package com.example.mychildsunwell  
  
import android.app.Activity  
import android.app.DatePickerDialog  
import android.content.Context  
import android.widget.DatePicker  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.foundation.text.KeyboardOptions  
import androidx.compose.material.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.text.input.KeyboardType  
import androidx.compose.ui.unit.dp  
import com.example.mychildsunwell.ui.InfoCell  
import com.example.mychildsunwell.ui.InfoDetailView  
import java.util.\*  
// ... (VaccinesDueSection implementation - to be provided in the next part)  
// ... (imports and other composables from parts 1 and 2)  
import androidx.compose.material.AlertDialog  
import androidx.compose.ui.text.font.FontWeight  
import com.example.mychildsunwell.InfoArray  
  
// ... (imports from part 1)  
import android.Manifest  
import android.content.pm.PackageManager  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Search*import androidx.compose.ui.text.input.TextFieldValue  
  
// Data class for Vaccine  
data class Vaccine(  
 val name: String,  
 val dueDate: Date  
)  
  
  
@Composable  
fun ContentView() {  
 var ageText by *remember* **{** *mutableStateOf*("") **}** var components by *remember* **{** *mutableStateOf*(DateComponents()) **}** var currentDate by *remember* **{** *mutableStateOf*(Date()) **}** var selectedDate by *remember* **{** *mutableStateOf*(Date()) **}** var vaccinesDue by *remember* **{** *mutableStateOf*<List<Pair<String, Date>>>(*emptyList*()) **}** var isDatePickerVisible by *remember* **{** *mutableStateOf*(true) **}** var showingAlert by *remember* **{** *mutableStateOf*(false) **}** var selectedVaccines by *remember* **{** *mutableStateOf*<List<Pair<String, Date>>>(*emptyList*()) **}** var ageInMonths by *remember* **{** *mutableStateOf*(0) **}** var age by *remember* **{** *mutableStateOf*(0) **}** var vaccines by *remember* **{** *mutableStateOf*<List<Vaccine>>(*emptyList*()) **}** // Assuming you have a Vaccine data class  
 val imageName = "sickapp" // Replace with actual drawable resource ID  
 var vaccineGroups by *remember* **{** *mutableStateOf*<Map<String, List<Pair<String, Date>>>>(*emptyMap*()) **}** var processedDueDates by *remember* **{** *mutableStateOf*<Set<Date>>(*emptySet*()) **}** var childsName by *remember* **{** *mutableStateOf*("") **}** var selectedTab by *remember* **{** *mutableStateOf*(1) **}** var isAboutUsPopupPresented by *remember* **{** *mutableStateOf*(false) **}** var openCount by *remember* **{** *mutableStateOf*(0) **}** var showReviewAlert by *remember* **{** *mutableStateOf*(false) **}** var searchText by *remember* **{** *mutableStateOf*("") **}** // ... (rest of ContentView implementation)  
  
 // homeView Sub-view  
 class MyApp : View {  
  
 private val notify = NotificationHandler() // Assuming NotificationHandler is a class  
  
 @Composable  
 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"  
 )  
  
 private var contentInformation: List<information> = InfoArray.ukInfections // Assuming information is a data class  
  
 private val vaccineMonthsMap: Map<String, List<Int>> = *mapOf*(  
 "1st 6-in-1" *to listOf*(2),  
 "2nd 6-in-1" *to listOf*(3),  
 "3rd 6-in-1" *to listOf*(4),  
 "1st rotavirus" *to listOf*(2),  
 "2nd rotavirus" *to listOf*(3),  
 "1st men b" *to listOf*(2),  
 "2nd men b" *to listOf*(4),  
 "3rd men b" *to listOf*(12),  
 "1st pneumococcal" *to listOf*(3),  
 "2nd pneumococcal" *to listOf*(12),  
 "hib/men c" *to listOf*(12),  
 "1st mmr vaccine" *to listOf*(12),  
 "2nd mmr vaccine" *to listOf*(36),  
 "nasal flu" *to listOf*(36),  
 "pre-school boosters (4-in-1)" *to listOf*(36),  
 "hpv vaccine" *to listOf*(156),  
 "teenage booster (3-in-1)" *to listOf*(168),  
 "men acwy" *to listOf*(168)  
 )  
  
 init {  
 // Update styles  
 WindowCompat.setDecorFitsSystemWindows(window, false) // Assuming WindowCompat is available  
 val navBar = androidx.appcompat.app.AppCompatActivity(Theme.AppCompat.Light).*supportActionBar* navBar?.setBackgroundDrawable(ColorDrawable(Color.BLACK)) // Assuming ColorDrawable is available  
 navBar?.*title* = "Your App Title" // Assuming you have a title for the app  
 navBar?.setDisplayHomeAsUpEnabled(true) // Assuming you want a back button  
 }  
  
