Non-functional Requirements

Performance: The system should load the clothes list and individual item details quickly. Any changes in favorites and shopping bags should be updated promptly.

Security: User authentication details should be securely stored and managed using Firebase Authentication.

Usability: The application will be designed with a primary focus on iOS, while also ensuring compatibility with Android. The user interface should be intuitive and responsive. All buttons should be labeled for clarity, and the overall application structure should be straightforward to navigate.

Scalability: The system should be designed to handle growth in user base without impacting user experience. Firebase provides auto-scaling capabilities to handle such situations with their paid plans.

Availability: The system should be capable of supporting up to 100 concurrent users connected to Firebase Realtime Database on the free plan. This would ensure high availability of services for users.

User personas

Target users:
Our target users are Shoppers looking for a convenient way to shop (15 to 70 years olds)

Bob Burger, 23:

- Characteristics: He is a full-time college student who likes to dress in a unique and comfortable style.He is a decisive buyer who is picky on what he buys. He needs to find a place that provides unique clothing in one single app
- Needs: Challenges he faces is that not all clothing items are set at an affordable price. He wants a universal piece of clothing that can become one of his staple pieces in his closet. He also wants to be able to put the clothes he likes in a favorite list so when he decides to buy it will be quick for him.

Timmy Tim, 16:

- Characteristics: He is a high school teen starting his sophomore year. He wants a new set of clothes that he's gonna wear for the year. He's still trying to find a style that suits him. Since he's in highschool he does not have that much money to spend.
- Needs: Timmy wants a platform where he could get trendy clothes for a reasonable price. He wants a straightforward UI that is easy to navigate through and he wants a way

to filter out products based on a price range. He would also like it if the .app make suggestion to what clothes would look best for him

Linda Lynn, 43:

- Characteristics: She is a full-time marketing agent who works a 9 to 9 to keep her family secure. She needs a day to day basic outfit that can be dressed up or dressed down for comfort.
- Needs: Linda still wants to be seen as a fashionable person but also wants to dress for comfort. She wants to quickly be able to add featured items to her cart to ensure she doesn't take too much time out of her day.

High-level architecture

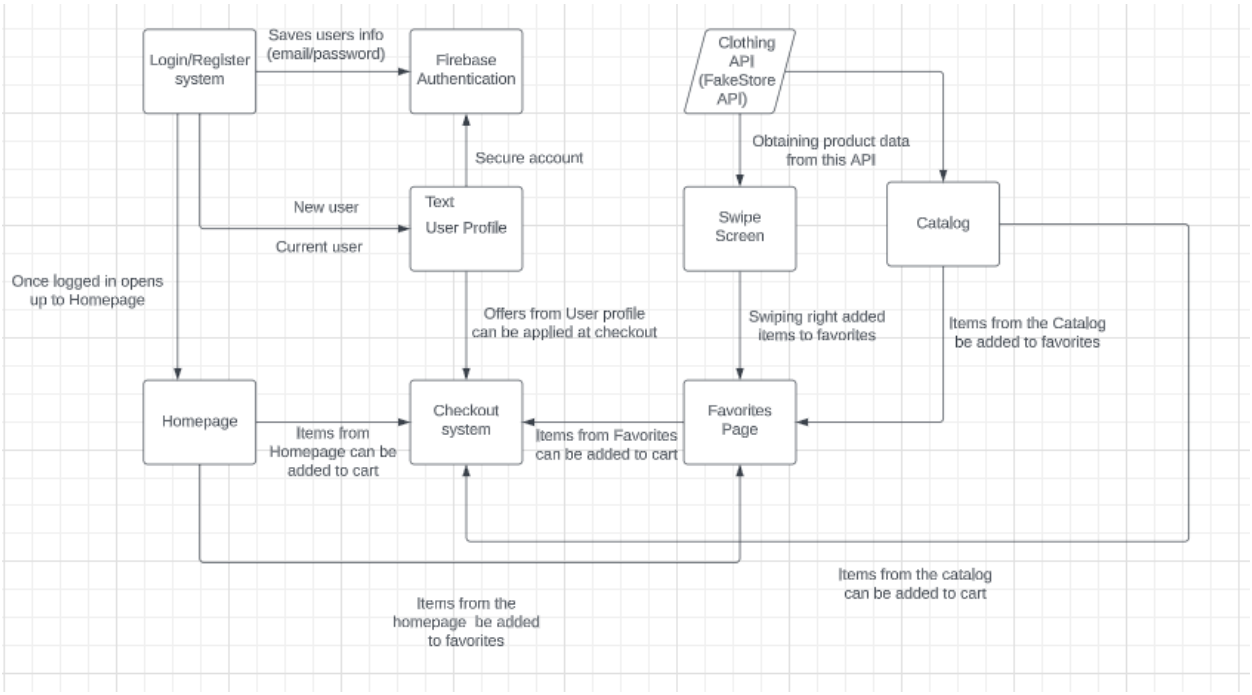
Overview of the system's architectural design:
For our application, we are using the React Native framework to organize our UI, business functionality such as what items could be displayed on our homepage, data communication which is like the login information/authentication, and data storage which is just managing how a user's data is stored.

Major components and their interactions:
The components we plan on having are a UI which is written in JavaScript. Our UI will be utilizing the React Native framework. The UI will be interacting with the firebase authentication to authenticate like the email and password when a user is first registering an account. In addition to that, we'll have a User Profile where it will store the user's data. To improve a user's shopping experience, we'll have a favorites page that will have the user's items that they plan on getting. Lastly, we plan on having a checkout/cart page that will allow the user to purchase their items placed in the cart.

Data storage and retrieval methods

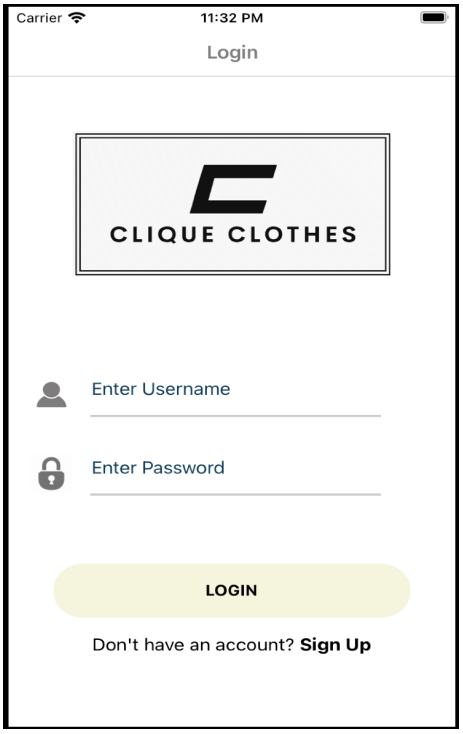
The users have a sign up page to sign-up information and is stored in Firebase Authentication. Once signed up, on each login, the system checks if the user's credentials exist in Firebase Authentication. User favorites, shopping cart items and profile details, are stored in Firebase's Cloud Firestore in a structured, NoSQL format, with each user having a unique document containing their respective data. For clothing items, the system fetches data from the FakeStore API and only caches the necessary data we will be using for our HomeScreen.

Data flow diagram


Login/Reglster system
Saves users info (email/password)
Firebase Authentication
Clothing API (FakeStore API)
Secure account
Obtaining product data from this API
New user
Text
User Profile
Swipe Screen
Catalog
Current user
Once logged in opens up to Homepage
Offers from User profile can be applied at checkout
Swiping right added items to favorites
Items from the Catalog be added to favorites
Homepage
Items from Homepage can be added to cart
Checkout system
Items from Favorites can be added to cart
Favorites Page
Items from the catalog can be added to cart
Items from the homepage be added to favorites

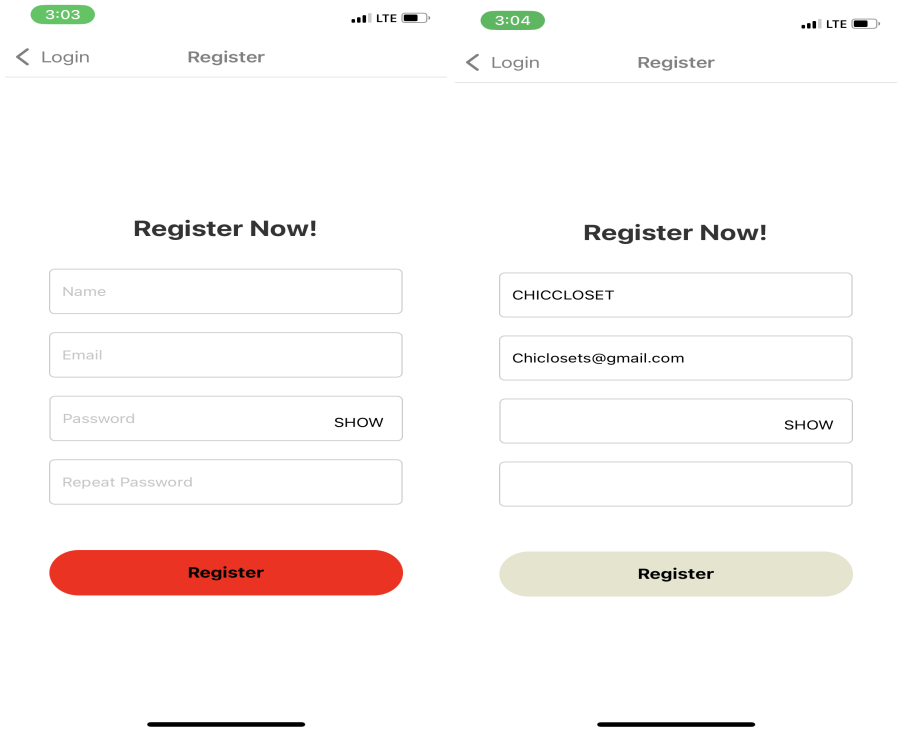
User Interface design

Login Screen:

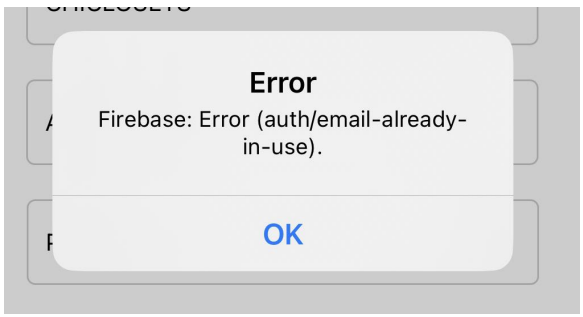


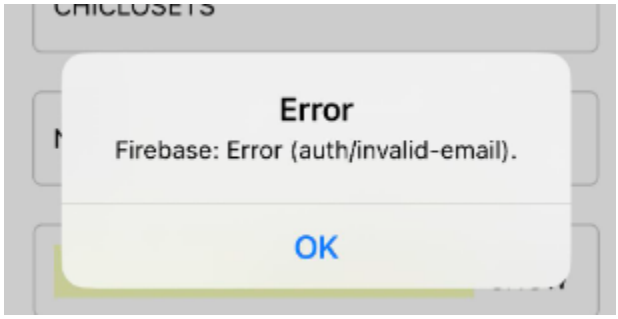
Users can login but only with the account that is already registered in the Database. We used firebase Authentication for this.

Register Screen:

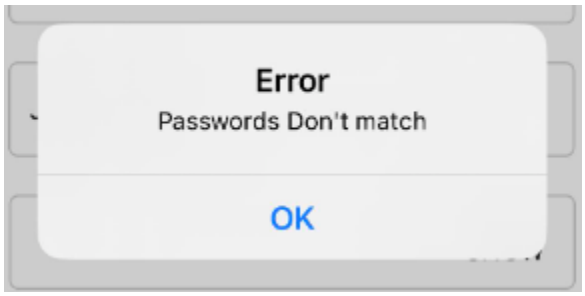


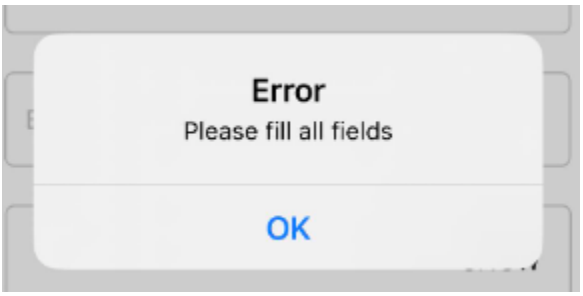
Users can input their name, email address and a password of their choice. Below are the listed errors users get if the following requirements are not met.





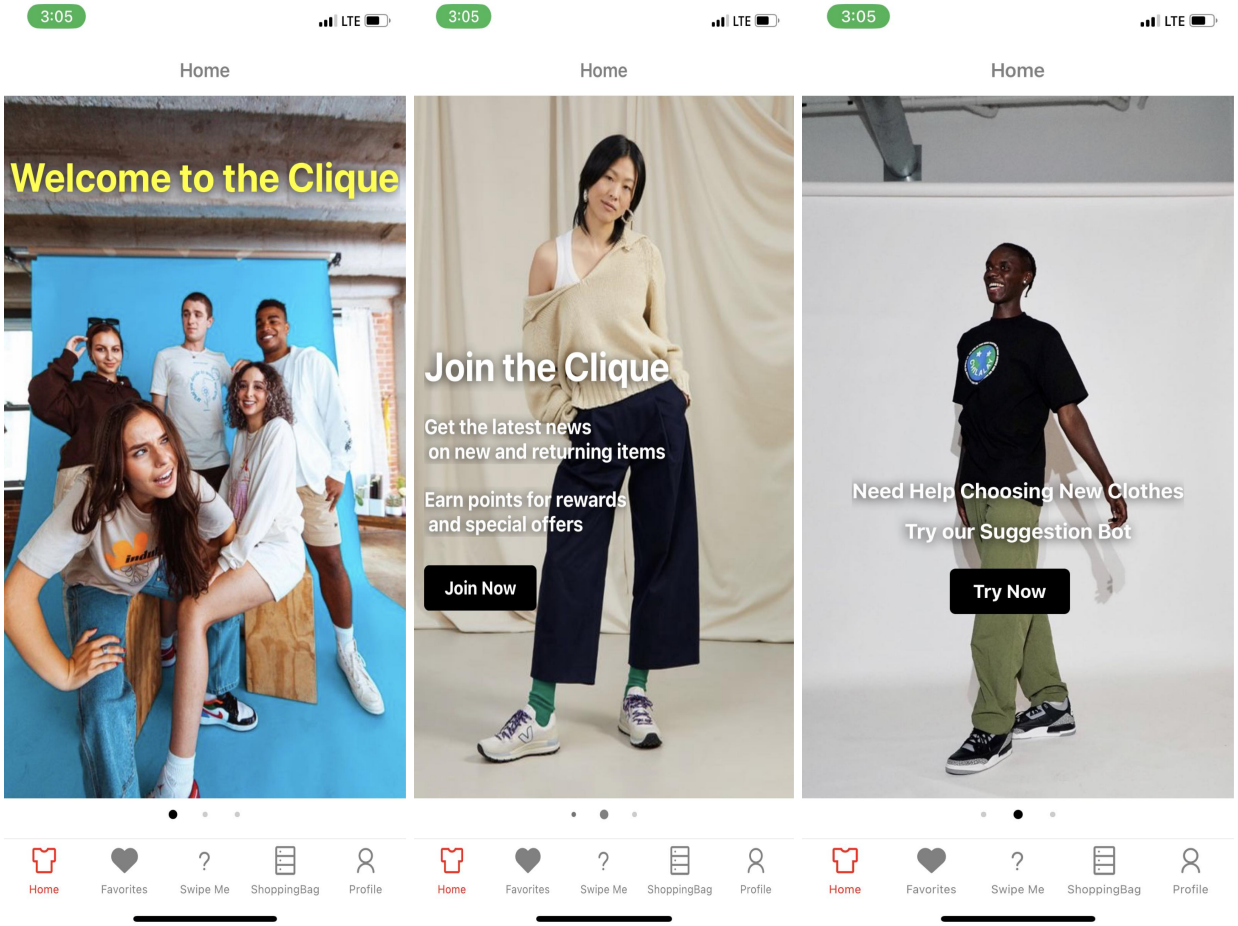
This will pop up if the email the user input has already been registered before and the one on the right will show up if the format for the email is not correct.



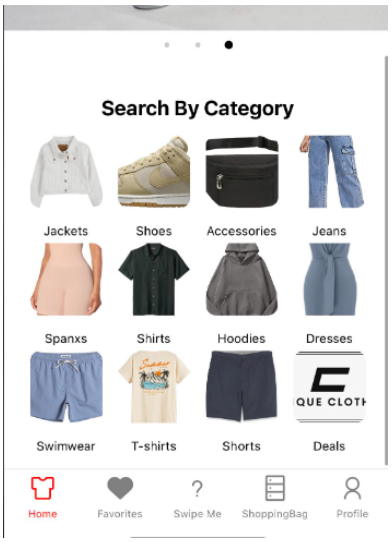


This will show up if the user input 2 different passwords, and the right will show up if not all fields are filled.

Home Page:

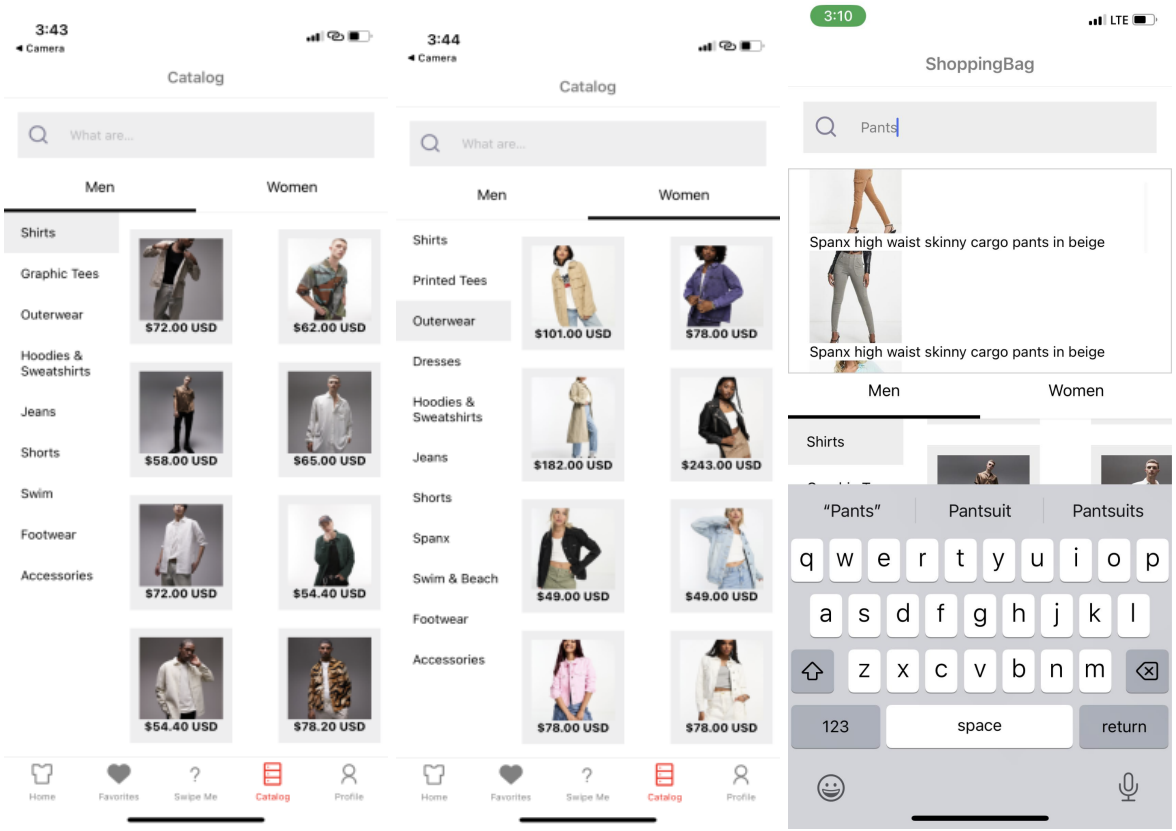


Once a user sign up or login the user will go straight to the homepage. In the homepage the user will see some image and then can scroll through it horizontally to see the other images and other features.

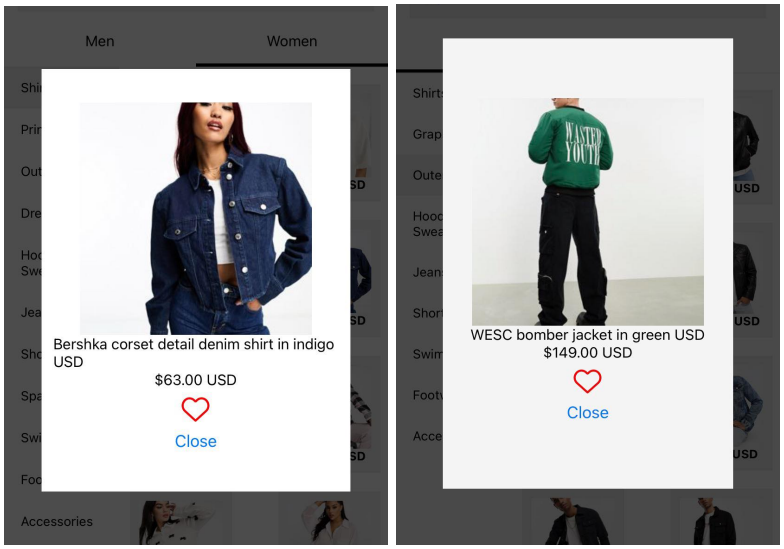


Once the user scrolls down, the user can see different categories and once they click on it, it will go and show those categories in the category screen.

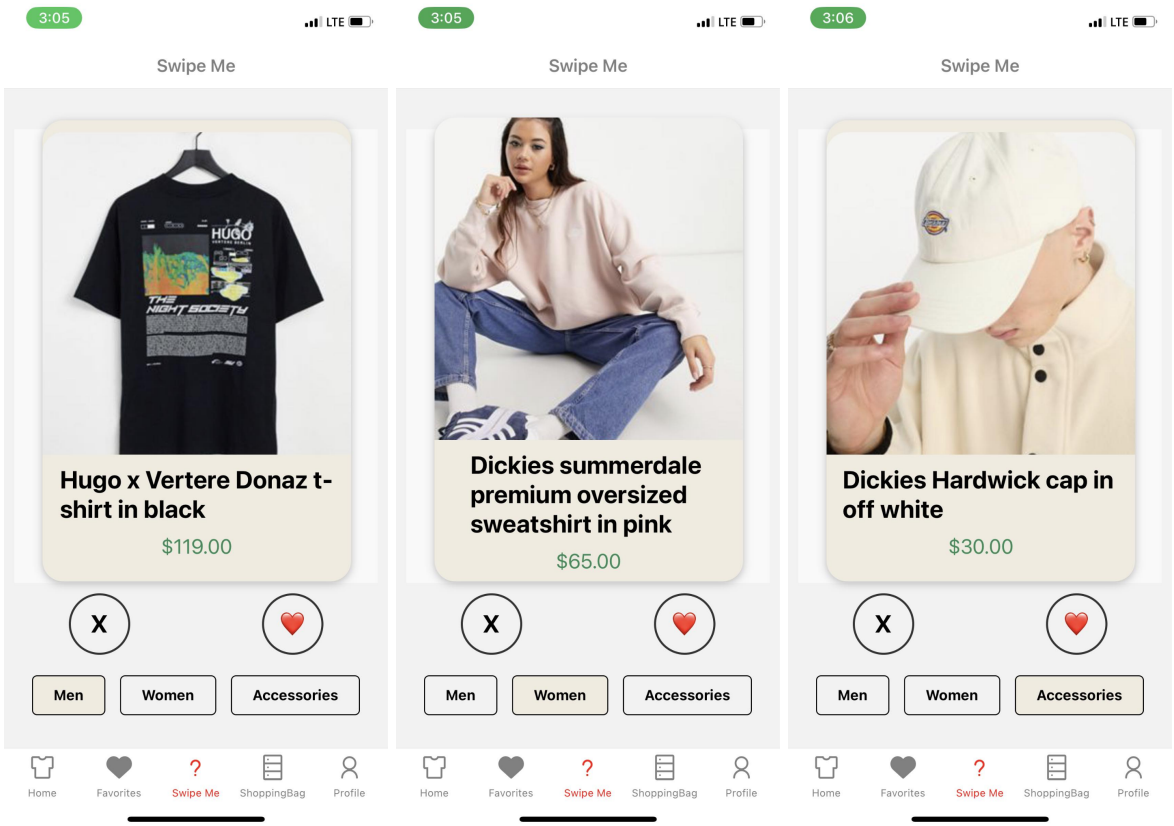
Catalog Screen:
Once the user clicks on the categories in the home screen it goes straight here and shows different categories of items. All our items are divided into men and womens and they are also filtered in subcategories for example if they click shirts and in the men category the user will only see listing for the men shirts, example goes on. We also implement a search bar where users can search for a specific item they want and users can click on it and add it to their favorites.



Users can also click on any of the listing and it will zoom it and they can add it to their favorites.



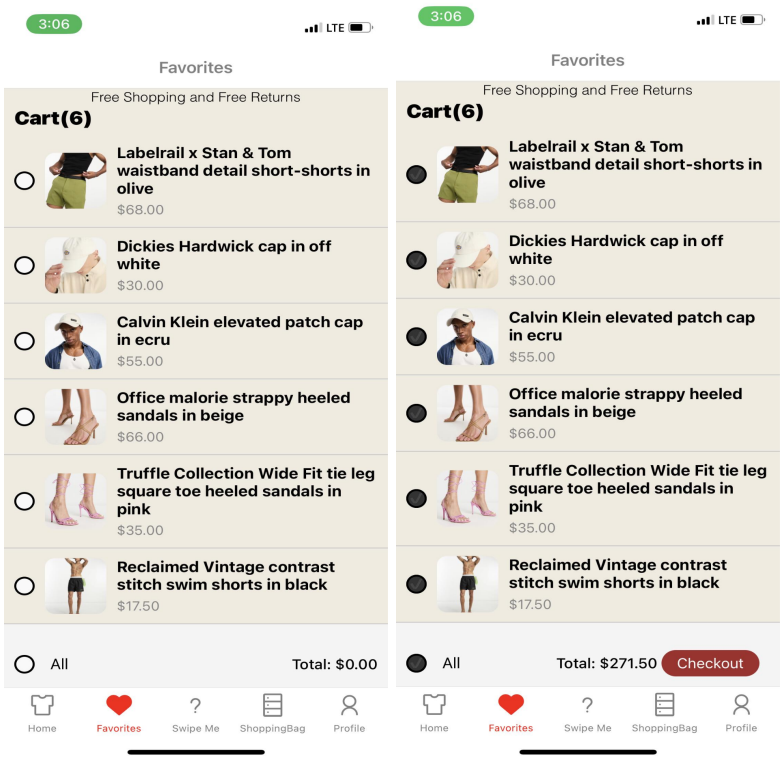
Swipe Screen:



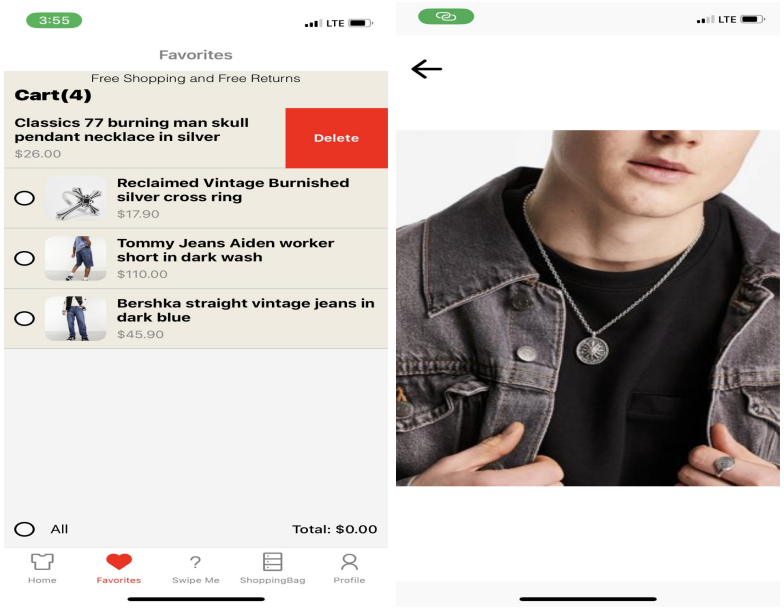
Swipe Screen is a unique feature we added that set us apart from the rest of the other online shopping sites. This idea is from the tinder app where swiping right means the user wants it and swiping left means the user does not. It is divided into 3 different categories (Men, Women, Accessories) that filter the products being shown. The algorithm for the swiping screen will be talked about below. Once an item that is swiped by the user to the right will go into the favorites screen.

