

Criterion C - Development

*** All the screenshots of the code along with the explanations in this document are from the final version of the product which had been developed after receiving constant feedback from the client and the supervisor during informal and formal sessions with them.

For evidence of the initial consultation with the client please check the files Appendix 1 and 2***

List of all advanced techniques incorporated meeting success criteria

Technique No	Advanced Technique	Purpose and explanation	Success Criteria	App module where complexity is implemented	Screenshot numbers for technique
1	Elimination of data duplication in usernames.	To allow new accounts to be created for customers without duplicating existing usernames.	7	Sign Up View	1.0, 1.1
2	Validation check- Authentication of password being at least 8 characters	To allow secure login for all customers	7	Sign Up View	1.0,1.2
3	Implemented presence check for all login fields	Shows an alert to the user to enter credentials during login	1,7	Login View	1.3, 1.4
4	Smooth navigation across the app through self-understandable tabs in the navigation bar.	Allows a user to go back to the home screen or previous screen from the current screen.	12	Table View	1.5, 1.6, 1.7
5	Parity in design- Implementation of standard icons throughout screens	It allows easy navigation because users will understand how to navigate to a desired screen by looking at the icon. Makes UI efficient	12	Table View Profile View Order View	1.5, 1.6
6	Integration of database with front-end using Swift Data and model context.	Allows data to be stored persistently and makes it permanent. Facilitates efficient inventory management	2,3,8,9,11	Swift Data	2.4,2.5

7	Improved UX/UI- Usage of Colorscheme environment	Allows better designing of the app so that it is easily viewed in light mode and dark mode of mobile.	12	Login View Table View	2.6,2.7
8	Extensibility- Integrating a photo picker as an option for admin	Allows the admin to handpick a photo from the gallery to visually depict the product, if required.	2,12,9	Table Item View Table View(for admin)	1.5, 1.8, 1.9
9	Quick Search- Implementation of SearchBar on the home screen	Allows all users to search through the list of products by product name and company name. Facilitates easier usability and navigation.	12	Table View	1.5, 2.0
10	Depicting stock status- Usage of icons to display whether reordering is required	A checkmark is displayed to the admin when an item is sufficient in supply however the checkmark disappears if the product is low in supply. It indicates to the admin that reordering is required	5	Table View(for admin) Table Item View	1.5, 2.1
11	Implementation of delete swipe action on the home screen for admin	The admin will be able to delete an item no longer available simply by swiping right on that item cell. (Ask for confirmation)	3	Table View(for admin)	1.5, 2.2, 2.3
12	Order details captured- Capturing and presenting the current date and time of the	Stores the time when an item is ordered to show the details of all orders to the admin and user	4,11	Order View Profile View	2.8, 2.9

	device				
13	Usage of green and red icons to indicate the status of orders for admin	Green and red circles are used to mark whether an order is completed or pending to the admin. It makes UI efficient and easy to understand.	11,12	Profile View(for admin)	3.1,3.2
14	Usage of alerts to remind admin of stock unavailability	If the admin tries to mark an order as completed but does not have enough stock then an alert is displayed to warn the client to update the stock. Prevents invalid entries.	4,5	Order View	2.8, 3.0
15	Usage of Stepper feature to make UI efficient and easier input of data	A Stepper lets the client click for input in forms rather than type on the keyboard. Saves more time.	12	Add Product Sheet Edit Supply View	3.5, 3.6
16	Usage of a “my favourites” list which acts as a specialized report	Customer can add a product to the list for more details about the product and track the stock of the product so that they can order when item is available.	12	Favourites View Table View(for customers) Table Item View	3.9, 4.0, 4.1, 4.2
17	Usage of alerts to warn customers if the item is unavailable.	In certain situations, the stock might be less than the quantity ordered. So the order is not processed and the customer is alerted to order when stock is available	11,12	Place Order View	3.7, 3.8

18	Usage of order tracker to show customer at which stage their order is.	A progress bar is used to show to the customer whether the order is “pending”, “packaged”, “being delivered” or “completed”. This allows the customer to estimate when their order may come based on how far along the progress bar is.	12	Profile View(for customers) Order View	3.3, 3.4
----	--	---	----	---	----------

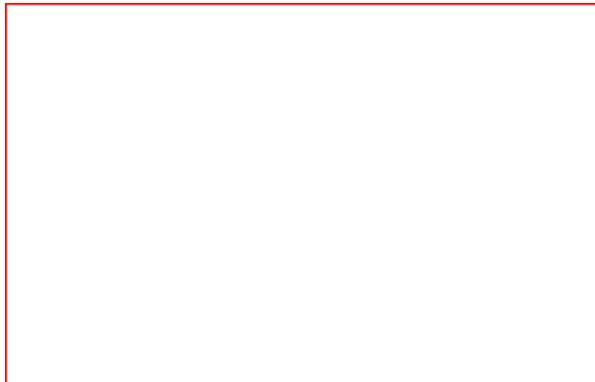
Format of a page

Sample_View_Name - View in which complexity used

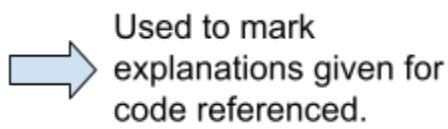
Sample_Picture - Picture of full view

Sample_Label - Figure number for pictures

Sample_Technique - Technique number in reference with table 1 along with picture of code of technique.



Part of the picture referenced.



Used to mark explanations given for code referenced.

Sample_Explaination



Important parts of code referenced inside red box.
Also used to represent important parts of view highlighted.

****The above page format is followed throughout this document for parity between all pages****

View: Sign Up View

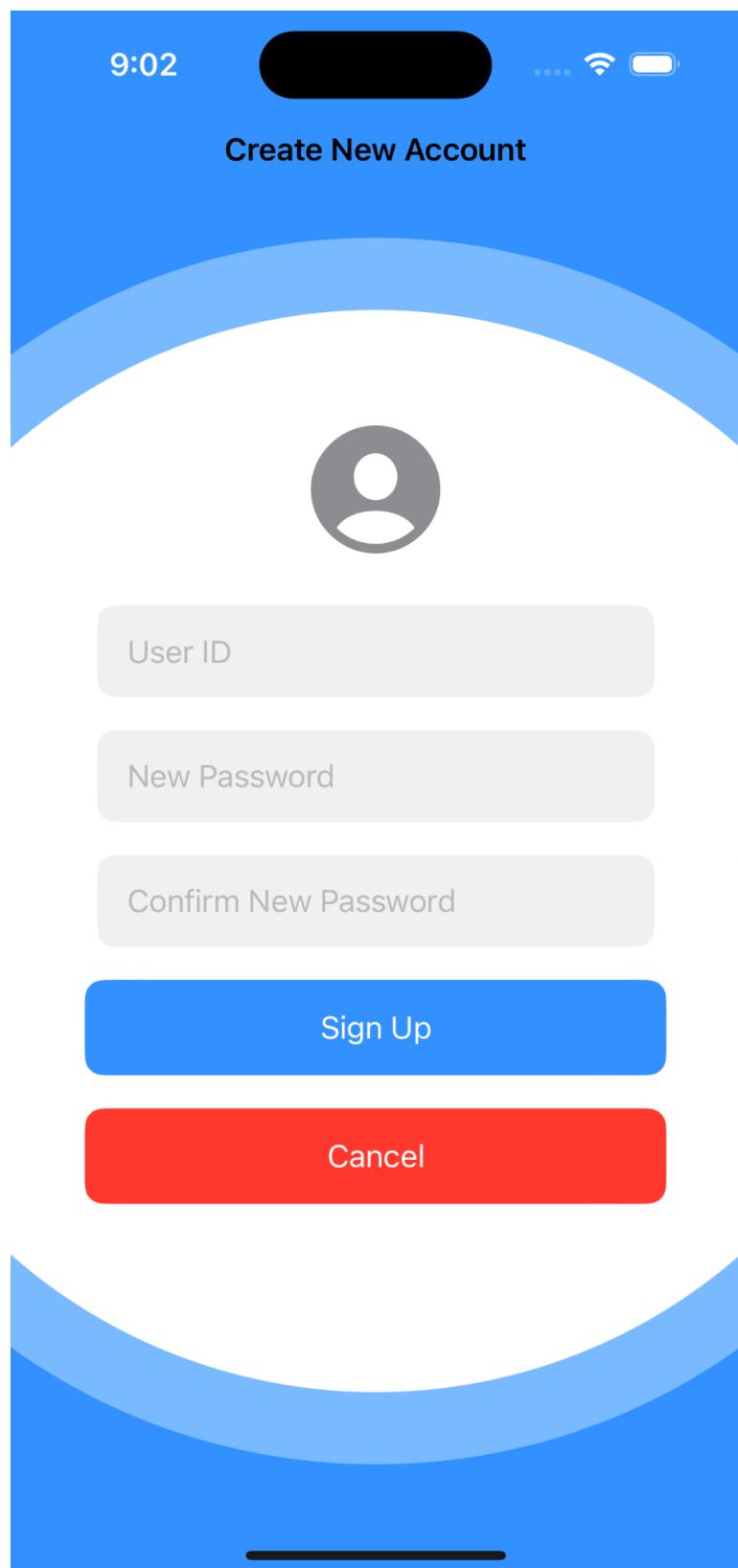


Fig 1.0

Technique 1:

```
func signUp() {
    if userId.isEmpty || newPassword.isEmpty || confirmPassword.isEmpty {
        showingEmptyAlert=true
    }

    if users.map({ $0.username }).contains(userId) {
        showingUsernameExistsAlert.toggle()
        return
    }

    else{
        if newPassword != confirmPassword {
            showingPasswordMismatchAlert = true
        }
        else if confirmPassword.count<8{
            showingPasswordShortAlert=true
        }

        else {
            // Handle sign up logic here
            addUser()
            //LoginView(navigationpath: $path)
            //dismiss()
        }
    }
}
```

This function checks whether the username entered exists in the database. If entered username exists in the map of usernames then the "showingUsernameExistsAlert" is displayed.

```
.alert("An account with this username already exists.", isPresented: $showingUsernameExistsAlert) {
    Button("Cancel", role: .cancel) {
        userId = ""
        newPassword = ""
        confirmPassword = ""
    }
}
```

If username doesn't exist then the user is added.