 @Composable  
 override fun Content(context: Context): Unit {  
 val selectedTab = *remember* **{** *mutableStateOf*(1) **}** // Assuming mutableStateOf is from androidx.compose.runtime  
  
 TabView(selectedItem = selectedTab.value) **{** Tab **{** homeView  
 .tabItem **{** Icon(Icons.Outlined.Home, contentDescription = "Home")  
 *Text*("Home")  
 **}** .tag(1)  
 **}** Tab **{** adviceView  
 .accentColor(Color.White)  
 .tabItem **{** Icon(Icons.Outlined.Microscope, contentDescription = "Infection")  
 *Text*("Infection")  
 **}** .tag(2)  
 **}** Tab **{** vaccinesView  
 .tabItem **{** Icon(Icons.Outlined.Syringe, contentDescription = "Vaccines")  
 *Text*("Vaccines")  
 **}** .tag(3)  
 **}** // Commenting out news section as per your code  
 /\*  
 Tab {  
 InfectionNews()  
 .accentColor(Color.White)  
 .tabItem {  
 Icon(Icons.Outlined.Newspaper, contentDescription = "News")  
 Text("News")  
 }  
 .tag(4)  
 }  
 \*/  
 **}** .background(*Color*(0.216f, 0.498f, 0.722f)) // Assuming Color is from androidx.compose.ui.graphics  
 .fillMaxWidth()  
 .onTabSelected **{** newTab **->** // Handle tab change if needed  
 **}** .overlay **{** if (isAboutUsPopupPresented.value) {  
 AboutUsPopup(isPresented = isAboutUsPopupPresented)  
 }  
 **}** }  
  
 // Assuming these are data classes or composables  
 private val homeView: @Composable () -> Unit = **{** /\* ... \*/ **}** private val adviceView: @Composable () -> Unit = **{** /\* ... \*/ **}** private val vaccinesView: @Composable () -> Unit = **{** /\* ... \*/ **}** private val isAboutUsPopupPresented = remember **{** mutable  
  
  
 val homeView = @Composable **{** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(*Color*(0xFF377DB5)),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.Center  
 ) **{** *Text*(  
 text = "Welcome to 2 Sick 4 School",  
 style = MaterialTheme.typography.h4,  
 fontWeight = FontWeight.Bold,  
 color = Color.White  
 )  
  
 *Image*(  
 painter = painterResource(id = imageName), // Assuming you have the image resource  
 contentDescription = null,  
 modifier = Modifier  
 .*fillMaxWidth*(0.5f)  
 .*height*(200.*dp*) // Adjust as needed  
 )  
  
 Row(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*),  
 horizontalArrangement = Arrangement.SpaceAround  
 ) **{** Button(  
 onClick = **{** selectedTab = 2 **}**,  
 modifier = Modifier  
 .*weight*(1f)  
 .*padding*(8.*dp*),  
 colors = ButtonDefaults.buttonColors(backgroundColor = Color.Green)  
 ) **{** Text("Should they go to school?", color = Color.White)  
 **}** *Button*(  
 onClick = **{** selectedTab = 3 **}**,  
 modifier = Modifier  
 .*weight*(1f)  
 .*padding*(8.*dp*),  
 colors = ButtonDefaults.buttonColors(backgroundColor = Color.Green)  
 ) **{** *Text*("When are the vaccines due?", color = Color.White)  
 **}  
 }** // Assuming these are already defined in your class  
 private var isAboutUsPopupPresented = *remember* **{** *mutableStateOf*(false) **}** private var openCount = 0  
 private var showReviewAlert = false  
  
