Perfecto, aquí tienes un **resumen claro y estructurado** de todo lo que hemos hecho (y también lo que **aún está pendiente o con errores**) para tu app de eventos llamada **FlashMeet**.

**✅ Objetivo del Proyecto**

Desarrollar una app moderna de eventos usando el **stack recomendado para Android**:

* **Jetpack Compose** para UI
* **Material Design 3**
* **Hilt** para DI
* **Retrofit** para APIs
* **Room / DataStore** para persistencia
* **Firebase** para autenticación y base de datos
* **Coroutines + Flow** para asincronía
* **CameraX** y otras APIs modernas

**🧱 Estructura del Proyecto**

Actualmente, el proyecto está dividido en módulos:

FlashMeet/

├── app/ → Módulo principal Android (UI y navegación)

├── core/ → Módulo de lógica o recursos compartidos

├── data/ → (posible módulo futuro) para repositorios, modelos, etc.

├── build.gradle.kts

├── settings.gradle.kts

├── gradle/libs.versions.toml → Usamos Version Catalog

**✅ Acciones Realizadas**

**1. Configuración del libs.versions.toml**

* Se definieron versiones para todas las librerías clave del stack moderno.
* Se agregaron plugins y libraries para Compose, Hilt, Retrofit, Firebase, etc.

🔧 *Aún tuvimos errores de referencias no resueltas en algunos alias.*

**2. Build Scripts**

* Se creó y ajustó el build.gradle.kts del proyecto raíz.
* Se añadieron plugins con alias(libs.plugins.xxx) utilizando Version Catalog.
* Se configuró correctamente el namespace, minSdk, targetSdk, compileSdk, packaging, etc.
* Se usó correctamente kotlinCompilerExtensionVersion.

🔧 *Varios errores en Unresolved reference relacionados con plugins, principalmente por malas referencias o falta de sincronización.*

**3. App module (app/build.gradle.kts)**

* Se integraron dependencias como compose-ui, material3, coil, navigation-compose, firebase-bom, etc.
* Se integró Hilt correctamente (aunque con advertencias por la versión o configuraciones duplicadas).
* Se usaron múltiples versiones para Firebase (algunas redundantes, otras obsoletas).

🔧 *Hay errores por dependencias duplicadas, plugins no aplicados correctamente o configuraciones no compatibles entre sí (por ejemplo, Compose + Kotlin 2.0).*

**4. Problemas Detectados**

* **Errores con Unresolved reference a Version Catalog** en varios libs.xxx.
* **Problemas con los plugins alias(...)** porque no se encontraban definidos correctamente en el .toml.
* **Errores de compilación** por viewModel, AndroidView, GoogleLoginButton, etc. no importados o definidos.
* **Conflictos de versiones** entre Kotlin, Compose Compiler y AGP.
* **Falta de dependencias importantes**, por ejemplo:
  + com.google.guava:guava para ListenableFuture
  + androidx.camera:camera-\*
  + lifecycle-viewmodel-compose
* **Módulo core sin tipo definido** (ni Android library ni Kotlin module).
* **Ausencia de AndroidManifest.xml en módulos Android Library**.

**5. Utilidades y comandos**

* Te mostré cómo obtener el **SHA-1** con ./gradlew signingReport.

**📝 Pendientes y Siguientes Pasos**

**🔁 Revisión y limpieza**

* Revisar y limpiar el archivo libs.versions.toml
* Asegurarse de que todas las referencias libs.xxx existan en el archivo TOML
* Eliminar duplicados de dependencias como coil, firebase-bom, material3, etc.

