Createventscreen:

package com.carlitoswy.flashmeet.presentation.event  
  
import android.net.Uri  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.Arrangement  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.lazy.LazyRow  
import androidx.compose.foundation.lazy.items  
import androidx.compose.material3.Button  
import androidx.compose.material3.ExperimentalMaterial3Api  
import androidx.compose.material3.LinearProgressIndicator  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.OutlinedTextField  
import androidx.compose.material3.Scaffold  
import androidx.compose.material3.Text  
import androidx.compose.material3.TextButton  
import androidx.compose.material3.TopAppBar  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.collectAsState  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.setValue  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.clip  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.unit.dp  
import androidx.hilt.navigation.compose.hiltViewModel  
import androidx.navigation.NavHostController  
import coil.compose.rememberAsyncImagePainter  
import com.carlitoswy.flashmeet.domain.model.AdOption  
import com.carlitoswy.flashmeet.presentation.shared.AdOptionChip  
import com.carlitoswy.flashmeet.presentation.shared.PendingEventData  
import com.carlitoswy.flashmeet.presentation.shared.SharedEventStateViewModel  
import com.carlitoswy.flashmeet.ui.navigation.Routes  
import com.google.firebase.auth.FirebaseAuth  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun CreateEventScreen(  
 navController: NavHostController,  
 createEventViewModel: CreateEventViewModel = hiltViewModel(),  
 sharedEventStateViewModel: SharedEventStateViewModel = hiltViewModel(),  
 onEventCreated: () -> Unit  
) {  
 var title by remember **{** *mutableStateOf*("") **}** var description by remember **{** *mutableStateOf*("") **}** var highlightedText by remember **{** *mutableStateOf*("") **}** var adOption by remember **{** *mutableStateOf*(AdOption.*NONE*) **}** var imageUri by remember **{** *mutableStateOf*<Uri?>(null) **}** var locationName by remember **{** *mutableStateOf*("") **}** val context = *LocalContext*.current  
 val userId = FirebaseAuth.getInstance().*currentUser*?.*uid* ?: "unknown"  
  
 val loading by createEventViewModel.isLoading.collectAsState()  
 val error by createEventViewModel.errorMessage.collectAsState()  
 val estimatedCost = createEventViewModel.estimateAdCost(adOption)  
  
 val cameraLauncher = rememberLauncherForActivityResult(ActivityResultContracts.TakePicturePreview()) **{** // *TODO: Guardar como archivo y convertir a Uri si lo necesitas* **}** val galleryLauncher = rememberLauncherForActivityResult(ActivityResultContracts.GetContent()) **{** uri **->** imageUri = uri  
 **}** Scaffold(topBar = **{** TopAppBar(  
 title = **{** Text("Crear Evento") **}**,  
 actions = **{** TextButton(onClick = **{** if (adOption == AdOption.*NONE*) {  
 createEventViewModel.createEvent(  
 title = title,  
 description = description,  
 adOption = adOption,  
 highlightedText = highlightedText,  
 imageUri = imageUri,  
 locationName = locationName,  
 onSuccess = onEventCreated  
 )  
 } else {  
 sharedEventStateViewModel.savePendingEvent(  
 PendingEventData(  
 title = title,  
 description = description,  
 imageUri = imageUri?.toString(),  
 locationName = locationName,  
 userId = userId,  
 adOption = adOption,  
 highlightedText = highlightedText  
 )  
 )  
 navController.navigate(Routes.PAYMENT)  
 }  
 **}**) **{** Text("Guardar")  
 **}  
 }** )  
 **}**) **{** padding **->** Column(  
 modifier = Modifier  
 .*padding*(padding)  
 .*padding*(16.*dp*)  
 .*fillMaxSize*(),  
 verticalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** OutlinedTextField(  
 value = title,  
 onValueChange = **{** title = **it }**,  
 label = **{** Text("Título") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
 OutlinedTextField(  
 value = description,  
 onValueChange = **{** description = **it }**,  
 label = **{** Text("Descripción") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
 OutlinedTextField(  
 value = highlightedText,  
 onValueChange = **{** highlightedText = **it }**,  
 label = **{** Text("Texto resaltado") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
 OutlinedTextField(  
 value = locationName,  
 onValueChange = **{** locationName = **it }**,  
 label = **{** Text("Ubicación del evento") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 Row(horizontalArrangement = Arrangement.spacedBy(16.*dp*)) **{** Button(onClick = **{** cameraLauncher.launch(null) **}**) **{** Text("Cámara") **}** Button(onClick = **{** galleryLauncher.launch("image/\*") **}**) **{** Text("Galería") **}  
 }** imageUri?.*let* **{** Image(  
 painter = rememberAsyncImagePainter(**it**),  
 contentDescription = null,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(200.*dp*)  
 .*clip*(MaterialTheme.shapes.medium)  
 )  
 **}** Text("Publicidad:")  
 LazyRow(horizontalArrangement = Arrangement.spacedBy(8.*dp*)) **{** *items*(AdOption.*entries*) **{** option **->** AdOptionChip(option, selected = adOption == option) **{** adOption = option  
 **}  
 }  
 }** Box(  
 modifier = Modifier  
 .*background*(MaterialTheme.colorScheme.secondary, MaterialTheme.shapes.small)  
 .*padding*(12.*dp*)  
 ) **{** Text(  
 "Costo estimado: $estimatedCost",  
 color = MaterialTheme.colorScheme.onSecondary  
 )  
 **}** if (error != null) {  
 Text(text = error!!, color = MaterialTheme.colorScheme.error)  
 }  
  
