SWEN325 Assignment 1 Report

Architecture

The receipt bowl app utilizes a relatively thin client, with the majority of data storage and processing occurring within the backend web service/database. All pages in the app interact in some way with the backend, to either authenticate users or retrieve/post data to the database.

A reduced threat to security is the first advantage of utilizing a thin client. As most data is stored on the cloud, malicious users must first break past the back end's user authentication system, before gaining access to any data. In the receipt bowl app, user authentication is managed via the web server, where on successful authentication the user is granted access to view and append group, and purchase data. Although not in this implementation, the architectural model also means users could only be given access to groups and by extension group purchases, which they have been added to.

Due to the nature and purpose of the app, utilizing a thin client reaps the benefits commonly associated with thin client implementations, and diminishes the disadvantages also associated with them. The primary purpose of the app is to store and process data for a group of users. All users need constant access to the data, and updates made by one user, should be instantly available to all users. To meet these requirements, the data must be stored in an online database. This means that the common disadvantage associated with thin clients, of always requiring internet access becomes irrelevant. The thin client architecture also means that the app is lightweight and portable. As a result the computational and storage requirements for the app are minimised, decreasing the load on the phone's internal storage and battery.

There are several components within the app where data is stored and processed on the client side. An example of this is the group creation component, which stores user group member data in a cache until the group is created. There are several reasons why this is done. The first is to reduce the number of writes made to the database. When members are added to a new group, they are cashed on the creator user's client. They are then submitted all at once when the new group is complete, rather than when each member is added. This not only reduces the number of writes necessary to create a group, but also the number of reads and deletions required to update it each time a member is added or deleted. This is beneficial, as databases such as firebase charge

based on reads, writes and deletions. The cache also makes the new group page more performant, with less waiting required for documents to be appended to a localised cache, rather than an external database, improving performance, and the overall user experience.

The source code for receipt bowl is minimal, and consists of two main element types; pages and services. Each page serves a different purpose within the app design, however they all are populated by and communicate with the backend web server and database via one of the two services. This reflects a thin client architectural design, as each page focuses on user experience, but defurs to the back end for most data storage and processing.

External Component: FireBase

Firebase was utilized to provide an authentication service and a persistent data storage service, for the receipt bowl app. As the architecture of the app is designed around a thin client model, the firebase services must do the bulk of data storage and processing, and is therefore an indispensable component of the app.

Authentication is managed using the firebase authentication component. Users can authenticate using their email address, and submit their name and a password into the system when logging in for the first time. User data is also stored in a users collection in a firestore database. This allows other components within the app to more easily reference user data. The collection can be accessed by the client using the user service, where user data can be added, removed and retrieved through a set of functions. When a user authenticates for the first time via the firebase authentication component, that users data is appended to the user collection. Their assigned UID, name and email address are all stored in a document in that collection.

Group data is stored in another firestore collection. This collection can be accessed by the client, using the group service. Here group data can be added, removed and retrieved through a set of functions. Each group document within the group collection contains the name of the group, a list of member uid’s and a sub collection of transactions for the group. Within the app client, group documents are added to the collection on the add group page. This creates a new group document with the users selected group name and a list of uids of the users added. This is one example of the use of the users collection. Although not completely implemented, each users uid could

be referenced and added by searching the user collection for their name or email. Group collection data is retrieved by the app client on the group list page. Here the name field of all group documents are displayed on a list of button components. When clicked this gives the user access to the purchase sub collection for that group.

Purchase data is stored in a subcollection of each group document in the group collection. This collection can be accessed by the client, using the same group service used for accessing the group collection. In this case however, the relevant functions can only add and retrieve the purchases of the group. Similar to the group collection, each purchase collection has purchase documents added on the add purchase page. This creates a new purchase document with the users selected purchase name and cost. Purchase collection data is retrieved by the app client on the purchase list page for each group. Here the purchase name and purchase cost fields are displayed as list elements, to be reviewed.