Favorite Screen:

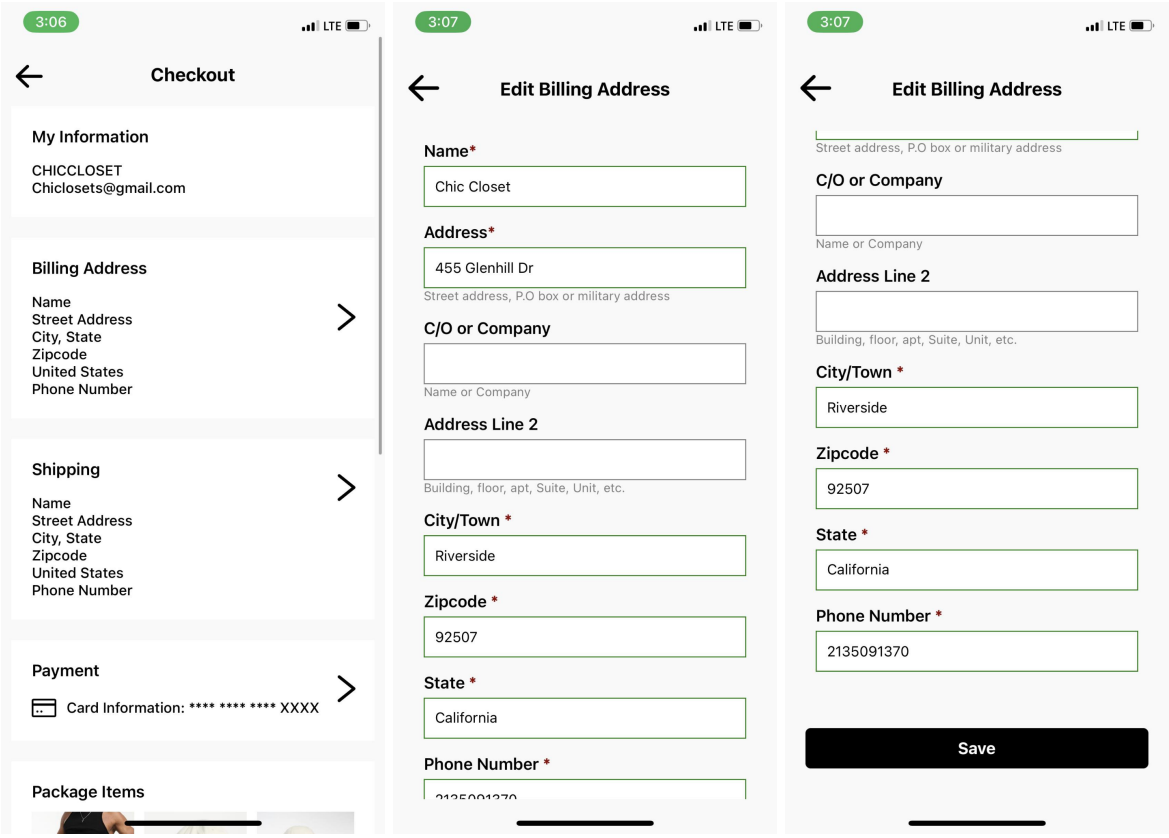
In this screen the items that are swiped to the right goes straight here, as well as the one that the user favorites in the catalog section.



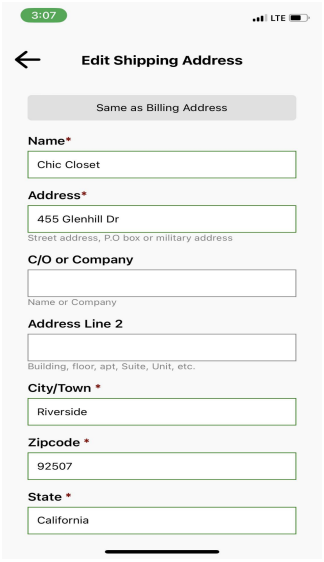
Once all the items are in the favorites we give the users an option to select the items they want to checkout. Users won't be able to see the checkout button unless at least one of the items is selected for checkout. The total sum of the selected items are also being shown.



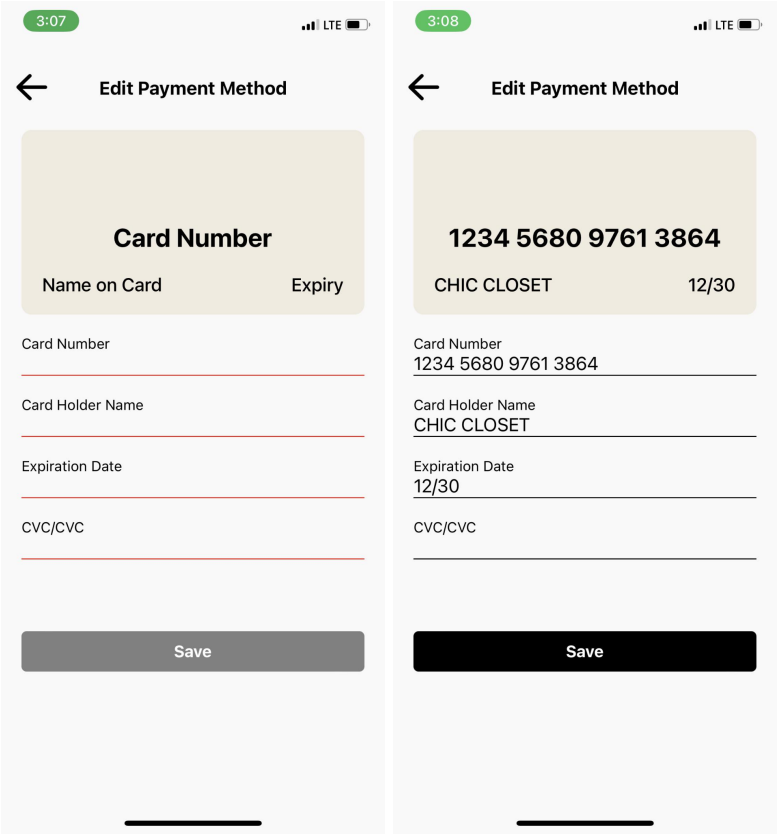
The user also has an option to delete it if they swipe the item on the right and can click on the image to show a close up look.
Once the user is ready to checkout after selecting the items they want to purchase it will bring them to an Order Confirmation page.



Users get to input their billing address, shipping information and their payment method. For the billing address we created a regex for most inputs in order to only accept correct formats such as zip codes and phone numbers where we limit zipcode to 5 digits and phone numbers to 10.

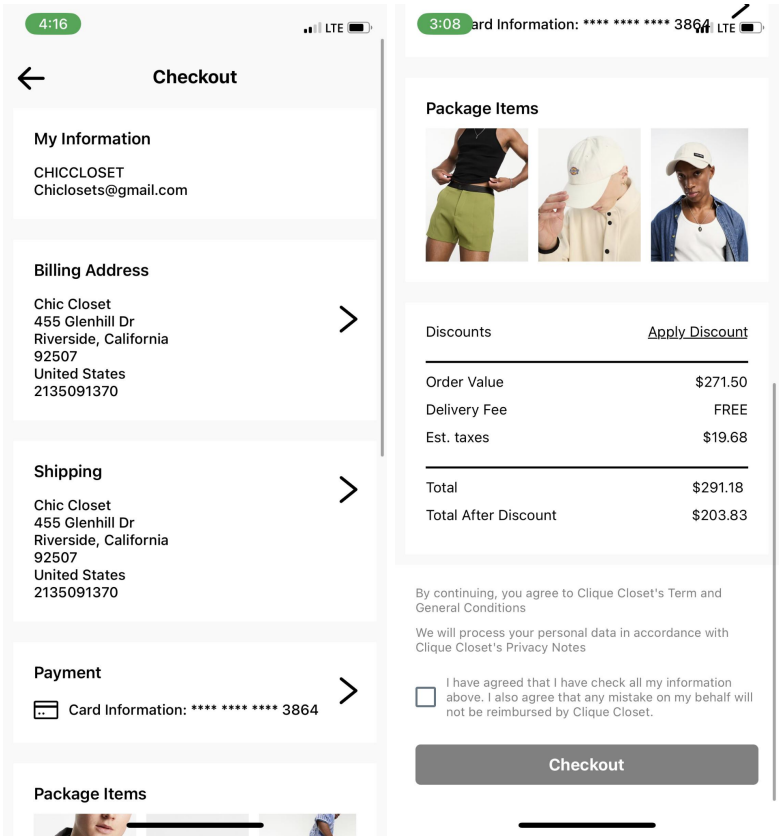


Sometimes users have a different address to their shipping address so we make sure they can input a new address if it's different but we also added a same to billing address button that will fill up their shipping address with their billing address if they want to.

