**import** SwiftUI

**import** MessageUI

**import** UIKit

**import** EventKit

**import** Social

**import** StoreKit

**struct** ContentView: View {

@State **private** **var** ageText: String = ""

@State **var** calendar = Calendar.current

@State **var** components = DateComponents()

@State **private** **var** currentDate = Date()

@State **var** selectedDate = Date()

@State **private** **var** vaccinesDue: [(String, Date)] = []

@State **var** isDatePickerVisible = **true**

@State **var** showingAlert = **false**

@State **private** **var** selectedVaccines: [(String, Date)] = []

@Environment(\.requestReview) **var** requestReview: RequestReviewAction

@State **private** **var** ageInMonths: Int = 0 // Declare ageInMonths outside the function

@State **private** **var** age: Int = 0

@State **private** **var** vaccines: [Vaccine] = []

**let** imageName = "sickapp" // Create a separate variable for the image name

@State **private** **var** vaccineGroups: [String: [(String, Date)]] = [:]

@State **private** **var** processedDueDates: Set<Date> = []

@State **private** **var** childsName: String = "" // Declare childsName as a state property

@State **private** **var** selectedTab = 1

@State **private** **var** isAboutUsPopupPresented = **false**

@State **private** **var** openCount = 0

@State **private** **var** showReviewAlert = **false**

@State **private** **var** searchText: String = ""

@State **var** eventStore: EKEventStore?

**let** notify = NotificationHandler()

@Environment(\.openURL) **var** openURL

**private** **var** email = SupportEmail(toAddress: "mat.itunesconnect@outlook.com", subject: "Submit your suggestion", messageHeader: "Please tell us what other infection you would liek covered", body: "Write which infection you would like to be included here")

**var** contentInformation: [information] = InfoArray.ukInfections

**let** vaccineMonthsMap: [String: [Int]] = [

"1st 6-in-1": [2],

"2nd 6-in-1": [3],

"3rd 6-in-1": [4],

"1st rotavirus": [2],

"2nd rotavirus": [3],

"1st men b": [2],

"2nd men b": [4],

"3rd men b": [12],

"1st pneumococcal": [3],

"2nd pneumococcal": [12],

"hib/men c": [12],

"1st mmr vaccine": [12],

"2nd mmr vaccine": [36],

"nasal flu": [36],

"pre-school boosters (4-in-1)": [36],

"hpv vaccine": [156],

"teenage booster (3-in-1)": [168],

"men acwy": [168]

]

**init**() {

UINavigationBar.appearance().largeTitleTextAttributes = [.foregroundColor:UIColor.init(Color(.white))]

UINavigationBar.appearance().barTintColor = .black

UINavigationBar.appearance().titleTextAttributes = [.foregroundColor: UIColor.white]

}

**var** body: **some** View {

TabView(selection: $selectedTab) {

Group {

homeView

.tabItem {

Image(systemName: "house")

Text("Home")

}

.tag(1)

adviceView

.accentColor(.white)

.tabItem {

Image(systemName: "microbe")

Text("Infection")

}

.tag(2)

.tabViewStyle(DefaultTabViewStyle())

vaccinesView

.tabItem {

Image(systemName: "syringe")

Text("Vaccines")

}

.tag(3)

// InfectionNews()

// .accentColor(.white)

// .tabItem {

// Image(systemName: "newspaper")

// Text("News")

// }

// .tag(4)

}

.toolbarBackground(Color("AccentColor"), for: .tabBar)

.toolbarBackground(.visible, for: .tabBar)

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.onChange(of: selectedTab) { newTab **in**

// Handle tab change if needed

}

.sheet(isPresented: $isAboutUsPopupPresented) {

AboutUsPopup(isPresented: $isAboutUsPopupPresented)

}

}

// Sub-views

**var** homeView: **some** View {

NavigationStack {

GeometryReader { geometry **in**

VStack(alignment: .center, spacing: 20) {

Text("Welcome to 2 Sick 4 School")