 if (loading) {  
 LinearProgressIndicator(modifier = Modifier.*fillMaxWidth*())  
 }  
 **}  
 }**}

createventviewmodel:

package com.carlitoswy.flashmeet.presentation.createevent  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import com.carlitoswy.flashmeet.domain.model.FlashEvent  
import com.carlitoswy.flashmeet.domain.repository.FlashEventRepository  
import com.carlitoswy.flashmeet.domain.usecase.GetSignedInUserUseCase  
import dagger.hilt.android.lifecycle.HiltViewModel  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.asStateFlow  
import kotlinx.coroutines.launch  
import java.util.UUID  
import javax.inject.Inject  
  
// State class to hold the input field values  
data class EventState(  
 val title: String = "",  
 val description: String = "",  
 // You might add state for loading, errors, success for the submission process here too  
 // val isLoading: Boolean = false,  
 // val submissionError: String? = null,  
 // val isSubmitted: Boolean = false  
)  
  
// You might also define a separate sealed class for submission state:  
// sealed class EventSubmissionState {  
// object Idle : EventSubmissionState()  
// object Loading : EventSubmissionState()  
// object Success : EventSubmissionState()  
// data class Error(val message: String) : EventSubmissionState()  
// }  
  
  
@HiltViewModel  
class CreateEventViewModel @Inject constructor(  
 private val eventRepository: FlashEventRepository,  
 private val getSignedInUserUseCase: GetSignedInUserUseCase // Use case to get the currently signed-in UserProfile  
) : ViewModel() {  
  
 // StateFlow for the input fields  
 private val \_eventState = *MutableStateFlow*(EventState())  
 val eventState = \_eventState.*asStateFlow*()  
  
 // Optional: StateFlow for the event submission process state  
 // private val \_submissionState = MutableStateFlow<EventSubmissionState>(EventSubmissionState.Idle)  
 // val submissionState = \_submissionState.asStateFlow()  
  
 fun onTitleChanged(newTitle: String) {  
 \_eventState.value = \_eventState.value.copy(title = newTitle)  
 }  
  
 fun onDescriptionChanged(newDescription: String) {  
 \_eventState.value = \_eventState.value.copy(description = newDescription)  
 }  
  
 fun submitEvent() {  
 *viewModelScope*.*launch* **{** // Optional: Update submission state to Loading  
 // \_submissionState.value = EventSubmissionState.Loading  
  
 val user = getSignedInUserUseCase() // Get the signed-in user profile  
 if (user == null) {  
 // Handle case where user is not signed in  
 // Optional: Update submission state to Error  
 // \_submissionState.value = EventSubmissionState.Error("User not signed in.")  
 return@launch  
 }  
  
 val currentEventState = \_eventState.value  
  
 // Ensure title and description are not empty (optional, add validation)  
 if (currentEventState.title.*isBlank*() || currentEventState.description.*isBlank*()) {  
 // Optional: Update submission state to Error  
 // \_submissionState.value = EventSubmissionState.Error("Title and description cannot be empty.")  
 return@launch // Or show a message to the user  
 }  
  
