**Day 2: SwiftUI Basics and UI Building**

**Task 1: Design the main user interface using SwiftUI to display a list of transactions.**

import SwiftUI

struct Transaction: Identifiable {

var id = UUID()

var title: String

var amount: Double

}

struct TransactionListView: View {

let transactions: [Transaction]

var body: some View {

NavigationView {

List(transactions) { transaction in

VStack(alignment: .leading) {

Text(transaction.title)

.font(.headline)

Text("$\(transaction.amount)")

.foregroundColor(transaction.amount >= 0 ? .green : .red)

}

}

.navigationBarTitle("Transactions")

}

}

}

// Test Preview

struct ContentView\_Previews: PreviewProvider {

static var previews: some View {

TransactionListView(transactions: [

Transaction(title: "Salary", amount: 3000),

Transaction(title: "Rent", amount: -1000),

Transaction(title: "Groceries", amount: -200),

Transaction(title: "Bonus", amount: 500)

])

}

}

**Task 2: Implement a form to add new transactions, categorizing them as either income or expenses.**

struct AddTransactionView: View {

@State private var title = ""

@State private var amount = ""

@State private var isIncome = true

var body: some View {

NavigationView {

Form {

Section(header: Text("Transaction Details")) {

TextField("Title", text: $title)

TextField("Amount", text: $amount)

.keyboardType(.numberPad)

}

Section(header: Text("Category")) {

Toggle("Income", isOn: $isIncome)

}

Section {

Button("Save") {

// Save the transaction

}

}

}

.navigationBarTitle("Add Transaction")

}

}

}

**Day 3: SwiftUI Layout and Views**

**Task 1: Use SwiftUI's layout system to create a dashboard view that shows the user's current balance and spending trends.**

struct DashboardView: View {

let currentBalance: Double

let spendingTrends: [Double] // Assuming it's an array of spending amounts over time

var body: some View {

VStack {

Text("Current Balance: $\(currentBalance)")

.font(.title)

.padding()

// Display spending trends chart here

// You can use a library like SwiftUICharts or implement your own custom view

}

.navigationBarTitle("Dashboard")

}

}

**Task 2: Integrate SwiftUI modifiers to customize the look and feel of the dashboard according to user preferences.**

// Example of adding a background color to the dashboard

struct DashboardView: View {

let currentBalance: Double

let spendingTrends: [Double]

let backgroundColor: Color // Added for customization

var body: some View {

VStack {

Text("Current Balance: $\(currentBalance)")

.font(.title)

.padding()

// Display spending trends chart here

}

.navigationBarTitle("Dashboard")

.background(backgroundColor) // Apply background color

}

}

**Day 4: User Interaction and Navigation**

**Task 1: Develop interaction logic that allows users to edit or delete transactions from the list.**

struct TransactionListView: View {

@State private var transactions: [Transaction] // State to hold transactions

var body: some View {

NavigationView {

List {

ForEach(transactions) { transaction in

NavigationLink(destination: TransactionDetail(transaction: transaction)) {

TransactionRow(transaction: transaction)

}

}

.onDelete(perform: deleteTransaction) // Enable swipe to delete

}

.navigationBarTitle("Transactions")

.navigationBarItems(trailing: EditButton()) // Enable edit mode

}

}

// Method to delete transaction

func deleteTransaction(at offsets: IndexSet) {

transactions.remove(atOffsets: offsets)

}

}

struct TransactionRow: View {

let transaction: Transaction

var body: some View {

VStack(alignment: .leading) {

Text(transaction.title)

.font(.headline)

Text("$\(transaction.amount)")

.foregroundColor(transaction.amount >= 0 ? .green : .red)

}

}

}

struct TransactionDetail: View {

let transaction: Transaction

var body: some View {

// Transaction detail view

// You can display transaction details here with an option to edit

}

}

**Task 2: Set up navigation flows to move between the dashboard, transaction list, and transaction creation form.**

struct ContentView: View {

var body: some View {

TabView {

DashboardView()

.tabItem {

Image(systemName: "chart.pie.fill")

Text("Dashboard")

}

TransactionListView(transactions: []) // Pass transactions here

.tabItem {

Image(systemName: "list.bullet")

Text("Transactions")

}

AddTransactionView()

.tabItem {

Image(systemName: "plus.circle.fill")