.font(.largeTitle)

.fontWeight(.bold)

.foregroundColor(.white)

.multilineTextAlignment(.center)

.padding(.top, 40.0)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

Image(imageName)

.resizable()

.aspectRatio(contentMode: .fit)

.frame(width: geometry.size.width \* 0.5, height: geometry.size.height \* 0.5)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

GeometryReader { geometry **in**

VStack(alignment: .center, spacing: 10) {

Spacer() // Add spacer at the top for vertical centering

// Button to go to the advice tab

Button(action: {

selectedTab = 2 // Navigate to the advice tab

}) {

Text("Should they go to school?")

.frame(width: geometry.size.width \* 0.8, height: geometry.size.height \* 0.2)

.background(Color.green)

.foregroundColor(.white)

.cornerRadius(10)

.font(.title3)

}

// Button to go to the vaccines tab

Button(action: {

selectedTab = 3 // Navigate to the vaccines tab

}) {

Text("When are the vaccines due?")

.frame(width: geometry.size.width \* 0.8, height: geometry.size.height \* 0.2)

.background(Color.green)

.foregroundColor(.white)

.cornerRadius(10)

.font(.title3)

}

// Button to open the "About Us" window

Button(action: {

isAboutUsPopupPresented.toggle() // Toggle the state to show/hide the popup

}) {

Text("About Us")

.frame(width: geometry.size.width \* 0.8, height: geometry.size.height \* 0.2)

.background(Color.green)

.foregroundColor(.white)

.cornerRadius(10)

.font(.title3)

}

Spacer() // Add spacer at the bottom for vertical centering

}

.frame(maxHeight: .infinity) // Set the frame to max height

.frame(maxWidth: .infinity)

// .padding

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.onAppear {

// Increment the open count every time the tab appears

openCount += 1

// Check if the open count is equal to 3

**if** openCount == 3 {

// If it's 3, reset the count and show the review alert

openCount = 0

showReviewAlert = **true**

}

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.alert(isPresented: $showReviewAlert) {

Alert(

title: Text("Leave a Review"),

message: Text("Thank you for using our app! Would you like to leave a review?"),

primaryButton: .default(Text("Leave Review")) {

**if** **let** scene = UIApplication.shared.connectedScenes.first **as**? UIWindowScene {

SKStoreReviewController.requestReview(in: scene)

}

},

secondaryButton: .cancel(Text("Later"))

)

}

}

}

}

**var** adviceView: **some** View {

NavigationStack() {

VStack {

SearchBar(searchText: $searchText)

List(contentInformation.filter { info **in**

searchText.isEmpty || info.title.localizedCaseInsensitiveContains(searchText)

}, id: \.id) { info **in**

NavigationLink(destination: InfoDetailView(info: info)) {

InfoCell(info: info)

}

}

.navigationTitle("Common UK infections")

.scrollContentBackground(Visibility.hidden)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.toolbar{

ToolbarItem(placement: .navigationBarTrailing){

Button {

email.send(openURL: openURL)

} label: {

HStack {

Text("Add an infection")

}

.foregroundColor(.white)

}

}

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

**var** vaccinesView: **some** View {

NavigationStack() {

**if** isDatePickerVisible {

DatePickerSection(isDatePickerVisible: $isDatePickerVisible, selectedDate: $selectedDate, childsName: $childsName, checkVaccines: checkVaccines)

} **else** {

VaccinesDueSection(

ageText: $ageText,

isDatePickerVisible: $isDatePickerVisible,

vaccinesDue: $vaccinesDue,

showingAlert: $showingAlert,

eventStore: EKEventStore(),

checkVaccines: **self**.checkVaccines,

selectedDate: selectedDate,

currentDate: currentDate, // Corrected parameter order

handleClickHereButton: handleClickHereButton,

selectedVaccines: $selectedVaccines,

vaccineGroups: vaccineGroups,

vaccineMonthsMap: vaccineMonthsMap,

childsName: childsName

)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.onAppear

{

initializeEventStore()