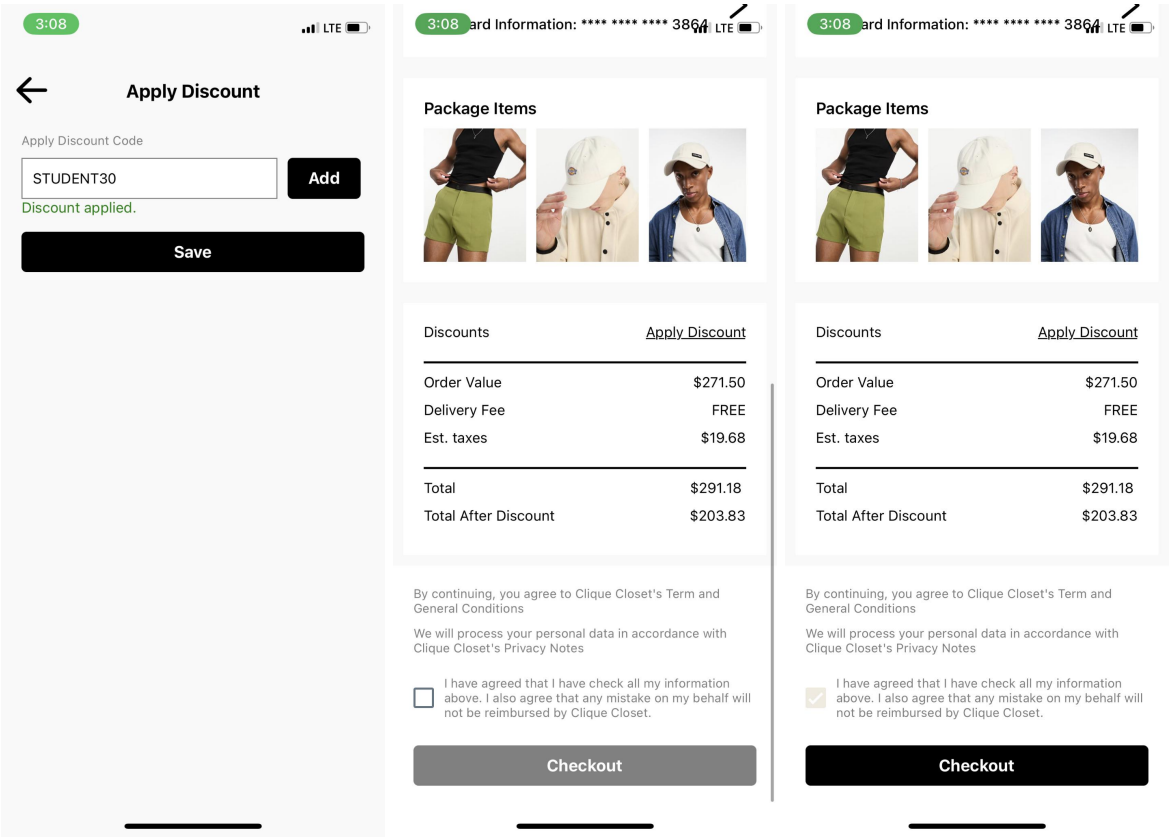


This is the payment method where users can input their card information. This one is just a dummy design and does not really check if the card is valid or not . We just added a regular expression for the formatting of the card number, the expiration date and the cvc to make sure we get the correct format we need.

Once the user filled up all the billing information, shipping information and the payment information it will be updated real time and will look like this:

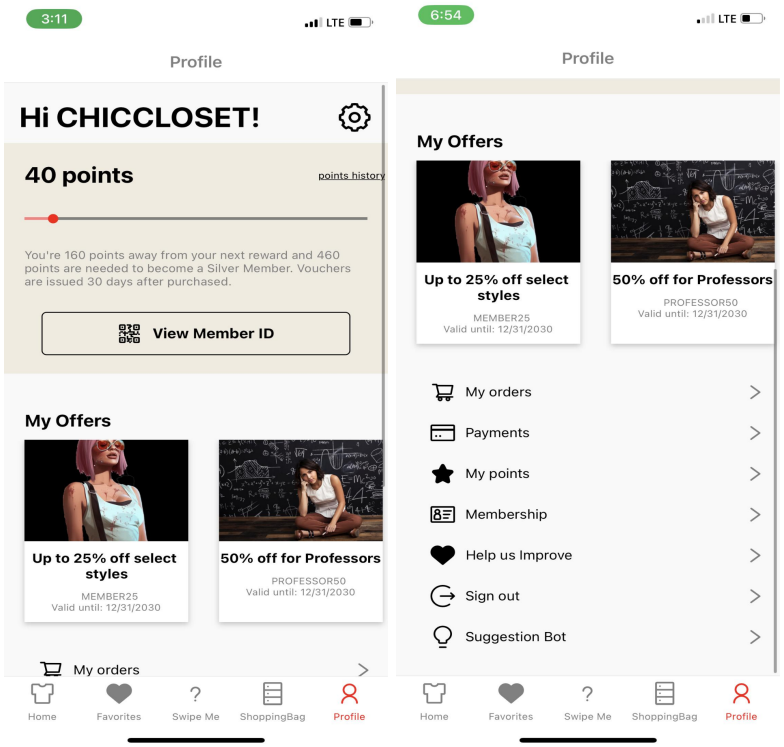


The user will see an updated Order confirmation page and they can also see the images of the items they selected to purchase. Users can also see the total before tax and how much the tax will be. The estimated taxes vary depending on the states they are in, therefore putting Arizona will give the user an estimated tax of 0 since they do not tax over there. Users can also apply a discount code where they'll be able to see by the profile screen.



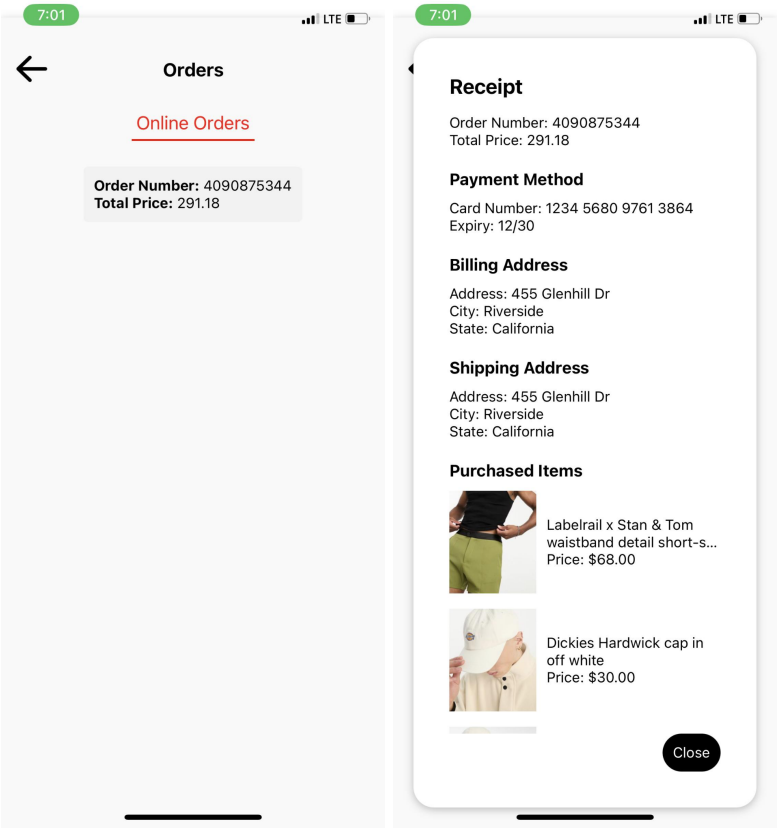
The user can apply a valid discount code, once they press add and the discount code is valid it will show a green text below stating that the discount is applied and if the discount code is invalid it will show a red text stating invalid code. Once they press save or the back arrow the user will see an updated total after discount which is the price they have to pay. Not having a valid discount code will show that the total after discount is the same as total. For the user to be able to click the checkout they have to read all the agreements and need to click on the checkbox agreeing that we are not going to reimburse them if ever they mess up their shipping address.

Profile Screen:
This is the profile screen! We can see here a heart warming Hi to the User. It will always be Hi {userName}. We can also see the points which go up every time the user purchases an item. They can also press View member ID that just outputs a dummy QR code which members can use to scan if ever we expand and do an in-store. Below that points card we can see some deals for the user which they can also scroll to the left to see more. They can use those codes to get the discounts when checking out.