// ... inside your Content composable  
  
 Column(modifier = Modifier.*fillMaxSize*()) **{** // Column for vertical layout  
  
 // Your existing tab views here  
  
 *Spacer*(modifier = Modifier.*weight*(1f)) // Spacer for vertical centering  
  
 *Row*(modifier = Modifier.*fillMaxWidth*().*padding*(horizontal = 16.*dp*)) **{** *Button*(  
 onClick = **{** isAboutUsPopupPresented.value = true **}**, // Toggle state on click  
 modifier = Modifier  
 .width(geometry.size.width \* 0.8f)  
 .height(50.*dp*) // Assuming you want a fixed height  
 .background(Color.Green)  
 .foregroundColor(Color.White)  
 .cornerRadius(10.*dp*)  
 .font(fontSize = MaterialTheme.typography.h6.fontSize) // Assuming h6 is the desired font size  
 ) **{** *Text*("About Us")  
 **}  
 }** if (showReviewAlert) {  
 Alert(  
 title = *Text*("Leave a Review"),  
 message = *Text*("Thank you for using our app! Would you like to leave a review?"),  
 onDismissRequest = **{** showReviewAlert = false **}**, // Dismiss on button click or outside tap  
 confirmButton = **{** Button(onClick = **{** val scene = window.scene as? WindowScene ?: return@Button  
 SKStoreReviewController.requestReview(in: scene)  
 **}**) **{** *Text*("Leave Review")  
 **}  
 }**,  
 dismissButton = **{** *Button*(onClick = **{** showReviewAlert = false **}**) **{** *Text*("Later")  
 **}  
 }** )  
 }  
 **}**// Assuming AboutUsPopup is a composable  
 if (isAboutUsPopupPresented.value) {  
 AboutUsPopup(isPresented = isAboutUsPopupPresented)  
 }  
  
 **}  
 }** // adviceView Sub-view  
 val adviceView = @Composable **{** *Column*(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(*Color*(0xFF377DB5))  
 ) **{** SearchBar(searchText = searchText)  
  
 *LazyColumn* **{** items(contentInformation.*filter* **{** info **->** searchText.*isEmpty*() || info.title.contains(searchText, ignoreCase = true)  
 **}**) **{** info **->** NavigationLink(destination = **{** *InfoDetailView*(info = info) **}**) **{** InfoCell(info = info)  
 **}  
 }  
 }  
 }  
 }** @Composable  
 fun vaccinesView() {  
 *Column*(modifier = Modifier.*fillMaxSize*()) **{** // Column for vertical layout  
 NavigationStack **{** if (isDatePickerVisible) {  
 DatePickerSection(  
 isDatePickerVisible = isDatePickerVisible,  
 selectedDate = selectedDate,  
 childsName = childsName,  
 checkVaccines = checkVaccines  
 )  
 } else {  
 VaccinesDueSection(  
 ageText = ageText,  
 isDatePickerVisible = isDatePickerVisible,  
 vaccinesDue = vaccinesDue,  
 showingAlert = showingAlert,  
 eventStore = EKEventStore(), // Assuming EKEventStore is available  
 checkVaccines = this.checkVaccines, // Reference 'this' for member functions  
 selectedDate = selectedDate,  
 currentDate = currentDate, // Corrected order  
 handleClickHereButton = handleClickHereButton,  
 selectedVaccines = selectedVaccines,  
 vaccineGroups = vaccineGroups,  
 vaccineMonthsMap = vaccineMonthsMap,  
 childsName = childsName  
 )  
 .background(*Color*(0.216f, 0.498f, 0.722f)) // Assuming Color is from androidx.compose.ui.graphics  
 .fillMaxWidth()  
 }  
 **}** .background(*Color*(0.216f, 0.498f, 0.722f))  
 .fillMaxWidth()  
 .onAppear **{** initializeEventStore()  
 **}  
 }** }  
  
  
 // SearchBar Composable  
 @Composable  
 fun SearchBar(searchText: String) {  
 var text by *remember* **{** *mutableStateOf*(searchText) **}** *OutlinedTextField*(  
 value = text,  
 onValueChange = **{** newText **->** text = newText  
 **}**,  
 label = **{** *Text*("Search") **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*),  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),  
 singleLine = true,  
 trailingIcon = **{** *Icon*(  
 imageVector = Icons.Default.*Search*,  
 contentDescription = null  
 )  
 **}** )  
 }  
  
 // AboutUsPopup Composable  
 @Composable  
 fun AboutUsPopup(isPresented: Boolean) {  
 if (isPresented) {  
 *AlertDialog*(  
 onDismissRequest = **{** isPresented = false **}**,  
 title = **{** *Text*("Disclaimer") **}**,  
 text = **{** *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.") **}**,  
 confirmButton = **{** *Button*(onClick = **{** isPresented = false **}**) **{** *Text*("Close")  
 **}  
 }** )  
 }  
 }  
  