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

**struct** SearchBar: View {

@Binding **var** searchText: String

**var** body: **some** View {

HStack {

TextField("Search", text: $searchText)

.padding(.horizontal, 10)

.padding(.vertical, 4)

.background(Color(.white))

.cornerRadius(8)

.padding(.horizontal)

.padding(.top, 10)

.padding(.bottom, 10)

.foregroundColor(.black)

Spacer()

Image(systemName: "magnifyingglass")

.padding(.trailing, 20)

.foregroundColor(.gray)

}

.background(Color(.white))

.cornerRadius(10)

.padding(.horizontal, 20)

.padding(.vertical, 4)

}

}

**struct** AboutUsPopup: View {

@Binding **var** isPresented: Bool

**var** body: **some** View {

VStack {

Text("Disclaimer")

.font(.title)

.fontWeight(.bold)

.foregroundColor(.white)

.padding()

Text("This app has been created to give general guidance to parents on common UK illnesses, and inform the user when their children(s) vaccinations are due. This app has been created by a doctor in the UK using their years of experience and has been created by using UK medical guidance and UK school policies. The information provided in this app should not replace professional medical advice, diagnosis, or treatment. Please seek medical assistance if you feel your child is unwell. We do not take any liability for any damages or harm arising from the use of the app's information. This app does not constitute a doctor-patient relationship, and is for general information only. While efforts have been made to provide accurate and up-to-date information, medical knowledge evolves, and the app may not always reflect the latest developments. Users should consult healthcare professionals for specific medical concerns and to use their judgment when applying the app's information to their unique situations.")

.foregroundColor(.white)

.padding()

Button("Close") {

isPresented.toggle()

}

.frame(width: 100, height: 40)

.background(Color.green)

.foregroundColor(.white)

.cornerRadius(10)

.padding()

}

.frame(maxWidth: .infinity, maxHeight: .infinity)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

}

}

**struct** DatePickerSection: View {

@Binding **var** isDatePickerVisible: Bool

@Binding **var** selectedDate: Date

@Binding **var** childsName: String

**var** checkVaccines: () -> Void

**var** body: **some** View {

VStack {

**if** isDatePickerVisible {

VStack {

List {

Section {

Text("Put in your childs name and date of birth and we will show you what vaccinations are due and when")

.font(.title2)

.multilineTextAlignment(.center)

.padding(10)

}

Section {

DatePicker("Select Date of Birth", selection: $selectedDate, in: ...Date(), displayedComponents: [.date])

.datePickerStyle(WheelDatePickerStyle())

.labelsHidden()

.frame(maxWidth: .infinity, alignment: .center)

.padding(.horizontal, 20)

.fixedSize(horizontal: **false**, vertical: **true**)

}

Section {

// Add a TextField for entering the child's name

TextField("Enter Child's Name", text: $childsName)

.padding(.horizontal, 20)

.fixedSize(horizontal: **false**, vertical: **true**)

.padding(10)

}

Section {

Button("Click here to see what vaccines are due") {

isDatePickerVisible = **false**

checkVaccines() // Call the action here

}

.padding()

.fixedSize(horizontal: **false**, vertical: **true**)

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

.onChange(of: selectedDate) { newValue **in**

print("Selected Date:", newValue)

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

**struct** AddToCalendarButton: View {

**var** selectedVaccines: [String: [(String, Date)]]

**var** isEventAlreadyAdded: (String, Date) -> Bool

**var** saveEventIfNotAdded: (String, Date, String, Int) -> Void

**var** body: **some** View {

Button(action: {

**for** (\_, vaccines) **in** selectedVaccines {

**for** (vaccine, dueDate) **in** vaccines {

**if** !isEventAlreadyAdded(vaccine, dueDate) {

saveEventIfNotAdded(vaccine, dueDate, "YourAgeTextHere", 0)