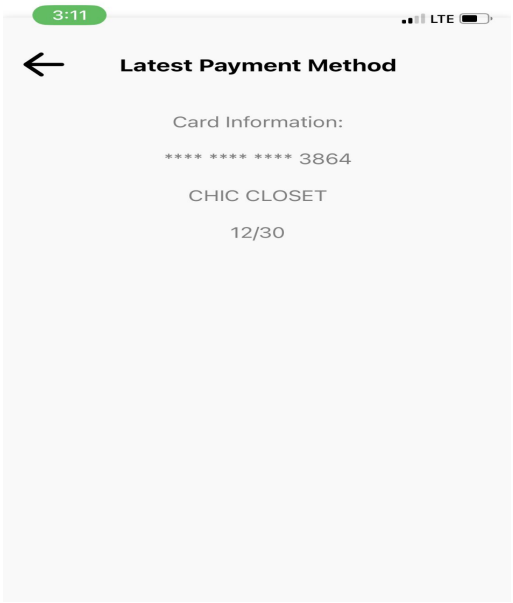


We can see different sets of buttons and all have different functionalities.
Let's start with the one that's too obvious: the **Sign Out** button. With the sign out button the user will be logged out and it will return to the sign in page. In the sign in page once the user signs in again we guarantee that they will still see what they left off. Everything in the favorites will still stay there as well as the rest of the purchased history and even the index swiped so that the user does not have to swipe with what they saw before logging off the app. Everything is stored in the database.

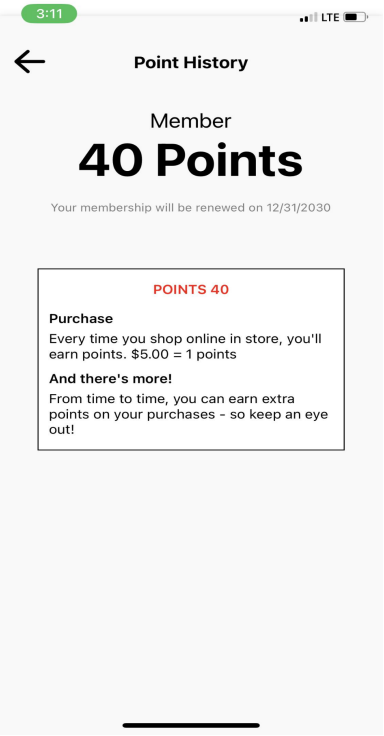
Next is, ***My Orders***, where they can see their purchase history by showing the items, total price, and the billing address and payment method used.



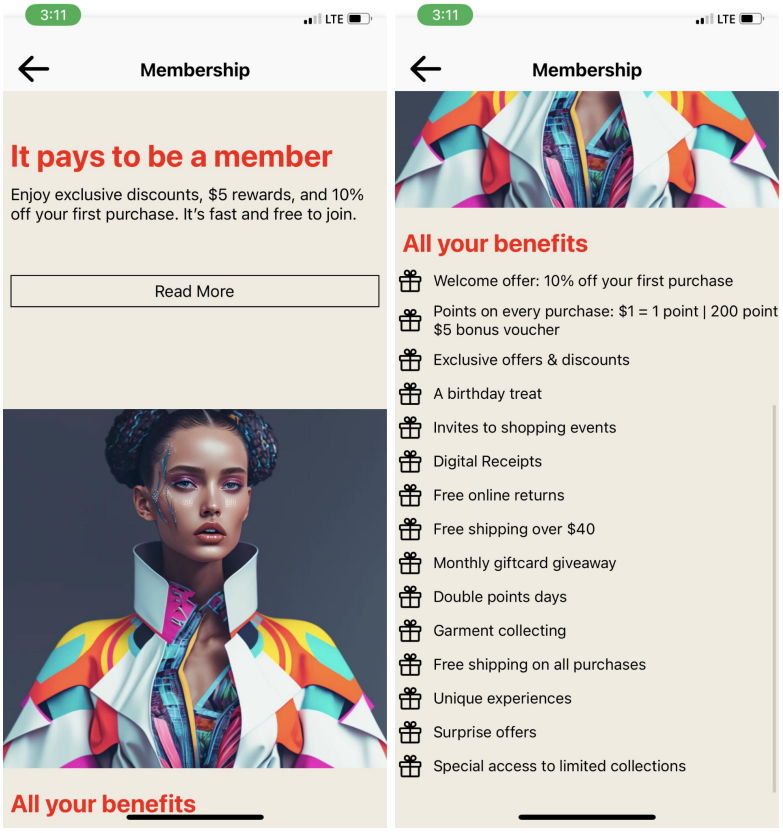
Once a user clicks the **Payment Method** button they will see the latest payment method they used for their order. If they have not ordered anything it will say that No Result.



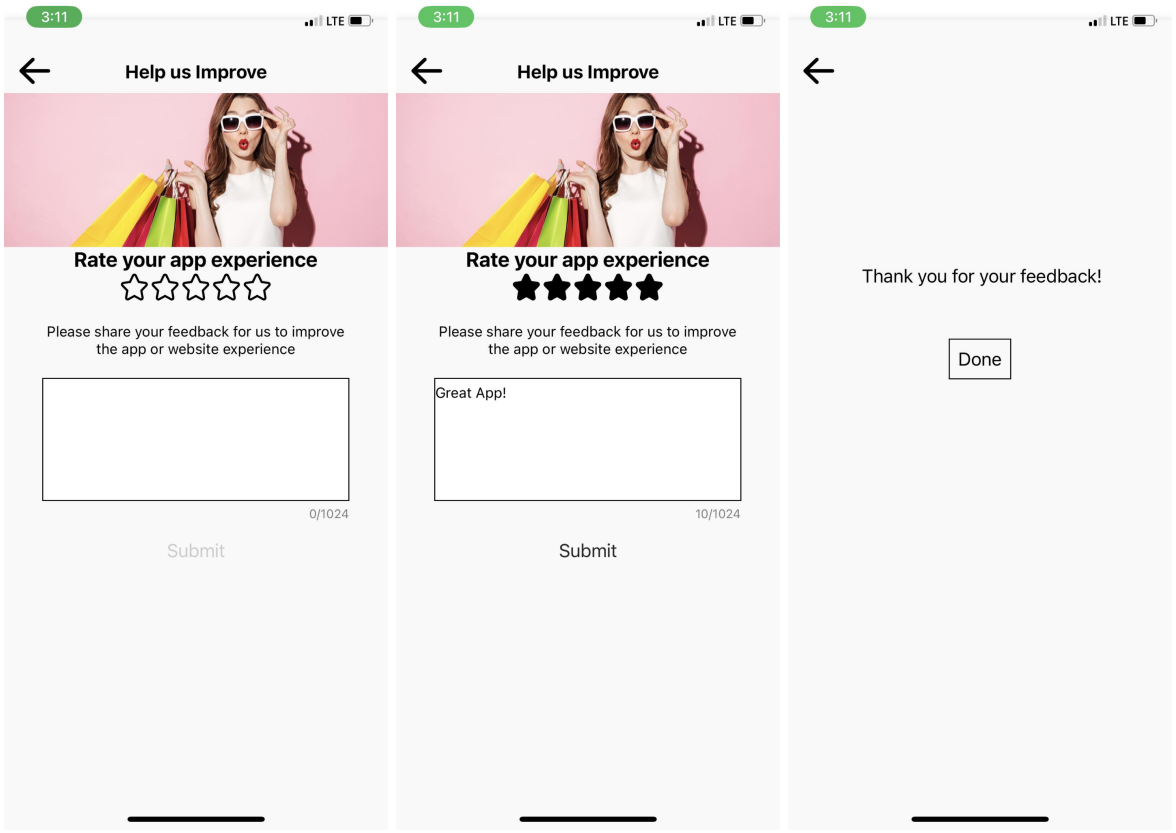
The user can also see more information about their Points, by clicking **My Points** they will see



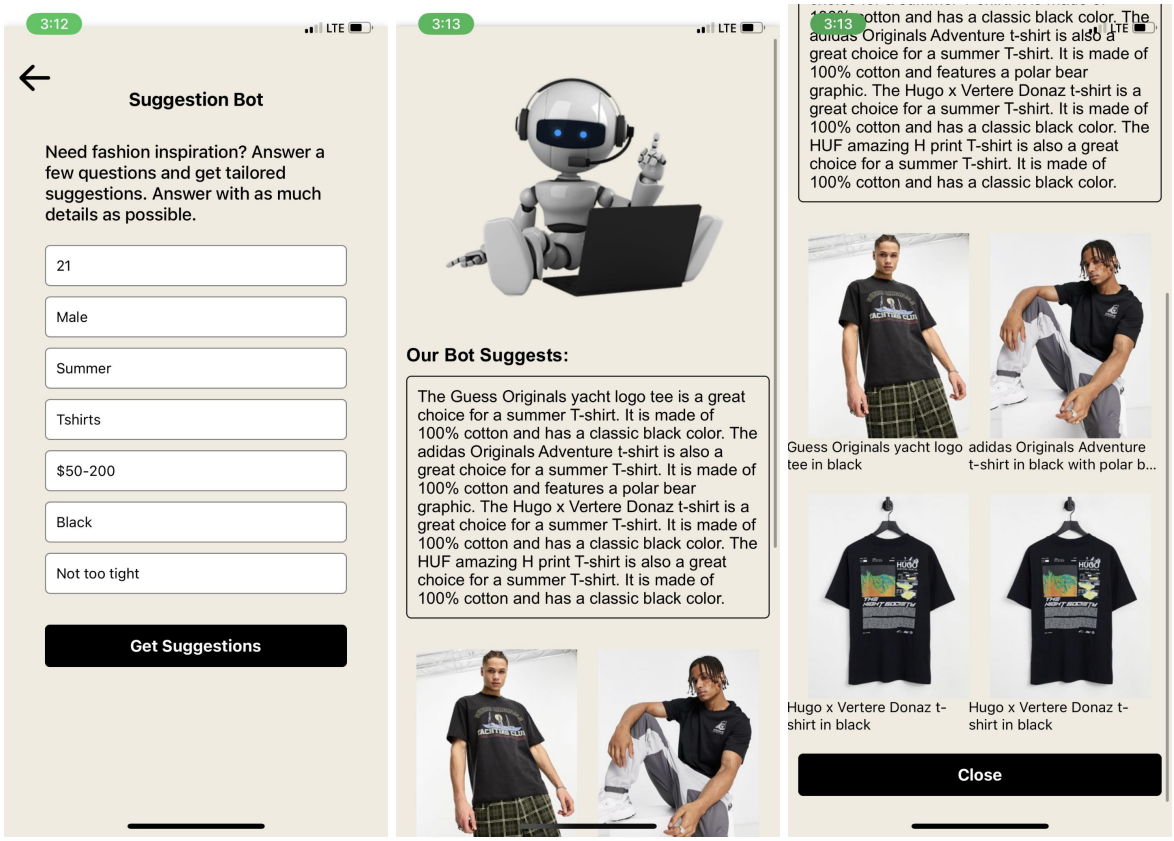
They can also see what their membership includes by clicking the **Membership** button. Most of this information is just added for the aesthetic view and will not be implemented.



