Class Name : CSC680 IOS application
Teammates : Ting Feng(Christine), Hugo Gomez, Jacob Torres, Yonatan Leake

Git repo : https://github.com/JTTorres-code/CSC680-Final-Project
ID : ChristineLoveCoding
ID: JTTorres-code
ID: ItsHugo28
ID: Yontheezus

MileStone 1 :

1. SetUp Git repo, intro meeting
2. SetUp ReadMe in Git repo : ReadMe should have an overview of the project as well as the name and student ID of each teammate.
3. Invite others to join git repo
4. Description of the project
5. Project tutorial resources sharing, learning the study material, setting up the weekly meeting for update progress.

MileStone 2 :

1. Design → Design all the features, using figma to do the protocols
2. Mission → What kind of problem are we trying to solve ?
3. Implementation → The major coding part, write a lot of codes to implement the key features.
4. Test → Test the application using various methods.
5. Deployment(optional) → We can try to ship the application to the apple store.

MileSone 3

1. Submission : Code + Documentation.
2. Class Demonstration : Presentation.

✅ Checklist App – Final Project (CSC 660/680)

📱 Project Description
This is a simple Checklist App that allows users to create and manage checklists. Users can add tasks, mark them as complete, and delete them. It's designed to help with productivity, grocery lists, or any task-oriented workflow. All data is saved locally using `UserDefaults`.

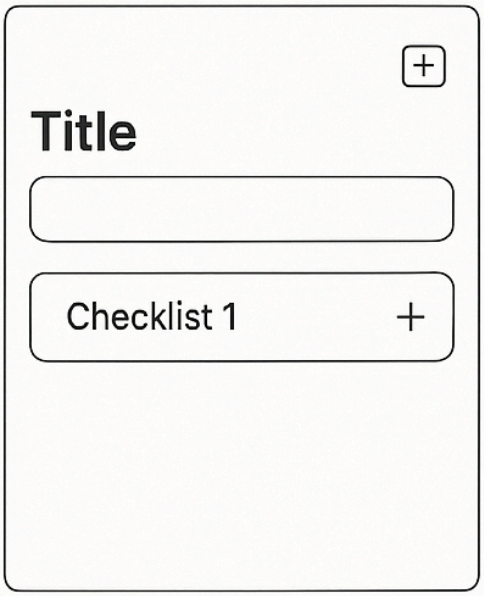
🚀 Must-Have Features
- [x] Create a checklist
- [x] Add items to a checklist
- [x] Mark items as complete/incomplete
- [x] Delete items
- [x] Save data locally using `UserDefaults`

✨ Nice-to-Have Features
- [x] Create multiple named checklists
- [x] Reorder checklist items
- [x] Support dark/light mode
- [x] Custom icons or colors for checklists

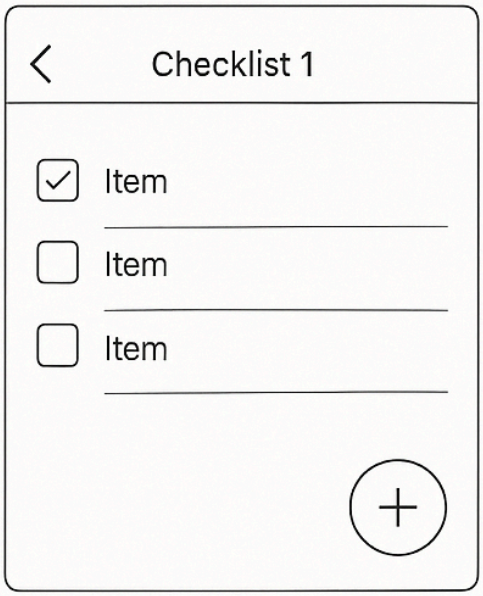
🖼️ Wireframes (In Progress)
1. **Home Screen** – Displays all checklists
2. **Checklist Detail View** – Shows checklist items and options
3. **Add Item Screen** – Simple UI to add a new item

🛠️ Tools & Technologies
- Swift / SwiftUI
- Xcode
- Apple Human Interface Guidelines
- Local persistence using `UserDefaults`