**⚙️ Configuración de módulos**

* Asegurar que core/ sea un módulo válido (Android library o Kotlin)
* Agregar AndroidManifest.xml en core/src/main/ si es módulo Android
* Verificar todos los build.gradle.kts para que estén sincronizados

**🛠️ Corrección de errores de código**

* Importar correctamente viewModel, HomeScreen, GoogleLoginButton, etc.
* Agregar dependencias faltantes como CameraX, Guava, LifecycleViewModel, etc.
* Resolver conflicto entre @Preview de Compose y CameraX

**🔐 Firebase y autenticación**

* Confirmar que google-services está correctamente aplicado
* SHA-1 ya obtenido, listo para configurar en Firebase Console

**💡 Recomendación General**

Dado que ya tenemos muchos errores acumulados, te recomiendo lo siguiente:

**Opción 1: Limpiar el proyecto**

* Hacer backup de código fuente
* Eliminar todos los build.gradle.kts, settings.gradle.kts, y libs.versions.toml
* Volver a generar desde Android Studio (New Project with Jetpack Compose + Kotlin DSL)
* Reintegrar dependencias con libs.versions.toml correctamente definido

**Opción 2: Corregimos paso a paso**

* Te puedo dar los archivos .toml, build.gradle.kts (raíz y módulos) **100% funcionales**
* Probamos en limpio y adaptamos tu código fuente

**✅ RESUMEN DEL PROGRESO ACTUAL**

**🧱 1. Estructura general y navegación**

* ✅ Implementamos el NavGraph con AnimatedNavHost y transiciones personalizadas (slide + fade).
* ✅ Flujo de navegación completo entre: Splash → Welcome → Onboarding → Login/Register → Home.
* ✅ Organización modular en screens y presentation.

**👋 2. Pantalla Welcome**

* ✅ Pantalla con animación Lottie (welcome\_anim.json).
* ✅ Textos internacionalizados (stringResource(...)).
* ✅ Navegación al onboarding y marcado de estado con OnboardingPrefs.

**🚀 3. Pantalla Onboarding**

* ✅ 3 slides con animaciones Lottie.
* ✅ Navegación con botón "Siguiente" y "Comenzar".
* ✅ Textos internacionalizados.
* ✅ Estado de finalización guardado con OnboardingPrefs.

**🔐 4. Login y Registro**

* ✅ Login con email y Google.
* ✅ Validación local de email y contraseña.
* ✅ Internacionalización completa (es/en/fr).
* ✅ Icono de Google personalizado (R.drawable.ic\_google\_logo).
* ✅ Registro con validación de:
  + Formato de email
  + Contraseña mínima
  + Coincidencia de contraseñas

**🌍 5. Internacionalización (i18n)**

* ✅ Soporte para **español**, **inglés**, y **francés**.
* ✅ Todos los textos importantes están en strings.xml en sus respectivos values-XX.
* ✅ MainActivity aplica LocaleManager desde SharedPreferences.
* ✅ Componente LocalAppLanguage para acceder al idioma actual.

**⚙️ 6. Pantalla de Configuración**

* ✅ Opción para cambiar el idioma desde la app.
* ✅ Recrea la actividad para aplicar cambios.
* ✅ Se reemplazó el selector de radio buttons por un menú desplegable (DropdownMenu).
* ✅ SettingsScreen totalmente integrada.

**💡 7. Tema Claro / Oscuro (En proceso)**

* 🟡 Próximo paso: agregar opción para cambiar el tema (Light / Dark / Auto).
* 🟡 Se detectará desde ViewModel + DataStore, con soporte de recomposición global.

**🔐 8. Firebase + Stripe**

* ✅ FirebaseAppCheck y PlayIntegrityAppCheckProviderFactory configurados en MainActivity.
* ✅ Stripe inicializado con clave pública para pagos.

**📦 Estado general: Muy avanzado y limpio**

* La base es sólida: escalable, moderna y profesional.
* Sigues buenas prácticas: Hilt, State, separación de responsabilidades.
* Internacionalización ✅
* UI amigable y pulida ✅
* Lottie y animaciones ✅
* Google Sign-In ✅

**🧱 1. Pantalla de creación de flyers (CreateFlyerScreen)**

Creamos una UI con Jetpack Compose que permite al usuario:

* Ingresar título y descripción.
* Seleccionar imagen desde cámara o galería.
* Elegir color de fondo y tipo de fuente.
* Agregar texto destacado.
* Seleccionar una opción de publicidad (AdOption).
* Visualizar una **vista previa en tiempo real** del flyer.
* Mostrar el **costo estimado** según la opción publicitaria elegida.
* Seleccionar el método de pago (por ahora solo mostramos botones).