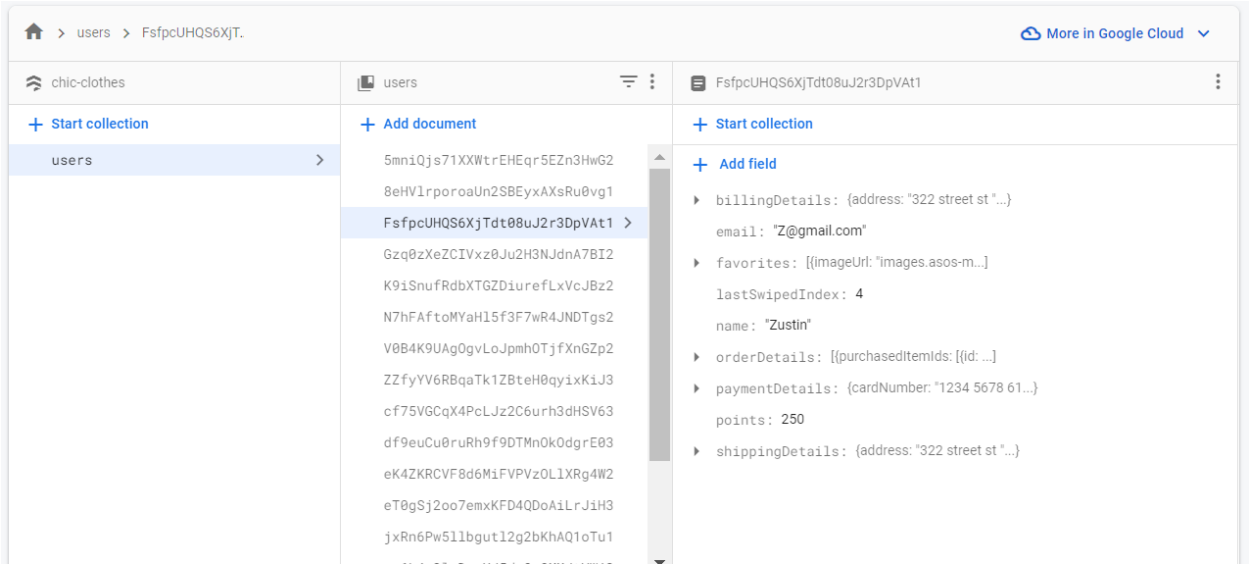
The user can also give us some feedback on how the app is by pressing the **Help Us Improve** button. For this one user can pick how much start they think the app is and can only submit if there are some comments in the textbox. The text box also has a limit of 1024 characters.



Lastly, users can get outfits suggestions from our **Suggestion Bot** by using chat GPT api and by adding an algorithm to make it more efficient that will be described in the algorithms part of this section. The user will be asked some information about what they need and within those question the Bot will try the best fit by giving out a description and outputting some items in our list of items that fits the description of what the user needs



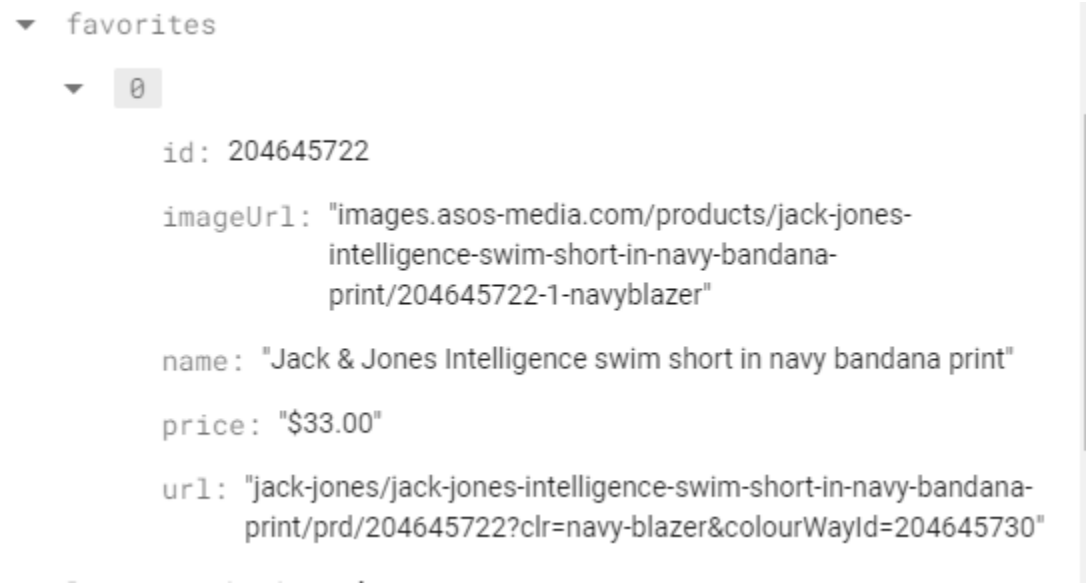
DB schema design



Each users that signs up they get to have their unique ID and within each user we are saving their **billing address** that contains:

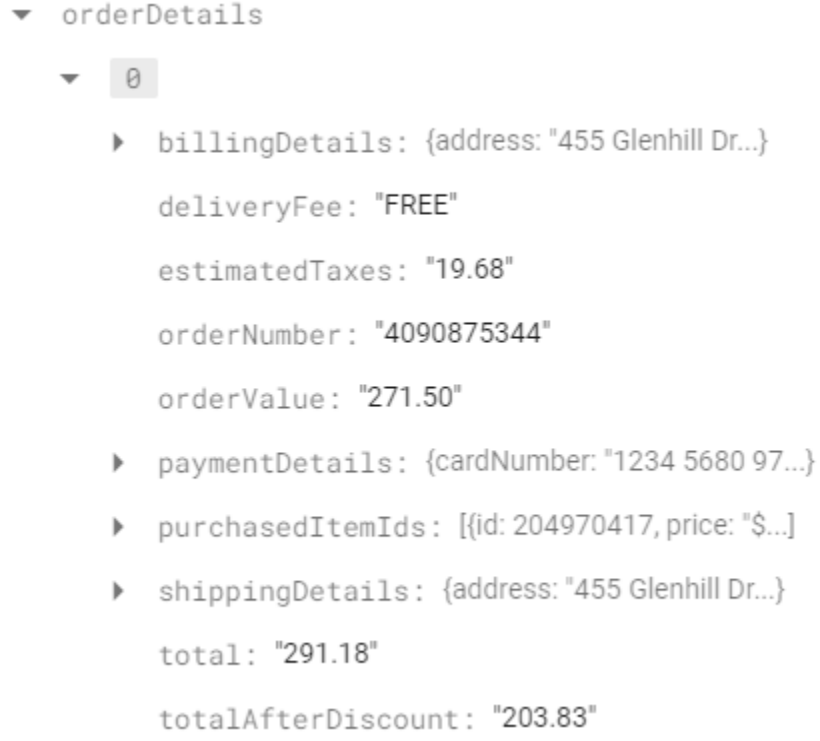
Email that contains the email they used to sign up.

Favorites array that stores what they currently have in their favorites screen:



Last swiped index so that they do not repeat the items in the swiping feature.