 // Create the FlashEvent object  
 val newEvent = FlashEvent(  
 id = UUID.randomUUID().toString(),  
 title = currentEventState.title,  
 description = currentEventState.description,  
 creatorId = user.uid, // Use user.uid for creatorId  
 // FIX: Use user.name and provide a default value if user.name is nullable  
 // Assuming FlashEvent.createdBy is String (non-nullable)  
 createdBy = user.name ?: user.email ?: "Unknown User" // Use name, fallback to email, then "Unknown User" if both are null  
 )  
  
 // Save the event via the repository  
 try {  
 eventRepository.createEvent(newEvent)  
 // Optional: Update submission state to Success  
 // \_submissionState.value = EventSubmissionState.Success  
  
 // Clear input fields after successful submission (optional)  
 \_eventState.value = EventState()  
  
 } catch (e: Exception) {  
 // Handle creation error  
 // Optional: Update submission state to Error  
 // \_submissionState.value = EventSubmissionState.Error(e.localizedMessage ?: "Failed to create event")  
 }  
 **}** }  
  
 // Optional function to reset submission state  
 // fun resetSubmissionState() {  
 // \_submissionState.value = EventSubmissionState.Idle  
 // }  
}

event:

package com.carlitoswy.flashmeet.domain.model  
  
data class Event(  
 val id: String = "",  
 val title: String = "",  
 val description: String = "",  
 val adOption: String = "NONE",  
 val highlightedText: String = "",  
 val createdBy: String = "",  
 val timestamp: Long = System.currentTimeMillis(),  
 val locationName: String = "",  
 val category: EventCategory = EventCategory.*OTHER*, // 🔥 Ahora es enum  
 val imageUrl: String? = null,  
  
 // ✅ Campos para ubicación  
 val country: String = "",  
 val city: String = "",  
 val postal: String = "",  
  
 val latitude: Double = 0.0,  
 val longitude: Double = 0.0  
)  
  
// ✅ Conversión para Firestore  
fun EventCategory.toFirestoreString(): String = this.name  
  
fun String?.toEventCategory(): EventCategory =  
 EventCategory.values().*find* **{ it**.name.*equals*(this, ignoreCase = true) **}** ?: EventCategory.*OTHER*

eventrepository:

// com.carlitoswy.flashmeet.domain.repository.EventRepository.kt  
package com.carlitoswy.flashmeet.domain.repository  
  
import com.carlitoswy.flashmeet.domain.model.Event  
import kotlinx.coroutines.flow.Flow  
  
interface EventRepository {  
 fun getMyEvents(): Flow<List<Event>>  
 fun getNearbyEvents(latitude: Double, longitude: Double): Flow<List<Event>>  
 suspend fun createEvent(event: Event)  
 // AÑADE ESTA LÍNEA:  
 suspend fun searchEvents(country: String, city: String, postal: String, category: String, date: String): List<Event>  
  
 suspend fun getEventById(eventId: String): Event?  
 suspend fun updateEvent(event: Event)  
  
}

Eventrepositoryimpl:

package com.carlitoswy.flashmeet.data.repository  
  
import android.util.Log  
import com.carlitoswy.flashmeet.domain.model.Event  
import com.carlitoswy.flashmeet.domain.repository.EventRepository  
import com.carlitoswy.flashmeet.utils.await  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.firestore.FirebaseFirestore  
import com.google.firebase.firestore.Query  
import kotlinx.coroutines.channels.awaitClose  
import kotlinx.coroutines.flow.Flow  
import kotlinx.coroutines.flow.callbackFlow  
import java.text.ParseException  
import java.text.SimpleDateFormat  
import java.util.Calendar  
import java.util.Locale  
import javax.inject.Inject  
import javax.inject.Singleton  
  