**📊 2. Estimación de costo**

* Implementamos lógica en CreateFlyerViewModel.kt para calcular el costo con base en AdOption seleccionado.
* Mostramos un **badge llamativo** con el costo estimado dentro de la pantalla.

**💳 3. Integración con Stripe (pago en línea)**

**Backend simulado:**

* Simulamos un backend que devuelve un clientSecret para el PaymentIntent.

**StripePaymentScreen:**

* Creamos una pantalla llamada StripePaymentScreen.kt.
* Se integró Stripe Payment Sheet con:
  + rememberPaymentSheetLauncher
  + PaymentConfiguration.init(...)
  + PaymentSheet.Configuration(...)
* Mostramos el estado del pago (✅ exitoso, ❌ error, ⚠️ cancelado).

**🧭 4. Navegación**

* Registramos la pantalla de pagos en AppNavGraph.
* Creamos constante Routes.PAYMENT.
* Listos para navegar desde CreateFlyerScreen hacia StripePaymentScreen según la opción de publicidad.

**🚧 5. Próximos pasos sugeridos**

1. ✅ **Conectar navegación al flujo de pago:**  
   Cuando el usuario elija una opción que requiere pago (BANNER, PROMOTED, etc.), navegar a Routes.PAYMENT.
2. 🧪 **Validar flujo completo:**
   * Crear flyer → seleccionar opción con pago → pantalla Stripe → pagar → confirmar éxito.
3. 🌐 **Reemplazar backend simulado:**  
   Llamar a un endpoint real (por ejemplo, usando Firebase Cloud Functions o tu servidor) para crear PaymentIntent.
4. 🧾 **Agregar soporte para PayPal u otros métodos** (opcional después de Stripe).

**✅ Resumen de avances en FlashMeet (hasta ahora)**

**1. Pantalla de mapa con eventos cercanos**

* Integramos GoogleMap usando maps-compose.
* Mostramos la ubicación del usuario (FusedLocationProviderClient).
* Añadimos marcadores personalizados (🟧 naranja) para eventos cargados desde Firestore.
* Cada marcador navega a la pantalla de detalle del evento al hacer click.
* Activamos isMyLocationEnabled y myLocationButtonEnabled.

**2. FABs funcionales (Speed Dial)**

* Implementamos un FAB principal tipo menú (☰).
* Al expandirse, muestra:
  + 🔍 FAB para navegar a búsqueda de eventos.
  + ➕ FAB para crear un nuevo evento.
* Corrigimos una superposición de capas que ocultaba los FABs por un GoogleMap duplicado ✅.
* Reposicionamos el FAB para que no se solape con el botón de “mi ubicación”.

**3. Pantalla de detalle de evento**

* Visualiza:
  + Imagen del flyer (Coil con rememberAsyncImagePainter)
  + Título, descripción, ubicación, tipo de promoción y texto destacado.
* Se añadió un botón **"Editar"** si el evento pertenece al usuario actual.
* Este botón redirige a EditEventScreen con el eventId como argumento.

**4. Pantalla para editar evento (EditEventScreen)**

* Carga el evento desde Firestore al entrar (por ID).
* Permite editar título, descripción, promoción, texto destacado, imagen y ubicación.
* Usa EditEventViewModel para:
  + Obtener evento desde repositorio.
  + Actualizarlo y guardar cambios.
* Flujo funcional de navegación: Detalle ➡️ Editar ➡️ Volver.

**5. Navegación organizada**

* Se definieron rutas nuevas en Routes.kt, como:
  + edit\_event/{eventId}
  + event\_detail/{eventId}
* En AppNavGraph.kt agregamos los composable() correspondientes.
* Se incluyó fun editEventWithId(eventId: String) para construir rutas limpias y reutilizables.

**6. Revisión de permisos y errores**

* Detectamos problemas de permisos de ubicación y los gestionamos con LocationPermissionHandler.
* Agregamos logs (Log.d, Log.e) para diagnósticos más claros.
* Identificamos errores derivados del doble GoogleMap y de permisos faltantes (SecurityException).

