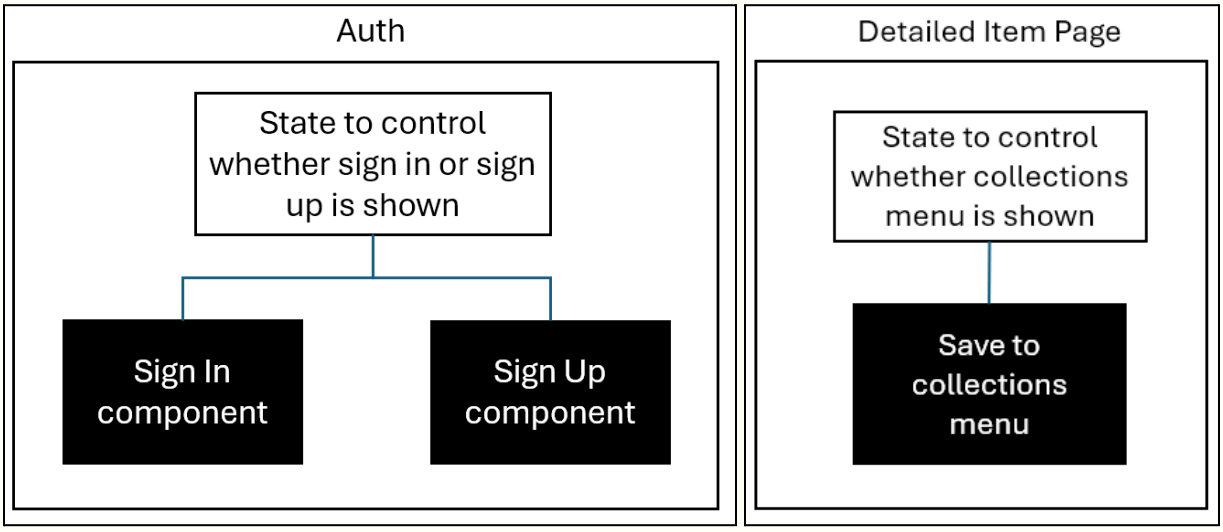


One example for this would be the item cards used in our app. Our home tab displays an item card (IC) component by default, and doubling tapping on the item card displays a detailed item card (DIC) component which shows more information about an item.

To create the detailed item card component, we extended upon the already existing IC component by importing it into the DIC component and adding components around the IC without modifying it.

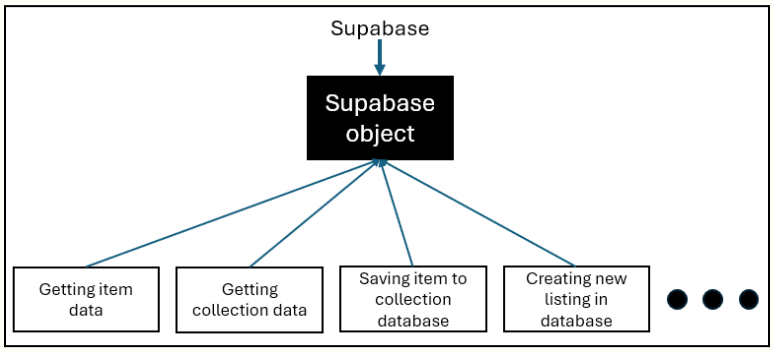
Compound Components

Compound component is a React design pattern where related components are placed inside a parent component, which in turn controls their behavior. This pattern helps in cleaner and more readable code with predictable UI interaction results.
We utilized the compound component pattern in our user sign in and sign up page. The “Auth” component holds both the “Sign Up” component and the “Sign In” component, and whichever of those two child components are displayed is based on the state of the “Auth” component. Pressing the “switch tab” button on either one of those child components switches the state of the parent component, which in turn changes which of the two components is rendered.
This pattern is also utilized in the detailed item page. The component rendering the detailed item page has a detailed item card and a collections menu (displaying the collections for the item to be added) as its child. Whether or not the collections menu is shown is dependent on the state of the parent component, and pressing the “add to collections” button or “close collections menu” button changes this state.