Fig 1.1

Technique 2:

```
else{
    if newPassword != confirmPassword {
        showingPasswordMismatchAlert = true
    }

    else if confirmPassword.count<8{
        showingPasswordShortAlert=true
    }
}
```

This function checks whether the confirm password and new password fields match. If they do then it checks whether the password entered is at least 8 characters. An alert is displayed to the user when the condition is not met.

Password cannot be less than eight characters.

OK

Fig 1.2

View: Log In View

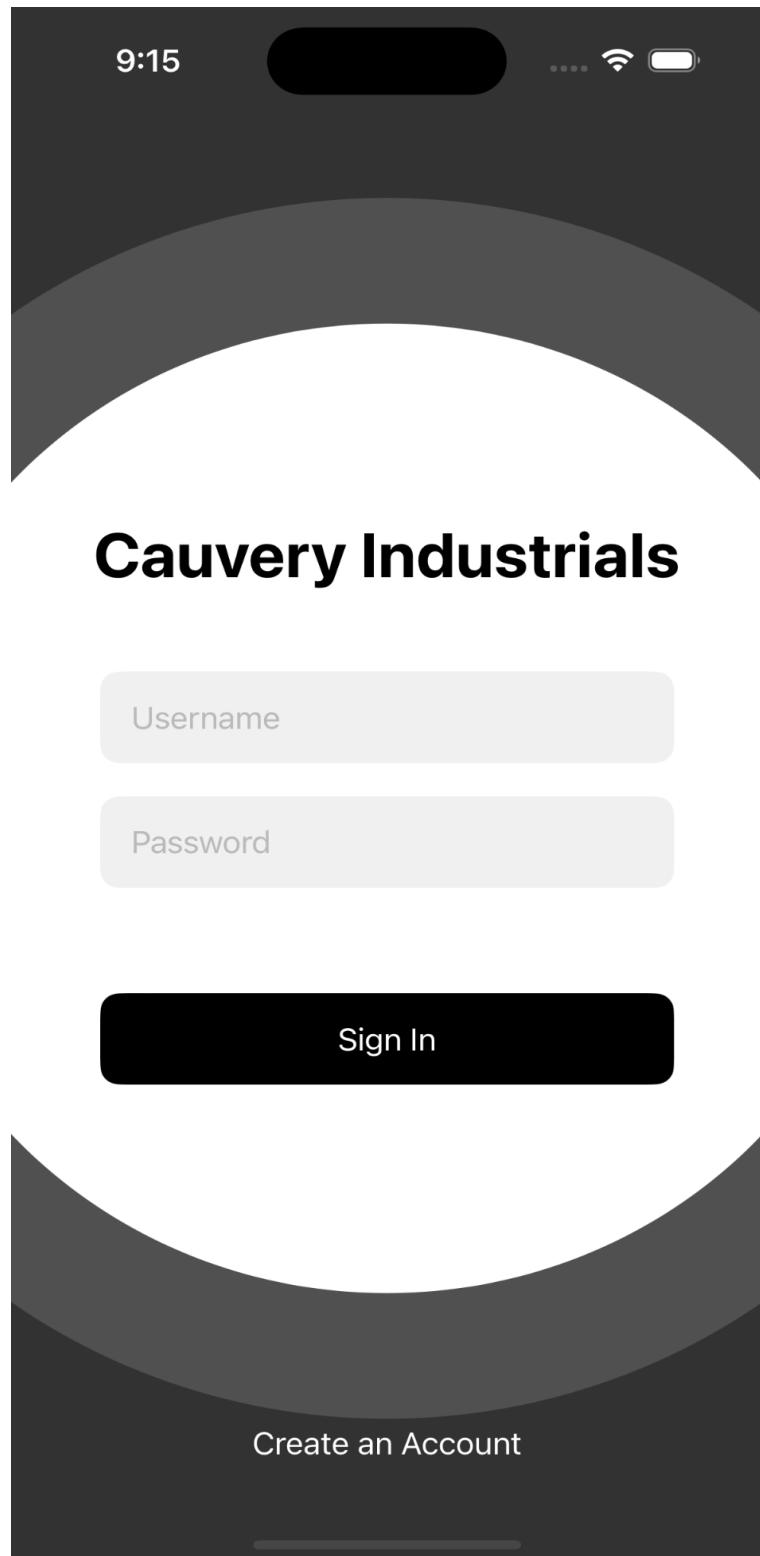


Fig 1.3

Technique 3:

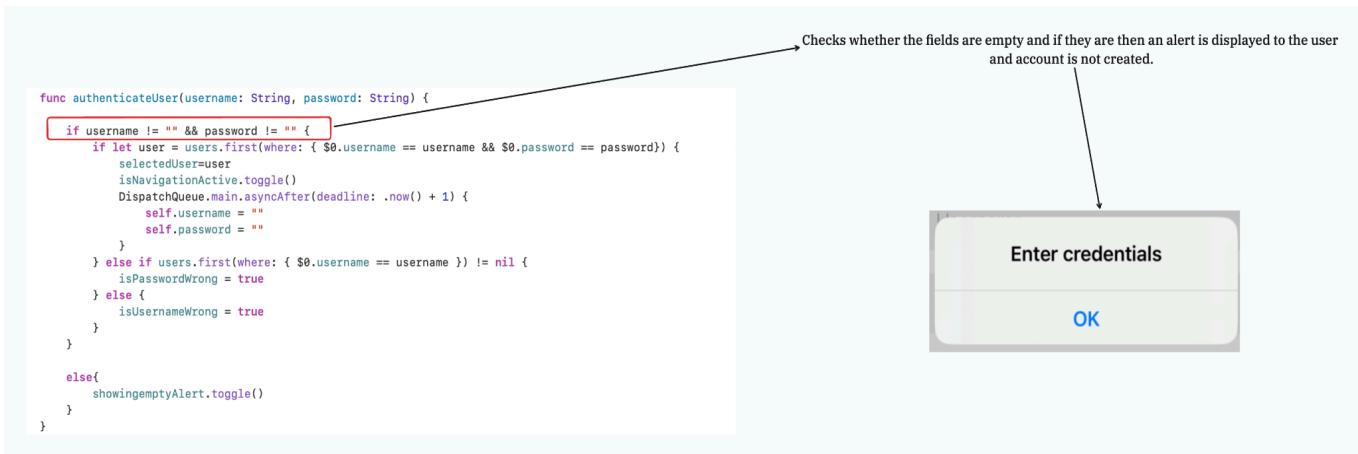


Fig 1.4

View: Table View(for Admin)

9:16

...

Search

Head Drilling Screw >
Kisha Hardware

Hexagonal Bolt ✓ >
Bolts and More

Square Bolt ✓ >
Bolts and Co

Screw Head ✓ >
Avanti Hardware

Tail Drilling Screw >
Avanti Hardware

Drywall Screw ✓ >
Bolts and Co

Machine Bolt >
Hubli Hardware

+

Click any product to update supply

Fig 1.5

Technique 4&5:

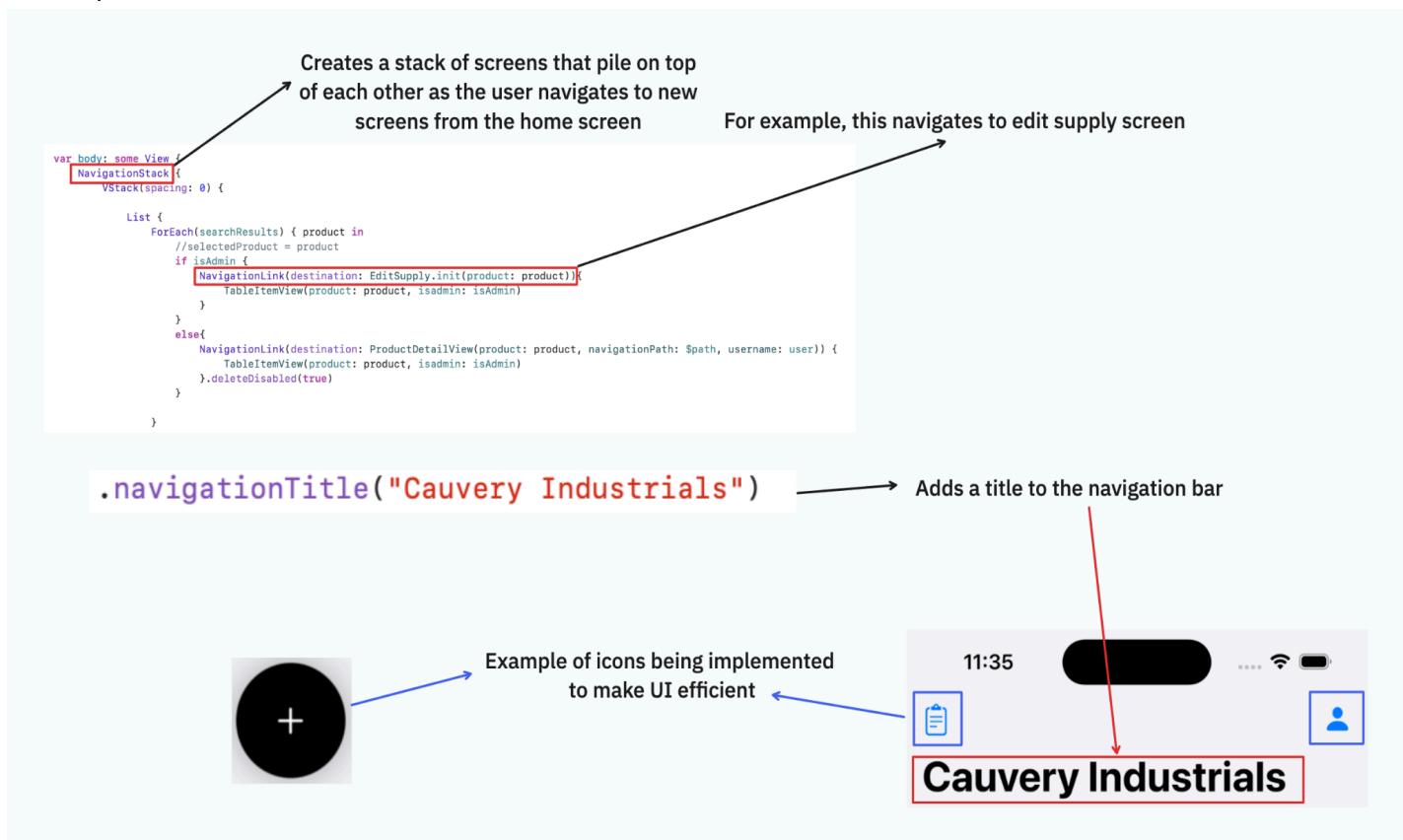


Fig 1.6

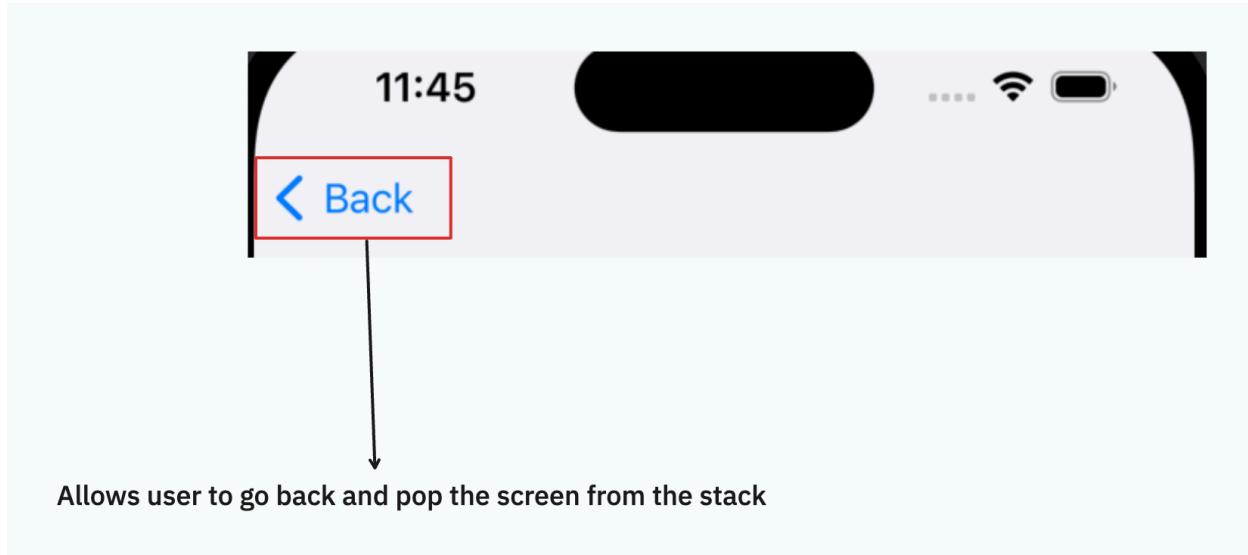


Fig 1.7

Technique 8:

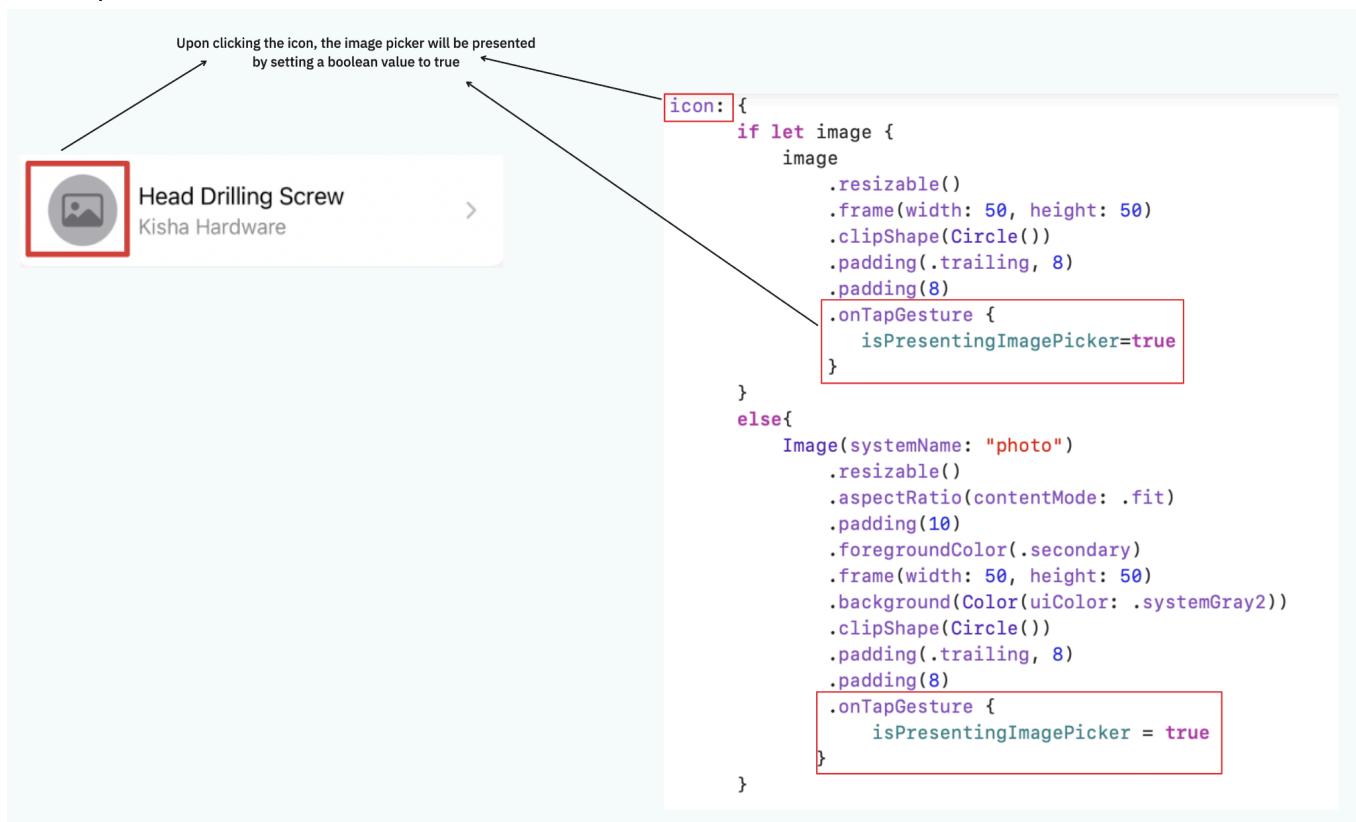


Fig 1.8



Fig 1.9

Technique 9:

The search bar is made case insensitive locally, so that the user can search for a product in small and capital case.

```

var searchResults: [Product] {
    if searchText.isEmpty {
        return products
    } else {
        return products.filter { $0.name.localizedCaseInsensitiveContains(searchText) }
    }
}

List {
   ForEach(searchResults) { product in
        //selectedProduct = product
        if isAdmin {
            NavigationLink(destination: EditSupply.init(product: product)){
                TableItemView(product: product, isAdmin: isAdmin)
            }
        } else{
            NavigationLink(destination: ProductDetailView(product: product, navigationPath: $path, username: user)){
                TableItemView(product: product, isAdmin: isAdmin)
            }.deleteDisabled(true)
        }
    }
}

```

Checks if search bar is empty. If search bar has text then it filters the list of products according to the searchtext

Fig 2.0

Technique 10:

Checks whether product has insufficient supply. If condition is met then checkmark is made invisible otherwise checkmark is presented.

```

Label {
    VStack{
        VStack(alignment: .leading) {
            Text(product.name)
            Text(product.companyName)
                .font(.subheadline)
                .foregroundColor(.secondary)
        }
        Spacer()
        if product.current_supply <= product.min_supply{
            checkmark.opacity(0)
        }else if isAdmin{
            checkmark.opacity(1)
        }
        //.padding(.trailing,10)
    }
}

```

In this example, the product "Hexagonal Bolt" has insufficient supply whereas "Head Drilling Screws" has sufficient supply

Fig 2.1

Technique 11:



Fig 2.2

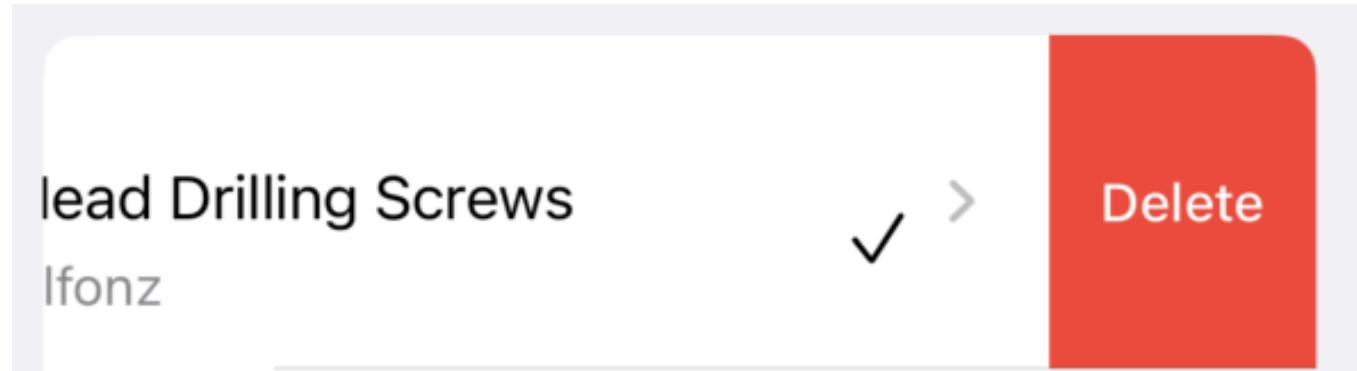


Fig 2.3

Swift Data (No view)

Technique Number 6:

```
Imports the library for handling the database
import SwiftData

@Model
class Product: Identifiable {
    //var id: String
    var name: String
    var companyName: String
    var inOrder: Order?
    var currentSupply: Int
    var minSupply: Int
    var price: Int
    @Attribute(.externalStorage) var imageData: Data?

Creates a model class in the
database for Products.

Initialising the class with
all variables

    // var image: Data?
    //
    // init(name: String, companyName: String, inOrder: Order? = nil, currentSupply: Int, minSupply: Int, price: Int, imageData: Data? = nil) {
        self.name = name
        self.companyName = companyName
        self.inOrder = inOrder
        self.currentSupply = currentSupply
        self.minSupply = minSupply
        self.imageData = imageData
        self.price = price
        // self.image = image
    }

    extension Product {
        var image: Image? {
            if let imageData, let uiImage = UIImage(data: imageData) {
                return Image(uiImage: uiImage)
            }
            return nil
        }
    }
}
```

Fig 2.4

```
Creates a container class
for the app

import SwiftUI
import SwiftData
@main
struct finalTestApp: App {
    //let persistenceController = PersistenceController.shared
    let container: ModelContainer
    @StateObject var lnManager = LocalNotificationManager()

    init() {
        let config = ModelConfiguration(isStoredInMemoryOnly: true)
        if let container = try? ModelContainer(for: User.self, Order.self, Product.self, configurations: config) {
            self.container = container
        } else {
            fatalError("Failed to initialize")
        }
    }
    var body: some Scene {
        WindowGroup {
            ContentView()
                .modelContainer(container)
                .environmentObject(lnManager)
                //.environment(\.managedObjectContext, persistenceController.container.viewContext)
        }
        .modelContainer(for: Product.self)
        // .modelContainer(for: Order.self)
    }
}
```

Fig 2.5

View: LoginView(in dark mode)

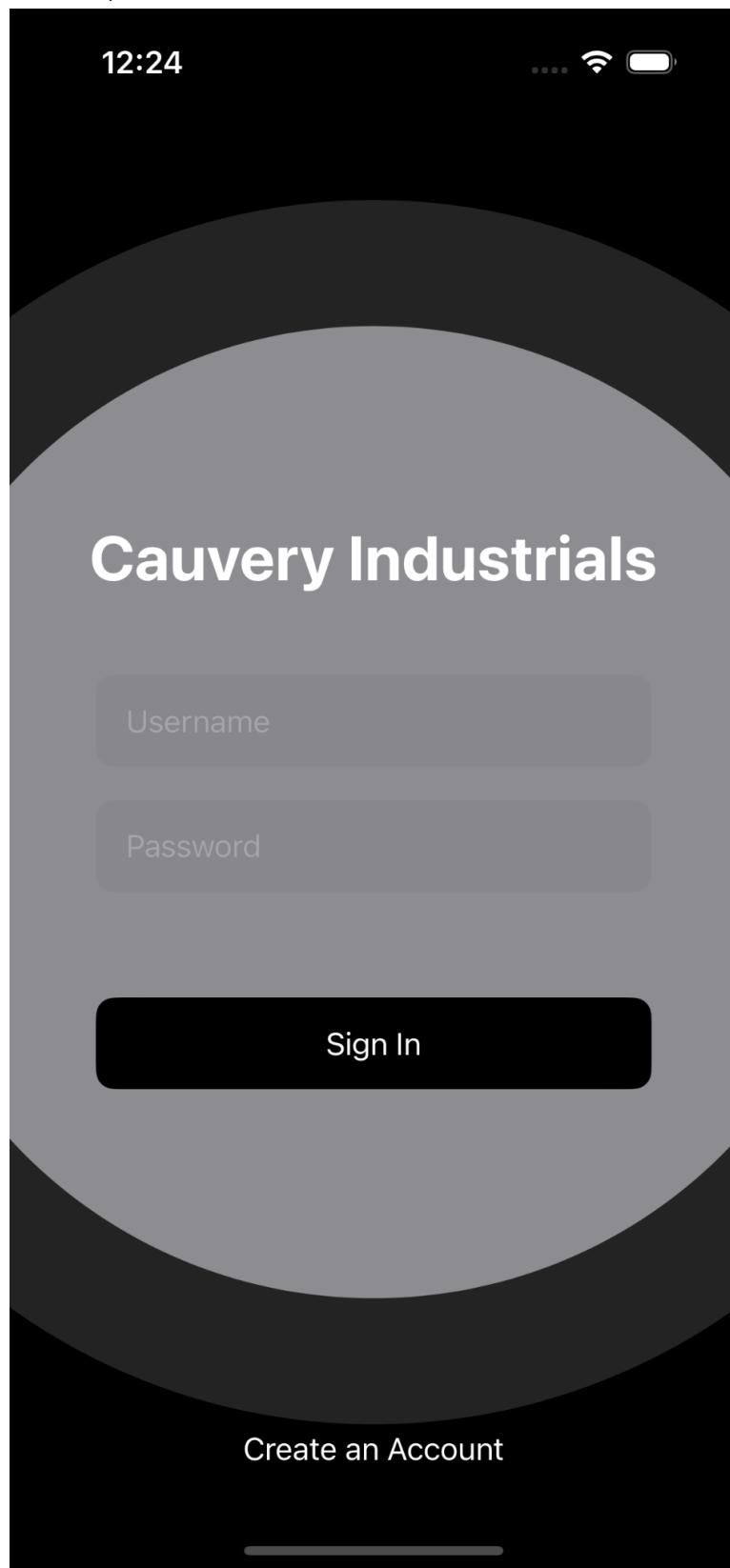


Fig 2.6

Technique 7:

```
struct LoginView: View {  
    @Environment(\.modelContext) var modelContext  
    @Environment(\.colorScheme) var colorScheme  
  
    Color.black.opacity(0.8)  
        .ignoresSafeArea()  
    Circle()  
        .scale(1.7)  
        .foregroundColor(.white.opacity(0.15))  
    Circle()  
        .scale(1.35)  
        .foregroundColor(colorScheme == .light ? .white : Color(uiColor: .systemGray))
```

Imports the colorScheme environment for usage in the app.

Using colorscheme to manipulate colors based on dark mode and light mode.

Fig 2.7

View: Orders View

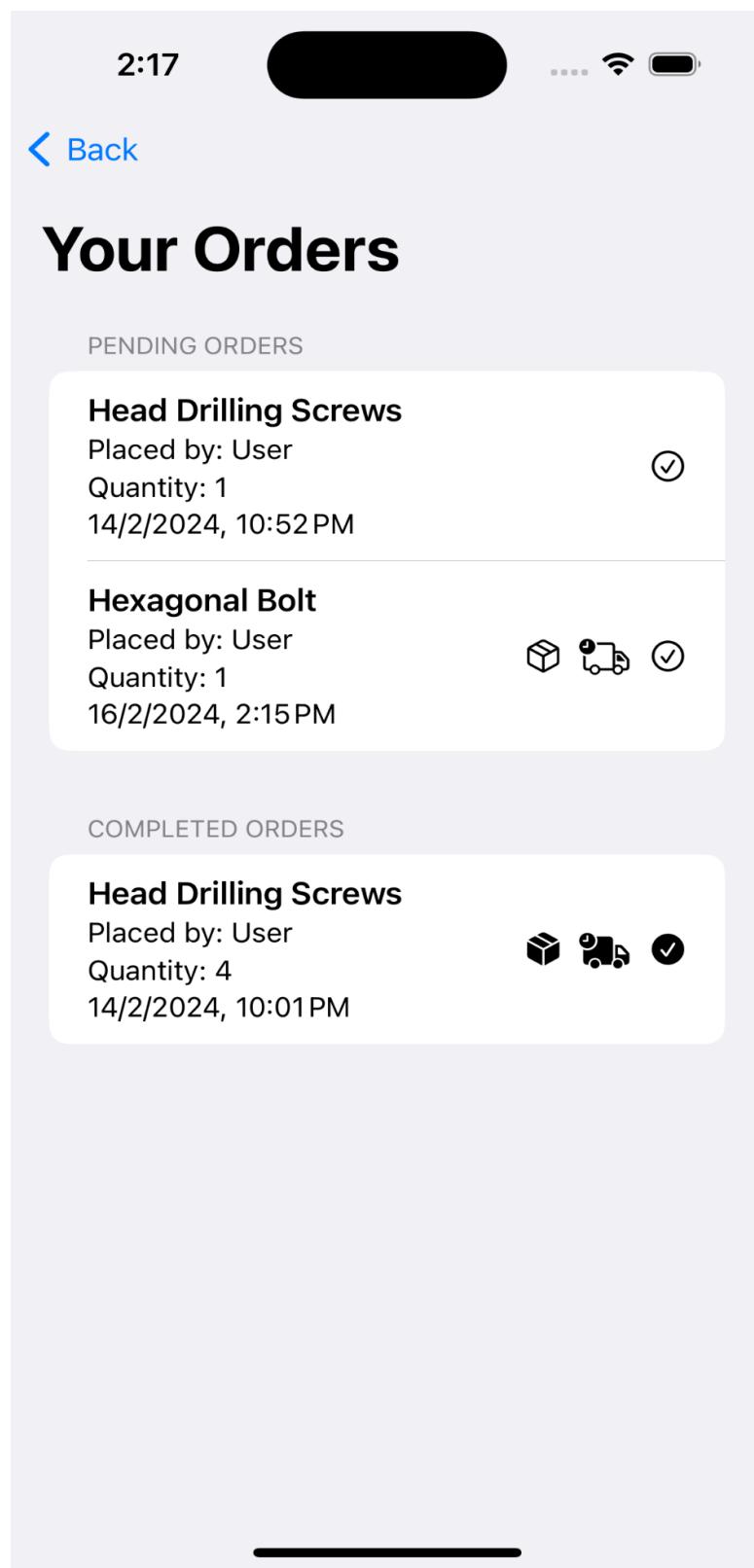


Fig 2.8

Technique 12:

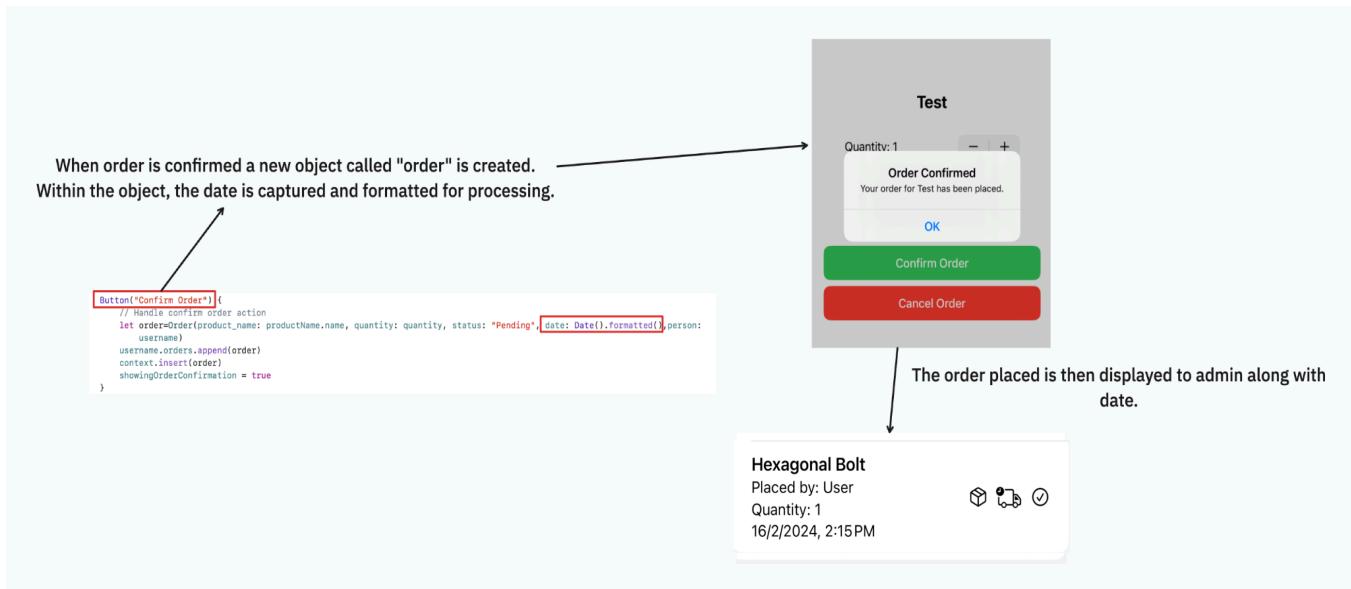


Fig 2.9

Technique 14:

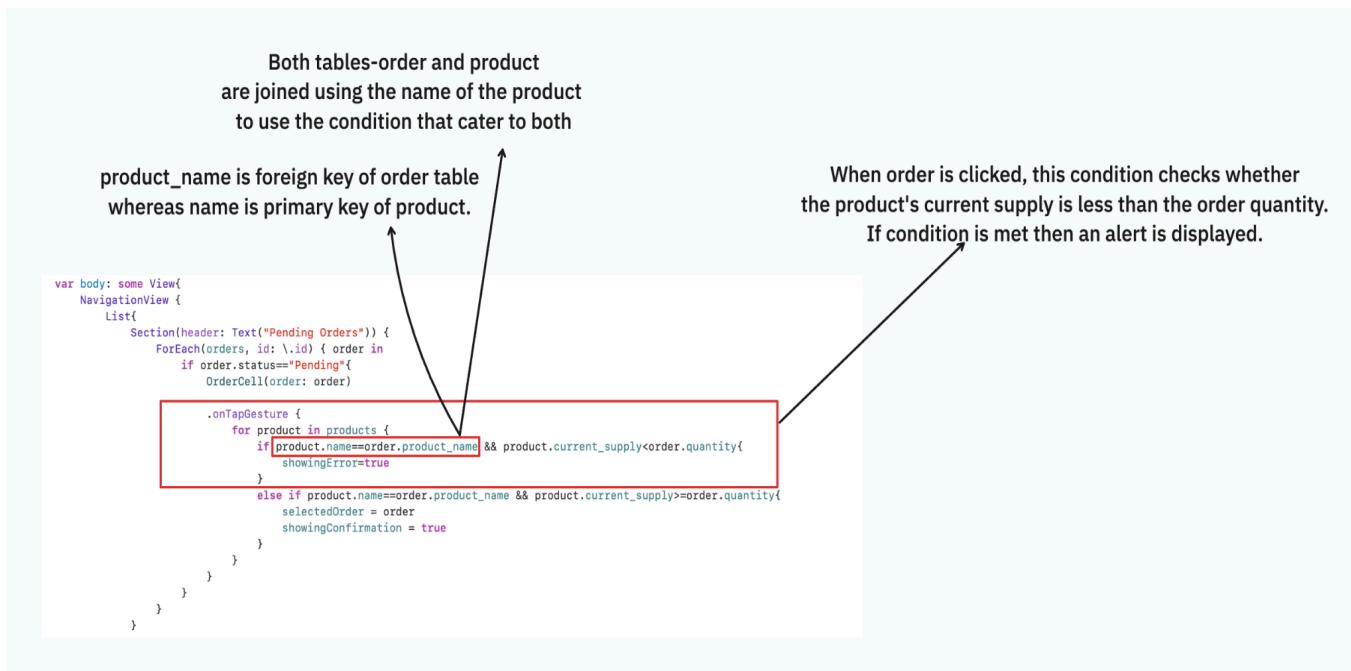


Fig 3.0

View: Profile View (for Admin)

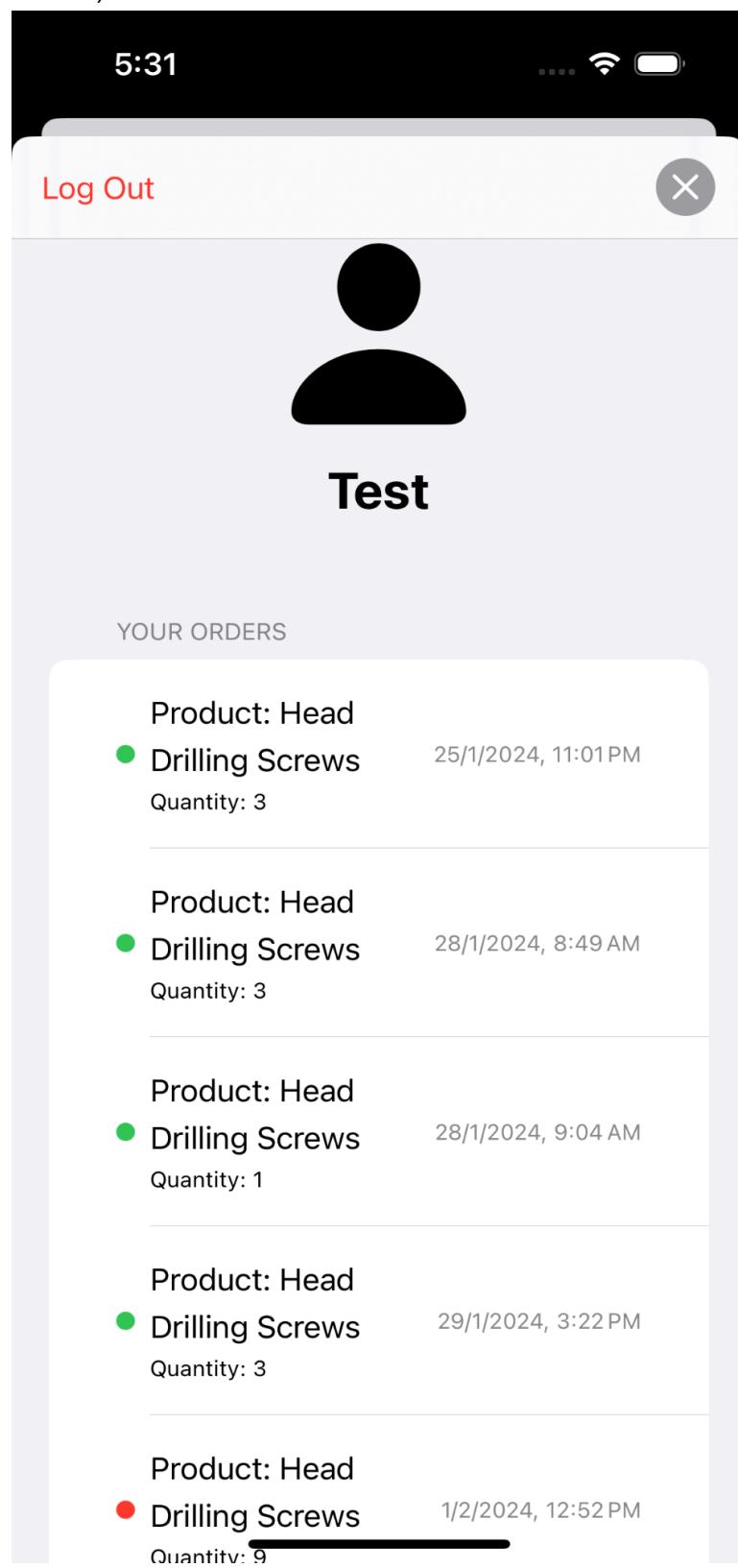


Fig 3.1

Technique 13:



Fig 3.2

View: Profile View (for customers)

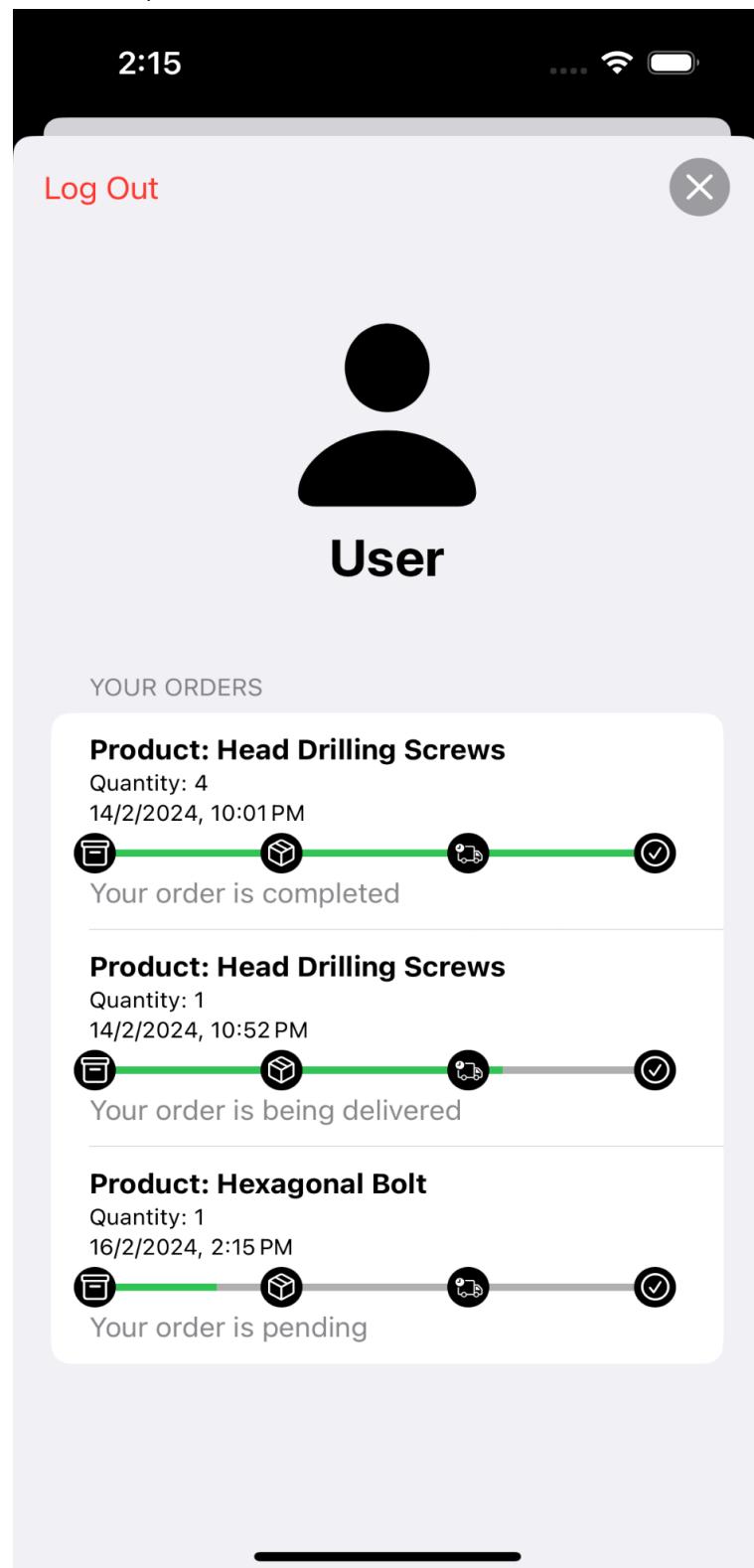


Fig 3.3

Technique 18:

For every order the admin has the choice to mark order as "packaged", "being delivered" or "completed".

By default the order is in pending status

```

if(order.status=="Pending"){
    shippingbox
        .onTapGesture{
            showingPackaged=true
        }
    delivery
        .onTapGesture{
            showingDelivered=true
        }
    completed
        .onTapGesture{
            showingConfirmation=true
        }
}
else if(order.status=="Packaged"){
    shippingbox.opacity(0)
    delivery
        .onTapGesture {
            showingDelivered=true
        }
    completed
        .onTapGesture {
            showingConfirmation=true
        }
}
else if(order.status=="Being Delivered"){
    shippingbox.opacity(0)
    delivery.opacity(0)
    completed
        .onTapGesture {
            showingConfirmation=true
        }
}
else if(order.status=="Completed"){
    completed.opacity(0)
    Image(systemName: "shippingbox.fill")
    Image(systemName: "truck.box.badge.clock.fill")
    Image(systemName: "checkmark.circle.fill")
        .foregroundColor(.black)
}
}

```

The admin can click any icon to mark it as the designated status

After marking the status, the order tracker or progress bar is updated accordingly.

Icons to mark each stage.

Array used to iterate through.

Product: Head Drilling Screws

Quantity: 1

Date: 14/2/2024, 10:52 PM

Your order is being delivered

Text is also used to explain each stage if icon is not enough.

```

struct ProgressCell: View {
    @State var progress:Double
    @State var progressImages:[String] = ["archivebox","shippingbox","truck.box.badge.clock","checkmark.circle"]
    var body: some View {
        VStack {
            ZStack(alignment: .leading) {
                Rectangle()
                    .foregroundColor(Color.black.opacity(0.3))
                    .frame(height: 4)
                Rectangle()
                    .foregroundColor(Color.green)
                    .frame(width: CGFloat(progress) * 300, height: 4)
            }
            HStack(spacing: 76) {
                ForEach(0...4) { index in
                    ZStack {
                        Circle()
                            .fill(Color.black)
                            .frame(width: 22, height: 22)
                        Image(systemName: "\\(progressImages[index])")
                            .resizable()
                            .aspectRatio(contentMode: fit)
                            .frame(width: 15, height: 15) // Size of the image
                            .foregroundColor(.white)
                    }
                }
            }
        }
    }
}

```

For example, at order.status="packaged" the progress bar will be 50% along and so on.

```

if(order.status=="Pending"){
    ProgressCell(progress: 0.25)
        .frame(width: 300, height: 4)
        .padding(.bottom, 5)
        .padding(.top, 5)
}
else if(order.status=="Packaged"){
    ProgressCell(progress: 0.5)
        .frame(width: 300, height: 4)
        .padding(.bottom, 5)
        .padding(.top, 5)
}
else if(order.status=="Being Delivered"){
    ProgressCell(progress: 0.75)
        .frame(width: 300, height: 4)
        .padding(.bottom, 5)
        .padding(.top, 5)
}
else if(order.status=="Completed"){
    ProgressCell(progress: 1.0)
        .frame(width: 300, height: 4)
        .padding(.bottom, 5)
        .padding(.top, 5)
}
Text("Your order is \((order.status.lowercased())")
    .foregroundColor(.secondary)
    .font(.subheadline)
}