**🧩 ¿Qué está funcionando ahora?**

| **Módulo** | **Estado** |
| --- | --- |
| Mapa con eventos | ✅ OK |
| FABs funcionales | ✅ OK |
| Detalle de evento | ✅ OK |
| Edición de evento | ✅ OK |
| Navegación entre pantallas | ✅ OK |

**📌 ¿Qué podemos hacer después?**

1. **Animaciones para los FABs**
2. **Filtrado de eventos por categoría o distancia**
3. **Mostrar lista de eventos junto al mapa**
4. **Eventos destacados (resaltados visualmente o primero en lista/mapa)**
5. **Subida real de imágenes a Firebase Storage**

**✅ OBJETIVO GENERAL**

**Modernizar y estabilizar la navegación completa de tu app**, asegurando:

* Transiciones animadas ✨
* Flujo fluido de autenticación 🧭
* Rutas organizadas correctamente 🗂️
* Eliminación de crashes y errores de navegación ❌
* Código limpio y modular siguiendo buenas prácticas 💡

**🔧 CAMBIOS Y MEJORAS HECHAS**

**🧩 1. Refactor de AuthNavGraph**

* Lo convertimos en una **función de extensión** de NavGraphBuilder ✅
* Se integró dentro del AppNavGraph usando navigation() y no como un composable suelto (💥 causa del crash original).

**🎬 2. Animaciones entre pantallas**

* Se aplicaron transiciones:
  + slideInHorizontally / slideOutHorizontally
  + fadeIn / fadeOut
* Animaciones suaves en:
  + SplashScreen
  + LoginScreen
  + RegisterScreen

**🌊 3. SplashScreen moderno**

* Añadimos:
  + AnimatedVisibility
  + scaleOut + fadeOut
* Validamos onboarding y login:
* when {
* !hasSeenOnboarding -> navController.navigate(Routes.ONBOARDING)
* currentUser == null -> navController.navigate(Routes.LOGIN)
* else -> navController.navigate(Routes.HOME)
* }

**🔐 4. Login y Registro con transiciones**

* Se mejoró LoginScreen y RegisterScreen con:
  + Animaciones de entrada/salida
  + Validaciones de campos
  + Navegación segura con popUpTo(0) para evitar crashes 🔥

**⚙️ 5. Revisión completa de AppNavGraph**

* Aseguramos que todas las rutas estaban registradas correctamente
* Eliminamos composables huérfanos o mal ubicados
* Se verificó que Routes.HOME, Routes.AUTH, etc. estén definidos en el NavHost

**💣 6. Crash Fix: "navigation destination splash is not a direct child..."**

* Solucionado asegurando que startDestination = Routes.SPLASH
* Verificamos que Routes.SPLASH esté correctamente dentro del NavHost

**📄 7. MainActivity.kt final optimizado**

* Usamos rememberNavController() solo una vez
* Definimos startDestination correctamente
* Proporcionamos el idioma con CompositionLocalProvider
* Añadimos botón de prueba de crash Crashlytics (opcional)

**📈 8. 🎨 Diagrama visual generado**

* Creamos un **diagrama de flujo** visual de navegación actual de tu app (pantallas principales y flujo Auth incluidos)

**📌 Resultado**

Tu app ahora tiene:

* Un flujo de navegación robusto y modular 🔐
* Animaciones modernas ✨
* Validaciones bien estructuradas ✅
* Navegación sin crashes 🛡️
* Un código más limpio y mantenible 🧼

**🔹 1. Modelo y lógica de eventos**

* Creamos/ajustamos el modelo Event con:
  + Datos base: id, title, description, timestamp, etc.
  + Ubicación (latitude, longitude, city, country, etc.).
  + Soporte para **categorías como enum (EventCategory)**.
  + Campos para publicidad: adOption, highlightedText.
* Helpers:
  + toFirestoreString() para guardar categoría en Firestore.
  + toEventCategory() para mapear strings a enums.

**🔹 2. Filtros y UI**

* EventCategoryFilterChips.kt → chips horizontales para filtrar eventos por categoría.
* MapSearchBar.kt → barra de búsqueda con Google Places + sugerencias de **eventos en Firestore**.
* EventDetailBottomSheet.kt → hoja inferior con detalle del evento + botón “Navegar en Google Maps”.
* Fix: quitamos un **toggle sobrante de “eventos destacados”** que causaba errores de parámetros.