}

}

}

}) {

Text("Add to Calendar")

.font(.title2)

.foregroundColor(.white)

.padding() // Add some padding around the text

.padding(.horizontal, 5) // Add horizontal padding

.padding(.vertical, 5) // Add vertical padding

}

.background(Color.green)

.cornerRadius(10)

}

}

**private** **func** initializeEventStore() {

**if** eventStore == **nil** {

eventStore = EKEventStore()

eventStore?.requestAccess(to: .event) { success, error **in**

**if** success {

print("Event store access granted.")

} **else** {

print("Event store access denied: \(error?.localizedDescription ?? "Unknown error")")

}

}

}

}

**struct** VaccinesDueSection: View {

@Binding **var** ageText: String

@Binding **var** isDatePickerVisible: Bool

@Binding **var** vaccinesDue: [(String, Date)]

@Binding **var** showingAlert: Bool

@State **private** **var** showAlert = **false**

@State **private** **var** showEventAddedAlert = **false**

**var** eventStore: EKEventStore

**var** checkVaccines: () -> Void

**var** selectedDate: Date

**var** currentDate: Date

**var** handleClickHereButton: ([String: [(String, Date)]]) -> Void

@Binding **var** selectedVaccines: [(String, Date)] // Keep this as a binding

@State **private** **var** uniqueSelectedVaccines = Set<String>()

**var** vaccineGroups: [String: [(String, Date)]]

**var** vaccineMonthsMap: [String: [Int]]

**var** childsName: String

// Add a property to store the sorted vaccine groups

**var** sortedVaccineGroups: [(key: String, value: [(String, Date)])] {

**let** order = [

"Vaccines due at 2 months:",

"Vaccines due at 3 months:",

"Vaccines due at 4 months:",

"Vaccines due at 1 year:",

"Vaccines due at 3 years:",

"Vaccines due at 13 years:",

"Vaccines due at 14 years:"

]

**return** order.compactMap { key **in**

**guard** **let** value = vaccineGroups[key] **else** { **return** **nil** }

**return** (key, value)

}

}

**func** saveEvent(eventStore: EKEventStore, for vaccine: String, at date: Date, ageText: String, ageInMonths: Int, childsName: String) {

// Check if the event already exists on the specified date, time, and with the given child's name

**if** isEventAlreadyAdded(eventStore: eventStore, for: vaccine, at: date, childsName: childsName) {

print("Vaccine already present in the calendar")

// Show a warning message here if needed

} **else** {

**let** event = EKEvent(eventStore: eventStore)

event.title = "\(vaccine) for \(childsName)"

// Set startDate to 6 am on the calculated due date

**let** calendar = Calendar.current

**let** startHour = 6

**let** startMinute = 0

**let** startDate = calendar.date(bySettingHour: startHour, minute: startMinute, second: 0, of: date)!

event.startDate = startDate

// Set endDate to 10 minutes after the start time

**let** endDate = calendar.date(byAdding: .minute, value: 10, to: startDate)!

event.endDate = endDate

event.calendar = eventStore.defaultCalendarForNewEvents

**do** {

**try** eventStore.save(event, span: .thisEvent)

print("Event for \(vaccine) added to the calendar on \(date) with start time 6 am and duration 10 minutes and name \(childsName).")

showEventAddedAlert = **true**

showAlert = **false** // Ensure this is set to false

} **catch** **let** error **as** NSError {

print("Error adding event to calendar: \(error)")

}

}

}

// Function to check if the event already exists on the specified date, time, and with the given child's name