Reflective Report

As my first project in ionic, and first time working with angular, my goal for this project was to create a basic but functional app for managing shared costs between a group of people. To achieve this goal my plan was to simply dive into development within the ionic framework, and iteratively add features to the app up until completion.

On starting the project I immediately found working with the ionic framework very easy and rewarding. Ion components could be added to a page very easily, and configured based on the abundant documentation available for each of the different ion components. The first problems came when extending the project past the user interface of a single page. As I had no experience with an angular project, the structure of the project, with multiple different files per page were very confusing, and my lack of experience using typescript added to this issue. This was one of the most significant hurdles for me to get over in this project. In particular routing between pages and accessing external libraries/frameworks were difficult problems to solve. As a result of these issues, I have learned a lot about the angular framework and how it works, by studying a stream of online tutorials, documentation and other resources.

Once I had a basic understanding of the ionic framework, I moved on to attempting to implement a user authentication system and a firestore collection via firebase. The authentication component went relatively smoothly. This was because I used an open source component called firebase UI. Once imported, the firebase component could be placed into a page, and took care of all authentication components, including communicating with the database, and registering new users. Calls from this component

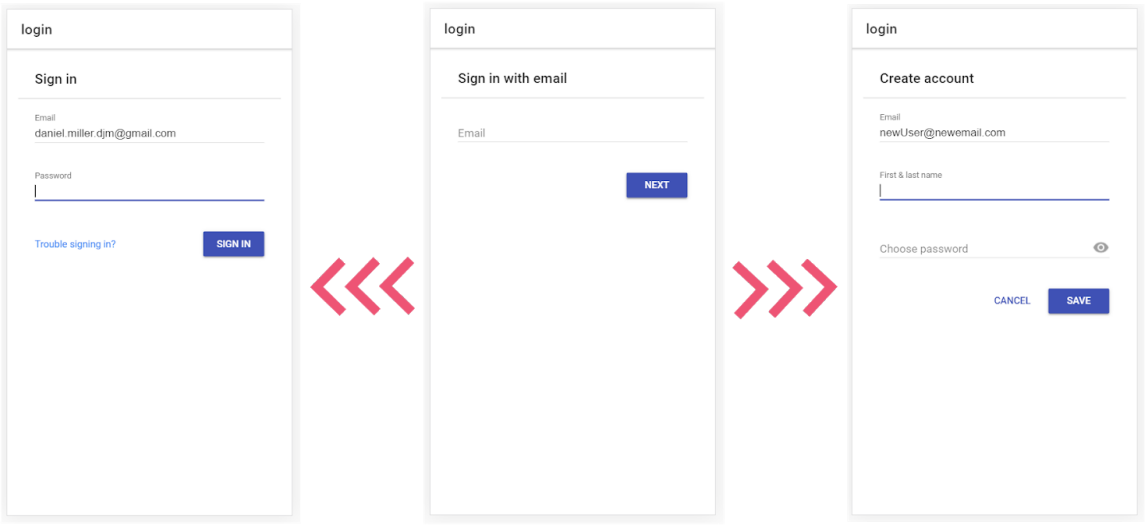
could then be used to route the user onto a new page. The firestore component was not as easy as this however. Communication to the firestore was done through two services in the ionic application. One for a group data collection, and one for a user data collection. The group service worked well after a long implementation process, and is used to append group data, to the collection, and retrieve all documents within the collection to display within several ion-list components. The user service however required the retrieval of a single users data, from their UID. Although a lot of work into this component, due to time constraints it was never completed.

Overall I have found ionic to be a very powerful tool for quickly creating, good looking mobile applications. Due to my lack of experience with angular, it is hard to comment on it, but I have found it overall very complicated, with a range of different files and components needing to be taken care of. The external component I implemented was very difficult to work with at first, however towards the end of development, my understanding has increased dramatically. In retrospect, due to my lack of experience around these frameworks, I think I was overambitious in my app description. I would have had more success if I had chosen a simpler concept, and focused on the basics of implementing an ionic application.