**🔹 3. HomeViewModel + HomeScreen**

* HomeViewModel maneja:
  + Ubicación del usuario (FusedLocationProviderClient).
  + Carga de eventos cercanos (eventRepository.getNearbyEvents).
  + Filtros por categoría.
  + **Alertas en tiempo real** → vibración + notificación local cuando aparecen eventos nuevos.
* HomeScreen:
  + Muestra el mapa con GoogleMap.
  + Renderiza marcadores con colores distintos si son **destacados**.
  + FAB tipo radar para acciones rápidas.
  + BottomSheet para detalles del evento.
  + Animaciones de radar (ondas expansivas) cuando hay alertas o evento seleccionado.

**🔹 4. Pagos (Stripe)**

* Creamos StripePaymentManager.kt:
  + init() con publishable key.
  + simulateTestPayment() para pruebas rápidas.
  + confirmRealPayment() para pagos reales.
  + createTestCard() para usar la tarjeta 4242.
* Creamos PaymentSummaryScreen:
  + Muestra resumen del evento pendiente (PendingEventData).
  + Simula pago con Stripe en **modo test**.
  + Maneja estados: procesando, éxito, error.
* Plan: conectarlo con Firebase Functions para crear **PaymentIntents** de verdad.

**🔹 5. Integración y estados compartidos**

* SharedEventStateViewModel guarda temporalmente los eventos que necesitan pago.
* Al guardar un evento con **adOption distinto de NONE**, redirige a PaymentSummaryScreen para procesar.

**🔹 6. Errores corregidos**

* ignoreCase mal usado en comparación de enums → arreglado usando .name.equals(..., ignoreCase = true).
* Duplicados de navigateWithGoogleMaps → detectamos que había dos archivos (MapUtils.kt y MapsUtils.kt) → **queda solo uno**.
* Parametros showOnlyHighlighted y onHighlightedToggle eliminados de EventDetailBottomSheet y actualizados en HomeScreen.
* Importaciones rojas corregidas (color, icon, toCategoryEnum ya resueltos en EventCategory).
* Limpieza de funciones duplicadas con **Build > Clean > Rebuild**.

✅ Con esto ya tenemos:

* **Mapa interactivo** con búsqueda, filtros y alertas.
* **Sistema de eventos con publicidad destacada**.
* **Flujo inicial de pagos con Stripe** (simulación lista, real en camino).
* **UI intuitiva** con chips, bottom sheet y radar.

👉 Próximos pasos que habíamos hablado:

1. Terminar **Firebase Functions** para crear PaymentIntent con la secret key segura.
2. Conectar PaymentSummaryScreen a esa función en lugar de usar un clientSecret falso.
3. Crear la pantalla **“Mis Eventos”** para ver si un evento está pagado o pendiente.
4. Pulir detalles de UI (búsqueda más compacta, banners de anuncios, etc.).

# Pagos (Stripe)

* Elegimos **Opción A (tu backend propio)** para crear PaymentIntents.
* Definimos PaymentApi + PaymentViewModel para pedir el **clientSecret**.
* Mantuvimos StripePaymentManager (init/confirm/simulación) y limpiamos firmas para que no choquen.
* Solucionamos conflictos de clases duplicadas y de DI (Hilt).

# DI / Hilt

* Eliminamos bindings duplicados (Retrofit/OkHttp/PaymentApi) que daban **DuplicateBindings**.
* Dejamos un solo módulo de red estable.

# Home (UX del mapa)

* Integramos **clustering** con DefaultClusterRenderer custom:
  + Marcadores con **badge ❤️** si “me interesa”.
  + **Clúster** con contador + badge de interesados.
* Añadimos **ClusterPreviewBar** (popover abajo) con “Acercar / Ver lista”.
* **EventListSheet**: lista ordenable (Distancia/Nombre/Fecha) + filtro “Solo destacados”.
* **EventPeekHandle** (“Eventos cerca (N)”) para abrir la lista rápido.
* **FeaturedStoriesRow** (carrusel “historias” de eventos destacados).
* Botón **centrar mapa** separado del FAB de acciones; la **search bar ya no tapa**.
* **Tema oscuro/claro del mapa** con toggle persistente (VM + DataStore).
* Animaciones de **radar/sonar** para feedback visual.
* CTA “**Cómo llegar**” (Google Maps) en cards/detalle.
* Mensaje elegante si **no hay permisos** de ubicación.