**func** isEventAlreadyAdded(eventStore: EKEventStore, for vaccine: String, at date: Date, childsName: String) -> Bool {

**let** startDate = Calendar.current.date(bySettingHour: 6, minute: 0, second: 0, of: date)!

**let** endDate = Calendar.current.date(bySettingHour: 6, minute: 10, second: 0, of: date)!

**let** predicate = eventStore.predicateForEvents(withStart: startDate, end: endDate, calendars: **nil**)

**let** events = eventStore.events(matching: predicate)

print("isEventAleadyAdded")

showEventAddedAlert = **false** // Ensure this is set to false

showAlert = **true**

**return** events.contains { event **in**

**let** isSameTitle = event.title == "\(vaccine) for \(childsName)"

**let** isSameStartDate = event.startDate == startDate

**let** isSameEndDate = event.endDate == endDate

**return** isSameTitle && isSameStartDate && isSameEndDate

}

}

**func** saveEventIfNotAdded(eventStore: EKEventStore, for vaccine: String, at date: Date, ageText: String, ageInMonths: Int) {

**if** !isEventAlreadyAdded(eventStore: eventStore, for: vaccine, at: date, childsName: childsName) {

// showAlert = true

saveEvent(eventStore: eventStore, for: vaccine, at: date, ageText: ageText, ageInMonths: ageInMonths, childsName: childsName)

} **else** {

print("Event for \(vaccine) already exists on \(date).")

// Show an error message here if needed

}

}

**func** calculateAgeText(from startDate: Date, to endDate: Date) -> String? {

**let** calendar = Calendar.current

**let** components = calendar.dateComponents([.year, .month], from: startDate, to: endDate)

**if** **let** years = components.year, **let** months = components.month {

**if** years > 0 {

**return** "Your child is \(years) year\(years > 1 ? "s" : "") and \(months) month\(months > 1 ? "s" : "") old"

} **else** {

**return** "Your child is \(months) month\(months > 1 ? "s" : "") old"

}

}

**return** **nil**

}

**var** body: **some** View {

VStack() {

HeaderView(ageText: $ageText, isDatePickerVisible: $isDatePickerVisible)

Button("Change DOB") { isDatePickerVisible = **true** }

.buttonStyle(MyButtonStyle())

VaccinesDueButton(checkVaccines: checkVaccines, selectedVaccines: $selectedVaccines)

.padding()

// Display child's age based on the selected date

**if** **let** ageText = calculateAgeText(from: selectedDate, to: currentDate) {

Text(ageText)

.font(.title)

.foregroundColor(.white)

// .padding(.top, 10)

.padding(.bottom, 10)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

Spacer()

.frame(height: 0) // Adjust height as needed

VStack() {

ScrollView {

VStack(alignment: .leading, spacing: 0) {

// Use the sortedVaccineGroups property here

ForEach(sortedVaccineGroups, id: \.key)

{ identifier, vaccines **in**

**let** ageGroup = identifier

Section(header: Text(ageGroup).foregroundColor(.white).font(.headline).padding(.vertical, 10)) {

ForEach(vaccines, id: \.0) { vaccineTuple **in**

**let** (vaccineName, \_) = vaccineTuple

VStack(alignment: .leading, spacing: 10) {

Text(vaccineName).foregroundColor(.white)

}

.padding(.horizontal, 20)

.padding(.vertical, 10)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

}

}

}

// .padding()

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

}

.onAppear {

// This might not be necessary

checkVaccines()

print("ScrollView content loaded. vaccineGroups: \(vaccineGroups)")

print("ScrollView content loaded. sortedVaccineGroups: \(sortedVaccineGroups)")

}

.onChange(of: selectedDate) { newDate **in**

checkVaccines()

}

.onChange(of: currentDate) { newDate **in**

checkVaccines()

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity, maxHeight: .infinity)

// Add the AddToCalendarButton below the ScrollView

// .padding(.top, 20)

// .padding(.bottom, 20)

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxHeight: .infinity)

Spacer()

.frame(height: 0) // Adjust height as needed

VStack() {

**if** !vaccineGroups.isEmpty {

AddToCalendarButton(selectedVaccines: vaccineGroups) { vaccine, date **in**

isEventAlreadyAdded(eventStore: **self**.eventStore, for: vaccine, at: date, childsName: childsName)