Text("Add Transaction")

}

}

}

}

Usage:

struct TransactionListView: View {

@ObservedObject var transactionManager: TransactionManager // Inject the manager

var body: some View {

NavigationView {

List {

ForEach(transactionManager.transactions) { transaction in

TransactionRow(transaction: transaction)

}

.onDelete(perform: deleteTransaction)

}

.navigationBarTitle("Transactions")

.navigationBarItems(trailing: EditButton())

}

}

// Method to delete transaction

func deleteTransaction(at offsets: IndexSet) {

transactionManager.transactions.remove(atOffsets: offsets)

}

}

**Day 5: State Management and Combine Framework**

**Task 1: Implement state management using the Combine framework to update the user interface reactively as transactions are added or modified.**

import Combine

class TransactionManager: ObservableObject {

@Published var transactions: [Transaction] = []

// Method to add transaction

func addTransaction(\_ transaction: Transaction) {

transactions.append(transaction)

}

// Method to delete transaction

func deleteTransaction(at index: Int) {

transactions.remove(at: index)

}

// Load transactions from a data source

func loadTransactions() {

// Call your data service here to fetch transactions asynchronously

// For this example, let's assume we have a TransactionService

let transactionService = TransactionService()

transactionService.fetchTransactions()

.sink(receiveValue: { [weak self] transactions in

self?.transactions = transactions

})

.store(in: &cancellables)

}

private var cancellables = Set<AnyCancellable>()

init() {

loadTransactions()

}

}

Usage:

struct TransactionListView: View {

@ObservedObject var transactionManager = TransactionManager()

var body: some View {

NavigationView {

List {

ForEach(transactionManager.transactions) { transaction in

Text("\(transaction.title): \(transaction.amount)")

}

}

.navigationBarTitle("Transactions")

}

}

}

**Task 2: Create publishers and subscribers that handle the loading and saving of transaction data in real-time.**

import Combine

class TransactionManager: ObservableObject {

@Published var transactions: [Transaction] = []

private let transactionService = TransactionService()

private var cancellables = Set<AnyCancellable>()

// Method to load transactions

func loadTransactions() {

transactionService.fetchTransactions()

.sink(receiveValue: { [weak self] transactions in

self?.transactions = transactions

})

.store(in: &cancellables)

}

// Method to save transaction

func saveTransaction(\_ transaction: Transaction) {

transactionService.saveTransaction(transaction)

.sink(receiveCompletion: { \_ in }, receiveValue: { \_ in })

.store(in: &cancellables)

}

init() {

loadTransactions()

}

}

Usage:

struct AddTransactionView: View {

@ObservedObject var transactionManager = TransactionManager()

@State private var newTransactionTitle = ""

@State private var newTransactionAmount = ""

var body: some View {

NavigationView {

Form {

Section(header: Text("Transaction Details")) {

TextField("Title", text: $newTransactionTitle)

TextField("Amount", text: $newTransactionAmount)

.keyboardType(.decimalPad)

}

Section {

Button("Save") {

guard let amount = Double(newTransactionAmount) else { return }

let transaction = Transaction(title: newTransactionTitle, amount: amount)

transactionManager.saveTransaction(transaction)

newTransactionTitle = ""

newTransactionAmount = ""

}

}

}

.navigationBarTitle("Add Transaction")

}

}

}

**Day 6: Lists, Animation, and Adaptivity**

**Task 1: Build dynamic lists that display transactions with animations to visually distinguish between income and expenses.**

struct TransactionRow: View {

let transaction: Transaction

var body: some View {

VStack(alignment: .leading) {

Text(transaction.title)

.font(.headline)

Text("$\(transaction.amount)")

.foregroundColor(transaction.amount >= 0 ? .green : .red)

}

.animation(.default) // Add animation

}

}

**Task 2: Implement gesture handling to interact with list items, such as swipe to delete**

struct TransactionRow: View {

let transaction: Transaction

var onDelete: () -> Void // Closure to handle delete action

var body: some View {

VStack(alignment: .leading) {

Text(transaction.title)

.font(.headline)

Text("$\(transaction.amount)")

.foregroundColor(transaction.amount >= 0 ? .green : .red)

}

.gesture(

DragGesture()

.onEnded { \_ in

self.onDelete() // Call onDelete closure

}

)

}

}