 @Composable  
 fun DatePickerSection(  
 isDatePickerVisible: Boolean, // Assuming isDatePickerVisible is no longer a Binding  
 selectedDate: State<Date>, // Assuming selectedDate is a state holder  
 childsName: String,  
 checkVaccines: () -> Unit, // Lambda for action  
 ) {  
 if (isDatePickerVisible) {  
 *Column*(modifier = Modifier.*fillMaxSize*()) **{** // Column for vertical layout  
 *List* **{** Section **{** *Text*(  
 text = "Put in your child's name and date of birth and we will show you what vaccinations are due and when",  
 fontSize = MaterialTheme.typography.h2.fontSize, // Assuming h2 is desired font size  
 textAlign = TextAlign.Center,  
 modifier = Modifier.*padding*(10.*dp*)  
 )  
 **}** Section **{** DatePicker(  
 "Select Date of Birth",  
 selection = selectedDate.value, // Access value from State  
 onDateSelected = **{** newDate **->** selectedDate.value = newDate // Update state  
 *println*("Selected Date:", newDate) // Print for debugging  
 **}**,  
 displayMode = DatePickerMode.Date, // Assuming Date corresponds to DatePickerMode.Date  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*align*(Alignment.CenterHorizontally)  
 .*padding*(horizontal = 20.*dp*)  
 .fixedSize(minWidth = 0.*dp*, minHeight = 0.*dp*) // Fixed size modifiers might not be needed  
 )  
 **}** Section **{** *TextField*(  
 value = childsName,  
 onValueChange = **{** newName **->** childsName = newName **}**, // Update name  
 label = **{** *Text*("Enter Child's Name") **}**,  
 modifier = Modifier  
 .*padding*(horizontal = 20.*dp*)  
 .fixedSize(minWidth = 0.*dp*, minHeight = 0.*dp*) // Fixed size modifiers might not be needed  
 .padding(10.*dp*)  
 )  
 **}** Section **{** *Button*(onClick = **{** isDatePickerVisible = false  
 checkVaccines()  
 **}**) **{** *Text*("Click here to see what vaccines are due")  
 **}** .padding()  
 .fixedSize(minWidth = 0.*dp*, minHeight = 0.*dp*) // Fixed size modifiers might not be needed  
 **}  
 }** .background(*Color*(0.216f, 0.498f, 0.722f)) // Assuming Color is from androidx.compose.ui.graphics  
 .fillMaxWidth()  
 **}** }  
 }  
  
  
  
// AddToCalendarButton Composable  
@Composable  
fun AddToCalendarButton(  
 selectedVaccines: Map<String, List<Pair<String, Date>>>,  
 isEventAlreadyAdded: (String, Date) -> Boolean,  
 saveEventIfNotAdded: (String, Date, String, Int) -> Unit  
) {  
 *Button*(  
 onClick = **{** for ((\_, vaccines) in selectedVaccines) {  
 for ((vaccine, dueDate) in vaccines) {  
 if (!isEventAlreadyAdded(vaccine, dueDate)) {  
 saveEventIfNotAdded(vaccine, dueDate, "YourAgeTextHere", 0) // You'll need to replace "YourAgeTextHere" and 0 with the correct values  
 }  
 }  
 }  
 **}**,  
 modifier = Modifier.*padding*(16.*dp*),  
 colors = ButtonDefaults.buttonColors(backgroundColor = Color.Green)  
 ) **{** *Text*("Add to Calendar", color = Color.White)  
 **}**}  
  