UX Decision 1: User input management

When requesting a user to fill out forms or enter information, it is important not to overwhelm the user with too many inputs or options at the same time, and to guide the user through the process with an obvious and natural progression. There are two main pages within the receipt bowl app where the user is required to enter information. The login page, and the new group page. Both of these pages handle the input process in a different way, aiming to reduce the cognitive load required from the user, to make the overall process quicker and easier.

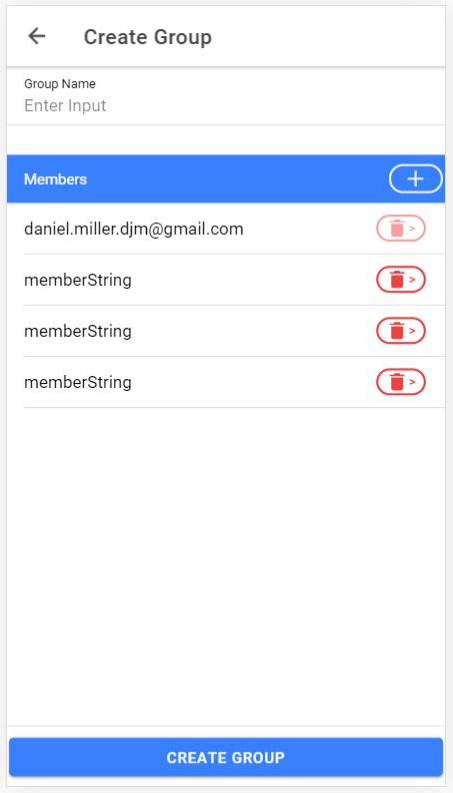
The login page aims to make the authentication process easier, by only displaying relevant fields for each step of the login process. The fields for each consecutive step are rendered based off the input of the previous step. In the case of the login process, the first screen asks for the user's email. If the email has already been registered, a single field for the password will be presented to the user. If the user has not registered the user will be presented with the multiple fields required for registering. This declutters the UI, and makes it very clear what the user is supposed to do at any one time. Removing any guesswork or problem solving on the part of the user, reduces the effort required to log in to the app, and improves the overall user experience.



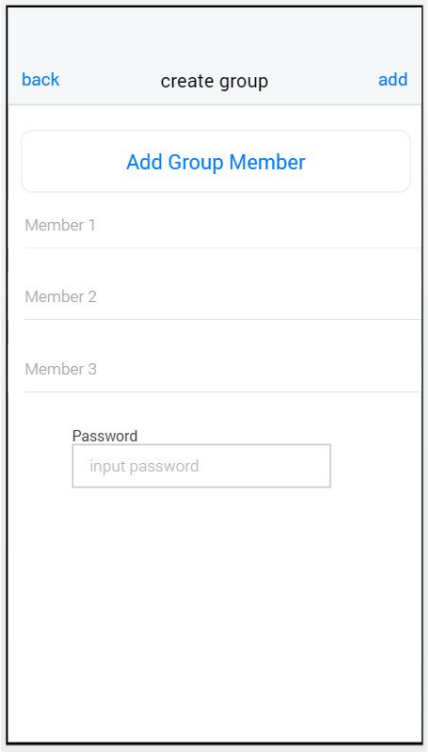
An alternative option for this was to have the user take manual control over the process and display all fields to them. For the login process, this would involve manually selecting to either login or register, then fill out the relevant forms. This is not ideal, as it puts more work onto the user. Although still a relatively simple process, taking away as must effort as possible from a repetitive and boring task such as logging in, can have a significant impact on improving user experience.

The new group page manages the flow of user input in a different way. Unlike the login page, it displays all necessary inputs at the same time. However, in order to declutter elements and clarify the input process, the page is split up by headers and ordered from top to bottom. Users are used to reading and writing from top to bottom, so will naturally

tend to do this automatically. The app takes advantage of this, and creates a natural flow of steps for the user to proceed through from the top of the page down. Finally the submission button is located at the bottom of the page as it is used last. This design is a benefit for user experience, as the flow of the page matches the way users will naturally complete a form. Resulting in a smooth and easy process.