@Singleton  
class EventRepositoryImpl @Inject constructor(  
 private val firestore: FirebaseFirestore  
) : EventRepository {  
  
 private val TAG = "EventRepo"  
  
 override fun getMyEvents(): Flow<List<Event>> = *callbackFlow* **{** val userId = FirebaseAuth.getInstance().*currentUser*?.*uid* ?: *run* **{** close(IllegalStateException("User not logged in"))  
 return@callbackFlow  
 **}** val subscription = firestore.collection("events")  
 .whereEqualTo("createdBy", userId)  
 .addSnapshotListener **{** snapshot, e **->** if (e != null) {  
 close(e)  
 } else {  
 val events = snapshot?.toObjects(Event::class.*java*).*orEmpty*()  
 Log.d(TAG, "getMyEvents → ${events.size} eventos obtenidos")  
 trySend(events)  
 }  
 **}** awaitClose **{** subscription.remove() **}  
 }** override suspend fun createEvent(event: Event) {  
 firestore.collection("events").document(event.id).set(event).await()  
 Log.d(TAG, "Evento creado: ${event.id}")  
 }  
  
 override fun getNearbyEvents(latitude: Double, longitude: Double): Flow<List<Event>> = *callbackFlow* **{** val query = firestore.collection("events")  
 .whereGreaterThanOrEqualTo("latitude", latitude - 0.1)  
 .whereLessThanOrEqualTo("latitude", latitude + 0.1)  
 .whereGreaterThanOrEqualTo("longitude", longitude - 0.1)  
 .whereLessThanOrEqualTo("longitude", longitude + 0.1)  
  
 val subscription = query.addSnapshotListener **{** snapshot, e **->** if (e != null) {  
 close(e)  
 return@addSnapshotListener  
 }  
 val events = snapshot?.toObjects(Event::class.*java*).*orEmpty*()  
 Log.d(TAG, "getNearbyEvents → ${events.size} eventos obtenidos cerca de $latitude,$longitude")  
 trySend(events)  
 **}** awaitClose **{** subscription.remove() **}  
 }** override suspend fun searchEvents(  
 country: String,  
 city: String,  
 postal: String,  
 category: String,  
 date: String  
 ): List<Event> {  
 return try {  
 var query: Query = firestore.collection("events")  
  
 if (country.*isNotEmpty*()) query = query.whereEqualTo("country", country)  
 if (city.*isNotEmpty*()) query = query.whereEqualTo("city", city)  
 if (postal.*isNotEmpty*()) query = query.whereEqualTo("postal", postal)  
 if (category.*isNotEmpty*() && category != "Todos") query = query.whereEqualTo("category", category)  
  
 // ✅ Filtro por fecha si se proporciona  
 if (date.*isNotEmpty*()) {  
 try {  
 val dateFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())  
 val selectedDate = dateFormat.parse(date)  
  
 val calendar = Calendar.getInstance().*apply* **{** *time* = selectedDate!!  
 set(Calendar.*HOUR\_OF\_DAY*, 0)  
 set(Calendar.*MINUTE*, 0)  
 set(Calendar.*SECOND*, 0)  
 set(Calendar.*MILLISECOND*, 0)  
 **}** val startOfDay = calendar.*timeInMillis* calendar.add(Calendar.*DAY\_OF\_MONTH*, 1)  
 val endOfDay = calendar.*timeInMillis* query = query  
 .orderBy("timestamp")  
 .whereGreaterThanOrEqualTo("timestamp", startOfDay)  
 .whereLessThan("timestamp", endOfDay)  
  