 private fun initializeEventStore() {  
 if (eventStore == null) {  
 eventStore = EKEventStore()  
 eventStore?.requestAccess(EKEntityType.Event) **{** success, error **->** if (success) {  
 *println*("Event store access granted.")  
 } else {  
 val errorMessage = error?.localizedDescription ?: "Unknown error"  
 *println*("Event store access denied: $errorMessage")  
 }  
 **}** }  
 }  
 @Composable  
 fun VaccinesDueSection(  
 ageText: String,  
 isDatePickerVisible: Boolean,  
 vaccinesDue: List<Pair<String, Date>>, // Assuming Pair is used for vaccine and date  
 showingAlert: Boolean,  
 eventStore: EKEventStore,  
 checkVaccines: () -> Unit,  
 selectedDate: Date,  
 currentDate: Date,  
 handleClickHereButton: (Map<String, List<Pair<String, Date>>>) -> Unit, // Updated type for handleClickHereButton  
 selectedVaccines: List<Pair<String, Date>>, // Binding for selected vaccines  
 uniqueSelectedVaccines: Set<String> = *mutableSetOf*(), // Initialize with mutable set  
 vaccineGroups: Map<String, List<Pair<String, Date>>>, // Assuming Pair is used  
 vaccineMonthsMap: Map<String, List<Int>>,  
 childsName: String,  
 ) {  
 val sortedVaccineGroups by *derivedStateOf* **{** val order = *listOf*(  
 "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:"  
 )  
 order.*mapNotNull* **{** key **->** vaccineGroups[key]?.*let* **{** value **->** key *to* value **}  
 }  
 }** var showAlert by *remember* **{** *mutableStateOf*(false) **}** // Remember state for showAlert  
 var showEventAddedAlert by *remember* **{** *mutableStateOf*(false) **}** // Remember state for showEventAddedAlert  
  
 *LaunchedEffect*(eventStore) **{** // LaunchedEffect for initialization  
 initializeEventStore()  
 **}** *Column* **{** // ... your existing UI elements ...  
  
 if (showAlert) {  
 // Show alert for adding vaccines to calendar  
 }  
  
 if (showEventAddedAlert) {  
 // Show alert for successful event addition  
 showEventAddedAlert = false // Reset state after displaying alert  
 }  
 **}** // Function to save event to calendar  
 fun saveEvent(vaccine: String, date: Date) {  
 if (isEventAlreadyAdded(eventStore, vaccine, date, childsName)) {  
 *print*("Vaccine already present in the calendar")  
 // Show warning message here if needed  
 } else {  
 val event = EKEvent(eventStore = eventStore)  
 event.title = "$vaccine for $childsName"  
  
 val calendar = Calendar.getInstance()  
 val startHour = 6  
 val startMinute = 0  
 val startDate = calendar.set(  
 Calendar.*HOUR\_OF\_DAY*, startHour  
 ).set(  
 Calendar.*MINUTE*, startMinute  
 ).set(  
 Calendar.*SECOND*, 0  
 ).time  
 event.startDate = startDate  
  
 val endDate = calendar.add(Calendar.*MINUTE*, 10).time  
 event.endDate = endDate  
  
 event.calendar = eventStore.defaultCalendarForNewEvents  
  
 try {  
 eventStore.save(event, span = EKEventSpan.ThisEvent)  
 *print*("Event for $vaccine added to the calendar...")  
 showEventAddedAlert = true  
 showAlert = false // Ensure this is set to false  
 } catch (error: Exception) {  
 *println*("Error adding event to calendar: $error")  
 }  
 }  
 }  
 }  
 fun isEventAlreadyAdded(  
 eventStore: EKEventStore,  
 for vaccine: String,  
 at date: Date,  
 childsName: String  
 ): Boolean {  
 val calendar = Calendar.getInstance()  
 val startDate = calendar.set(  
 Calendar.*HOUR\_OF\_DAY*, 6  
 ).set(  
 Calendar.*MINUTE*, 0  
 ).set(  
 Calendar.*SECOND*, 0  
 ).time.apply **{** time = date.*time* **}** val endDate = calendar.add(Calendar.*MINUTE*, 10).time.apply **{** time = date.*time* **}** val predicate = eventStore.predicateForEvents(withStart: startDate, end: endDate, calendars: null)  
 val events = eventStore.events(matching: predicate)  
 showEventAddedAlert = false // Ensure this is set to false  
  
 return events.any **{** event **->** val isSameTitle = event.title == "$vaccine for $childsName"  
 val isSameStartDate = event.startDate.time == startDate.time  
 val isSameEndDate = event.endDate.time == endDate.time  
 isSameTitle && isSameStartDate && isSameEndDate  
 **}** }  
  
 fun saveEventIfNotAdded(  
 eventStore: EKEventStore,  
 for vaccine: String,  
 at date: Date,  
 ageText: String,  
 ageInMonths: Int  
 ) {  
 if (!isEventAlreadyAdded(eventStore, for: vaccine, at: date, childsName = childsName)) {  
 saveEvent(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  
 }  
 }  
  