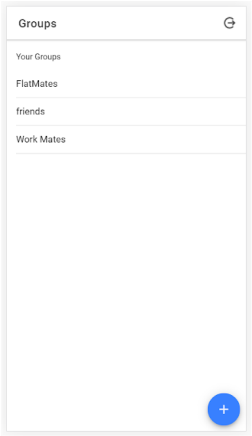
An alternative to this design was to place items elements randomly, or in an alternative non-linear layout. The problem with this, is that the order in which the user is intended to complete each element is not clear. This can result in more effort for the user, ultimately making the app annoying and hard to use.



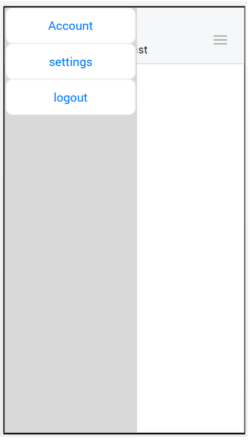
UX Decision 2: Simplistic and intuitive design

In order to make the purpose of the receipt bowl app clearer, and its use more intuitive and easier to understand, the apps overall layout and design is very minimal and simplistic. The total number of components on each page is minimised, with each supporting a single specific function. This design philosophy results in better user comprehension and engagement with the apps core functionality, and reduces the possibility of users becoming confused under the weight of a more complicated interface.

Current Implementation



Alternative Implementation



An example of this is the log out button to the right of the header in the group list page. This element has a single function; to log the user out of the app, and its purpose is intuitive to the user due to the icon, and its location on the header. An alternative to this logout component was to use a side menu. A side menu could have incorporated more functionality under a single button, including the logout button and other optional components, such as user settings etc. Popup menus are also common in many apps, and are thus intuitive for users to use. In the end the basic logout button was chosen over the pop out screen due to simplicity. The extra functionality provided by a popup menu was not deemed sufficient to warrant its implementation, as the additional complexity that comes with it increases the learning curve for new users, and could negatively impact the overall experience.

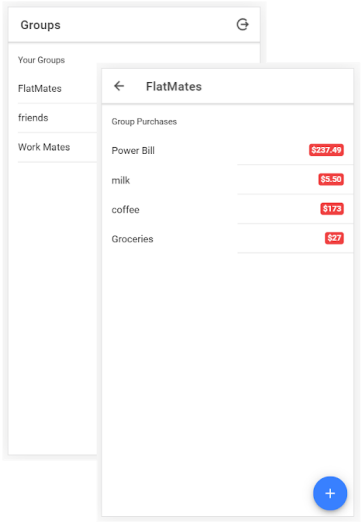
UX Decision 3: Utilizing user's prior knowledge

The receipt bowl app uses familiar screens, components, and repeated layouts, to help users apply both general logical knowledge and prior app experience in using the receipt bowl app. In implementing components such as these, new users can cut down their learning curve for the receipt bowl app, and begin to take full advantage of the apps features sooner.

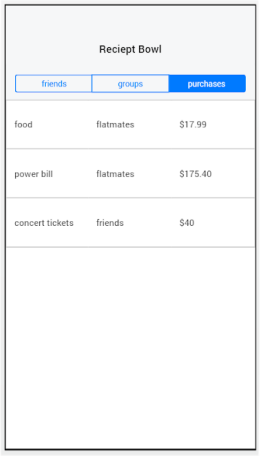
An example of this is the page structure and navigation utilized within the receipt bowl app. The pages which display data within the app are nested. This directly resembles the file system utilized on most computers, where directories contain subdirectories and files. Receipt bowl is similar to this with its group list and purchase list pages. The group list page contains a set of groups, these groups in turn contain a purchase list, which

contain a set of purchases. This nested page structure provides users with a familiar navigation system, that they are already comfortable with using. In the receipt bowl app, users can not only navigate inward by clicking on a list item or subdirectory, but also return to the previous layer by clicking on the back button. These are present in the upper left corner of each relevant page, and work the same as back buttons found in other applications and file systems.