```

Fig 3.4

View: Add Product Sheet

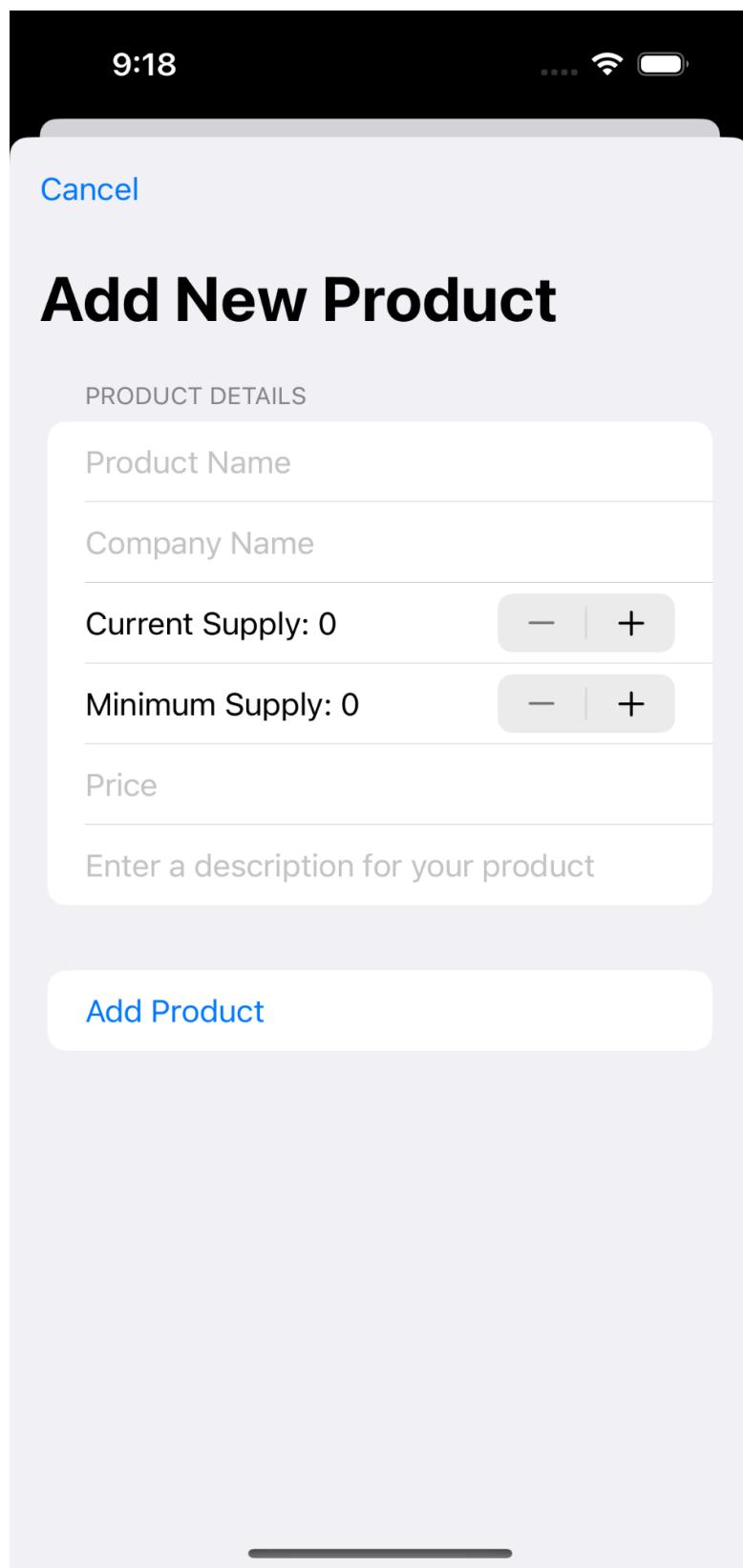


Fig 3.5

Technique 15:

The diagram illustrates a code snippet for a stepper component and its corresponding user interface (UI) representation.

Code Snippet:

```
Form {
  Section(header: Text("Product Details")) {
    TextField("Product Name", text: $newProductName)
    TextField("Company Name", text: $newCompanyName)
    Stepper("Current Supply: \$(currentSupply)", value: $currentSupply, in: 0...Int.max)
    Stepper("Minimum Supply: \$(minimumSupply)", value: $minimumSupply, in: 0...Int.max)
    TextField("Price:", text: $price)
    //Stepper("Price: \$(price).00", value: $price, in:0...Int.max)
  }
}
```

Annotations:

- An annotation points from the code snippet to the UI representation, stating: "The stepper changes between values of 0 to infinity It facilitates more efficient UI for data input".

UI Representation:

Two stepper components are shown:

- Current Supply:** Value 0, with a range from - to +.
- Minimum Supply:** Value 0, with a range from - to +.

Fig 3.6

View: Place Order View

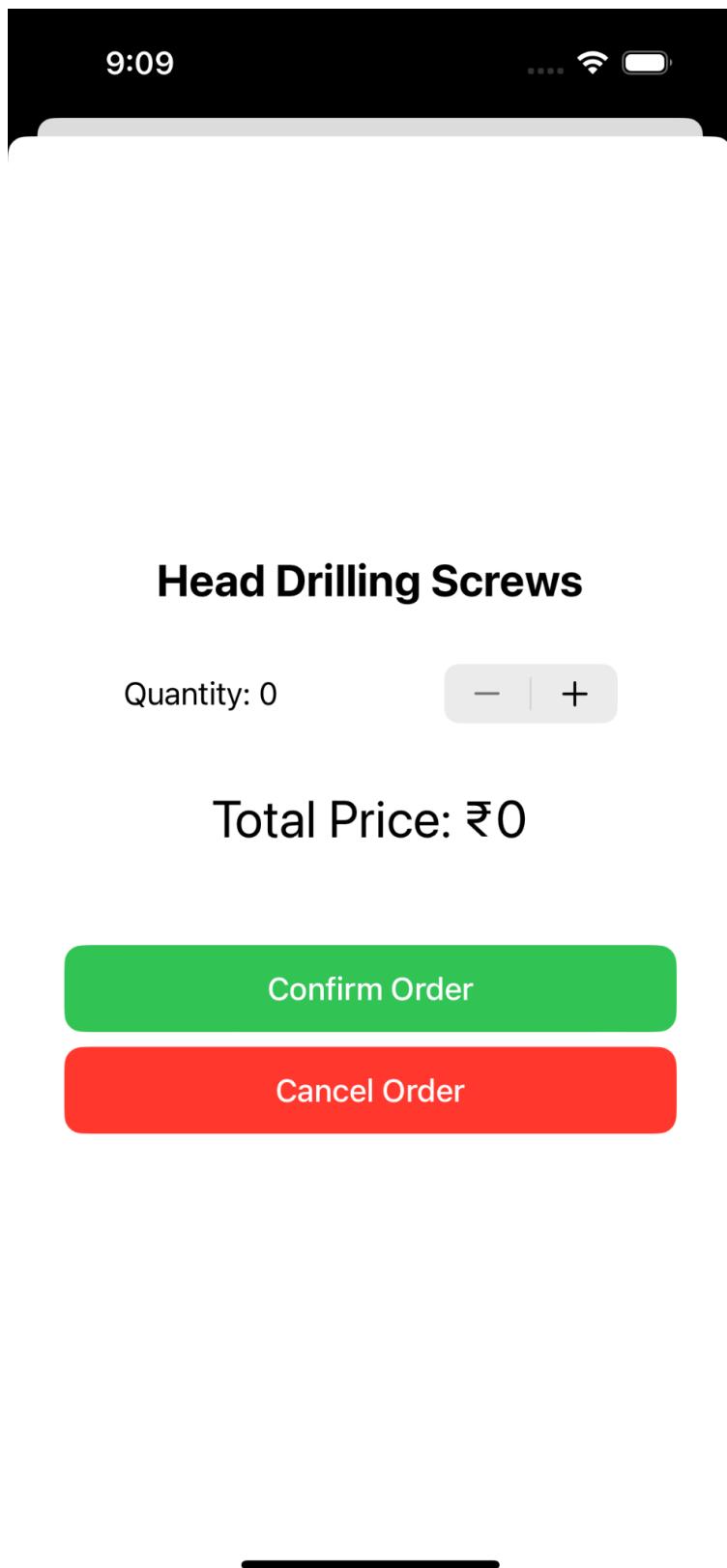


Fig 3.7

Technique 17:

If the order quantity is more than the current supply of the product than the alert "showingOrderNotAvailable" is displayed and order is deleted.

```
let order=Order(product_name: product.name, quantity: quantity, status: "Pending", date: Date().formatted(),person: username)
if order.quantity==0{
    showingZeroOrder=true
    context.delete(order)
}
else{
    username.orders.append(order)
    context.insert(order)
    if order.quantity>product.current_supply{
        showingOrderNotAvailable=true
        context.delete(order)
    }
    else{
        showingOrderConfirmation=true
    }
}
```

Try Again
Your order could not be processed because currently item is unavailable, try ordering in some time. Sorry for the inconvenience.

OK

Fig 3.8

View: Table View (for customers)

9:20

...

Search

Head Drilling Screw
Kisha Hardware

Hexagonal Bolt
Bolts and More

Square Bolt
Bolts and Co

Screw Head
Avanti Hardware

Tail Drilling Screw
Avanti Hardware

Drywall Screw
Bolts and Co

Machine Bolt
Hubli Hardware

Click top right to track all your orders

The image shows a mobile application interface for a hardware store. At the top, there's a header with the store name 'Cauvery Industrials' in large bold letters. Below the header is a search bar with a magnifying glass icon and the word 'Search'. The main content area displays a list of seven hardware items, each with a small image, the item name, and the supplier name. To the right of each item name is a red heart icon with a white outline and a greater-than symbol (>). The items listed are: 'Head Drilling Screw' from 'Kisha Hardware', 'Hexagonal Bolt' from 'Bolts and More', 'Square Bolt' from 'Bolts and Co', 'Screw Head' from 'Avanti Hardware', 'Tail Drilling Screw' from 'Avanti Hardware', 'Drywall Screw' from 'Bolts and Co', and 'Machine Bolt' from 'Hubli Hardware'. At the bottom of the list, there's a note: 'Click top right to track all your orders'. The top of the screen shows a status bar with the time '9:20', signal strength, battery level, and a blue icon.