} saveEventIfNotAdded: { vaccine, date, ageText, ageInMonths **in**

saveEventIfNotAdded(eventStore: **self**.eventStore, for: vaccine, at: date, ageText: ageText, ageInMonths: ageInMonths)

}

.padding()

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

ZStack {}

.alert(isPresented: $showAlert) {

Alert(title: Text("Warning"), message: Text("Vaccine is already present in the calendar."), dismissButton: .default(Text("OK")))

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

ZStack {}

.alert(isPresented: $showEventAddedAlert) {

Alert(title: Text("Success"), message: Text("Vaccine added to the calendar."), dismissButton: .default(Text("OK")))

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

.listStyle(.plain)

}

}

**struct** HeaderView: View {

@Binding **var** ageText: String

@Binding **var** isDatePickerVisible: Bool

**var** body: **some** View {

Text(ageText)

.font(.title)

.multilineTextAlignment(.center)

.foregroundColor(.white)

.padding(.vertical)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

**struct** MyButtonStyle: ButtonStyle {

**func** makeBody(configuration: Configuration) -> **some** View {

configuration.label

.padding(.all)

.foregroundColor(.white)

.background(Color.red)

.cornerRadius(10)

}

}

**struct** Vaccine {

**let** name: String

**let** dueDate: Date

}

**struct** VaccinesDueButton: View {

**var** checkVaccines: () -> Void

@Binding **var** selectedVaccines: [(String, Date)]

**var** body: **some** View {

Text("Vaccines Due")

.font(.title)

.foregroundColor(.white)

.background(Color(red: 0.216, green: 0.498, blue: 0.722))

.frame(maxWidth: .infinity)

}

}

**func** calculateMonthsRemaining(from startDate: Date, to endDate: Date) -> Int {

**let** calendar = Calendar.current

**let** components = calendar.dateComponents([.month], from: startDate, to: endDate)

**return** components.month ?? 0

}

**func** checkVaccines() {

**let** today = Date()

**var** calculatedDueDates: [Date] = []

// Calculate age in months

**let** ageComponents = calendar.dateComponents([.month], from: selectedDate, to: today)

ageInMonths = ageComponents.month ?? 0

print("age of child in months: \(ageInMonths)")

**for** (vaccine, months) **in** vaccineMonthsMap {

**if** **let** lastMonth = months.last, lastMonth >= ageInMonths {

**if** **let** dueDate = calculateDueDateForVaccine(vaccine, at: ageInMonths) {

calculatedDueDates.append(dueDate)

}

}

}

calculatedDueDates.sort()

// Move this line outside the loop

**var** uniqueSelectedVaccines = Set<String>()

// Move this line outside the loop

**var** vaccineGroups = [String: [(String, Date)]]()

**for** dueDate **in** calculatedDueDates {

print("Processing vaccines for due date: \(dueDate)")

**if** **let** vaccines = calculateVaccine(for: dueDate, ageInMonths: ageInMonths) {

print("Processing vaccines inside loop for due date: \(dueDate)")

**for** (vaccine, ageText) **in** vaccines {

**let** vaccineKey = "\(vaccine)-\(dueDate.timeIntervalSince1970)"

**if** !uniqueSelectedVaccines.contains(vaccineKey) {

uniqueSelectedVaccines.insert(vaccineKey)

addVaccine(for: vaccine, inAgeGroup: ageText, to: &vaccineGroups, currentDate: dueDate, ageInMonths: ageInMonths)

}

}

}

}

// Clear the set after processing all due dates

uniqueSelectedVaccines.removeAll()

print("vaccineGroups list:")

print(vaccineGroups)

// Sort the vaccineGroups based on the age group identifier

**let** sortedVaccineGroups = vaccineGroups.sorted { $0.key < $1.key }

// Clear vaccineGroups and add the sorted groups

vaccineGroups.removeAll()