# Intereses del usuario

* Nuevo EventInterestsViewModel (Firestore: users/{uid}/interests) para **marcar/desmarcar** y **escuchar** intereses en tiempo real.
* Icons y badges sincronizados con ese estado.

# Navegación & Deep Links

* Rutas nuevas: HOME\_FOCUS (+ helpers para navegar con focusId/lat/lon).
* EventDeepLinkScreen: carga por eventId y **redirige** a Home centrando mapa + abriendo detalle.
* AppNavGraph: deep links (https://flashmeet.app/e/{id}, /event/{id}, flashmeet://event/{id}).
* **Manifest** actualizado con intent-filters (https + **custom scheme**).

# Manifest / Permisos

* Limpiamos permisos y metadatos (Maps, FCM, Wallet).
* FileProvider correcto.
* Cámara no requerida.
* Añadimos **VIBRATE**.

# Errores y crasheos solucionados

* SecurityException por isMyLocationEnabled → ahora se **habilita solo con permiso**.
* MapEffect fuera del GoogleMap → **reubicado** para evitar UiApplier → MapApplier.
* Colores (parseColor, WHITE) y otros imports corregidos.
* EventDetailBottomSheet ahora recibe SheetState y soporta **interés** + **distanceLabel**.
* buildFeatures { buildConfig true } para el error de BuildConfig.

# Próximos pasos recomendados

1. **App Links reales (https)**: subir /.well-known/assetlinks.json con tu **SHA-256** para que Chrome abra directo la app con https://flashmeet.app/event/<id>.
2. Ajustar el baseUrl real en tu módulo de red para el backend de Stripe.
3. (Opcional) Fallback: si llega un deeplink de un evento que no está en memoria, **cargarlo on-the-fly** y “inyectarlo” temporalmente para mostrar el detalle.
4. Pulir assets: map\_dark.json, iconos vectoriales, y medir rendimiento de bitmaps de marcadores en listas grandes.

# Resumen express de todo lo que montamos 💪

## Dominio, DNS y hosting

* Elegiste **123myway.com** (DNS en **Cloudflare**).
* Apuntamos el dominio a **Vercel**:
  + A @ → 76.76.21.21 y CNAME www → cname.vercel-dns.com (DNS only).
* Servimos /.well-known/assetlinks.json desde Vercel (apex y www) → **200 OK**.

## Repos y deploy

* Repo **GitHub**: paparatsi40/flashmeet-web con:
  + public/.well-known/assetlinks.json
* Proyecto **Vercel**: flashmeet-web con dominios:
  + flashmeet-web.vercel.app (staging), 123myway.com y www.123myway.com (prod).

## Android App Links & Deeplinks

* **Manifest**:
  + <intent-filter android:autoVerify="true"> para https://123myway.com y https://www.123myway.com en rutas /event y /e.
  + Fallback: flashmeet://event/{id} (siempre abre la app).
* **assetlinks.json**:
  + Incluye SHA-256 **debug** (ya verificada).
  + Dejamos slots para **release** y **Play App Signing**.
* Verificación con ADB:
  + pm verify-app-links y pm get-app-links → **verified** en tus dominios.

## Navegación y pantallas

* **AppNavGraph**:
  + Deep links para event/{eventId} con **query params** opcionales ?lat&lon.
* **EventDeepLinkScreen**:
  + Carga evento por eventId, prioriza lat/lon del link y navega a **HOME\_FOCUS** con foco.
* **Routes helpers**:
  + homeWithFocus(eventId?, lat?, lon?) → construye rutas limpias (sin null, con . en decimales).

## Compartir enlaces (growth)

* **DeeplinkBuilder**:
  + Genera URLs: https://{host}/event/{id} o /e/{id} + lat/lon + UTM opcionales.
  + shareEvent(context, ...) abre el chooser nativo.
* **EventDetailScreen**:
  + Botones **Compartir** y **Copiar link** con Snackbar.