Fig 3.9

Technique 16:

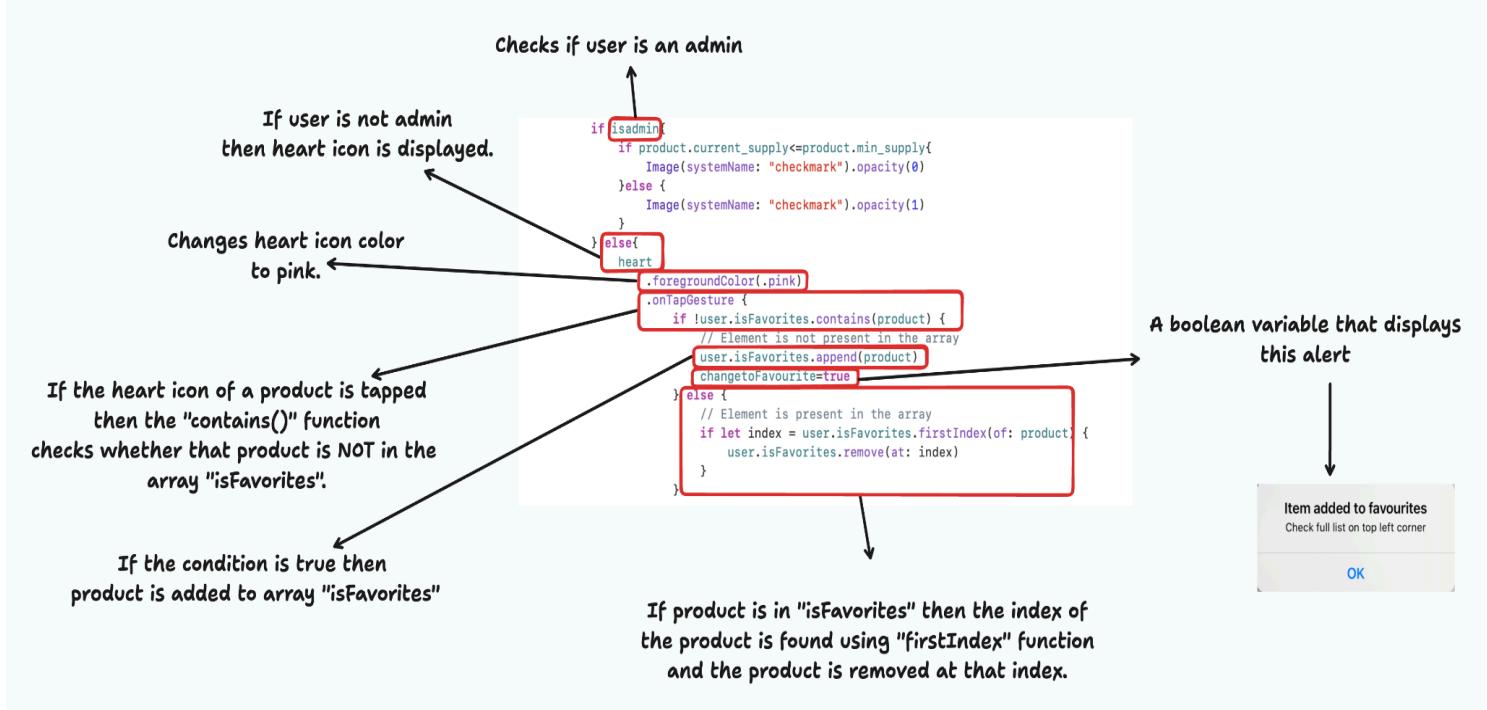


Fig 4.0

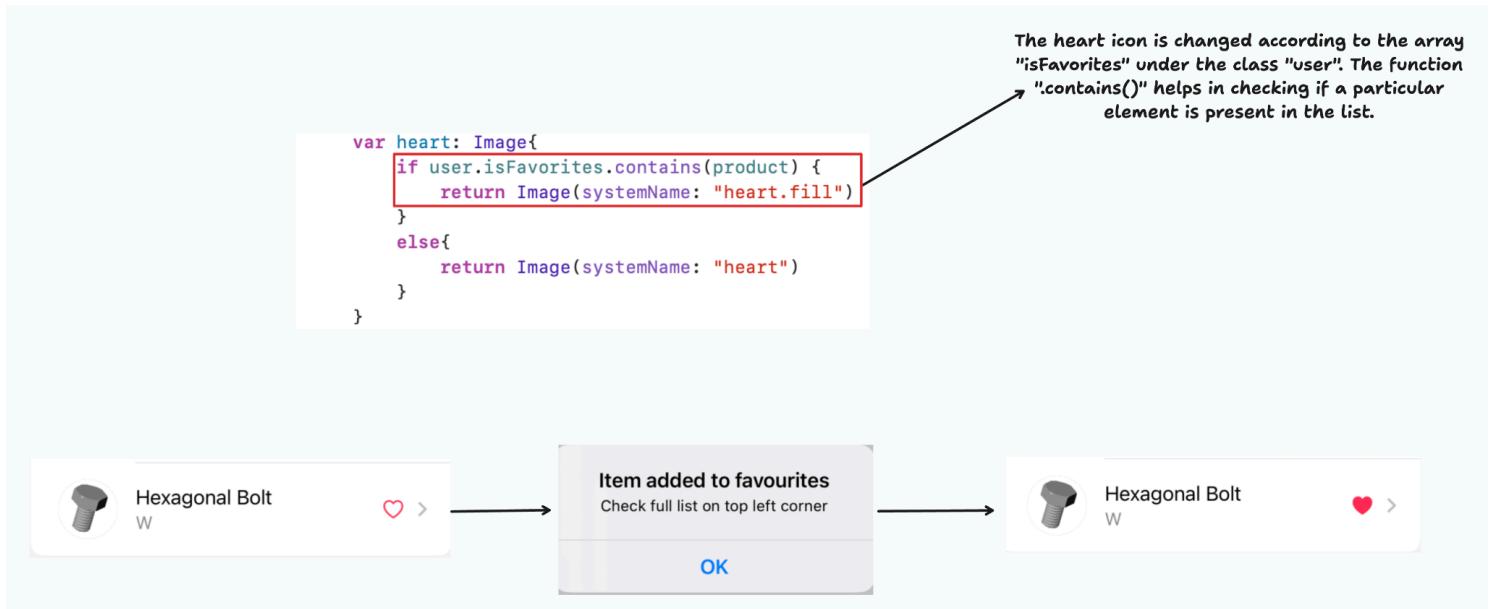


Fig 4.1

After being added the item is displayed in the view below:

View: Favourites View

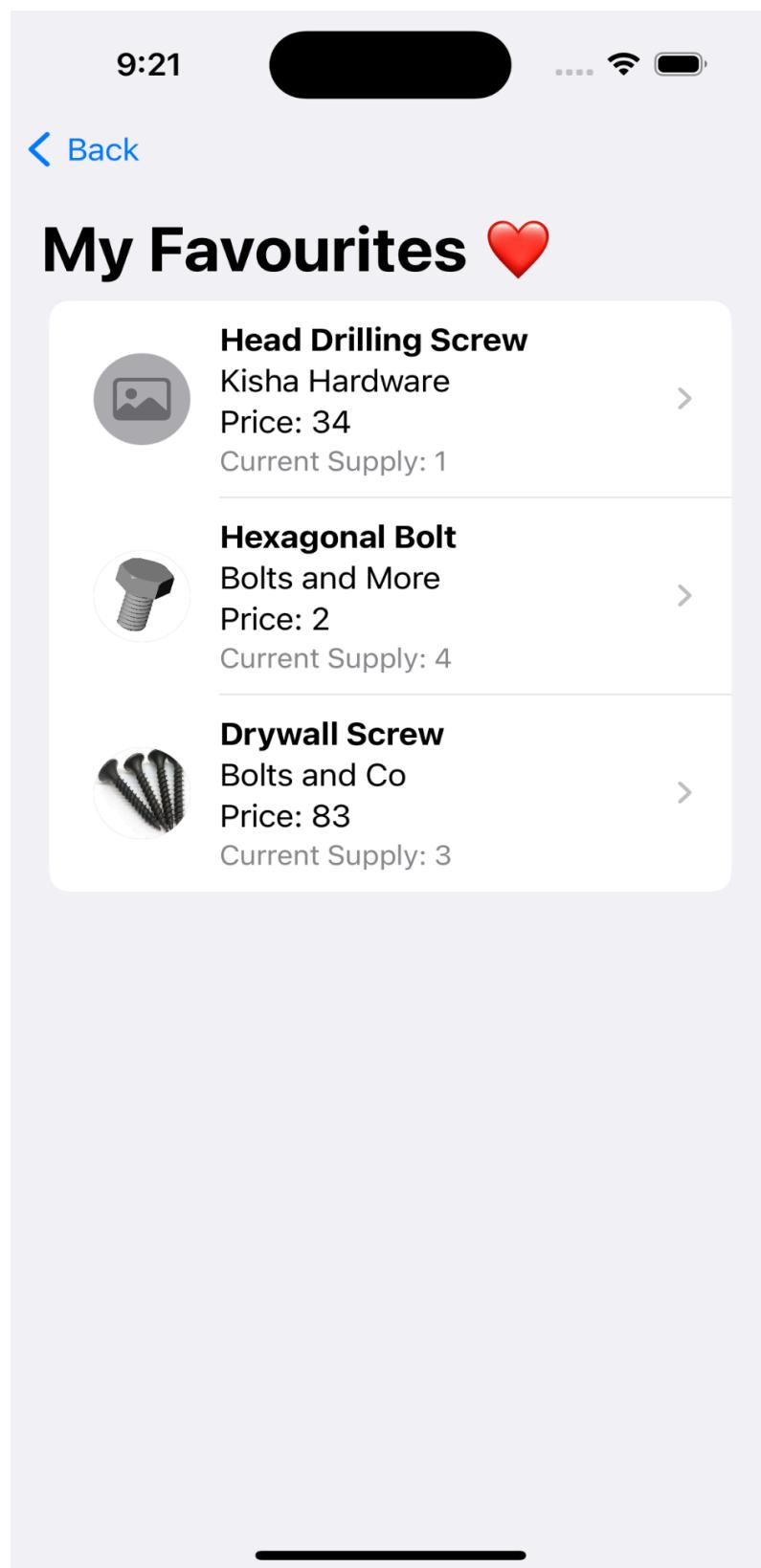


Fig 4.2