**for** (identifier, vaccines) **in** sortedVaccineGroups {

vaccineGroups[identifier] = vaccines

}

print("Ordered vaccineGroups:")

print(vaccineGroups)

// Ensure UI updates on the main thread

DispatchQueue.main.async {

**self**.vaccineGroups = vaccineGroups

// self.sortedVaccineGroups = sortedVaccineGroups

handleClickHereButton(vaccineGroups: vaccineGroups)

}

}

**func** calculateDueDateForVaccine(\_ vaccine: String, at age: Int) -> Date? {

**let** lowercaseVaccine = vaccine.lowercased()

**guard** **let** selectedMonthsToAddOptions = vaccineMonthsMap[lowercaseVaccine] **else** {

print("Error: No months specified for \(vaccine)")

**return** **nil**

}

// If there are no specified months, return nil

**guard** **let** firstMonth = selectedMonthsToAddOptions.first **else** {

print("Error: No months specified for \(vaccine)")

**return** **nil**

}

// Calculate the difference between due date and age

**let** differenceInMonths = firstMonth - age

**let** adjustedMonths = abs(differenceInMonths) // Use absolute value to handle negative differences

// Calculate the due date using the adjusted months for the vaccine

**let** dueDate = calendar.date(byAdding: .month, value: adjustedMonths, to: currentDate)

print("dueDate \(dueDate)")

// Ensure dueDate is not nil

**guard** **let** unwrappedDueDate = dueDate **else** {

print("Error: Unable to calculate due date for \(vaccine)")

**return** **nil**

}

**return** unwrappedDueDate

}

**func** calculateVaccine(for dueDate: Date, ageInMonths: Int) -> [(vaccine: String, ageGroup: String)]? {

**let** monthsRemaining = calculateMonthsRemaining(from: currentDate, to: dueDate)

print("Months remaining: \(monthsRemaining)")

**var** result: [(vaccine: String, ageGroup: String)] = []

**for** (vaccineName, specifiedMonths) **in** vaccineMonthsMap {

**guard** **let** ageGroup = ContentView.getAgeText(for: specifiedMonths.first ?? 0), specifiedMonths.last ?? 0 > ageInMonths **else** {

**continue**

}

// Check if the vaccine is due within the specifiedMonths range

**if** specifiedMonths.contains(where: { $0 >= monthsRemaining }) {

result.append((vaccineName, ageGroup))

print("specifiedMonths: \(result)")

}

}

**if** result.isEmpty {

print("No vaccines found for due date: \(dueDate)")

**return** **nil** // Return nil when no vaccines are found

}

**return** result

}

**static** **func** getAgeText(for monthsRemaining: Int) -> String? {

**switch** monthsRemaining {

**case** 2: **return** "Vaccines due at 2 months:"

**case** 3: **return** "Vaccines due at 3 months:"

**case** 4: **return** "Vaccines due at 4 months:"

**case** 12: **return** "Vaccines due at 1 year:"

**case** 36: **return** "Vaccines due at 3 years:"

**case** 156: **return** "Vaccines due at 13 years:"

**case** 168: **return** "Vaccines due at 14 years:"

**default**: **return** **nil**

}

}

**func** addDueDate(for name: String, at month: Int, to array: **inout** [Date]) {

**if** **let** dueDate = calculateDueDateForVaccine(name, at: age) {

array.append(dueDate)

// print("addDueDate Function: Added due date for \(name) at \(age) months: \(dueDate)")

} **else** {

print("No due date added for \(name) at \(age) months")

}

}

**func** addVaccine(for name: String, inAgeGroup ageGroup: String, to dict: **inout** [String: [(String, Date)]], currentDate: Date, ageInMonths: Int) {

// Calculate the due date for the vaccine

**guard** **let** dueDate = calculateDueDateForVaccine(name, at: ageInMonths) **else** {

print("Error: Unable to calculate due date for \(name)")

