Maura Donatelli & Ava Hatfield

Dr. Zhang

IOS Project 2/3 Documentation

8 December 2022

The Trylife Center Shopping Application digitizes the shopping experience for clients and client assistants at the Trylife Center in Lower Burrell, PA. Trylife Center is a non-profit, pro-life organization that provides families in need with free education, peer counseling, and material support. By participating in Trylife’s program, clients receive points they can use to spend on material supplies for them and their family. Material items can be purchased through Trylife’s shopping center where there is a current need for an application that client assistants can use to manage purchases.

Currently, Trylife Center manages the checkout experience by pen and paper. The cost of an item is found on a reference sheet with the name of item and number of points or items are individually marked with stickers. Points are recorded on physical documents called shopping trackers. Points are summed using a calculator. The process is very timely and has a high potential for errors. Client assistants must manually look for the name of unmarked items, which is the case for most items, on a reference sheet to find the number of points. The time to do this adds up fast when clients have a lot of items, which is not atypical. The name of the item may also not always be clear. For example, a client may want to purchase a baby carrier, but there are multiple different kinds of baby carriers, and it can be hard to identify which baby carrier is associated with which number of points. In addition, using a calculator or doing mental math to sum the points often results in incorrect totals. For this reason, their audit process includes two additional people checking the math before the file can be considered accurate and complete. The Trylife Center Shopping Application will digitize the checkout experience, eliminating errors and improving efficiency. By enabling users to digitally scan or manually type in barcodes to search for items, the hassle of having to look up the item by name on a reference sheet will no longer be a problem. Items will be totaled using a computer which will eliminate any potential mathematical errors and client assistants. The goal is to provide a user-friendly application for the client assistants at Trylife that will improve the checkout experience for them and their clients. We are honored to have received the opportunity to create a positive impact for the clients and client assistants at the Trylife Center.

Our application incorporates many unique and advanced features. When opening the application, users will get to choose their preferred way of scanning items. Users can either manually enter the barcode number or scan the barcode of the item using the device’s camera. Upon tapping the “Scan” button, users will be prompted for permission to use their camera (if it’s their first time using this feature). If permission is given, the app will open their camera automatically. Once the camera is held up to an item’s barcode, that item is added to the client’s shopping list. If a user is having an issue getting the barcode to scan, or prefers to manually enter the barcode number, they can do so by typing in the number and tapping the “Add item” button. Upon adding an item to cart, the app looks up the item by number and finds the item’s name and point value. The name and point value are displayed to the user immediately, along with the total. These intuitive features make item lookup and totaling much easier for users.

When constructing the TryLife application, many SwiftUI functionalities were implemented to reach the applications goals. One Swift element that was very critical to the functionality of our app was the use of optional variables. An optional variable was used to define the items in the shopping list. Making the variable “item” an optional TryLifeList type was important when determining whether an item is found when the user searches for it through the scanner or text input. It also helped us determine whether an item needed to be added to the item list (or shopping list) based on whether it was found or not (nil or not nil). Also, another optional variable was used within some of the views to assist in the navigation process. Many of the buttons throughout the app navigate to a sub view, and an optional “selectedSubView” variable is used to select and navigate to the assigned view.

We also utilized many UI elements such as NavigationView, ListView / List, ScrollView / Scrolling, ForEach loop, VStack, ZStack, HStack and more. These elements were valuable when focusing on the design and placement of items on the different views included in our application. For example, NavigationView enables the user to navigate to each page on a button tap. ListViews, ForEaches, and ScrollViews all make the process of displaying the shopping list on the pages possible. Specifically, ListView made deletion of elements in the list possible which is an especially important feature for users to have. Next, UITextView was used to display the information on the main page to the user. In addition, a text field was used to implement the manual barcode entry feature. Using the text field, the user can enter the barcode number of an item into a text box manually which triggers a search for the item. To implement the item lookup, parsing a JSON file was needed so that data on each item could be stored and retrieved. The JSON file stored the item’s name, number of points, and barcode number. The JSON had to be deserialized so it could be used to lookup the item via barcode.

The most advanced functionality to implement was the barcode scanning. This was the most crucial and key part of our application. To implement the barcode scanner into our application, we added a package dependency from an existing application from github (third party application). Once this was completed, we were able to add a info.plist value that ensures that a user will be asked for permission to use their device camera (NSCameraUsageDescription). The “CarBode” package dependency uses the AVFoundation library which imports access to barcode types that are scanned. Inside the ScanningView, a scanner was installed with the type of barcode we needed to scan (. code128). Then, the scanner is set on a time interval where every 5 seconds the event (or scan) will be triggered. Finally, once a barcode is found while the camera is displayed, it captures the value and searches for it in the JSON file. With the barcode scanner implemented, the once completely manual process of checking out items in the shop became entirely digitized.

We (Ava and Maura) both worked together effectively and diligently to build this application. As a team, we chose the design aspects of the app such as the layout, colors, and navigation of many elements. Ava worked on the home page and getting the manual barcode input to submit and then adding the item to the shopping list. Ava also worked on implementing the UITextView. Maura worked to get the text input to add to the shopping list of items. Maura additionally conducted research on the AVFoundation library and implemented the barcode scanner view. To implement the barcode scanner, Maura learned how to add a package dependency into the project from github. Also, after Maura reconstructed the navigation / hierarchy of the views included in the application, Ava was able to finish the front-end design elements to complete the final product.

Overall, the process of designing and creating this application went exceptionally well. We were able to work well as a team to complete the necessary components. Divvying the work between the us went smoothly and we were able to help each other with different tasks we got stuck on. However, we did encounter challenges when developing certain features of our app. When incorporating the NagivationView, one problem we encountered was nested NavigationViews. Continuous testing was needed to eliminate nested NavigationViews. To fix this problem, we needed to research and seek guidance from Dr. Zhang to make the navigations from one view to another work properly without becoming nested. A huge, unexpected challenge encountered throughout the whole process was the front-end design of the application. It took hours of tweaking different elements such as ScrollViews, ListViews, text elements, and spacing to get them to be placed in a way that could be properly displayed on all devices. The application functions very smoothly and effectively and meets the user’s needs. Ultimately, the application reaches the main goal of digitizing the current manual shopping process at the TryLife Center.

To close, it is valuable to discuss the possible improvements that could be made in the future. The design of the app (as far as colors, additional UI features) could be improved to a more advanced level. For example, this could include changing the fonts, layouts, and button designs to more closely match TryLife branding. As far as future versions of the app, this could include adding more elements such as a web view that will include a link to the options of items for sale in the store and images of what they look like to give users a better idea of what they are purchasing. Another feature that could be added is an additional page showing previous purchases. This will be useful in cases where a user might restart the app without documenting their summary. If the app keeps track of previous shopping summaries, the user will not have to worry about accidentally restarting the application. Another feature could be sending the receipt to the client’s email. Although we may make some improvements to make the application even more efficient later, the final product effectively meets the needs of the client assistants at the TryLife Center and also positively impacts the experience clients have when they shop at TryLife Center.

<https://github.com/mdonatelli22/Project2-Final.git>

^^ clone on xcode

**BARCODES TO TEST:**

Background pattern

Description automatically generated

Background pattern

Description automatically generated

Background pattern

Description automatically generated

Background pattern

Description automatically generated