Name contains the name of the user.
Order details that contains what the user selected to checkout:



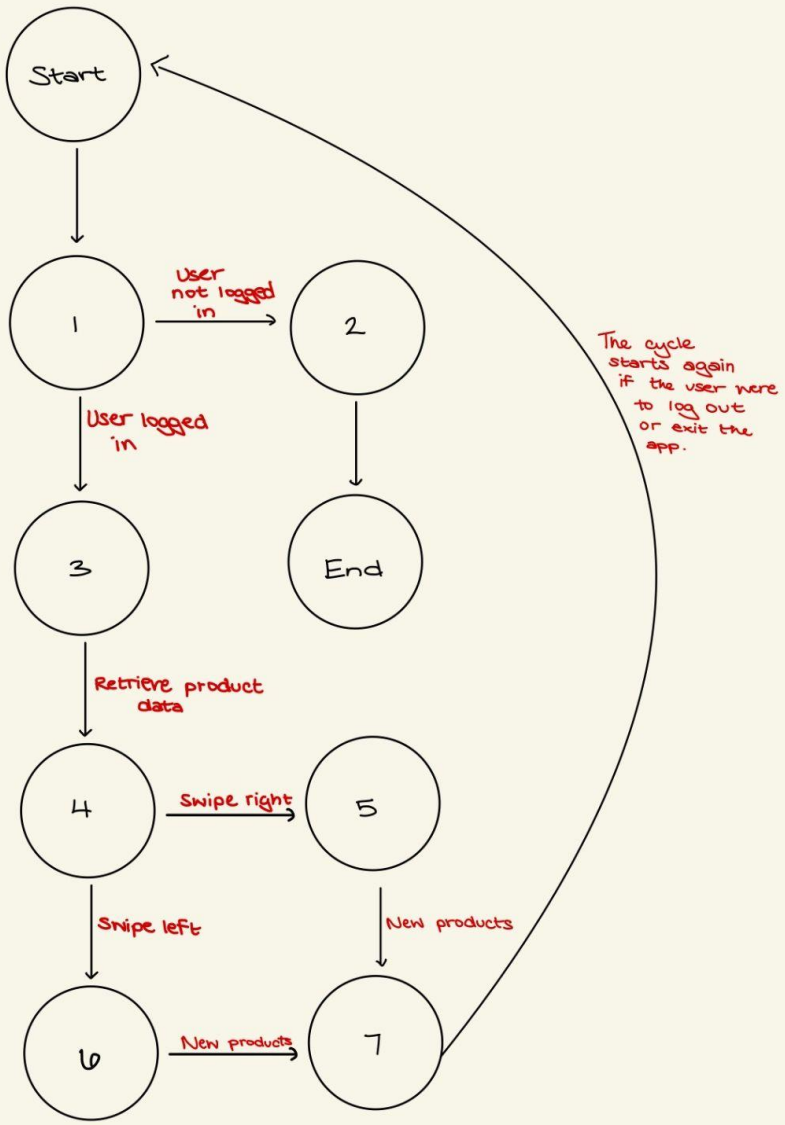
It contains all the items the user checks out which are in purchasedItemIds with the billing,shipping and the payment method with the total prices of that order too.

Payment Details that contain the card information of what the user entered.

Points store the current points of the user which they get from when they purchase an item.
Shipping details stores the address of where the package the user wants it to be delivered at

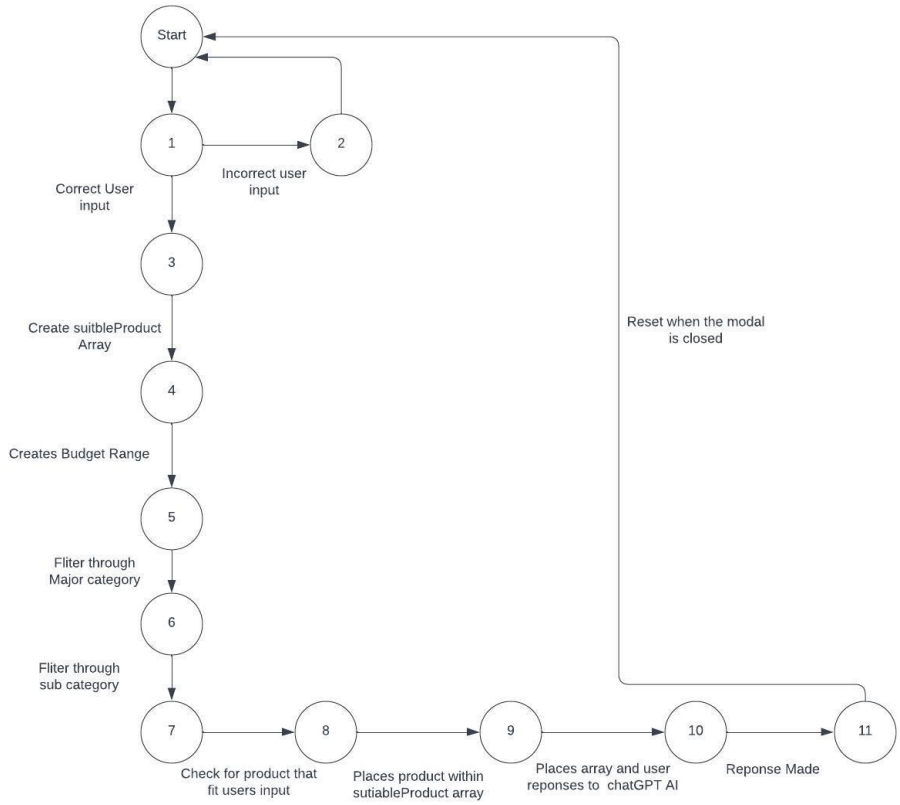
Algorithms

Control flow graph of major algorithms in our code (explanation/pseudocode)
Swipe Screen: (based on the handleSwipe function)



1. Checks if the user exists
2. If not then the session ends
3. If the user exists we move forward to retrieve product data
4. If the user swipes right
5. Add product to favorites
6. Ignore the product
7. New products will appear

FliterData Function for SuggestionBot



Data Structure

Major data structures in our code (Show snippets of actual code)


```
{
  "majorCategory":"Women",
  "subCategories": [
  {
   "categoryName":"Women Jackets & Coats",
   "products":[
      {
         "id":201778145,
         "imageUrl":"images.asos-media.com/products/levis-4-pockets-belted-denim-jacket-in-khaki/201778145-1-khaki",
         "name":"Levi's 4 pockets belted denim jacket in khaki",
         "price":"$101.00",
         "url":"levis/levis-4-pockets-belted-denim-jacket-in-khaki/prd/201778145?clr=khaki&colourWayId=201778173"
      },
      {
         "id":204444763,
         "imageUrl":"images.asos-media.com/products/bolongaro-trevor-acid-wash-denim-worker-jacket-in-purple-part-of-a-set/204444763-1-purpleacid",
         "name":"Bolongaro Trevor acid wash denim worker jacket in purple - part of a set",
         "price":"$78.00",
         "url":"bolongaro-trevor/bolongaro-trevor-acid-wash-high-waist-jean-and-denim-jacket-set-in-purple/grp/205417471?clr=purple-acid&colourWayId=2044
      },
      {
```


All of our product data is stored in a JSON file. We have majorCategories in there which are Women, Men, and Accessories. Products that fall into these categories are then broken up into subCategories. By storing our products in a JSON file we are able to have direct access to our own data.

Design Principles

Having the SOLID principles in mind, we looked through our code to pinpoint some examples of where we would apply SOLID principles.

- Single-responsibility principle
  - **SwipeMeScreen.js** is an example of SRP because in our code we separated data fetching, UI render, and authentication into different functions. Each function has its own single responsibility.
  - **data.JSON** is an example of SRP because its single responsibility is to store all the product data.

Refactoring

Original:

```
const checkPass = () => {
   // homeUsed = false;

   // console.log(homeUsedCheck);
   //spanx
   if (stat == prevStatus) {
     setPrevStatus(stat);
     if (passIndex == 11 && !homeUsedCheck && homeUsed) {
       // console.log("spanx");
       setcategoryState(1);
       setIndex(1);
       setHomeUsedCheck(true);
     }

     //dresses
     if (passIndex == 44 && !homeUsedCheck && homeUsed) {
       setcategoryState(4);
       setIndex(1);
       setHomeUsedCheck(true);
     }

     if (passIndex == 88 && !homeUsedCheck && homeUsed) {
       setcategoryState(20);
       setIndex(0);
```

```
        setHomeUsedCheck(true);
      }

      if (!homeUsedCheck && homeUsed) {
        lastPassIn = passIndex;
        setcategoryState(passIndex);
        setHomeUsedCheck(true);
        // console.log("homeUsedCheck");
        console.log(homeUsedCheck);
      }
    } else {
      if (passIndex == 11) {
        // console.log("spanx");
        setcategoryState(1);
        setIndex(1);
        setHomeUsedCheck(true);
      }

      //dresses
      else if (passIndex == 44) {
        // console.log("dresses");
        setcategoryState(4);
        setIndex(1);
        setHomeUsedCheck(true);
      } else if (passIndex == 88 && !homeUsedCheck && homeUsed) {
        setcategoryState(20);
        setIndex(0);
        setHomeUsedCheck(true);
      } else {
        setcategoryState(passIndex);
      }
      setPrevStatus(stat);
    }
  };
```

Refactored:

```
const checkPass = () => {
  const handleHomeUnused = () => {
```

```
    if (!homeUsedCheck && homeUsed) {
      setHomeUsedCheck(true);
      switch (passIndex) {
        case 11:
          setcategoryState(1);
          setIndex(1);
          break;
        case 44:
          setcategoryState(4);
          setIndex(1);
          break;
        case 88:
          setcategoryState(20);
          setIndex(0);
          break;
        default:
          setcategoryState(passIndex);
          break;
      }
    }
  };

  if (stat !== prevStatus) {
    setPrevStatus(stat);
    switch (passIndex) {
      case 11:
        setcategoryState(1);
        setIndex(1);
        setHomeUsedCheck(true);
        break;
      case 44:
        setcategoryState(4);
        setIndex(1);
        setHomeUsedCheck(true);
        break;
      default:
        setcategoryState(passIndex);
        break;
    }
  } else if (stat === prevStatus) {
    handleHomeUnused();
  }
};
```

This refactor code just removed some of the redundant logic and made it more readable to other developers.