Current Implementation

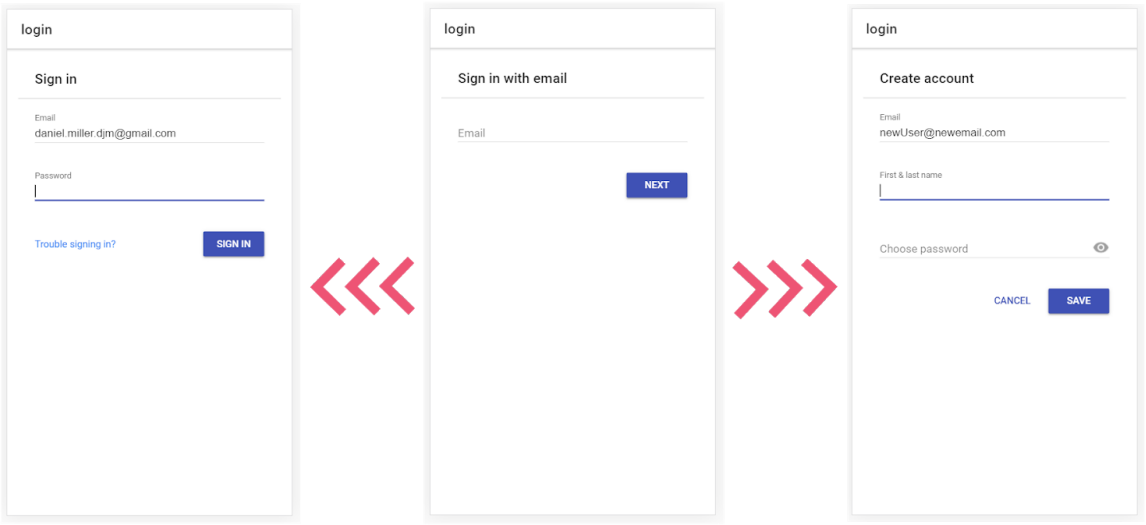


Alternative Implementation

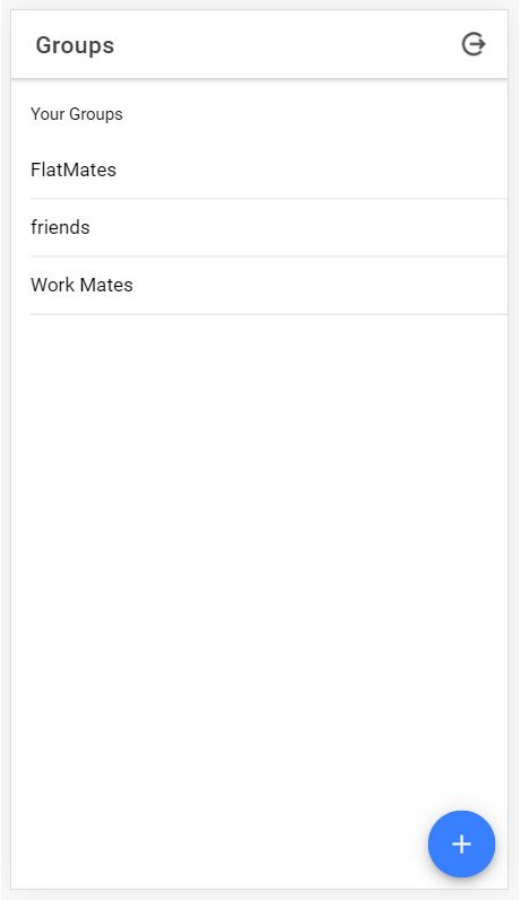


An alternative to this nested file structure was to store all data in a shallow page structure, with both groups and purchases available from the same page. This could allow for more efficient access to all pages, given that the user does not need to transverse into the nested files, and back out again to find what they are looking for. This structure is more confusing however, and increases the learning curve required for new users to pick up the app. If users cannot easily learn to operate something, they will often give up on it quickly for an alternative method. For this reason the traditional nested page structure was chosen, even with a small loss in efficiency.