 Log.d(TAG, "Filtro fecha → $startOfDay a $endOfDay")  
 } catch (e: ParseException) {  
 Log.w(TAG, "Error parseando fecha: $date → ${e.message}")  
 }  
 }  
  
 val snapshot = query.get().await()  
 val events = snapshot.*documents*.*mapNotNull* **{ it**.toObject(Event::class.*java*) **}** Log.d(TAG, "searchEvents → ${events.size} eventos encontrados con filtros: " +  
 "[country=$country, city=$city, postal=$postal, category=$category, date=$date]")  
  
 events  
 } catch (e: Exception) {  
 Log.e(TAG, "Error buscando eventos: ${e.message}", e)  
 *emptyList*()  
 }  
 }  
  
 override suspend fun getEventById(eventId: String): Event? {  
 return try {  
 firestore.collection("events").document(eventId)  
 .get().await()  
 .toObject(Event::class.*java*).*also* **{** Log.d(TAG, "getEventById($eventId) → ${**it** != null}")  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error obteniendo evento $eventId: ${e.message}")  
 null  
 }  
 }  
  
 override suspend fun updateEvent(event: Event) {  
 try {  
 firestore.collection("events").document(event.id).set(event).await()  
 Log.d(TAG, "Evento actualizado: ${event.id}")  
 } catch (e: Exception) {  
 Log.e(TAG, "Error actualizando evento ${event.id}: ${e.message}")  
 throw e  
 }  
 }  
}

createeventusecase:

package com.carlitoswy.flashmeet.domain.usecase  
  
import com.carlitoswy.flashmeet.domain.model.FlashEvent  
import com.carlitoswy.flashmeet.domain.repository.EventRepository  
import javax.inject.Inject  
  
class CreateEventUseCase @Inject constructor(  
 private val repository: EventRepository  
) {  
 suspend operator fun invoke(event: FlashEvent) {  
 repository.createEvent(event)  
 }  
}

cameracapturescreen:

package com.carlitoswy.flashmeet.ui.screens  
  
import android.net.Uri  
import android.util.Log  
import androidx.camera.core.CameraSelector  
import androidx.camera.core.ImageCapture  
import androidx.camera.core.ImageCaptureException  
import androidx.camera.lifecycle.ProcessCameraProvider  
import androidx.camera.view.PreviewView  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.Button  
import androidx.compose.material3.Text  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.platform.*LocalLifecycleOwner*import androidx.compose.ui.unit.dp  
import androidx.compose.ui.viewinterop.AndroidView  
import androidx.core.content.ContextCompat  
import androidx.lifecycle.LifecycleOwner  
import java.io.File  
import java.text.SimpleDateFormat  
import java.util.\*  
  
@Composable  
fun CameraCaptureScreen(  
 onImageCaptured: (Uri) -> Unit,  
 onError: (ImageCaptureException) -> Unit  
) {  
 val context = *LocalContext*.current  
 val lifecycleOwner: LifecycleOwner = *LocalLifecycleOwner*.current  
 val cameraProviderFuture = remember **{** ProcessCameraProvider.getInstance(context) **}** var imageCapture by remember **{** *mutableStateOf*<ImageCapture?>(null) **}** val outputDirectory = remember **{** File(context.*cacheDir*, "images").*apply* **{** mkdirs() **}  
 }** Column(modifier = Modifier.*fillMaxSize*()) **{** AndroidView(  
 factory = **{** ctx **->** val previewView = PreviewView(ctx)  
  
 cameraProviderFuture.addListener(**{** val cameraProvider = cameraProviderFuture.get()  
  
 val preview = androidx.camera.core.Preview.Builder().build().*apply* **{** setSurfaceProvider(previewView.*surfaceProvider*)  
 **}** imageCapture = ImageCapture.Builder().build()  
  
 val cameraSelector = CameraSelector.*DEFAULT\_BACK\_CAMERA* try {  
 cameraProvider.unbindAll()  
 cameraProvider.bindToLifecycle(  
 lifecycleOwner,  
 cameraSelector,  
 preview,  
 imageCapture  
 )  
 } catch (exc: Exception) {  
 Log.e("CameraCapture", "Use case binding failed", exc)  
 }  
  
