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-understand able 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

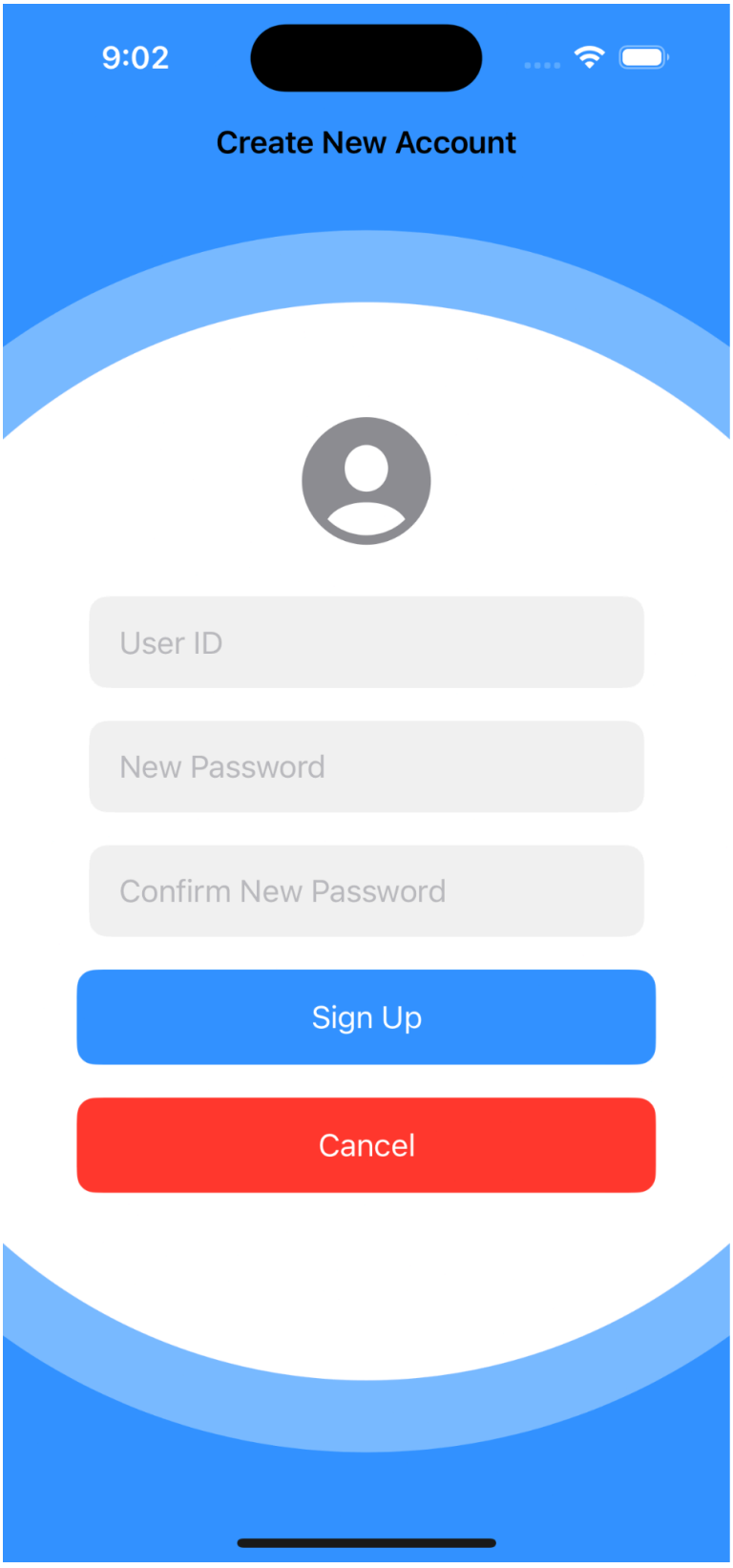


Fig 1.0

Technique 1:

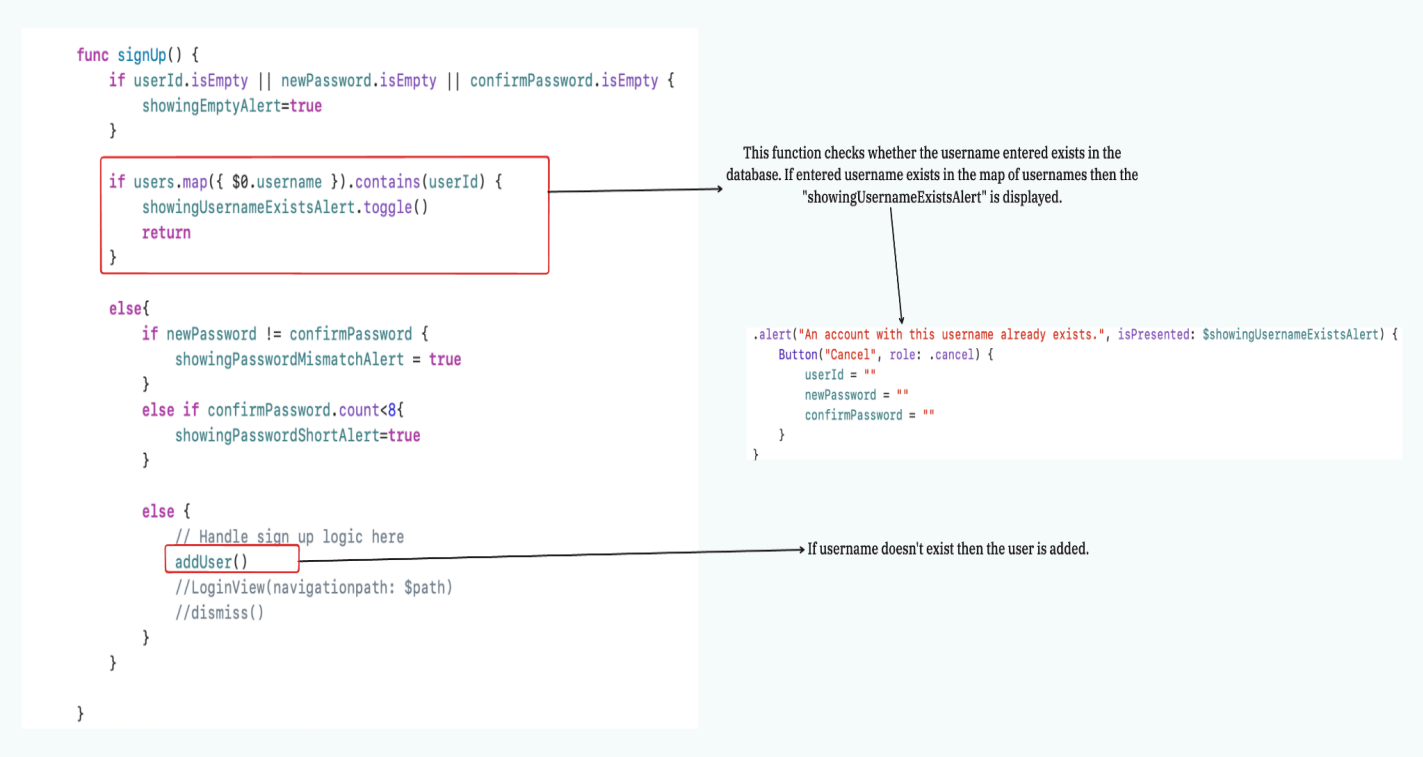


Fig 1.1

Technique 2:

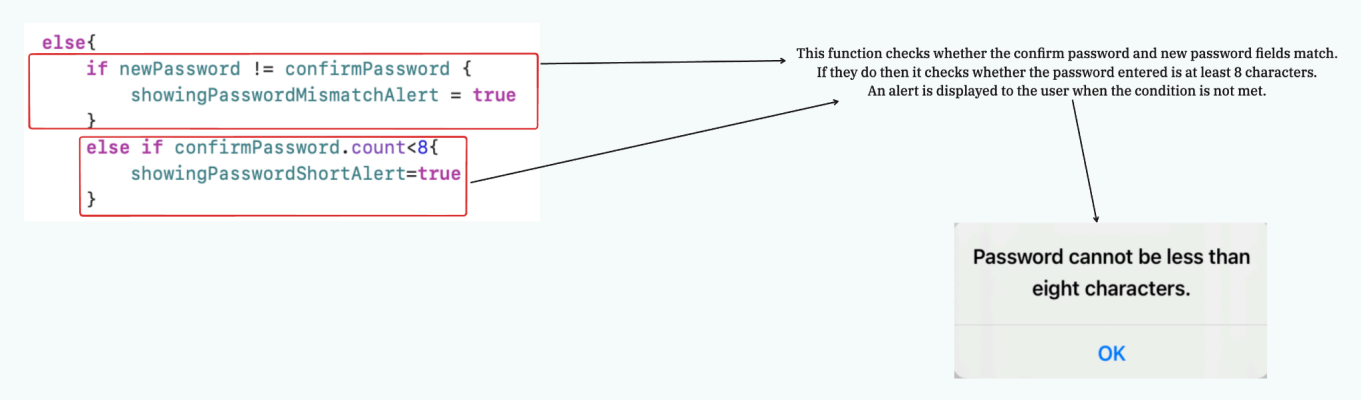


Fig 1.2

View: Log In View

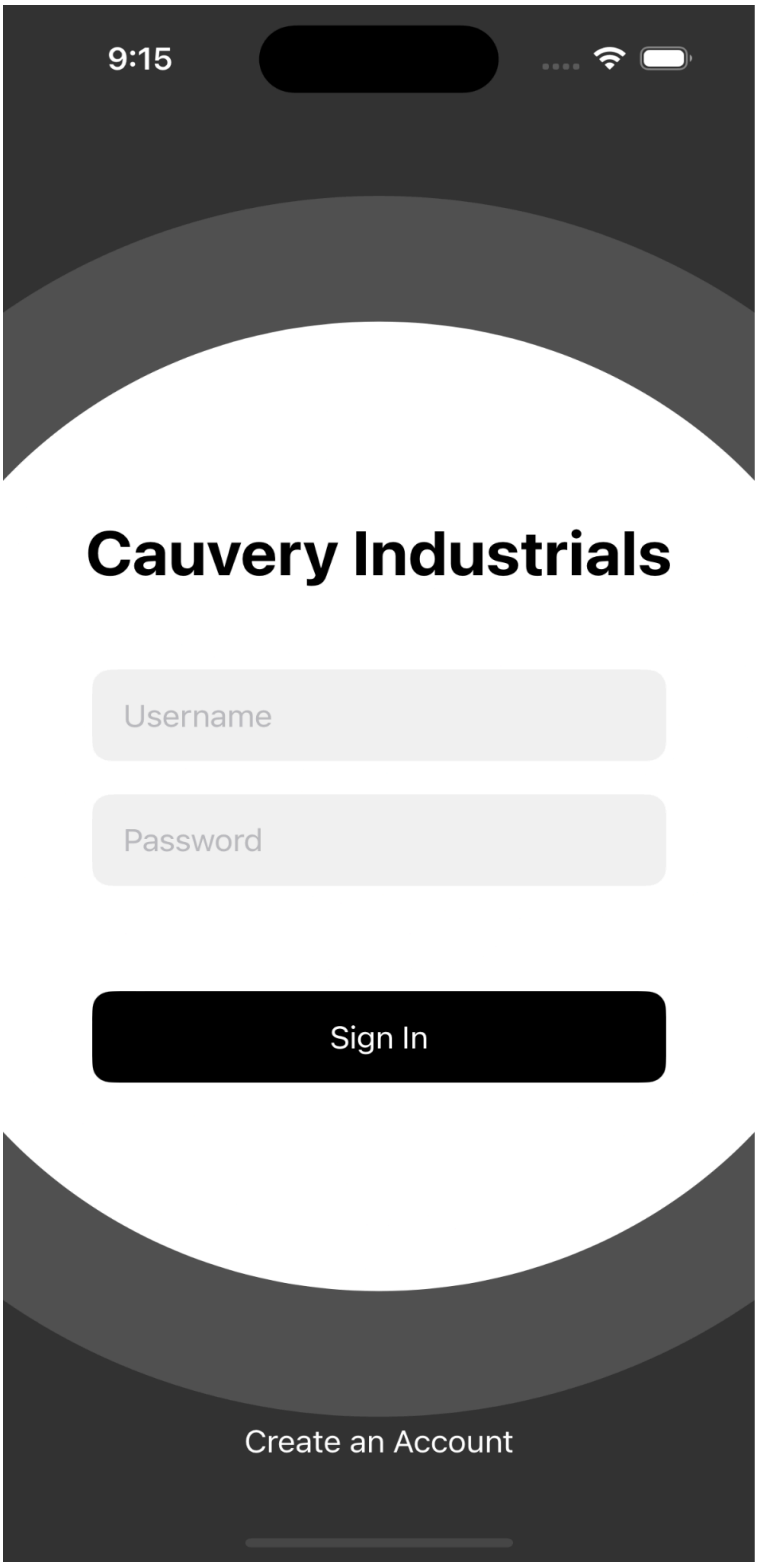
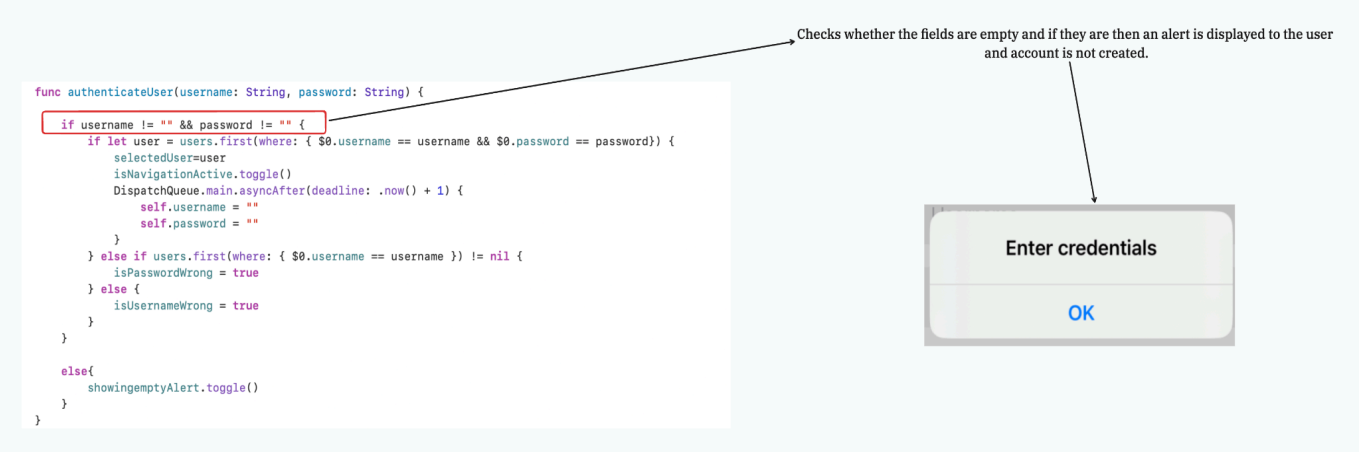


Fig 1.3

Technique 3:




```
func authenticateUser(username: String, password: String) {

    if username != "" && password != "" {
        if let user = users.first(where: { $0.username == username && $0.password == password}) {
            selectedUser=user
            isNavigationActive.toggle()
            DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
                self.username = ""
                self.password = ""
            }
        } else if users.first(where: { $0.username == username }) != nil {
            isPasswordWrong = true
        } else {
            isUsernameWrong = true
        }
    }

    else{
        showingemptyAlert.toggle()
    }
}
```

Fig 1.4

View: Table View(for Admin)

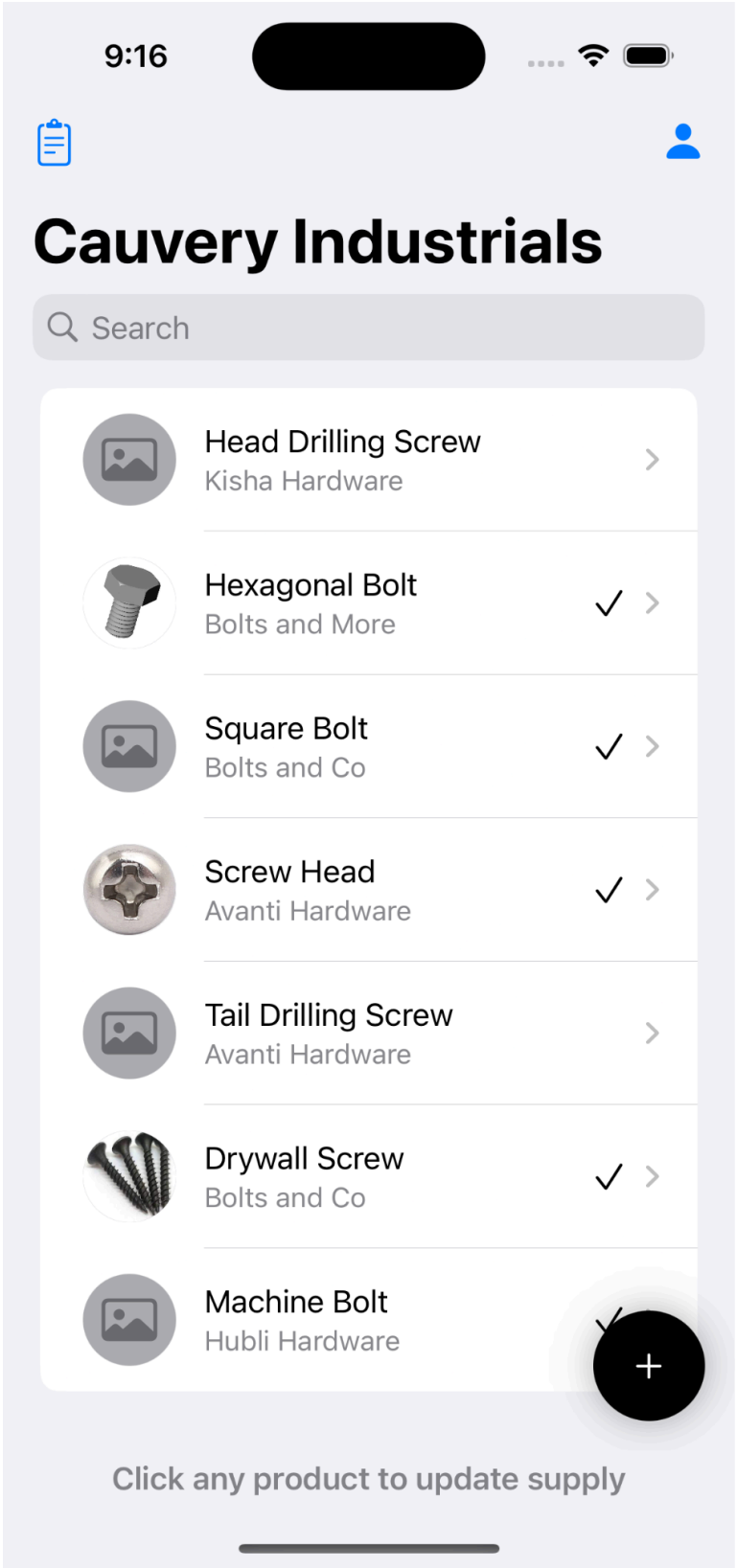


Fig 1.5

Technique 4&5:

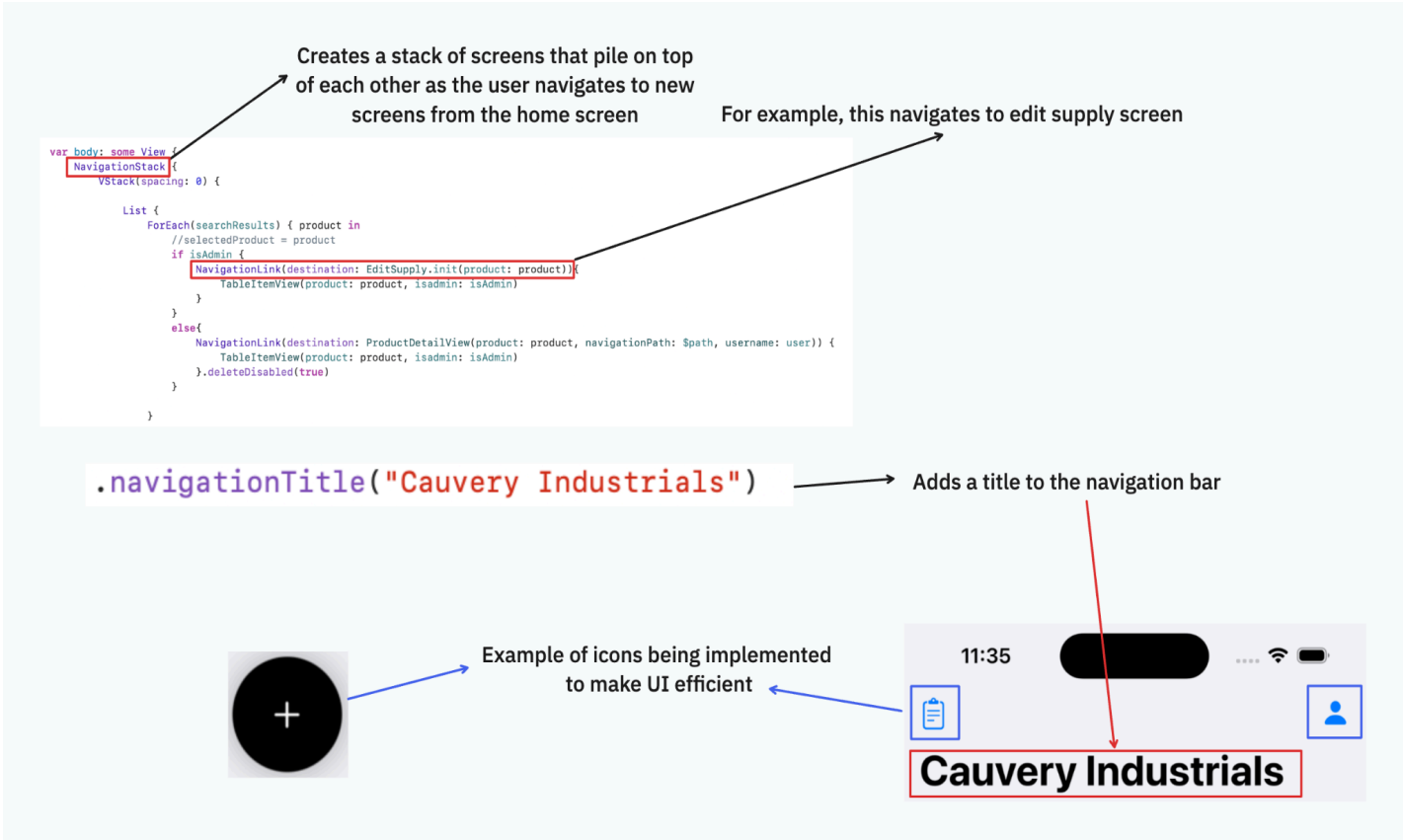


Fig 1.6

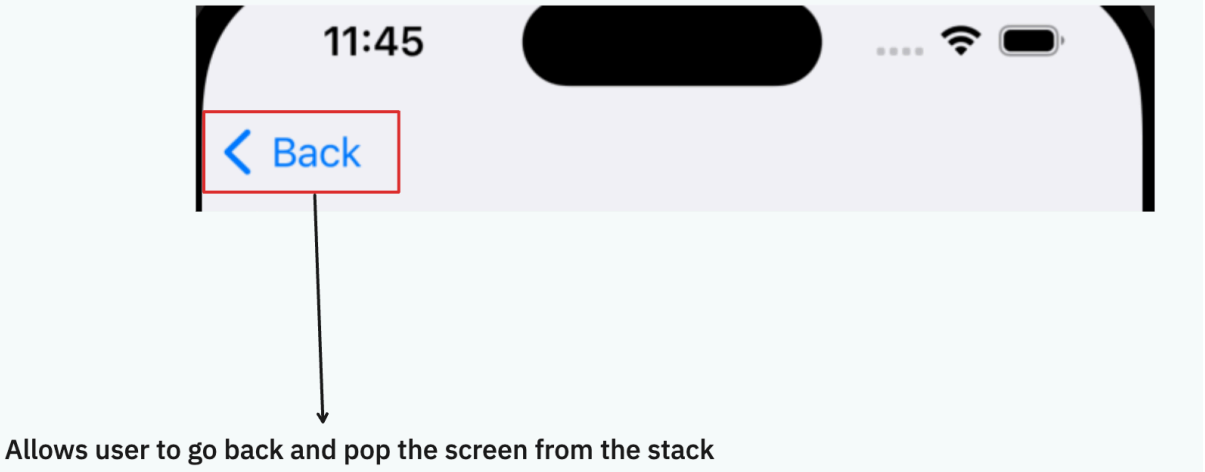


Fig 1.7

Technique 8:

Fig 1.8

Fig 1.9

Technique 9:

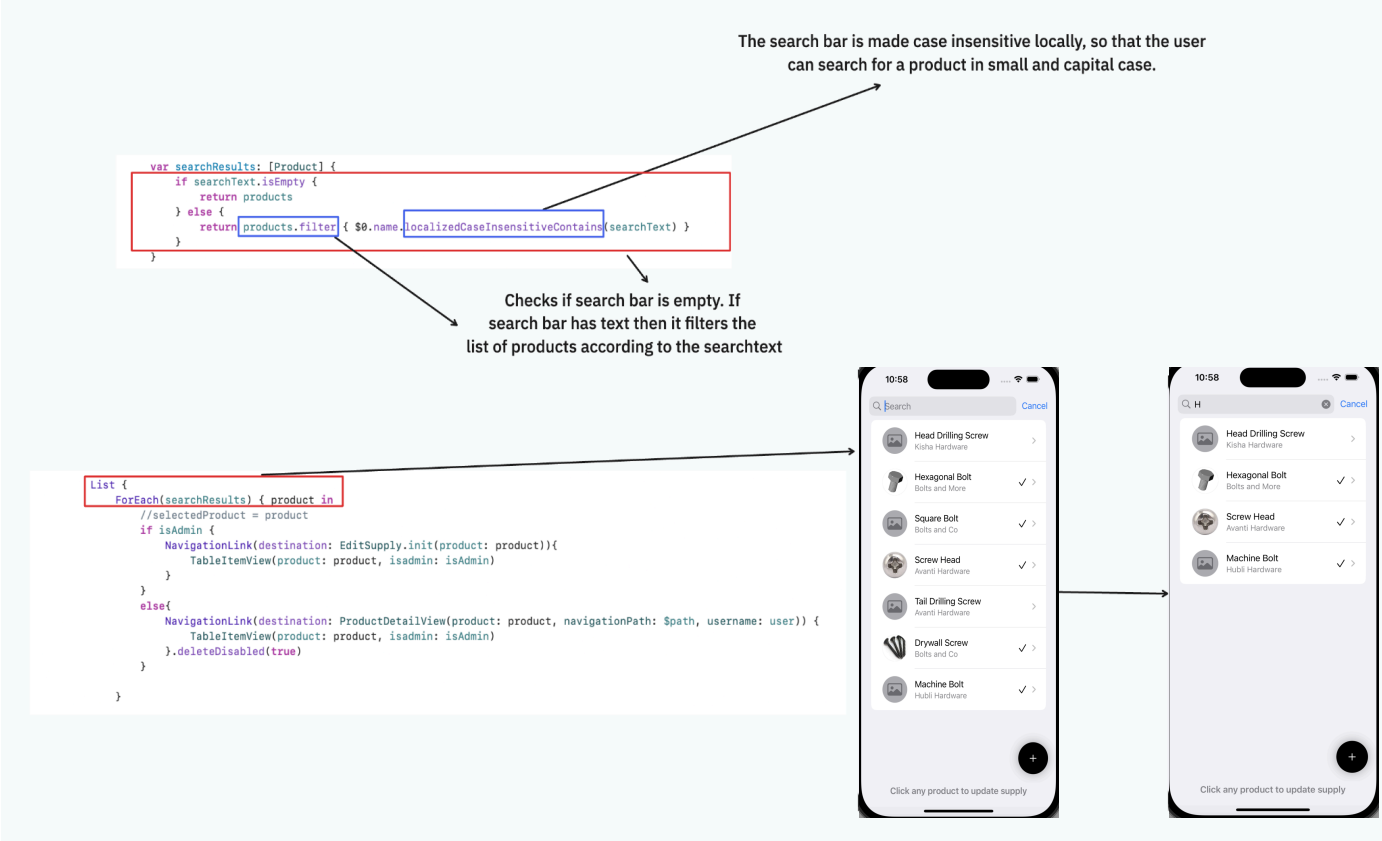


Fig 2.0

Technique 10:

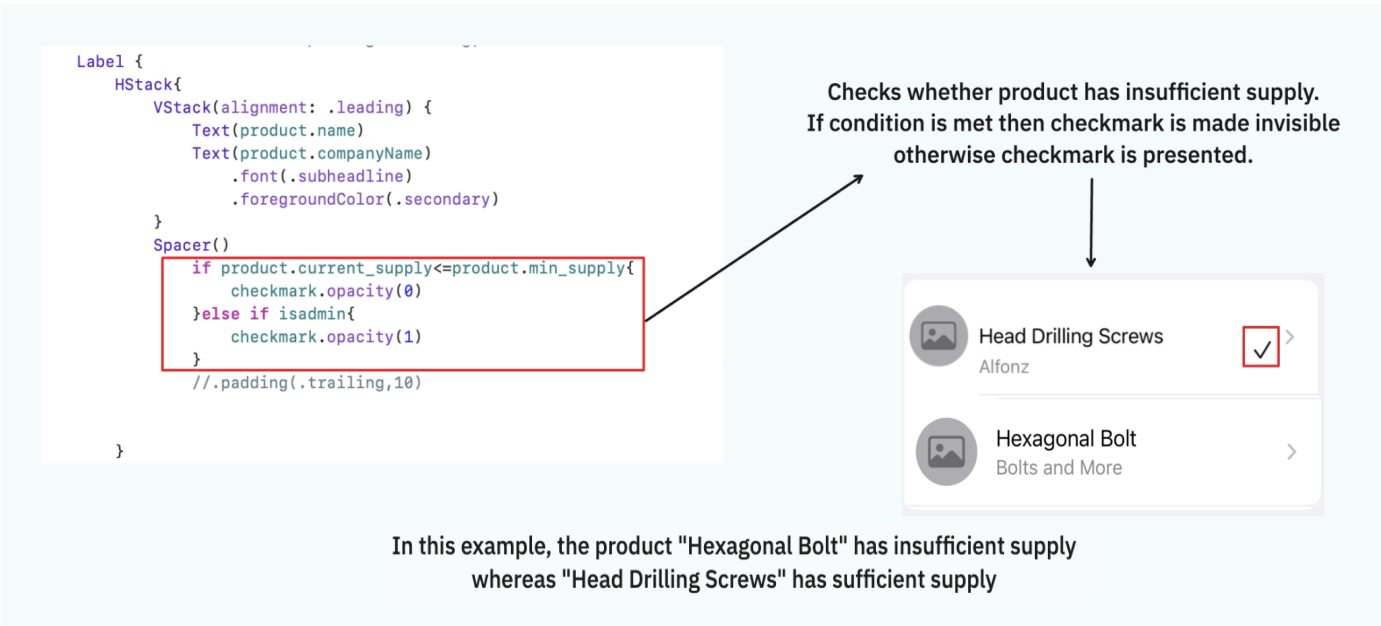


Fig 2.1

Technique 11:

Fig 2.2

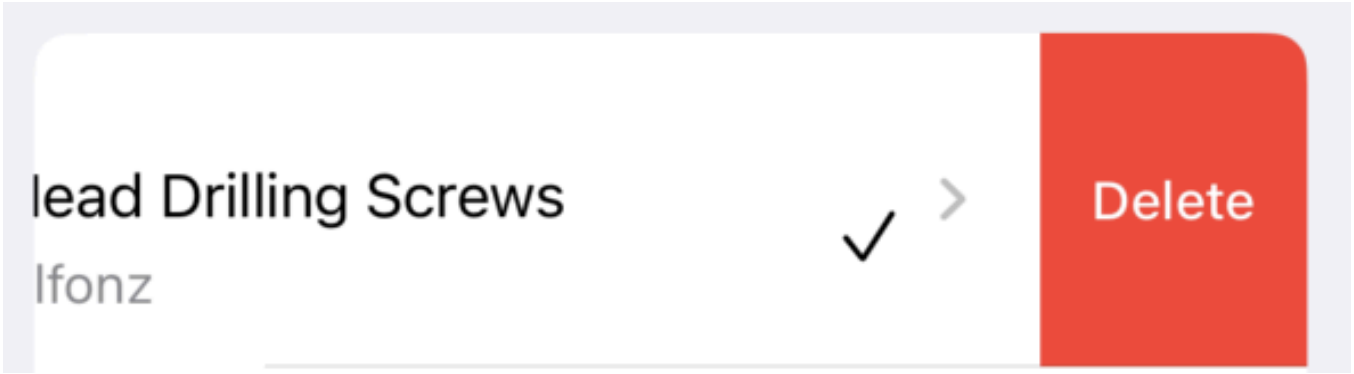
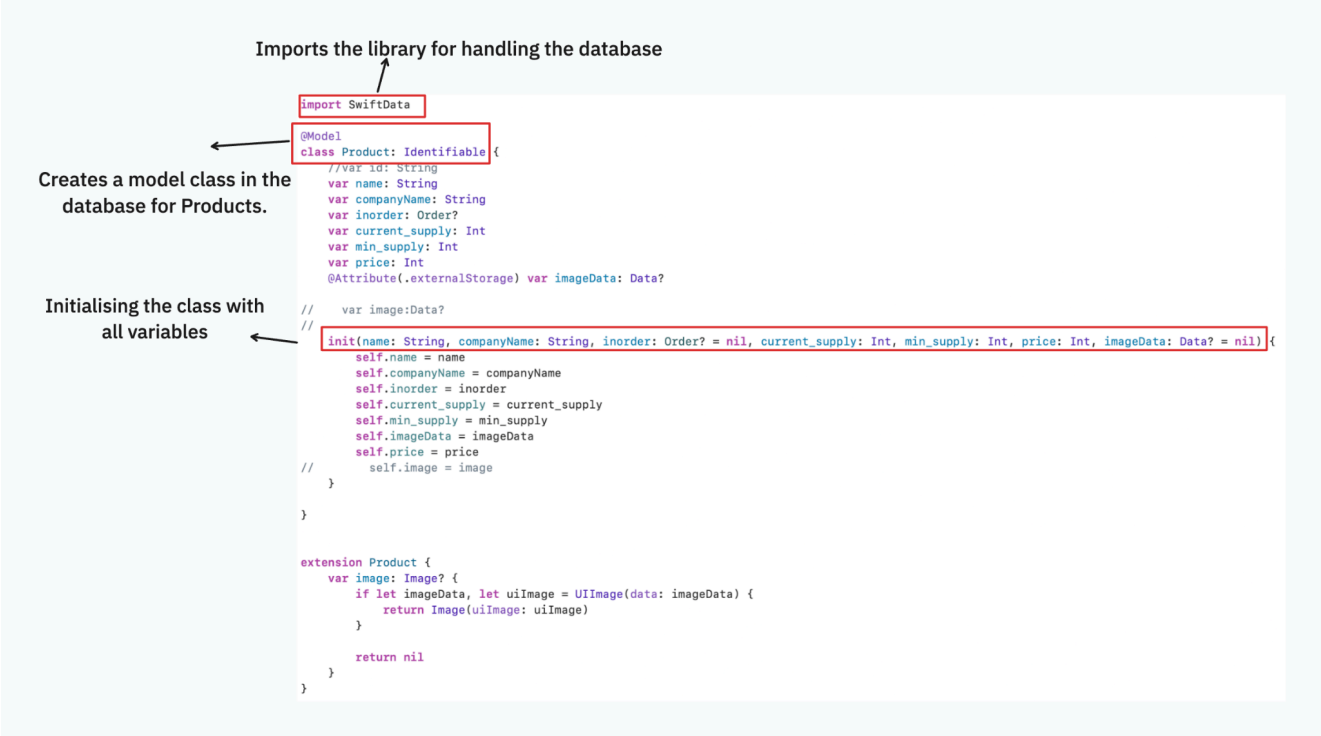


Fig 2.3

Swift Data (No view)
Technique Number 6:




```swift
import SwiftData

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

//    var image:Data?
//
    init(name: String, companyName: String, inorder: Order? = nil, current_supply: Int, min_supply: Int, price: Int, imageData: Data? = nil) {
        self.name = name
        self.companyName = companyName
        self.inorder = inorder
        self.current_supply = current_supply
        self.min_supply = min_supply
        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


```swift
import SwiftUI
import SwiftData
@main
struct final_testApp: 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)

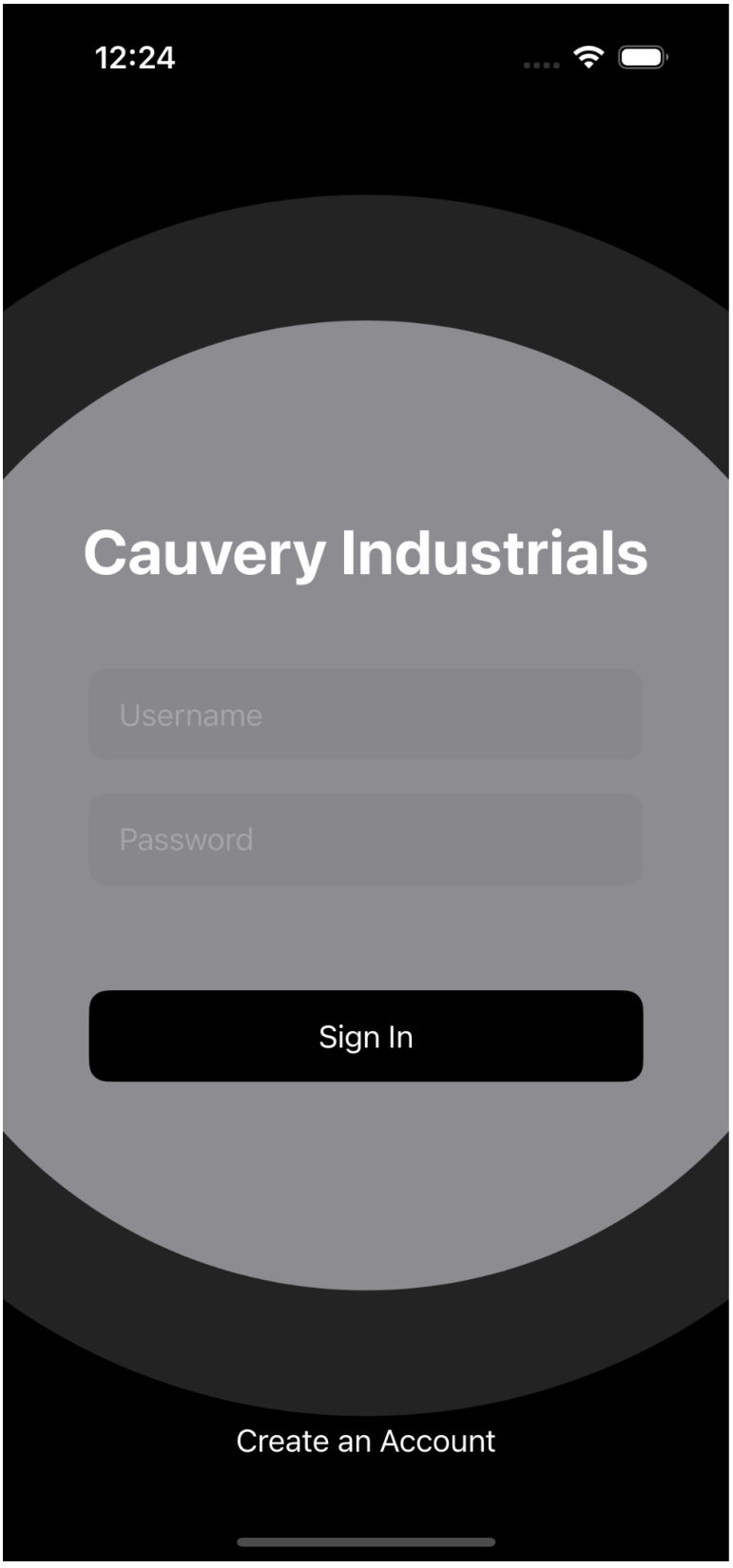


Fig 2.6

Technique 7:

Fig 2.7

View: Orders View

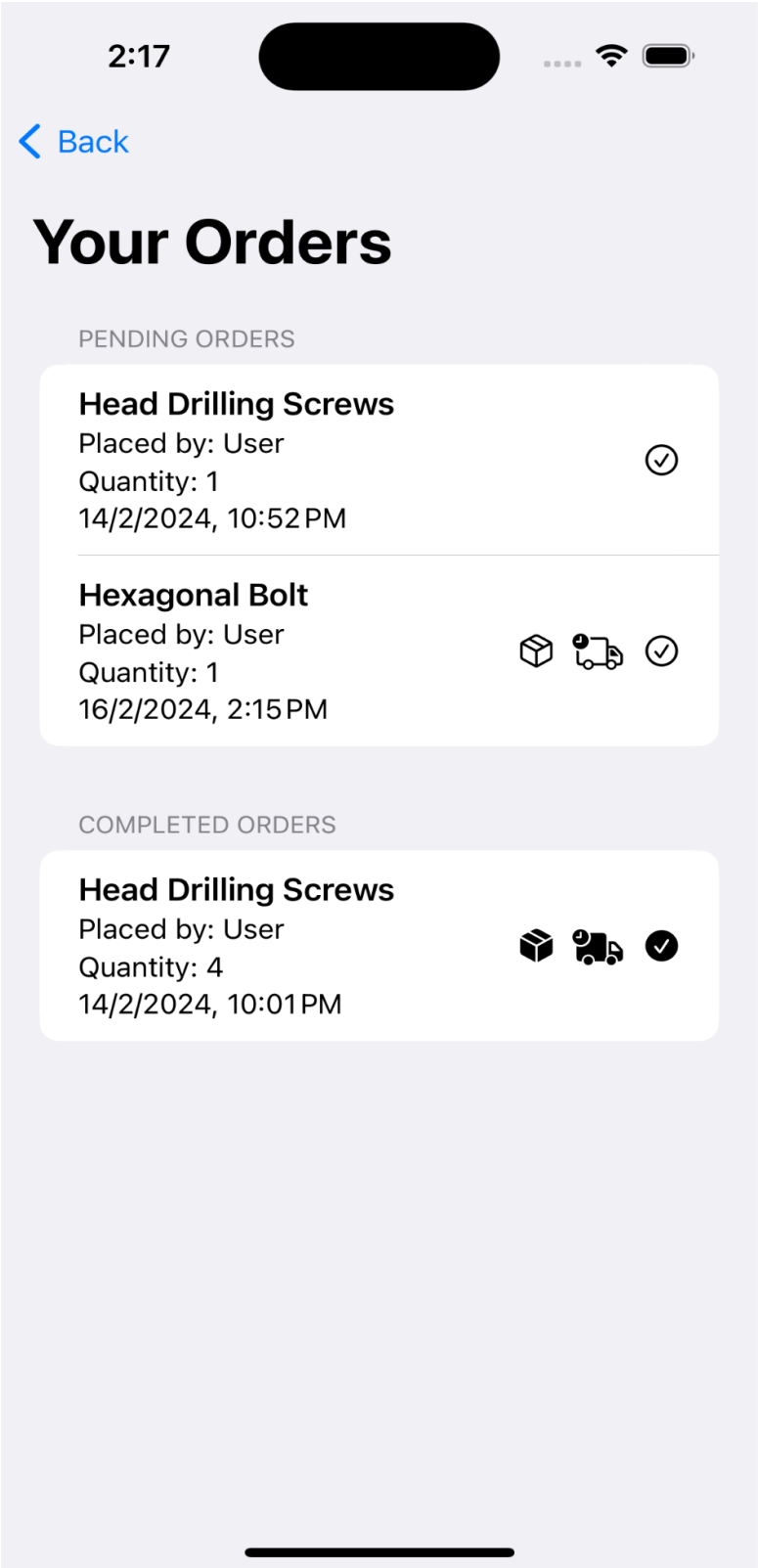


Fig 2.8

Technique 12:

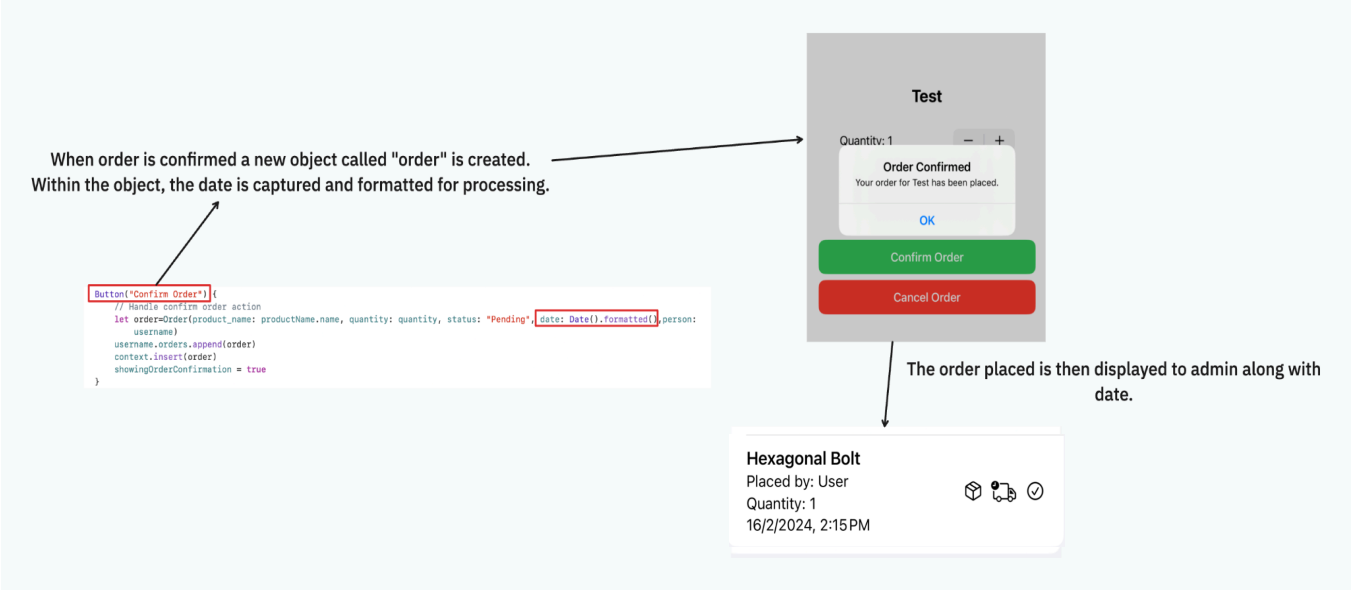


Fig 2.9

Technique 14:

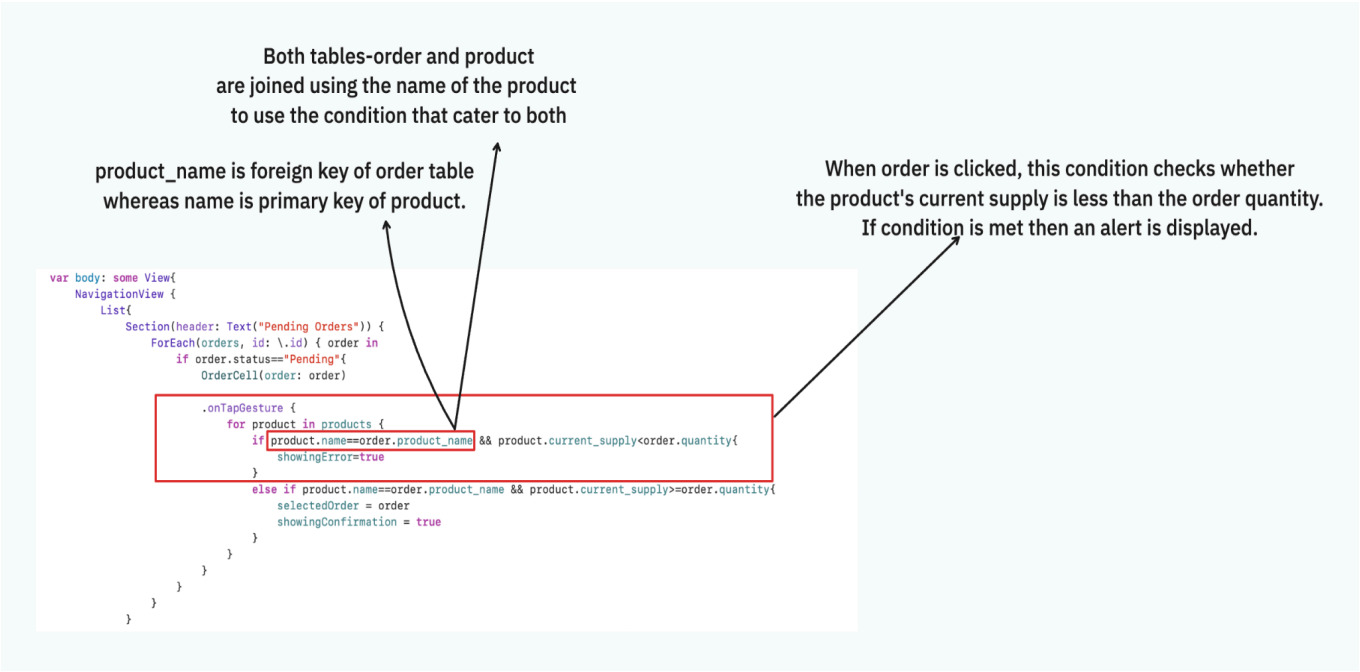


Fig 3.0

View: Profile View (for Admin)

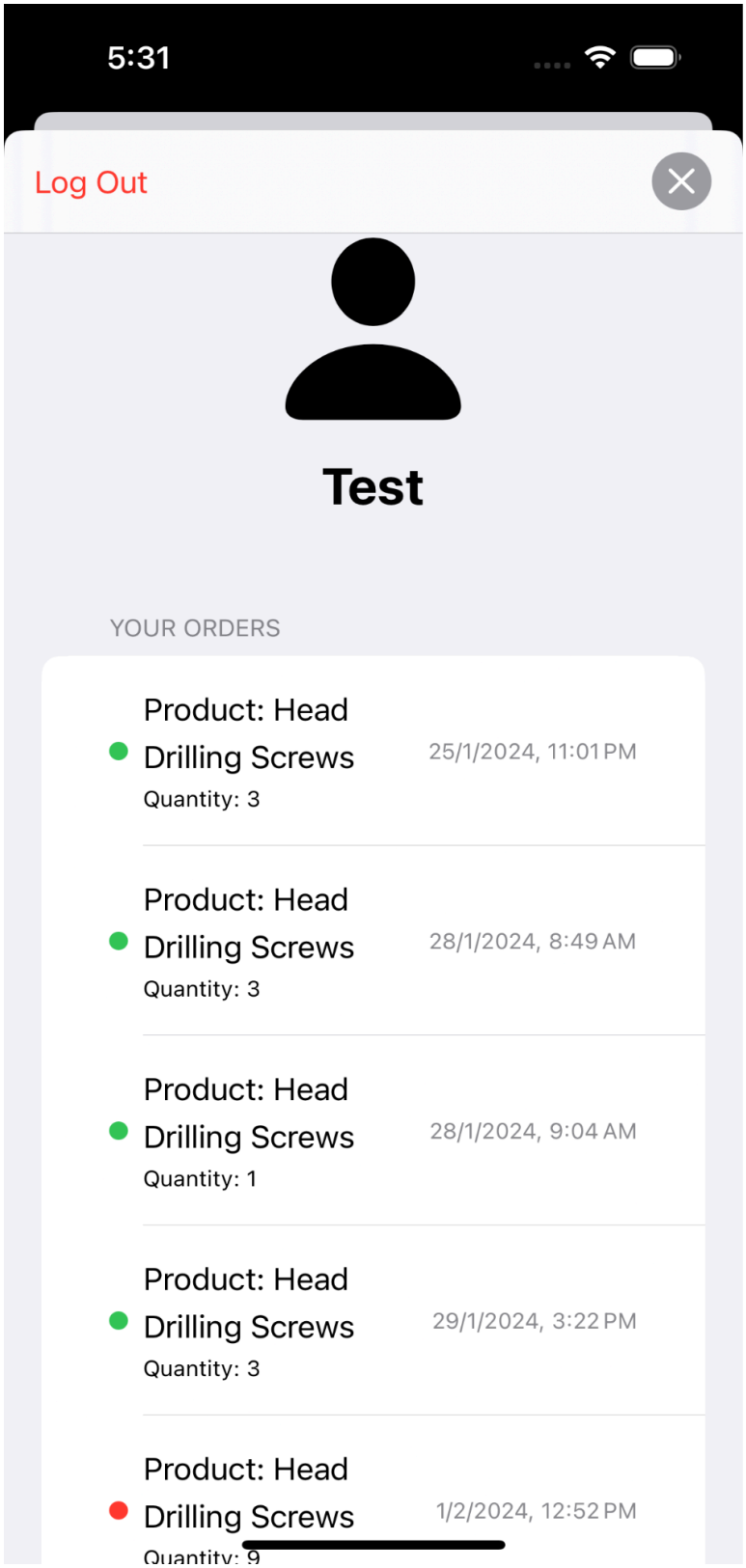


Fig 3.1

Technique 13:

Fig 3.2

View: Profile View (for customers)