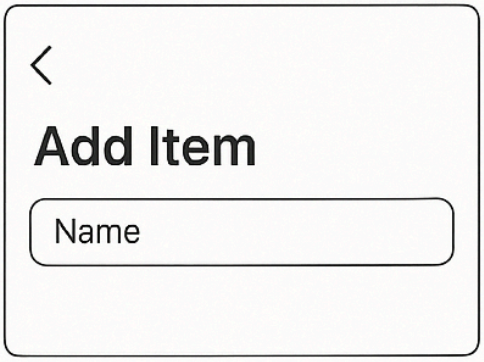
Prototype



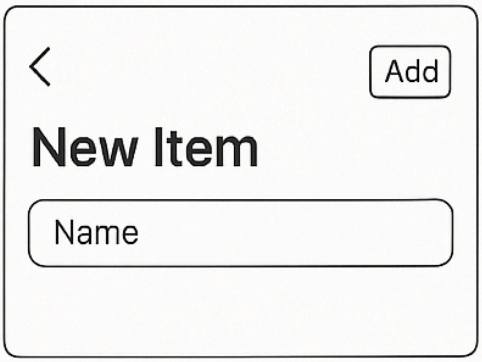
Home Screen



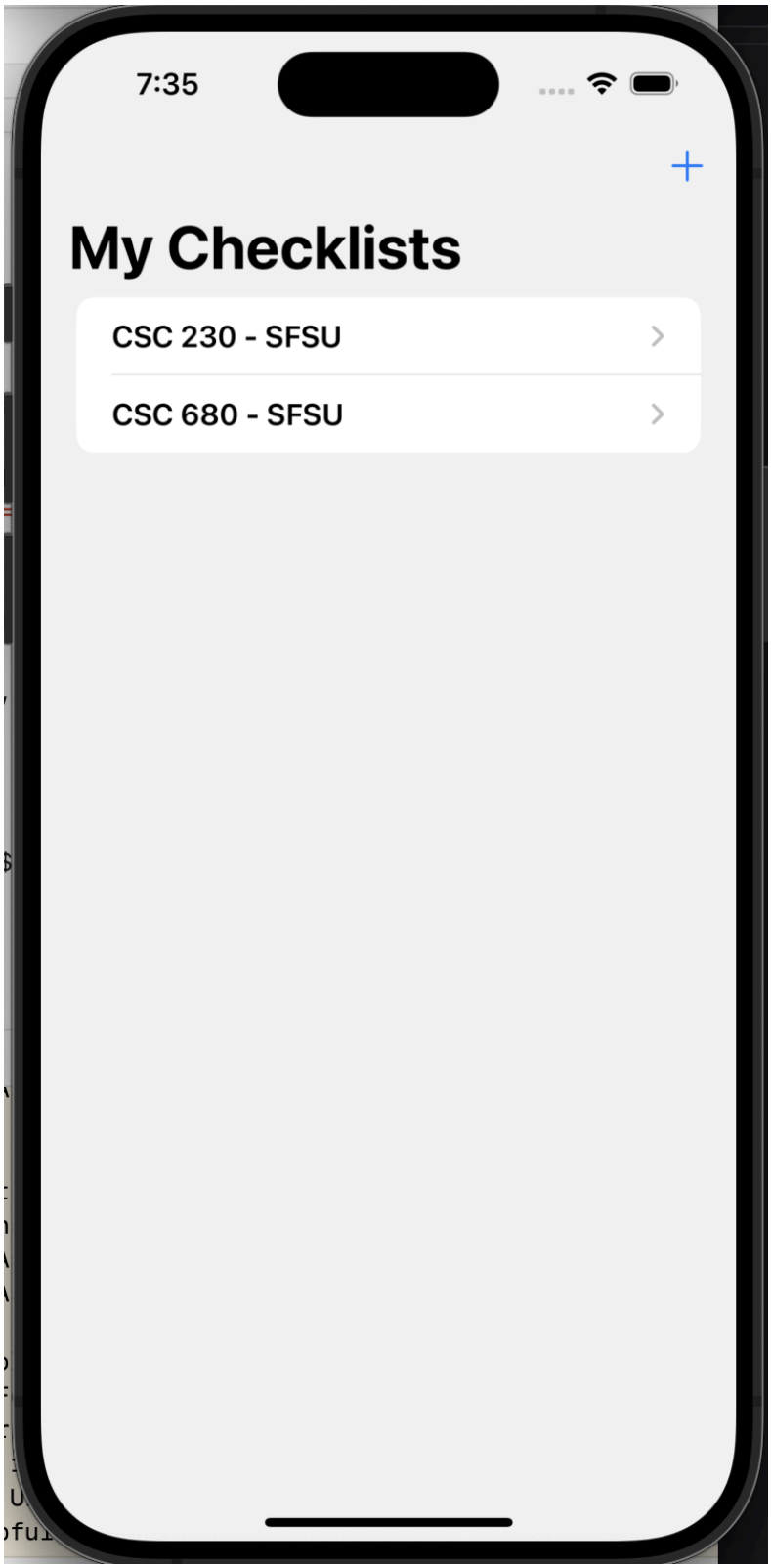
Checklist Detail

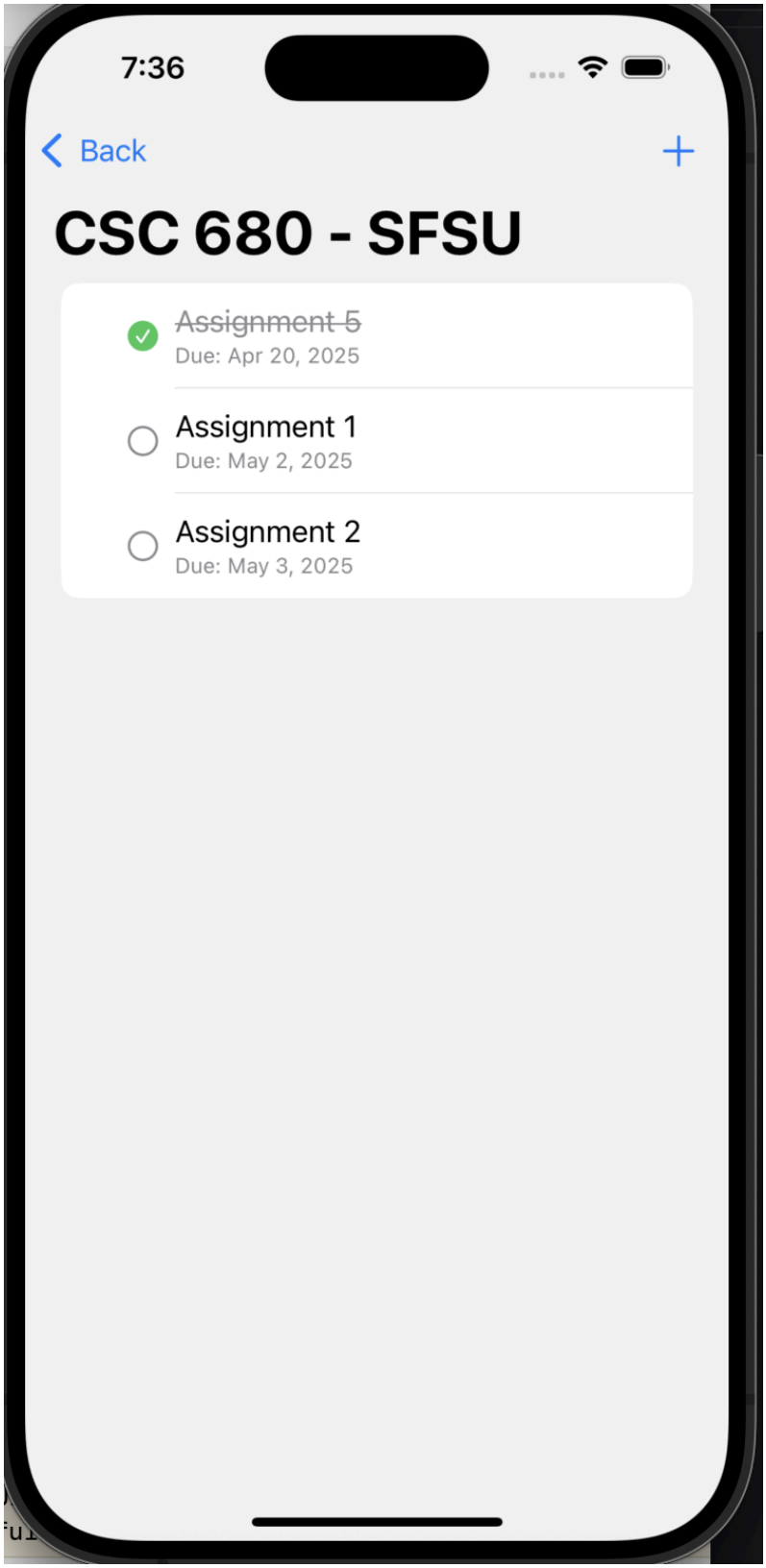


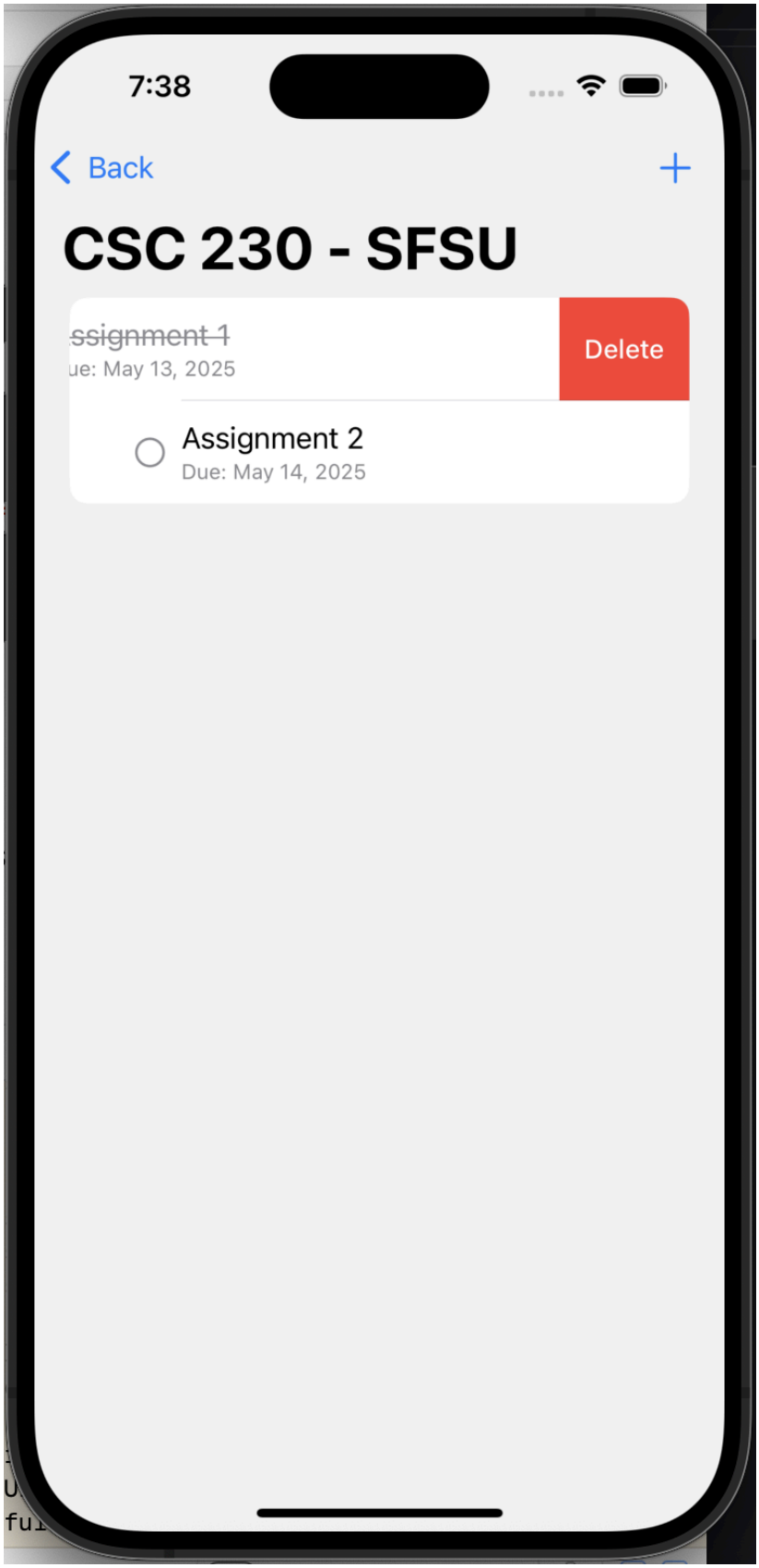
Add Item View

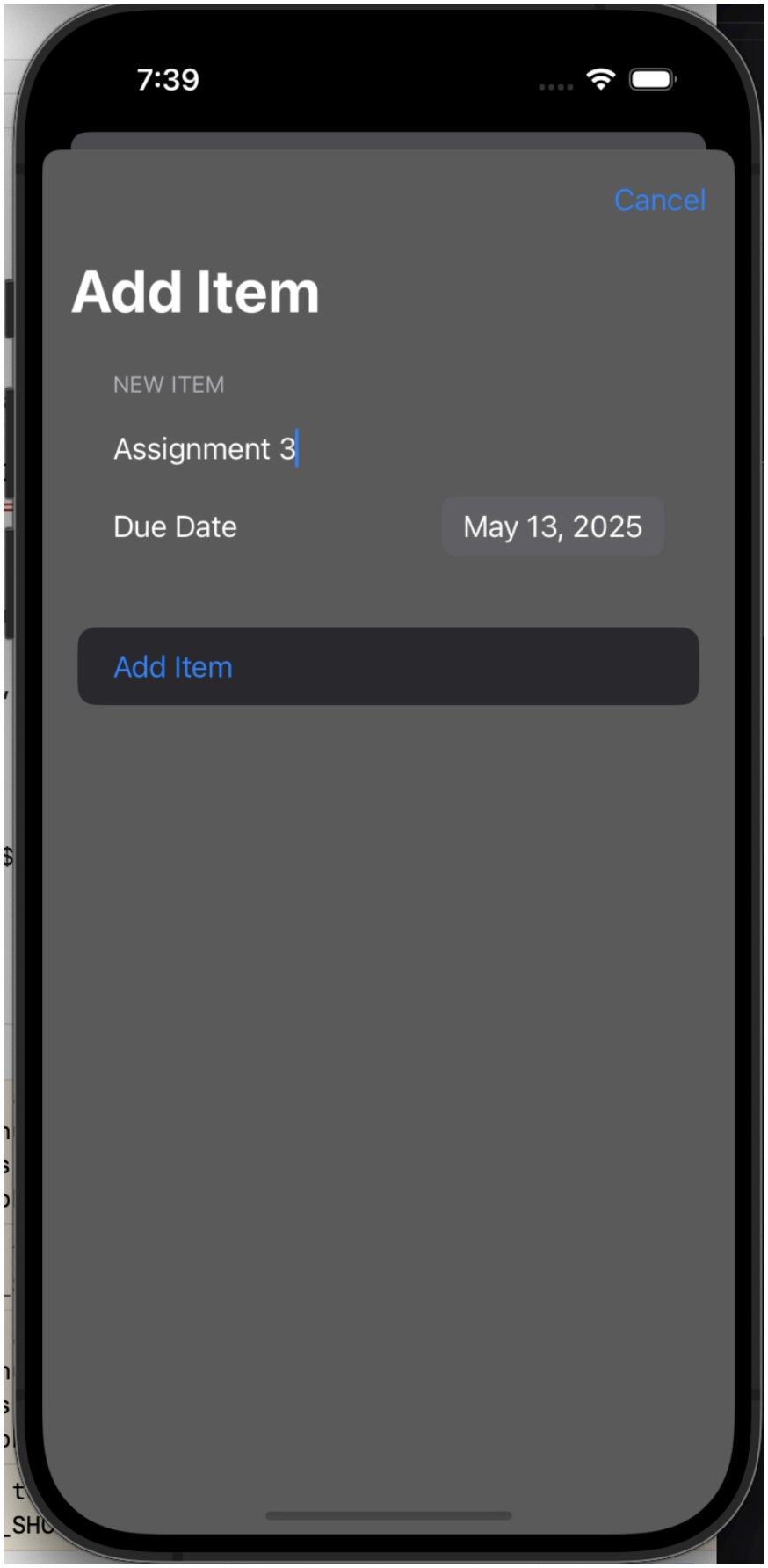


Add Item View


7:35
My Checklists
CSC 230 - SFSU
CSC 680 - SFSU


7:36
Back
CSC 680 - SFSU
Assignment 5
Due: Apr 20, 2025
Assignment 1
Due: May 2, 2025
Assignment 2
Due: May 3, 2025


7:38
Back
CSC 230 - SFSU
ssignment 1
ue: May 13, 2025
Delete
Assignment 2
Due: May 14, 2025