 fun calculateAgeText(from startDate: Date, to endDate: Date): String? {  
 val calendar = Calendar.getInstance()  
 val components = calendar.dateComponents(*listOf*(Calendar.*YEAR*, Calendar.*MONTH*), from = startDate, to = endDate)  
  
 val years = components.get(Calendar.*YEAR*)  
 val months = components.get(Calendar.*MONTH*)  
  
 return if (years != null && months != null) {  
 if (years > 0) {  
 "Your child is $years year${if (years > 1) "s" else ""} and $months month${if (months > 1) "s" else ""} old"  
 } else {  
 "Your child is $months month${if (months > 1) "s" else ""} old"  
 }  
 } else {  
 null  
 }  
 }  
  
 @Composable  
 fun HeaderView(  
 ageText: String,  
 isDatePickerVisible: Boolean,  
 ) {  
 *Text*(ageText)  
 .font(MaterialTheme.typography.h1) // Assuming h1 is desired font size  
 .multilineTextAlignment(TextAlign.Center)  
 .foregroundColor(Color.White)  
 .padding(vertical = 16.*dp*) // Using dp for padding  
 .background(*Color*(0.216f, 0.498f, 0.722f)) // Assuming Color is from androidx.compose.ui.graphics  
 .fillMaxWidth()  
 }  
 @Composable  
 object MyButtonStyle : ButtonStyle {  
  
 @Composable  
 override fun makeBody(configuration: Configuration): Content {  
 configuration.label  
 .padding(all = 16.*dp*) // Using dp for padding  
 .foregroundColor(Color.White)  
 .background(Color.Red)  
 .cornerRadius(10.*dp*)  
 }  
 }  
  
  
  
 // VaccinesDueButton Composable  
@Composable  
fun VaccinesDueButton(  
 checkVaccines: () -> Unit,  
 selectedVaccines: List<Pair<String, Date>> // or MutableList if you need to modify it  
) {  
 *Text*(  
 "Vaccines Due",  
 style = MaterialTheme.typography.h4,  
 color = Color.White,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*background*(*Color*(0xFF377DB5))  
 .*padding*(16.*dp*)  
 )  
}  
  
 fun calculateMonthsRemaining(from startDate: Date, to endDate: Date): Int {  
 val calendar = Calendar.getInstance()  
 val components = calendar.dateComponents(Calendar.*MONTH*, from = startDate, to = endDate)  
 return components.get(Calendar.*MONTH*) ?: 0  
 }  
 fun checkVaccines() {  
 val today = Date()  
 val calculatedDueDates = *mutableListOf*<Date>()  
  
 // Calculate age in months  
 val calendar = Calendar.getInstance()  
 val ageComponents = calendar.dateComponents(Calendar.*MONTH*, from = selectedDate, to = today)  
 val ageInMonths = ageComponents.get(Calendar.*MONTH*) ?: 0  
 *println*("age of child in months: $ageInMonths")  
  
 val uniqueSelectedVaccines = *mutableSetOf*<String>()  
 val vaccineGroups = *mutableMapOf*<String, List<Pair<String, Date>>>()  
  
 for ((vaccine, months) in vaccineMonthsMap) {  
 if (months.*isNotEmpty*() && months.*lastOrNull*() ?: 0 >= ageInMonths) {  
 val dueDate = calculateDueDateForVaccine(vaccine, at = ageInMonths)  
 if (dueDate != null) {  
 calculatedDueDates.add(dueDate)  
 }  
 }  
 }  
  
 calculatedDueDates.*sort*()  
  
 for (dueDate in calculatedDueDates) {  
 *println*("Processing vaccines for due date: $dueDate")  
  
 val vaccines = calculateVaccine(for = dueDate, ageInMonths = ageInMonths)  
 if (vaccines != null) {  
 *println*("Processing vaccines inside loop for due date: $dueDate")  
 for ((vaccine, ageText) in vaccines) {  
 val vaccineKey = "$vaccine-${dueDate.*time*}"  
 if (vaccineKey !in uniqueSelectedVaccines) {  
 uniqueSelectedVaccines.add(vaccineKey)  
 addVaccine(for: vaccine, inAgeGroup: ageText, to = vaccineGroups, currentDate: dueDate, ageInMonths = ageInMonths)  
 }  
 }  
 }  
 }  
  