 **}**, ContextCompat.getMainExecutor(ctx))  
  
 previewView  
 **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*aspectRatio*(3f / 4f)  
 )  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 Button(  
 onClick = **{** val photoFile = File(  
 outputDirectory,  
 "IMG\_${SimpleDateFormat("yyyyMMdd\_HHmmss", Locale.*US*).format(Date())}.jpg"  
 )  
  
 val outputOptions = ImageCapture.OutputFileOptions.Builder(photoFile).build()  
  
 imageCapture?.takePicture(  
 outputOptions,  
 ContextCompat.getMainExecutor(context),  
 object : ImageCapture.OnImageSavedCallback {  
 override fun onError(exc: ImageCaptureException) {  
 Log.e("CameraCapture", "Photo capture failed: ${exc.message}", exc)  
 onError(exc)  
 }  
  
 override fun onImageSaved(output: ImageCapture.OutputFileResults) {  
 val savedUri = Uri.fromFile(photoFile)  
 Log.d("CameraCapture", "Photo capture succeeded: $savedUri")  
 onImageCaptured(savedUri)  
 }  
 }  
 )  
 **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*)  
 ) **{** Text("Capturar Foto")  
 **}  
 }**}

flyereditorscreen:

package com.carlitoswy.flashmeet.ui.screens  
  
import android.Manifest  
import android.net.Uri  
import android.util.Log  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.foundation.background  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.Arrangement  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.layout.size  
import androidx.compose.foundation.lazy.LazyRow  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*CameraAlt*import androidx.compose.material.icons.filled.*Image*import androidx.compose.material3.ExperimentalMaterial3Api  
import androidx.compose.material3.Icon  
import androidx.compose.material3.IconButton  
import androidx.compose.material3.OutlinedTextField  
import androidx.compose.material3.Scaffold  
import androidx.compose.material3.Text  
import androidx.compose.material3.TextButton  
import androidx.compose.material3.TopAppBar  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.LaunchedEffect  
import androidx.compose.runtime.collectAsState  
import androidx.compose.runtime.getValue  
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.font.FontFamily  
import androidx.compose.ui.unit.dp  
import androidx.hilt.navigation.compose.hiltViewModel  
import androidx.navigation.NavHostController  
import coil.compose.AsyncImage  
import com.carlitoswy.flashmeet.presentation.flyer.FlyerEditorViewModel  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun FlyerEditorScreen(  
 navController: NavHostController,  
 viewModel: FlyerEditorViewModel = hiltViewModel()  
) {  
 val title by viewModel.title.collectAsState()  
 val description by viewModel.description.collectAsState()  
 val imageUri by viewModel.imageUri.collectAsState()  
 val bgColor by viewModel.bgColor.collectAsState()  
 val fontName by viewModel.fontName.collectAsState()  
 val status by viewModel.status.collectAsState()  
  