7:39
Cancel
Add Item
NEW ITEM
Assignment 3
Due Date
May 13, 2025
Add Item

📌 Must-Have Features & Time Estimates

Feature	Description	Est. Time
✅ Create a checklist	UI + local data model using `UserDefaults`	1.5 hrs
✅ Add items to checklist	Form input + append to data structure	1.5 hrs
✅ Mark items complete/incomplete	Toggle checkmark and save state	1 hr
✅ Delete items	Swipe to delete or long-press delete	1 hr
✅ Persist data with `UserDefaults`	Encode/decode checklist structure to local storage	2 hrs
✅ Build basic UI screens	NavigationView, List views, Add screens	2 hrs

Subtotal: ~9 hours

✨ Nice-to-Have Features & Time Estimates

Feature	Description	Est. Time
Create multiple checklists	Add list of checklist models and allow switching between them	2.5 hrs
Reorder checklist items	Implement drag-and-drop sorting	2 hrs
Dark/light mode toggle	Leverage SwiftUI environment or settings	0.5 hr
Custom icons/colors	Use SF Symbols and color picker	1.5 hrs

Subtotal (Optional): ~6.5 hours

📊 Summary

- **Total Must-Have Time**: ~9 hours

- **Total Nice-to-Have Time**: ~6.5 hours (optional)
- **Total (if doing everything)**: ~15.5 hours

🧱 Phase 1 – Coding Breakdown for the Checklist App

We'll start with **Must-Have Features Only**. Once those are done, we can explore optional ones.

✅ Task 1: Define the `ChecklistItem` Model

- A simple struct to represent each checklist item.
- Fields: `id`, `title`, `isCompleted`

✅ Task 2: Create the `ChecklistViewModel`

- A class that manages the array of items
- Functions:
    - Add item
    - Toggle item completion
    - Delete item
    - Save/load from `UserDefaults`

✅ Task 3: Build the `ChecklistView`

- Main list screen

- Displays checklist items
- Tap to toggle
- Swipe to delete

✅ Task 4: Add `AddItemView`

- Screen or sheet to add a new item
- TextField + Save button

✅ Task 5: Add `UserDefaults` Persistence

- Save checklist items
- Load on app launch

🔨 Let's Begin with Task 1: Define `ChecklistItem`

Here's the code for the model:

swift
CopyEdit

```
import Foundation

struct ChecklistItem: Identifiable, Codable {
    let id: UUID
    var title: String
    var isCompleted: Bool
```

```
    init(id: UUID = UUID(), title: String, isCompleted: Bool = false)
{
        self.id = id
        self.title = title
        self.isCompleted = isCompleted
    }
}
```