 // Clear the set after processing all due dates  
 uniqueSelectedVaccines.clear()  
  
 *println*("vaccineGroups list:")  
 *println*(vaccineGroups)  
  
 // Sort the vaccineGroups based on the age group identifier  
 val sortedVaccineGroups = vaccineGroups.*toList*().*sortedBy* **{ it**.first **}** // Clear vaccineGroups and add the sorted groups  
 vaccineGroups.clear()  
 sortedVaccineGroups.*forEach* **{** (identifier, vaccines) **->** vaccineGroups[identifier] = vaccines  
 **}** *println*("Ordered vaccineGroups:")  
 *println*(vaccineGroups)  
  
 // Ensure UI updates on the main thread (assuming you have a DispatchQueue instance)  
 if (Dispatchers.Main.isDispatchThread) {  
 this.vaccineGroups = vaccineGroups  
 // self.sortedVaccineGroups = sortedVaccineGroups // Assuming you don't need this  
 handleClickHereButton(vaccineGroups = vaccineGroups)  
 } else {  
 withContext(Dispatchers.Main) **{** this@VaccinesDueSection.vaccineGroups = vaccineGroups  
 // self.sortedVaccineGroups = sortedVaccineGroups // Assuming you don't need this  
 handleClickHereButton(vaccineGroups = vaccineGroups)  
 **}** }  
 }  
  
  
 fun calculateDueDateForVaccine(vaccine: String, at age: Int): Date? {  
 val lowercaseVaccine = vaccine.*lowercase*()  
  
 val selectedMonthsToAddOptions = vaccineMonthsMap[lowercaseVaccine]  
 ?: return *run* **{** *println*("Error: No months specified for $vaccine")  
 null  
 **}** // If there are no specified months, return nil  
 val firstMonth = selectedMonthsToAddOptions.*firstOrNull*()  
 ?: return *run* **{** *println*("Error: No months specified for $vaccine")  
 null  
 **}** // Calculate the difference between due date and age (absolute value)  
 val adjustedMonths = Math.abs(firstMonth - age)  
  
 // Calculate the due date using the adjusted months for the vaccine  
 val dueDate = Calendar.getInstance().add(Calendar.*MONTH*, adjustedMonths)  
 *println*("dueDate $dueDate")  
  
 return dueDate.time // Assuming dueDate is not null  
  
 }  
  
 fun calculateVaccine(for dueDate: Date, ageInMonths: Int): List<Pair<String, String>>? {  
 val monthsRemaining = calculateMonthsRemaining(from = currentDate, to = dueDate)  
 *println*("Months remaining: $monthsRemaining")  
  
 val result = *mutableListOf*<Pair<String, String>>()  
  
 for ((vaccineName, specifiedMonths) in vaccineMonthsMap) {  
 val firstMonth = specifiedMonths.*firstOrNull*() ?: continue  
 val ageGroup = ContentView.getAgeText(for = firstMonth) // Assuming ContentView.getAgeText exists  
 if (ageGroup == null || specifiedMonths.*lastOrNull*() ?: 0 <= ageInMonths) continue  
  
 if (specifiedMonths.*any* **{ it** >= monthsRemaining **}**) {  
 result.add(vaccineName *to* ageGroup)  
 *println*("specifiedMonths: $result")  
 }  
 }  
  
 return if (result.isEmpty()) {  
 println("No vaccines found for due date: $  
 object ContentView {  
 fun getAgeText(for monthsRemaining: Int): String? {  
 return when (monthsRemaining) {  
 2 -> "Vaccines due at 2 months:"  
 3 -> "Vaccines due at 3 months:"  
 4 -> "Vaccines due at 4 months:"  
 12 -> "Vaccines due at 1 year:"  
 36 -> "Vaccines due at 3 years:"  
 156 -> "Vaccines due at 13 years:"  
 168 -> "Vaccines due at 14 years:"  
 else -> null  
 }  
 }  
 }  
 fun addDueDate(for name: String, at month: Int, to array: MutableList<Date>) {  
 val dueDate = calculateDueDateForVaccine(name, at = month)  
 if (dueDate != null) {  
 array.add(dueDate)  
 // Use println(...) for logging if needed  
 } else {  
 *print*("No due date added for $name at $month months")  
 }  
 }  
 fun addVaccine(for name: String, inAgeGroup ageGroup: String, to dict: MutableMap<String, List<Pair<String, Date>>>, currentDate: Date, ageInMonths: Int) {  
 val dueDate = calculateDueDateForVaccine(name, at = ageInMonths) ?: *run* **{** *print*("Error: Unable to calculate due date for $name")  
 return  
 **}** val vaccineTuple = name *to* dueDate  
  