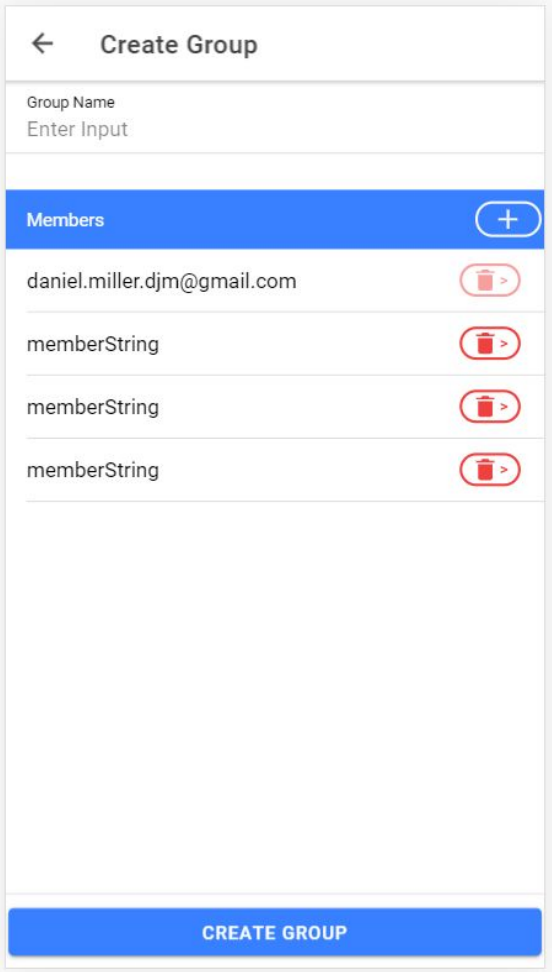
Appendix



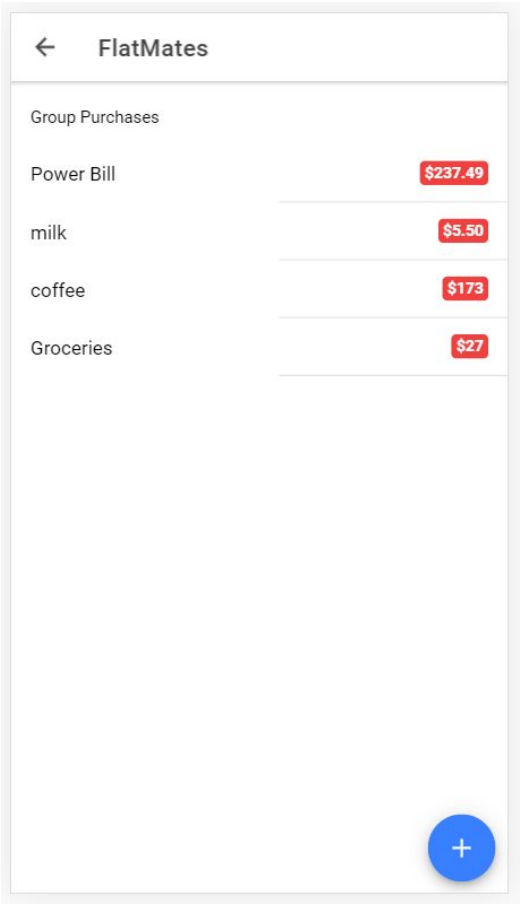
The login page is responsible for authenticating both new and existing users. The page consists of a header, indicating the role of the page to the user, and the firebase login element. The login element only presents relevant, and currently needed fields to the user at any time, removing any ambiguity about the login process. The login element first requests the email address of the user logging in. If the entered address has already been registered, a password field is added, along with a link for resetting the account password. If the email address has not been registered, a name and password field are added to the element, for the user to register.