## Notificaciones (FCM)

* **FlashMessagingService**:
  + Recibe data-only: {type=event, eventId, lat, lon, title, body}.
  + Notificación con PendingIntent principal **https** (App Links) y acción **custom scheme** de fallback.
* **FlashMeetApp**:
  + Loguea el **FCM device token** en onCreate() → adb logcat -s FCM.
* **Sender de pruebas** (Node):
  + tools/fcm-sender/send.js con **firebase-admin** (key privada en .gitignore).
  + Envío:
  + $env:FCM\_TOKEN='...'
  + node .\send.js
* Abrir ajustes de notificaciones:
  + adb shell am start -a android.settings.APP\_NOTIFICATION\_SETTINGS --es android.provider.extra.APP\_PACKAGE com.carlitoswy.flashmeet

## Qué falta (checklist corto) ✅

* Obtener **SHA-256 release** (keystore) y **Play App Signing** (Play Console).
* Actualizar assetlinks.json con esas huellas y **deploy** en Vercel.
* (Opcional) Quitar flashmeet-web.vercel.app del Manifest al pasar a prod.
* (Android 13+) Pedir POST\_NOTIFICATIONS en onboarding si aún no lo haces.

Claro. Aquí tienes un **resumen completo y estructurado** de lo que se ha hecho hasta ahora en tu proyecto:

**✅ 1. Dominio y Hosting**

* **Dominio adquirido**: 123myway.com
* **DNS gestionado en**: Cloudflare
* **Hosting actual**: Firebase Hosting (funcionando con firebase deploy)
* **Directorio público configurado**: public\_html
* **Archivos clave presentes**:
  + index.html (principal)
  + privacy.html (para cumplir con políticas de Google Play)
  + 404.html (opcional para errores personalizados)

**✅ 2. Enlaces y Navegación (Deeplinks + App Links)**

* **App Links configurados en AndroidManifest**:
  + Hosts: 123myway.com, www.123myway.com
  + Rutas soportadas: /event, /e
  + Filtro android:autoVerify="true" habilitado
* **Fallback scheme personalizado**: flashmeet://event/...
* **Pantalla puente** (EventDeepLinkScreen) que traduce deeplinks a navegación interna.
* **Helper** Routes.homeWithFocus(...) usado para centrar mapa al abrir por deeplink.

**✅ 3. Firebase & Notificaciones**

* **Firebase configurado** (proyecto: flashmeet-72bbf)
* **Firebase Cloud Messaging (FCM)** integrado:
  + FlashMessagingService registrado en AndroidManifest
  + Pruebas funcionando con tools/fcm-sender (script en Node.js)

**✅ 4. assetlinks.json y Verificación de Dominio**

* **assetlinks.json** alojado correctamente en:
* https://123myway.com/.well-known/assetlinks.json
* Contiene:
  + Nombre del paquete: com.carlitoswy.flashmeet
  + SHA-256 del certificado de desarrollo
  + SHA-256 del certificado de Play Store (Google App Signing)
* Resultado: App Links ya **verificados y funcionales** en dispositivos Android.

**✅ 5. Android App Bundle (AAB) y Play Console**

* .aab generado correctamente desde Android Studio.
* Subido a Google Play Console (versión interna / pruebas).
* Se cumplió con el requerimiento de privacidad:
  + **Política de privacidad alojada en**: https://123myway.com/privacy.html
  + Declarada en la sección:
    - **Play Console > App content > Privacy policy**

**✅ 6. Otros avances técnicos**

* Se migró de dominio anterior (flashmeet.com) a 123myway.com.
* Eliminación de entradas innecesarias del AndroidManifest (como staging).
* Soporte multi-idioma iniciado.
* Preparación para multiplataforma (iOS + web) considerada desde la arquitectura.

**🧭 Estado general: Etapa de desarrollo avanzada**

Estás muy bien encaminado. Tienes una app que:

* Está **conectada con el mundo web** vía enlaces seguros.
* Puede ser **compartida por links, notificaciones, y QR** sin fricción.
* Cumple requisitos técnicos de Play Store (política de privacidad, firma, permisos).
* Está **lista para pruebas internas o cerradas** en Google Play.