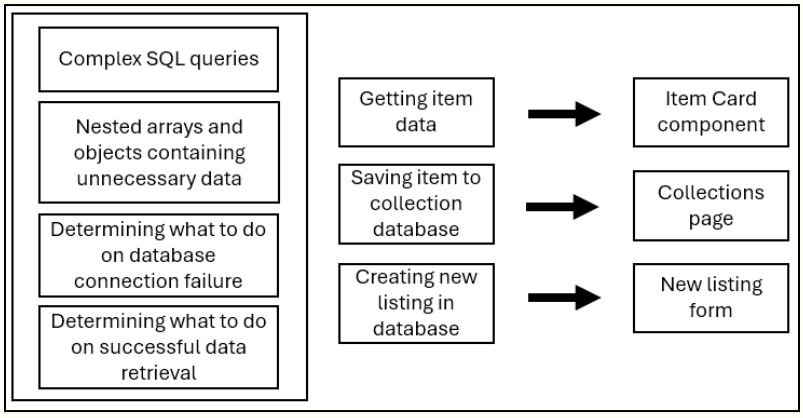
Singleton pattern

Singleton pattern is a pattern where only one instance of a class is instantiated. This helps to prevent name collision and enables lazy initialization.
We utilized the singleton pattern in communicating with the Supabase backend. A single utility function instantiates the Supabase backend object, and all the other utility functions that make a call to the backend goes through this Supabase backend object instead of creating a new Supabase object each time they make a call to the backend.



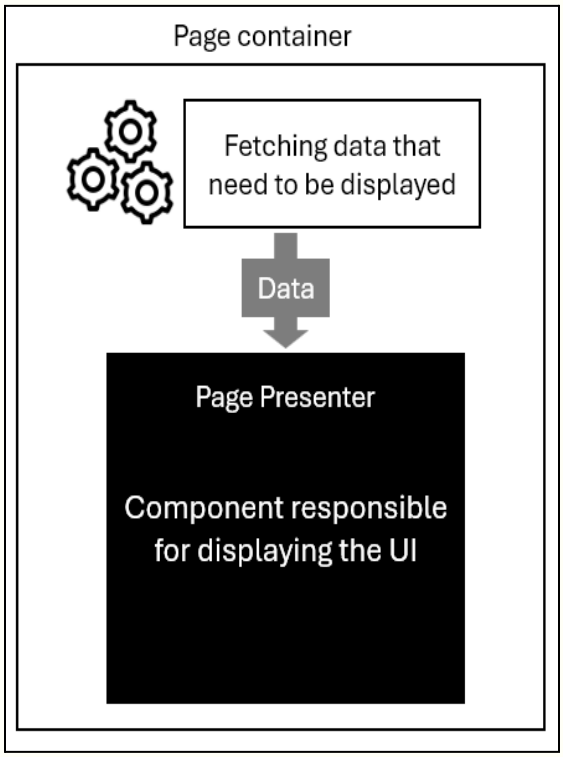
Facade Pattern

Facade pattern is a pattern where there is a simplified interface abstracting away the complex implementation details of what that interface is responsible for. This helps separate concerns and make the code more readable.
We used the facade pattern in our utility functions responsible for getting data from the back end. To get specific pieces of data from the Supabase backend, one needs to write complex SQL queries and then determine what happens on data retrieval failure or success. Our utility functions hide this detail from the components that require these data and enable the components to simply call and retrieve the prepared data from these functions.



Container-Presenter Pattern

The container-presenter pattern is a pattern where one component is responsible for handling the data fetching and state management (Container) and another component is responsible for the presentation of the UI (Presenter). This helps in separating the concerns of data management and UI rendering, which can make the application easier to manage and scale.
We used the container-presenter pattern in rendering all of our pages. All of our page components takes in a component which renders the UI based on the props given passed down to it. The parent component is responsible for fetching the data and passing it to the child component (container), and the child container then renders the UI with the data given by the parent (presenter).