 val existingVaccines = dict[ageGroup]  
 if (existingVaccines != null) {  
 if (existingVaccines.*none* **{ it**.first == name && **it**.second == dueDate **}**) {  
 *print*("Adding vaccine $name to age group $ageGroup")  
 existingVaccines.add(vaccineTuple)  
 dict[ageGroup] = existingVaccines  
 }  
 } else {  
 *print*("Adding vaccine $name to new age group $ageGroup")  
 dict[ageGroup] = *listOf*(vaccineTuple)  
 *println*("vaccineTuple $vaccineTuple")  
 }  
  
 vaccinesDue.clear() // Assuming vaccinesDue is a mutable list  
 }  
 fun calculateAgeDueText(from currentDate: Date, to dueDate: Date): String? {  
 val calendar = Calendar.getInstance()  
 val components = calendar.dateComponents(Calendar.*MONTH*, from = currentDate, to = dueDate)  
 val monthsRemaining = components.get(Calendar.*MONTH*) ?: return null  
  
 return if (monthsRemaining < 12) {  
 "Due in $monthsRemaining months"  
 } else {  
 val years = monthsRemaining / 12  
 val months = monthsRemaining % 12  
 "Due at $years year${if (years == 1) "" else "s"} and $months month${if (months == 1) "" else "s"}"  
 }  
 }  
 fun calculateDueDate(for vaccine: String, at ageInMonths: Int): Date? {  
 return when (ageInMonths) {  
 in 0..2 -> calculateDueDateForVaccine(vaccine, at = 2)  
 in 3..4 -> calculateDueDateForVaccine(vaccine, at = 4)  
 in 5..12 -> calculateDueDateForVaccine(vaccine, at = 12)  
 in 13..36 -> calculateDueDateForVaccine(vaccine, at = 36)  
 in 36..144 -> calculateDueDateForVaccine(vaccine, at = 144)  
 in 145..156 -> calculateDueDateForVaccine(vaccine, at = 156)  
 else -> null  
 }  
 }  
 fun handleClickHereButton(vaccineGroups: Map<String, List<Pair<String, Date>>>) {  
 val eventStore = this.eventStore ?: *run* **{** *print*("Error: Event store is not initialized.")  
 return  
 **}** for ((\_, vaccines) in vaccineGroups) {  
 for ((vaccine, dueDate) in vaccines) {  
 val ageDueText = calculateAgeDueText(from = currentDate, to = dueDate)  
 if (ageDueText != null) {  
 *print*("eventStore $eventStore") // Assuming you want to log the eventStore here  
 }  
 }  
 }  
  
 showingAlert = true  
 }  
 fun formattedDateString(from date: Date): String {  
 val dateFormatter = SimpleDateFormat("M/d/yy", Locale.getDefault()) // Adjust format if needed  
 return dateFormatter.format(date)  
 }  
 @Composable  
 data class InfoCell(val info: information) {  
 @Composable  
 fun Content() {  
 *Spacer*(modifier = Modifier.weight(1f))  
 *Row*(horizontalArrangement = Arrangement.spacedBy(5.*dp*)) **{** *Image*(  
 asset = painterResource(id = info.imageName),  
 contentDescription = info.title, // Set content description  
 modifier = Modifier  
 .resizable(AspectRatioContent.Fit)  
 .scaledToFit()  
 .size(100.*dp*, 70.*dp*)  
 .cornerRadius(4.*dp*)  
 .padding(vertical = 8.*dp*)  
 )  
 *Column*(modifier = Modifier.*weight*(1f)) **{** Text(  
 text = info.title,  
 fontWeight = FontWeight.SemiBold,  
 maxLines = 2,  
 modifier = Modifier.*padding*(5.*dp*).minimumScaleFactor(0.5f)  
 )  
 *Text*(  
 text = "Click here for more information",  
 fontSize = MaterialTheme.typography.subtitle1.fontSize,  
 modifier = Modifier.*padding*(5.*dp*)  
 )  
 **}  
 }** }  
 }