**return**

}

**let** vaccineTuple = (name, dueDate)

// Check if the identifier already exists in the dictionary

**if** **var** existingVaccines = dict[ageGroup] {

// Check if the vaccineTuple already exists in the identifier

**if** !existingVaccines.contains(where: { $0.0 == name && $0.1 == dueDate }) {

print("Adding vaccine \(name) to age group \(ageGroup)")

existingVaccines.append(vaccineTuple)

dict[ageGroup] = existingVaccines

}

} **else** {

// If the identifier doesn't exist, create a new entry with the vaccineTuple

print("Adding vaccine \(name) to new age group \(ageGroup)")

dict[ageGroup] = [vaccineTuple]

print("vaccineTuple \(vaccineTuple)")

}

// Clear vaccinesDue after processing the current vaccine

vaccinesDue.removeAll()

}

**func** calculateAgeDueText(from currentDate: Date, to dueDate: Date) -> String? {

**let** calendar = Calendar.current

**let** components = calendar.dateComponents([.month], from: currentDate, to: dueDate)

**guard** **let** monthsRemaining = components.month **else** {

**return** **nil**

}

**if** monthsRemaining < 12 {

**return** "Due in \(monthsRemaining) months"

} **else** {

**let** years = monthsRemaining / 12

**let** months = monthsRemaining % 12

**return** "Due at \(years) year\(years == 1 ? "" : "s") and \(months) month\(months == 1 ? "" : "s")"

}

}

**func** calculateDueDate(for vaccine: String, at ageInMonths: Int) -> Date? {

**switch** ageInMonths {

**case** 0...2:

**return** calculateDueDateForVaccine(vaccine, at: 2)

**case** 3...4:

**return** calculateDueDateForVaccine(vaccine, at: 4)

**case** 5...12:

**return** calculateDueDateForVaccine(vaccine, at: 12)

**case** 13...36:

**return** calculateDueDateForVaccine(vaccine, at: 36)

**case** 36...144:

**return** calculateDueDateForVaccine(vaccine, at: 144)

**case** 145...156:

**return** calculateDueDateForVaccine(vaccine, at: 156)

**default**:

**return** **nil**

}

}

**func** handleClickHereButton(vaccineGroups: [String: [(String, Date)]]) {

**guard** **let** eventStore = eventStore **else** {

print("Error: Event store is not initialized.")

**return**

}

**for** (\_, vaccines) **in** vaccineGroups {

**for** (vaccine, dueDate) **in** vaccines {

// Calculate age due text based on currentDate and dueDate

**if** **let** ageDueText = calculateAgeDueText(from: currentDate, to: dueDate) {

print("eventStore \(eventStore)")

}

}

}

showingAlert = **true**

}

**static** **func** formattedDateString(from date: Date) -> String {

**let** dateFormatter = DateFormatter()

dateFormatter.dateStyle = .short

dateFormatter.timeStyle = .none

**return** dateFormatter.string(from: date)

}

**struct** InfoCell: View {

**var** info: information

**var** body: **some** View {

Spacer()

HStack (spacing: 5) {

Image(info.imageName)

.resizable()

// .foregroundColor(/\*@START\_MENU\_TOKEN@\*/Color(red: 0.216, green: 0.498, blue: 0.722)/\*@END\_MENU\_TOKEN@\*/)

.scaledToFit()

.frame(width: 100)

.frame(height: 70)

.cornerRadius(4)

.padding(.vertical, 8)

VStack(alignment: .leading, spacing: 5) {

Text(info.title)

.fontWeight(.semibold)

.lineLimit(2)

.padding(5)

.minimumScaleFactor(0.5)

Text("Click here for more information")

.font(.subheadline)

.padding(5)

// .foregroundColor(.secondary)

}

}

}

}

**struct** ContentView\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

ContentView()

}

}

}