Unit testing

Our components and functions for the front and the back end have been unit-tested based on the smallest responsibility assigned to them.
To unit test our code, we used three tools - Jest, React Native Testing Library, and Husky.
We chose Jest as it was the most popular tool for testing Javascript code, and it came pre-installed with the Expo configuration we were using. Jest allowed us to automate unit testing, where we could run a Jest script in the background and it continuously ran tests whenever there was a change in our code. Jest also allowed us to mock and replace functions that were used to get data from the back end, allowing us to test for each component only what they're responsible for.
Although Expo documentation recommended using React Test Renderer, we chose to use the community-supported React Native Testing Library as it allowed us to simulate user interactions on the app much more intuitively while testing.
We used Husky, which ran testing scripts every time a commit was being made, to further automate our testing process.
Our React component tests were written based on three goals in mind - whether it is rendered properly or not, whether it displays props passed down to it properly or not, and whether it behaves properly on user interactions or not.

Common	SignIn	Renders correctly
		Pressing sign in calls the sign in function with the typed email and password
		Pressing sign up instead takes the user to the sign up tab
	SignUp	Renders correctly
		Pressing sign up calls the sign up function with the typed email, password, and Telegram username
		Pressing sign in instead takes the user to the sign in tab
	ItemCard	Renders correctly
		Displays the given item name properly
	DetailedItemCard	Renders correctly
		Displays the given item name properly
		Displays the given item gender properly
		Displays the given item category properly

Cart	Cart	Renders correctly
		Displays given items properly
		Deletes item from UI on pressing delete item
	CartItem	Renders correctly
		Calls the callback function on remove button press
		Calls the callback function on purchase button press

Collections	Collections	Renders correctly
		Displays the given collections list properly
		Tapping on a collection calls the callback function
	EditableSingleCollection	Renders correctly
		Displays the given items properly
		Deleting an item removes it from the UI and calls the callback function

Home	CardDeck	Renders correctly
		Displays the top card correctly based on the given starting card index
		Calls the callback function on swipes correctly
		Calls the callback function on double taps correctly
	ItemDetailsPage	Renders correctly
		Displays the given item name properly
		Displays the given item gender properly
		Displays the given item category properly
		Opens up collection menu on button press
		Calls the callback function with the item information on add to cart button press
		Calls the callback function with the item vendor information on contact seller button press
	CollectionMenu	Renders correctly
		Displays the given collections list properly
		Opens up new collection form on button press
	NewCollectionForm	Renders correctly
		Calls the callback function with the typed collection name on button press

Explore	Explore	Renders correctly
		Renders the given explore link children prop correctly
	ExploreLink	Renders correctly
		Calls the callback function with the given URL on press
	ExplorePage	Renders correctly
		Displays the given items correctly
		Double tapping on an item calls the callback function with the item information

Listings	Listings	Renders correctly
		Displays the settings correctly
		Tapping on the view my listings button calls the callback function
		Tapping on thecreate new listing button calls the callback function
	MyListings	Renders correctly
		Displays the given listing items correctly
		Editing and deleting a listing item hides the item from the UI and calls the callback function

Integration testing

Each one of our pages can be seen as a “composite component”, where multiple components come together to create one large page component. To ensure that these components work together properly, we performed integration testing on each of the six pages mentioned above - home, explore, cart, listings, profile, and collections.

Home	Opening the home page without prior signin takes user to the sign in tab
	Pressing the sign up instead button takes user to the sign up tab
	Signing up takes users to the home tab
	Swiping right on the item card shows the next item card
	Swiping left on the item card shows the next item card
	Double tapping on the item card directs users to the detailed item page
	Pressing add to collection on the detailed item page brings up the collections menu
	Pressing new collection on the collections menu brings up the new collection form
	Pressing visit seller site redirects user to an external site
Explore	Explore page displays explore tabs that lead to explore sub-pages correctly
	Tapping on an explore tab redirects users to that explore sub-page
	Tapping on an item redirects users to the item details page
	Closing the item details page redirects users to the previous explore sub-page
Cart	Tapping on "add to cart" on a detailed item page saves item to the cart in the UI and the database
	Pressing remove on the cart page removes item from the cart in the UI and the database
	Pressing purchase on the cart page alerts the user of the purchase and removes item from the cart in the UI and the database
Listings	Creating a new listing with the new listing form adds it to the "view my listings" page UI and adds the item to the database
	Removing a listing from the "view my listings" page removes it from the UI and the database
Profile	Pressing the "my collections" button takes the user to the collections page
	Changing email through the change email tab changes the email for the user in the database
	Changing password through the change password tab changes the password for the user in the database
	Pressing the "sign out" button signs the user out succesfully
Collections	Pressing the "save to collections" and choosing a random collection to save on the detailed item page saves the item in that collection in the UI and the database
	Removing an item from a collection removes it from the UI and the database