 val context = *LocalContext*.current  
  
 val galleryLauncher = rememberLauncherForActivityResult(  
 ActivityResultContracts.GetContent()  
 ) **{** uri: Uri? **->** uri?.*let* **{** viewModel.setImage(**it**) **} }** val permissionLauncher = rememberLauncherForActivityResult(  
 ActivityResultContracts.RequestPermission()  
 ) **{** granted **->** if (granted) {  
 navController.navigate("camera\_capture")  
 } else {  
 Log.e("FlyerEditor", "Permiso de cámara denegado")  
 }  
 **}** Scaffold(  
 topBar = **{** TopAppBar(  
 title = **{** Text("Crear Flyer") **}**,  
 actions = **{** TextButton(onClick = **{** viewModel.saveFlyer("demoUser") **}**) **{** Text("Guardar")  
 **}  
 }** )  
 **}** ) **{** padding **->** Column(  
 modifier = Modifier  
 .*padding*(padding)  
 .*fillMaxSize*()  
 .*padding*(16.*dp*),  
 verticalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** Box(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(200.*dp*)  
 .*background*(*Color*(bgColor)),  
 contentAlignment = Alignment.Center  
 ) **{** if (imageUri != null) {  
 AsyncImage(model = imageUri, contentDescription = "Flyer image")  
 } else {  
 Text("Toca un botón para agregar imagen")  
 }  
 **}** Row(horizontalArrangement = Arrangement.spacedBy(12.*dp*)) **{** IconButton(onClick = **{** permissionLauncher.launch(Manifest.permission.*CAMERA*)  
 **}**) **{** Icon(Icons.Default.*CameraAlt*, contentDescription = "Tomar foto")  
 **}** IconButton(onClick = **{** galleryLauncher.launch("image/\*")  
 **}**) **{** Icon(Icons.Default.*Image*, contentDescription = "Elegir imagen")  
 **}  
 }** OutlinedTextField(  
 value = title,  
 onValueChange = **{** viewModel.title.value = **it }**,  
 label = **{** Text("Título") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 OutlinedTextField(  
 value = description,  
 onValueChange = **{** viewModel.description.value = **it }**,  
 label = **{** Text("Descripción") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 Text("Color de fondo:")  
 LazyRow **{** val colors = *listOf*(Color.White, Color.Yellow, Color.Cyan, Color.LightGray)  
 items(colors.size) **{** i **->** Box(  
 modifier = Modifier  
 .*size*(40.*dp*)  
 .*background*(colors[i])  
 .*clickable* **{** viewModel.bgColor.value = colors[i].value.toInt() **}** )  
 **}  
 }** Text("Fuente:")  
 LazyRow **{** val fonts = *listOf*(  
 FontFamily.SansSerif *to* "Sans",  
 FontFamily.Serif *to* "Serif",  
 FontFamily.Monospace *to* "Mono"  
 )  
 items(fonts.size) **{** i **->** Text(  
 text = fonts[i].second,  
 fontFamily = fonts[i].first,  
 modifier = Modifier  
 .*padding*(8.*dp*)  
 .*clickable* **{** viewModel.fontName.value = fonts[i].second **}** )  
 **}  
 }** Spacer(modifier = Modifier.*weight*(1f))  
 **}  
 }** // Navegar al terminar  
 LaunchedEffect(status) **{** if (status == "ok") {  
 navController.popBackStack()  
 }  
 **}**}

flyereditorviewmodel:

package com.carlitoswy.flashmeet.presentation.flyer  
  
import android.net.Uri  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import com.carlitoswy.flashmeet.domain.model.Flyer  
import com.carlitoswy.flashmeet.utils.await  
import com.google.firebase.firestore.FirebaseFirestore  
import com.google.firebase.storage.FirebaseStorage  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.launch  
  
class FlyerEditorViewModel : ViewModel() {  
 private val firestore = FirebaseFirestore.getInstance()  
 private val storage = FirebaseStorage.getInstance().*reference* val title = *MutableStateFlow*("")  
 val description = *MutableStateFlow*("")  
 val imageUri = *MutableStateFlow*<Uri?>(null)  
 val bgColor = *MutableStateFlow*(0xFFFFFFFF.toInt())  
 val fontName = *MutableStateFlow*("SansSerif")  
  
 private val \_status = *MutableStateFlow*<String?>(null)  
 val status: StateFlow<String?> = \_status  
  
 fun setImage(uri: Uri) { imageUri.value = uri }  
  
 fun saveFlyer(createdBy: String = "anonymous") {  
 *viewModelScope*.*launch* **{** try {  
 val flyerId = firestore.collection("flyers").document().*id* val downloadUrl = imageUri.value?.*let* **{** uri **->** val fileRef = storage.child("flyers/$flyerId.jpg")  
 val uploadTask = fileRef.putFile(uri).await()  
 fileRef.*downloadUrl*.await().toString()  
 **}** ?: ""  
  
 val flyer = Flyer(  
 id = flyerId,  
 title = title.value,  
 description = description.value,  
 imageUrl = downloadUrl,  
 createdBy = createdBy,  
 timestamp = System.currentTimeMillis(),  
 bgColor = bgColor.value,  
 fontName = fontName.value  
 )  
  
 firestore.collection("flyers").document(flyerId).set(flyer).await()  
 \_status.value = "ok"  
 } catch (e: Exception) {  
 \_status.value = "Error: ${e.message}"  
 }  
 **}** }  
}