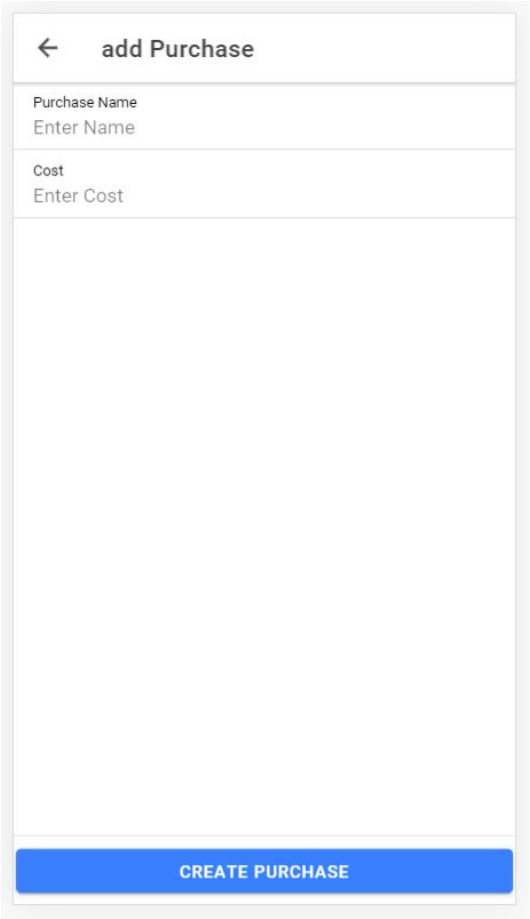
The group list page gives the user access to each of their groups, and allows them to create new groups. The main body of this page is taken up by a list, containing each group the user has access to. Each group element on the list is a button, which can be clicked to open the purchases page for that group. The header of this page contains a title, indicating to the user where they are within the app, and a logout button, to return to the login page. An add button is located on the bottom right corner of the screen, which redirects the user to the add group page, in order to create a new group.



The add group page allows the user to create a new group and add it to the database. The header of the page indicates to the user where they are in the app, along with a button to quickly return to the group list page. An input for the group's name is located beneath this. The input is labeled, and contains temporary text in order to make its purpose more clear. The list of group members is located beneath this, and takes up most of the remaining screen. The list has a colored header, to both describe what the list is, and segregate the list from the name input. The header also contains an add button used to append a user to the member list. Beneath the list header is the list body. This contains an element for each member in the group. Each element is labeled with the users email address and has a delete button, to remove that member from the list. The user creating the group is added to this list automatically, however their list entry has a disabled delete button. The final component is the submission button, which submits the group and returns the user to the group list page. This is large to attract attention and make its purpose clear. If the name field has not been correctly filled out, clicking this button will result in an alert asking the user to enter a valid name.The overall design of this page has the user working from top to bottom, filling in the components before adding the group. This is intuitive for users, as they are used to this flow in other areas, such as reading etc.



The purchase list page allows users to review purchases that have been submitted to the group. The header of the page indicates to the user where they are in the app, along with a back button to quickly return to the group list page. The body of the page is taken up by a list, containing each purchase for the group. Each list element contains the purchase name/description, and its price. The price tags are highlighted in red, to make them easier to read, and stand out. An add button is located on the bottom right corner of the screen. This redirects the user to the add purchase page, in order to add a new purchase to the group.



The add purchase page allows the user to create a new purchase and add it to the database, under the relevant group. The header of the page indicates to the user where they are in the app, along with a button to quickly return them to the group's purchase list page. Inputs for the purchase name and purchase cost are located beneath this. Each input is labeled, and contains temporary text in order to make their purpose clearer. The final component is the submission button, which submits the purchase and returns the user to the group's purchase list page. This is large to attract attention and make its purpose clear. If the name or cost fields have not been correctly filled out, clicking this button will result in an alert asking the user to enter valid inputs before creating the purchase. Like the add group page, the design of this page has the user working from top to bottom, filling in the components before adding the purchase. This is again, more intuitive for users, as they are used to this flow in other areas, such as reading etc.