Revision Control Software

Although both our team members had an idea of what the other was working on, the work we were doing was pretty much separated, with one of us focusing mainly on the machine learning side of the app while the other focused mainly on the front end and the back end of the app.

We used Git and GitHub, revision control softwares, to work on different features at the same time and merge changes without affecting each others' work.

We forked from the same repository, worked on the assigned parts, and created pull requests every time a meaningful part of the app was completed.

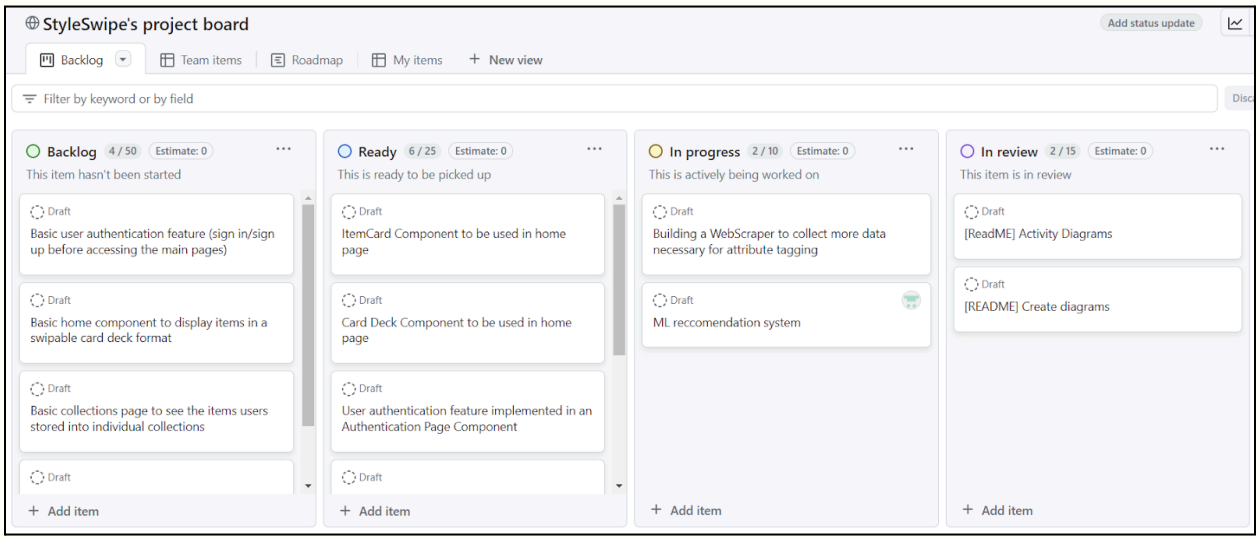
When a pull request is created, both team members would discuss the changes being made and merge the changes to the main branch or prevent the pull request if there were issues with it.

We also followed the principle of micro-committing, where we would commit changes to our local Git branch every time there was a tiny change.

This allowed us to revert to previous versions of our code if there was a mistake and fix them without undoing and redoing a lot of work.

Agile Development Methodologies

We employed agile development methodology in developing our application. We kept a backlog of features we planned to implement, which were broken down into smaller tasks that were assigned to each person. We then had a two week sprint where we worked on those tasks and gave each other feedback on the work that has been done, the goals that have been met, what goals have not been met, and what to do for the next sprint.



We used the GitHub Project Board to keep track of our goals and tasks

Trying out the app

To test our app on your Android or Apple mobile device,

1. Download **Expo Go** from the Google Play Store or Apple App Store
2. Scan the QR code below using the **Expo Go app if you're on Android** or using the **Camera app if you're on Apple**