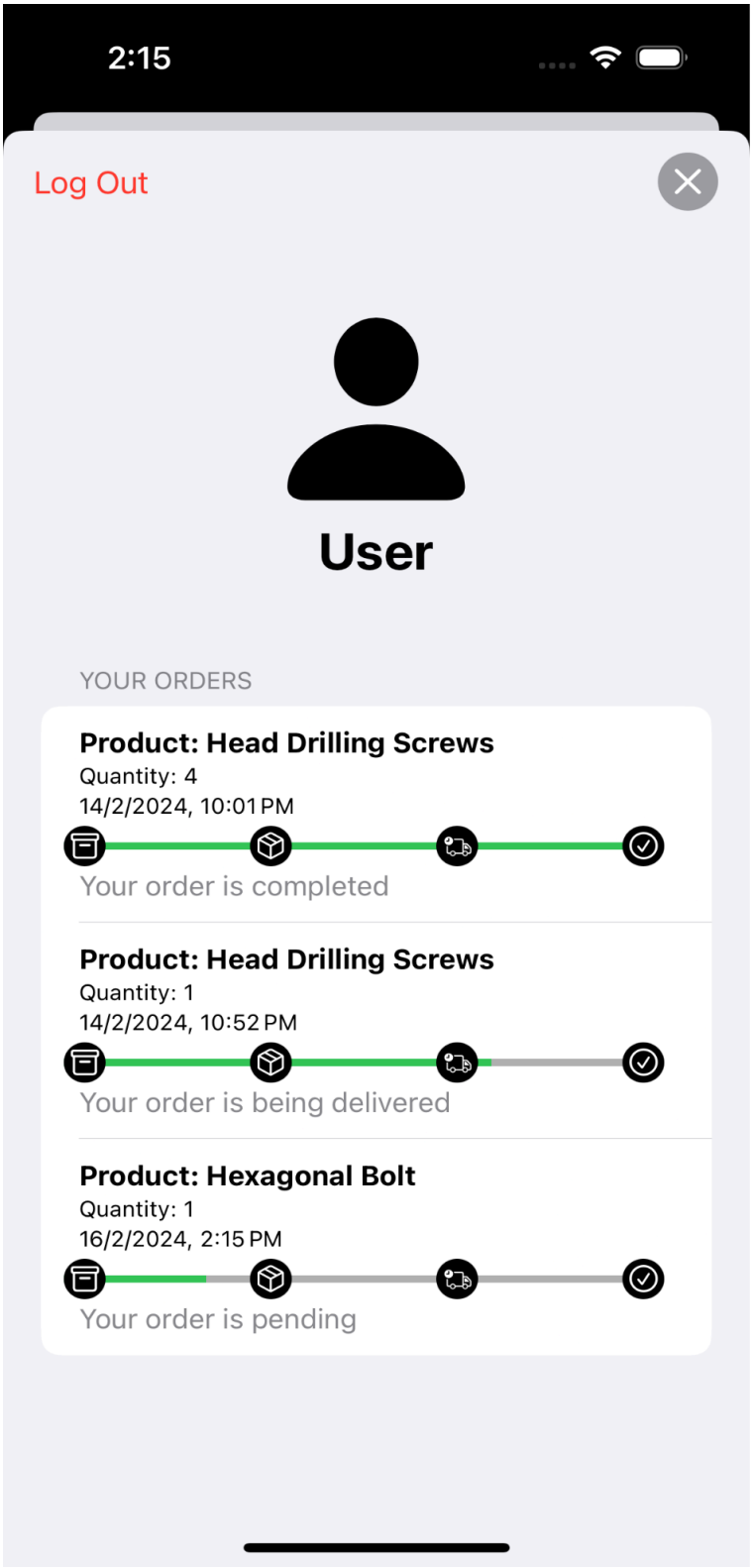
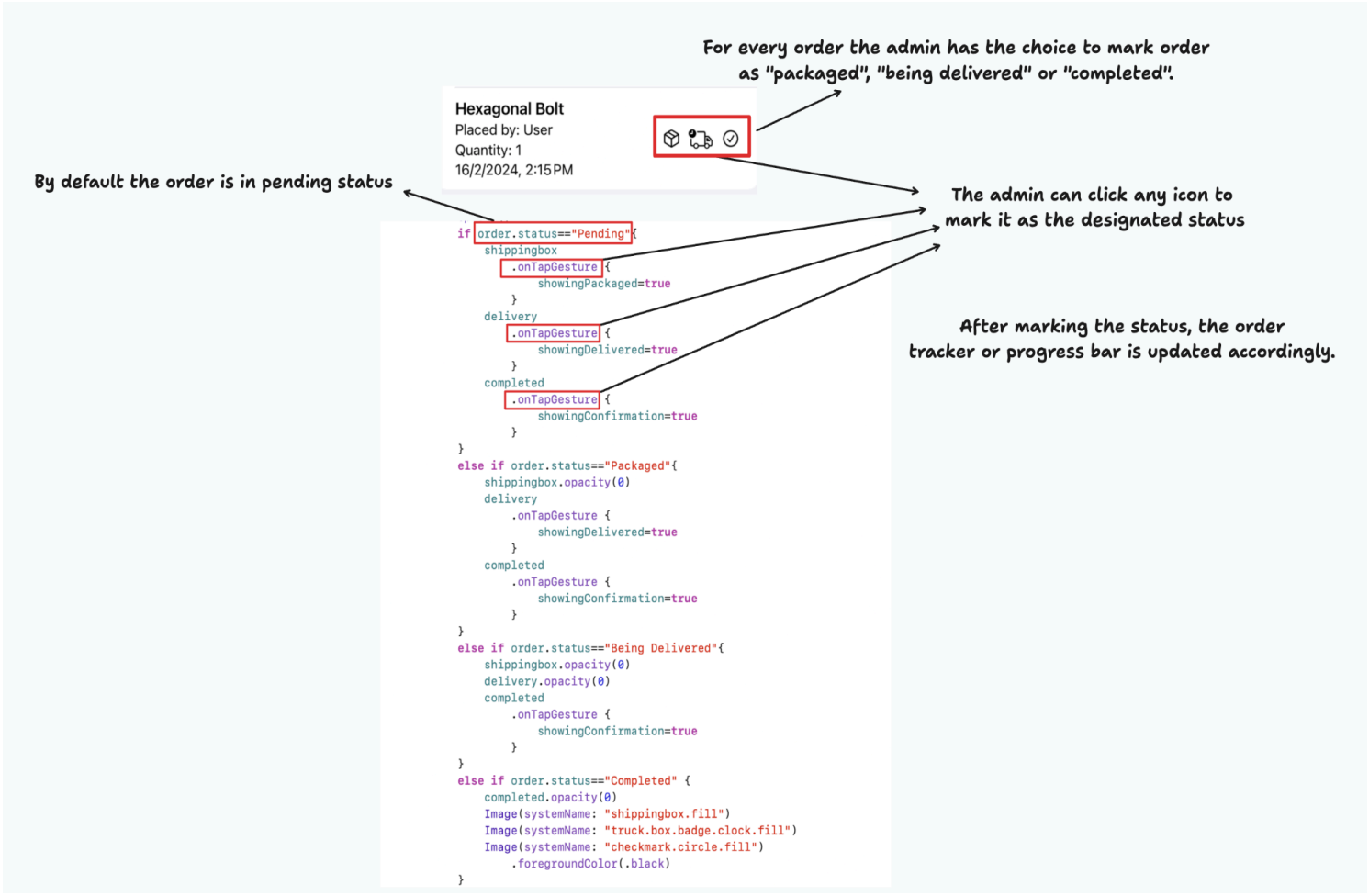


Fig 3.3

Technique 18:



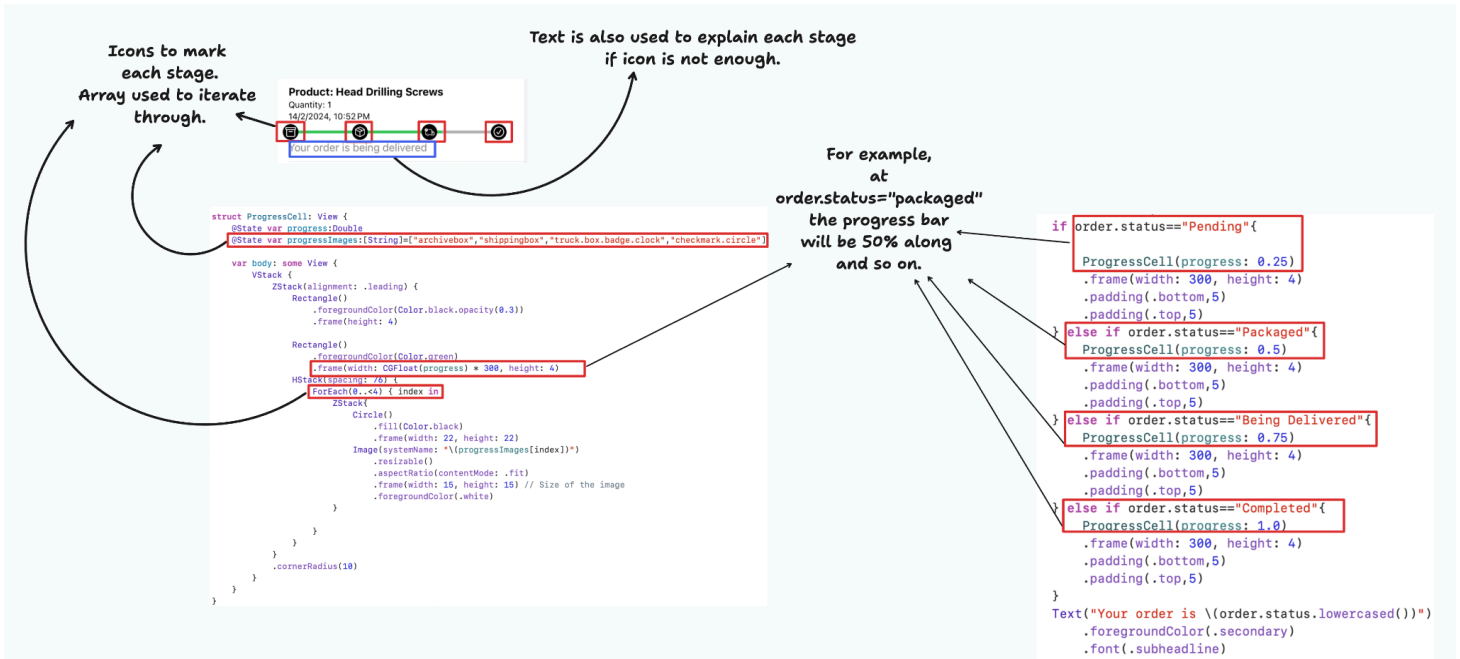


Fig 3.4

View: Add Product Sheet

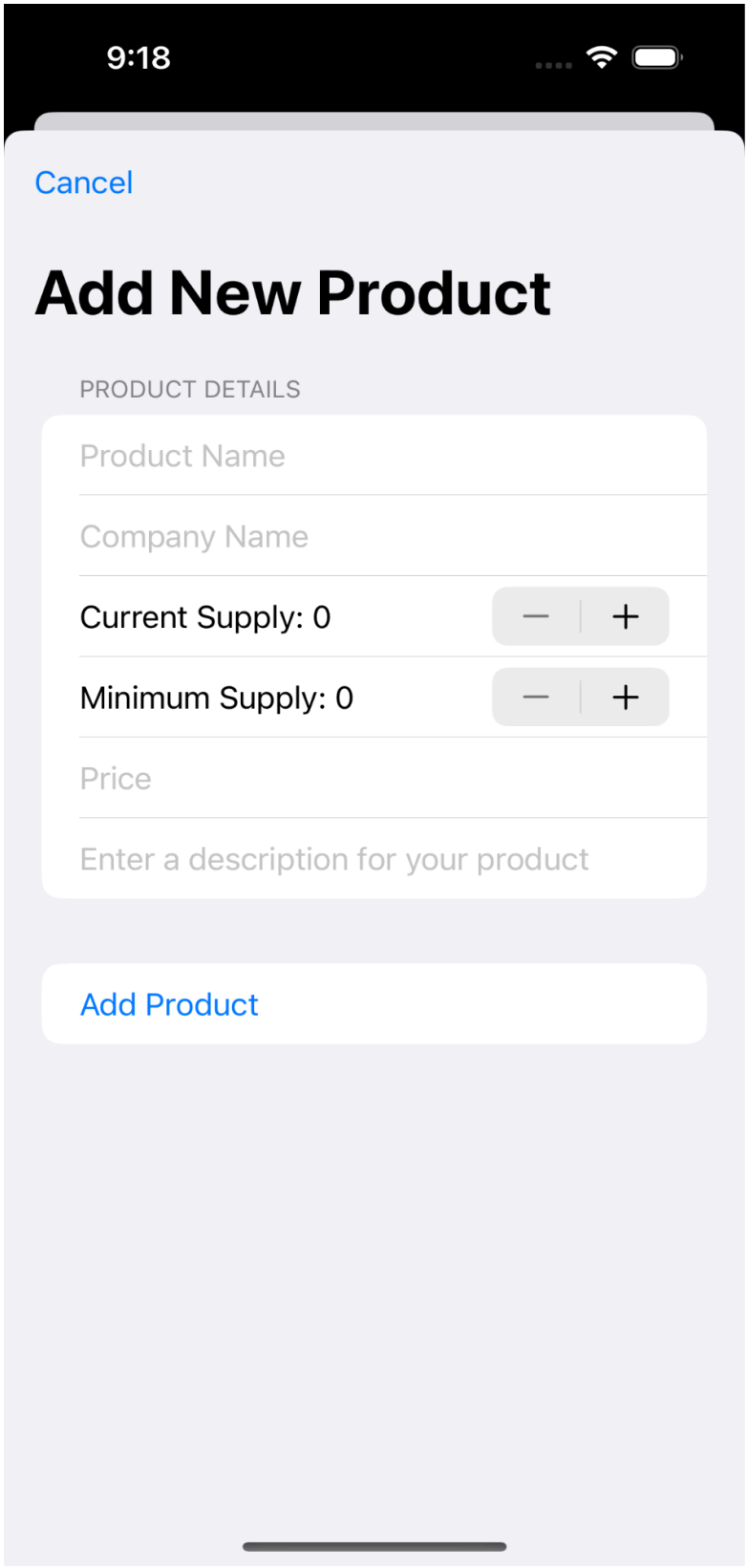


Fig 3.5

Technique 15:

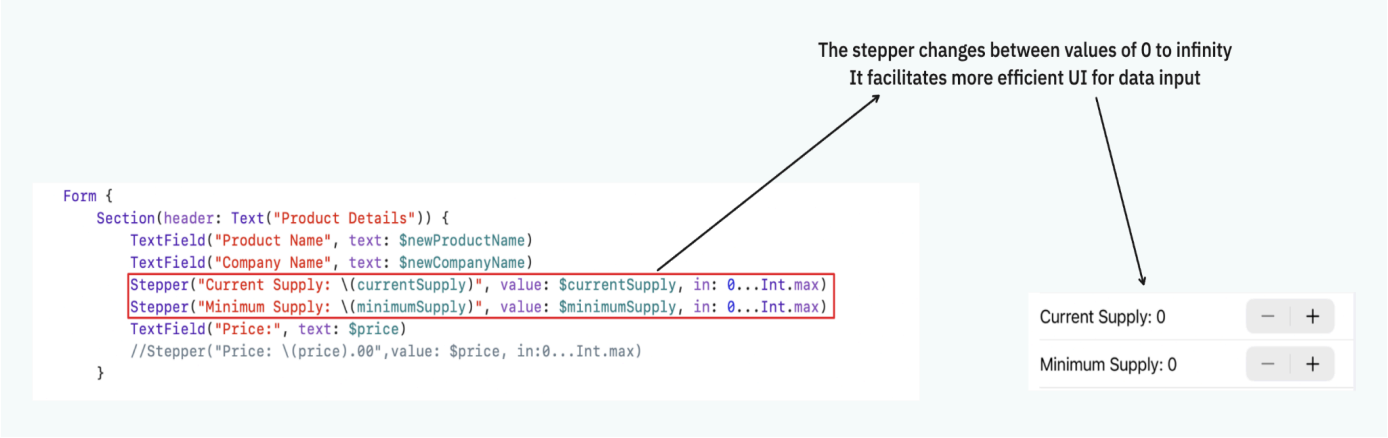


Fig 3.6

View: Place Order View

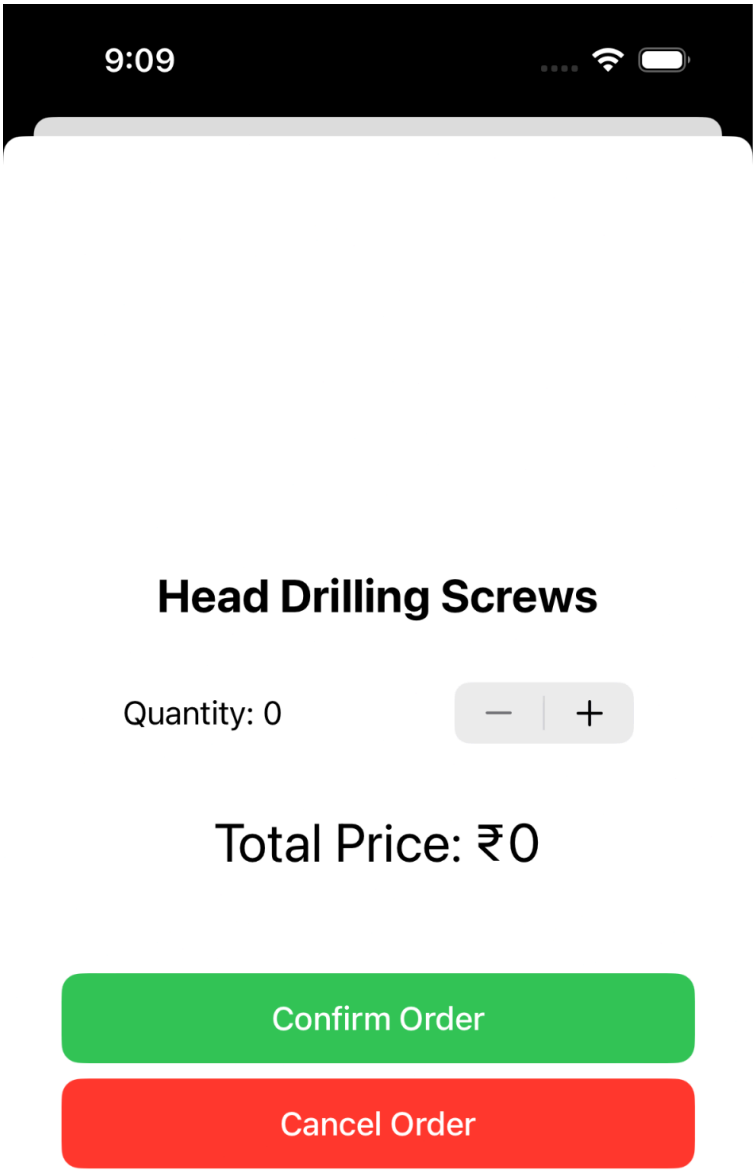


Fig 3.7

Technique 17:

Fig 3.8

View: Table View (for customers)

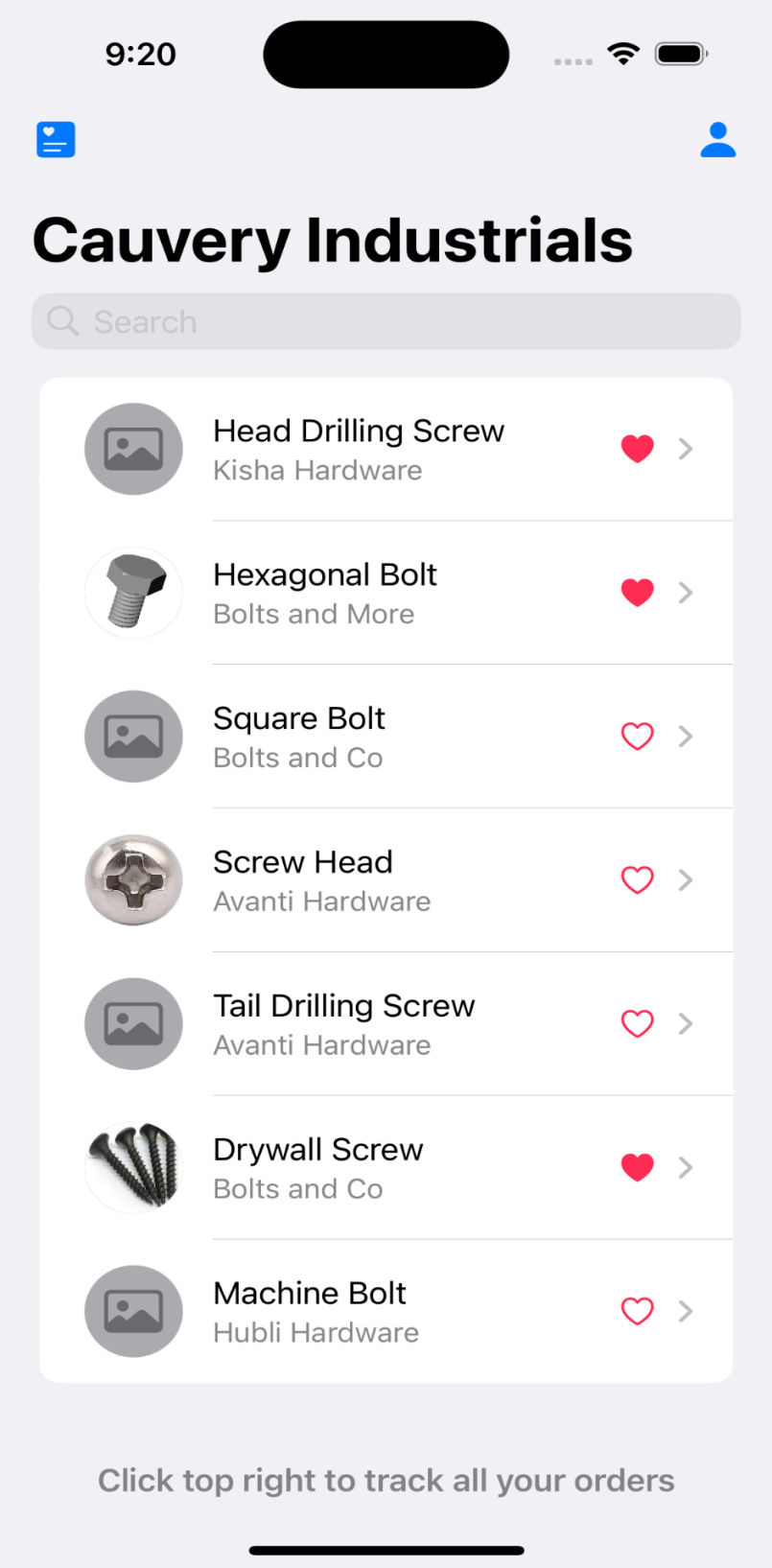


Fig 3.9

Technique 16:

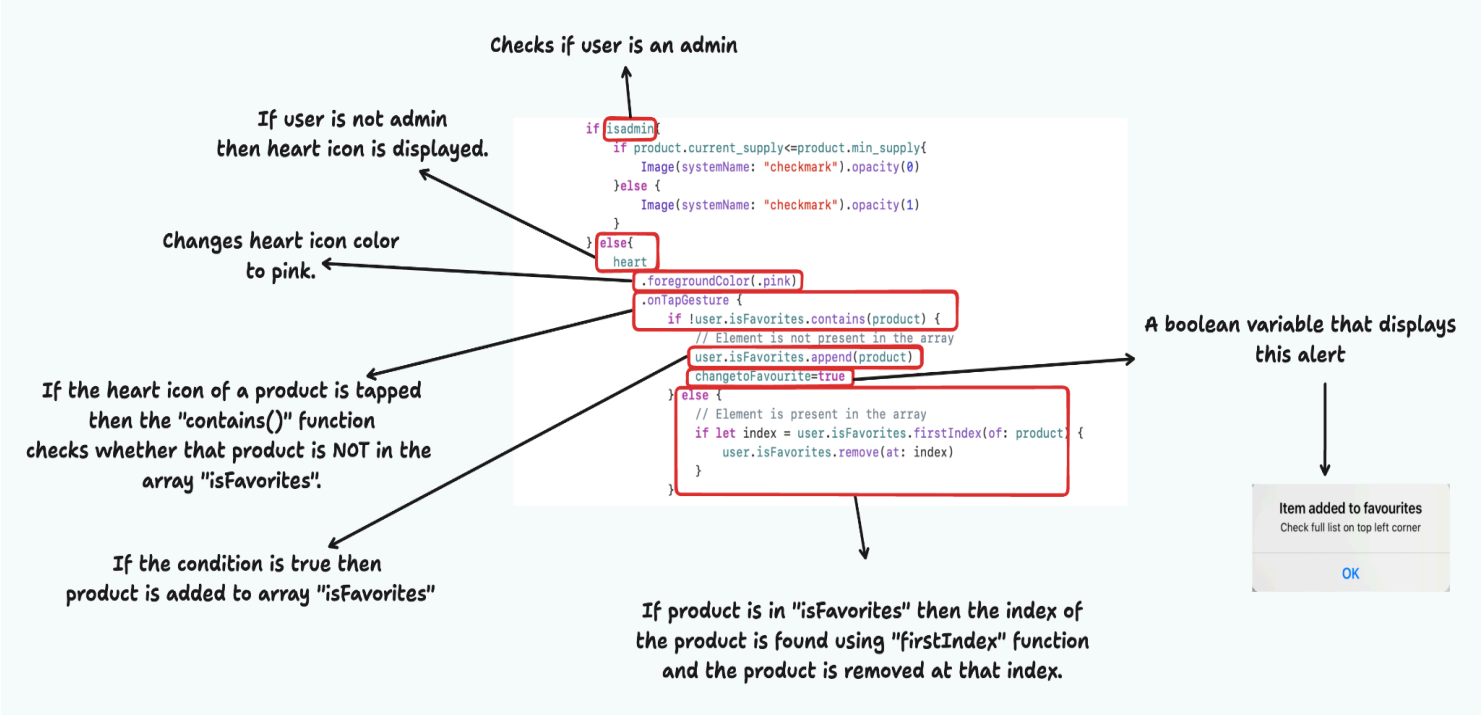


Fig 4.0

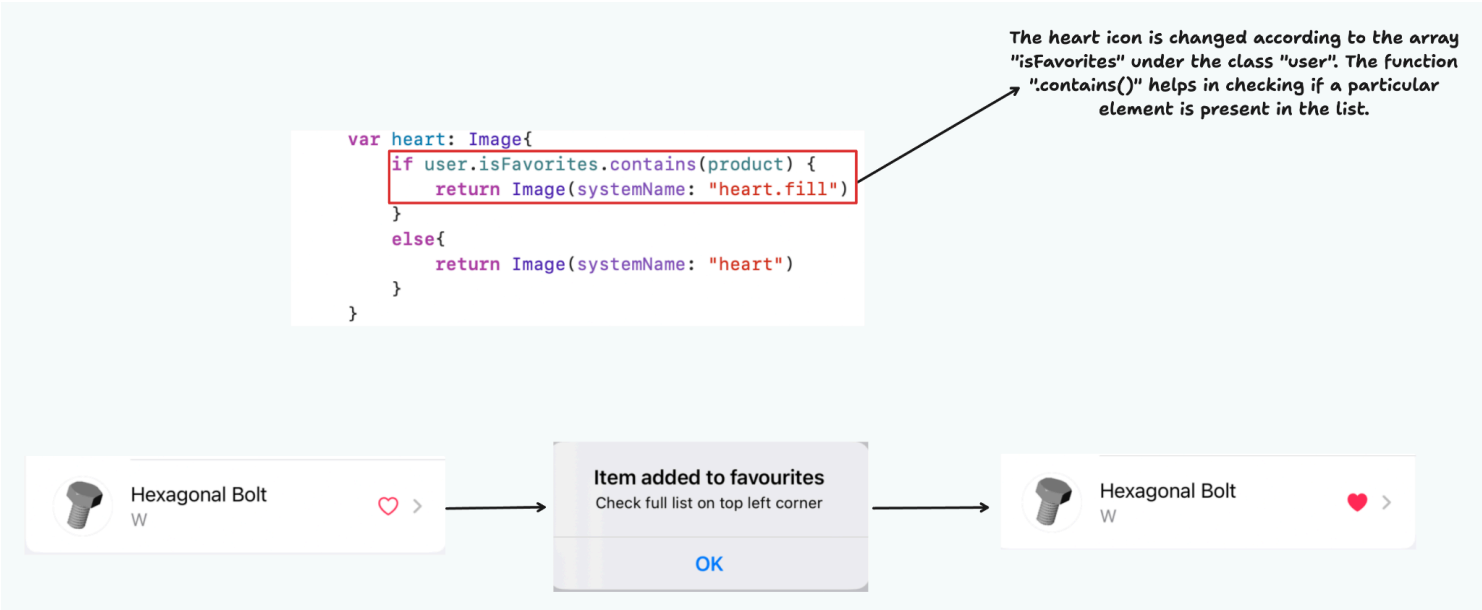


Fig 4.1

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

View: Favourites View

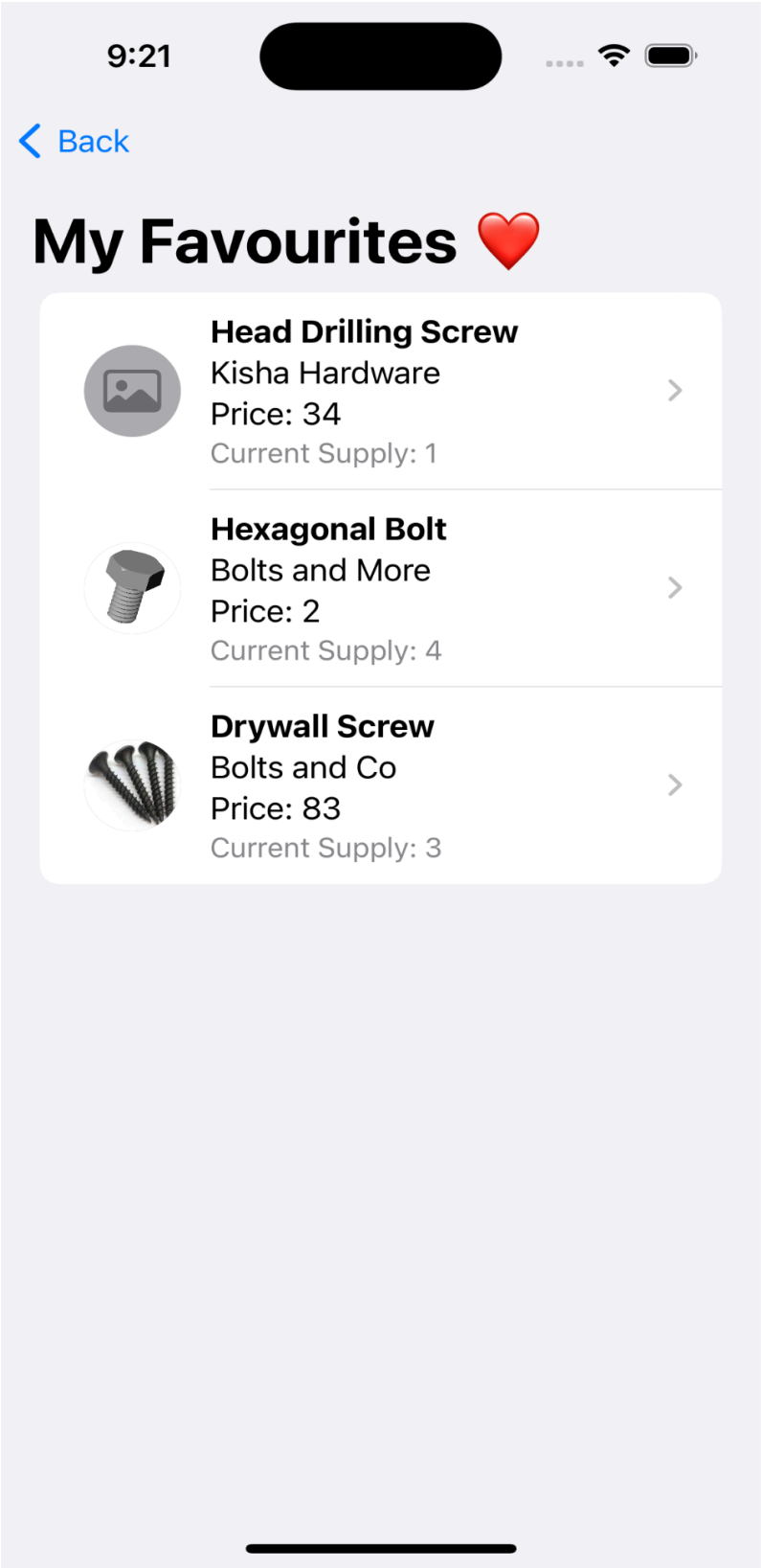


Fig 4.2