✅ Task 2: `ChecklistViewModel.swift`

swift
CopyEdit

```
import Foundation

class ChecklistViewModel: ObservableObject {
    @Published var items: [ChecklistItem] = [] {
        didSet {
            saveItems()
        }
    }

    private let itemsKey = "ChecklistItems"

    init() {
        loadItems()
    }

    // MARK: - Public Functions

    func addItem(title: String) {
        let newItem = ChecklistItem(title: title)
        items.append(newItem)
    }

    func toggleItem(_ item: ChecklistItem) {
        if let index = items.firstIndex(where: { $0.id == item.id }) {
```

```
            items[index].isCompleted.toggle()
        }
    }

    func deleteItem(at offsets: IndexSet) {
        items.remove(atOffsets: offsets)
    }

    // MARK: - Persistence

    private func saveItems() {
        if let encoded = try? JSONEncoder().encode(items) {
            UserDefaults.standard.set(encoded, forKey: itemsKey)
        }
    }

    private func loadItems() {
        if let data = UserDefaults.standard.data(forKey: itemsKey),
           let decoded = try?
JSONDecoder().decode([ChecklistItem].self, from: data) {
            items = decoded
        }
    }
}
```

🔄 Summary of What This Does:

- Loads items from `UserDefaults` when the app starts
- Saves any changes automatically via `didSet` on the `items` array
- Supports:
    - `addItem(title:)`
    - `toggleItem(_:)`
    - `deleteItem(at:)`

✅ Task 3: `ChecklistView.swift` (Main Screen)

swift
CopyEdit

```
import SwiftUI

struct ChecklistView: View {
    @StateObject private var viewModel = ChecklistViewModel()
    @State private var newItemTitle: String = ""
    @State private var showingAddItem = false

    var body: some View {
        NavigationView {
            List {
                ForEach(viewModel.items) { item in
                    HStack {
                        Image(systemName: item.isCompleted ?
"checkmark.circle.fill" : "circle")
                            .foregroundColor(item.isCompleted ? .green
: .gray)
                        Text(item.title)
                            .strikethrough(item.isCompleted)
                            .foregroundColor(item.isCompleted ? .gray
: .primary)
                    }
                    .contentShape(Rectangle()) // So the whole row is
tappable
                    .onTapGesture {
                        viewModel.toggleItem(item)
                    }
                }
                .onDelete(perform: viewModel.deleteItem)
            }
            .navigationTitle("My Checklist")
```

```
            .toolbar {
                ToolbarItem(placement: .navigationBarTrailing) {
                    Button(action: {
                        showingAddItem = true
                    }) {
                        Image(systemName: "plus")
                    }
                }
            }
            .sheet(isPresented: $showingAddItem) {
                AddItemView(viewModel: viewModel)
            }
        }
    }
}
```

✅ Summary of What This View Does:

- Uses a `NavigationView` with a `List` of checklist items.
- Taps toggle completion.
- Swipe to delete.
- "+" button opens `AddItemView` in a sheet.

✅ Task 4: `AddItemView.swift`

swift
CopyEdit

```
import SwiftUI

struct AddItemView: View {
```

```
    @Environment(\.presentationMode) var presentationMode
    @ObservedObject var viewModel: ChecklistViewModel
    @State private var title: String = ""

    var body: some View {
        NavigationView {
            Form {
                Section(header: Text("New Item")) {
                    TextField("Enter item title", text: $title)
                }

                Section {
                    Button("Add Item") {
                        if !title.trimmingCharacters(in:
.whitespaces).isEmpty {
                            viewModel.addItem(title: title)
                            presentationMode.wrappedValue.dismiss()
                        }
                    }
                    .disabled(title.trimmingCharacters(in:
.whitespaces).isEmpty)
                }
            }
            .navigationTitle("Add Item")
            .navigationBarItems(trailing: Button("Cancel") {
                presentationMode.wrappedValue.dismiss()
            })
        }
    }
}
```

✅ Features:

- Text field for new item input
- Adds item to `ChecklistViewModel`

- Dismisses the sheet on save or cancel
- Disables button if the field is empty