# Informe integral — Mi Partido

**Fecha:** 23 de septiembre de 2025
**Autor:** Equipo Mi Partido (con apoyo de asesoría técnica)
**Contexto:** Kotlin Multiplatform + Compose Multiplatform + Firebase (Auth, Firestore, Storage, Functions/Tasks)

---

## 0) Resumen ejecutivo

**Mi Partido** es una plataforma para fútbol amateur que conecta equipos cercanos, facilita coordinar partidos y, en el futuro, permitirá reservar canchas, gestionar ligas/torneos y llevar estadísticas. La app se construye con **Kotlin Multiplatform (KMP)** para compartir lógica entre Android e iOS, **Compose Multiplatform** para UI declarativa, y **Firebase** como backend administrado. El MVP prioriza:

1. **Onboarding + Autenticación** (Google, Apple, Facebook opcional, email/contraseña con verificación y reset in-app).
2. **Creación/gestión de perfil y equipo** (rol de capitán, logo, ubicación).
3. **Geolocalización + geohash** para descubrir equipos cercanos y **matching** para coordinar partidos.
4. **Chat/confirmaciones básicas de partido** y **agenda** mínima (sin pagos ni reservas de cancha en el MVP).

**Diferenciadores**: experiencia fluida (UI moderna con bottom bar flotante y swipe entre tabs), algoritmo de emparejamiento geográfico + preferencias, y una identidad visual carismática (mascota cabra) con potencial de **Rive** para micro-interacciones.

**Riesgos principales**: cumplimiento legal (datos personales/menores), precisión de permisos de ubicación, costos y escalabilidad de consultas geoespaciales, fricción de onboarding.

---

## 1) Propuesta de valor y mapa de funcionalidades

### 1.1 Propuesta de valor

- **Para equipos amateur**: encontrar rivales del mismo nivel cerca, coordinar horarios y confirmar asistencia sin caos en WhatsApp.
- **Para admins de canchas (futuro)**: tablero web/desktop con reservas, ingresos/gastos, disponibilidad.
- **Para ligas/torneos (futuro)**: inscripción, fixture, mesa de control, estadísticas.

### 1.2 Funcionalidades (MVP → Roadmap)

| Área | MVP | Roadmap 1 | Roadmap 2 |
|---|---|---|---|
| Auth | Google, Apple, Email/Pass (verificación + reset in-app), opcional Facebook | 2FA opcional | Inicio social ampliado (X, Instagram) |

| Área | MVP | Roadmap 1 | Roadmap 2 |
|---|---|---|---|
| Perfil | Usuario, Equipo (logo, colores, nivel, radio de juego), geohash | Stats básicas (PJ, G, E, P) | Historial completo + badges |
| Descubrir | Lista de equipos cercanos (geohash) | Filtros (nivel, disponibilidad) | Recomendaciones ML |
| Matching | Propuesta/aceptación de partido, chat mínimo | Calendario/agenda; recordatorios push | Reservas integradas con canchas |
| Canchas | — | Directorio básico + ratings | Integración de reservas + pagos |
| UX/UI | Bottom bar flotante, swipe/ pager, predictive back | Animaciones Rive (mascota) | Temas dinámicos, accesibilidad AA |
| Observabilidad | Firebase Analytics, Crashlytics | Evento de funil completo | A/B testing (Remote Config) |

# 2) Arquitectura técnica (KMP/Compose/Firebase)

## 2.1 Capas y módulos (Clean Architecture)

**Capas:**
- **domain**: entidades, use-cases, contratos de repositorio.
- **data**: fuentes (Firebase, local), mappers, DTOs, caché, error mappers.
- **presentation**: ViewModels (MVI/MVVM), estados (sealed), UI Compose, navegación.
- **platform**: expect/actual, servicios nativos (ubicación, Google/Apple/Facebook Sign-In, notificaciones).

**Módulos** (sugeridos):
- `:core:domain`, `:core:data`, `:core:platform`
- `:feature:auth`, `:feature:profile`, `:feature:teams`, `:feature:discover`, `:feature:match`, `:feature:` (opcional), `:feature:home`
- `:app:composeApp` (entry multiplaforma)

## 2.2 Dependencias fijadas (recomendadas)

- Kotlin **2.0.21**
- Compose Multiplatform **1.8.2**
- Navigation Compose KMP **2.9.0-beta05**
- Koin core **3.5.6** / Koin compose **1.1.5**
- kotlinx-serialization **1.7.3**
- activity-compose (Android) **1.9.x**

Mantener versiones en un `libs.versions.toml` para control centralizado.

## 2.3 Navegación y UX avanzada

- **Bottom bar flotante** con contenedor transparente, safe insets, indicador custom.

- **HorizontalPager** para swipe entre tabs (Home / Discover / Matches / Profile).
- **Predictive Back**:
- **Android**: Navigation Compose 2.8+ integra back predictivo; configurar `onBackPressedDispatcher` y `PredictiveBackHandler` en pantallas con gestos.
- **iOS**: integrar con `UINavigationController` (interactive pop). Compose MPP permite hookear eventos; respetar jerarquía para "swipe to go back".

### 2.4 Servicios nativos (expect/actual)

- **LocationService**: permisos, accuracy, single vs continuous updates.
- **AuthService**: Google/Apple/Facebook; manejo de credenciales y tokens; fallback email/contraseña.
- **Notifications**: topic "match-proposals-{teamId}".

---

## 3) Módulo `data` en detalle + `ViewModel`

Estilo profesor + senior: teoría → ejemplo → ejercicio → solución.

### 3.1 Explicación teórica

- **Repository pattern**: aísla dominio de fuentes de datos (Firebase, cache local).
- **DTO ↔ Entity mappers**: transforman formatos remotos a entidades de dominio; **nunca** filtrar lógica de negocio en DTOs.
- **Error mappers**: traducen excepciones de Firebase a errores de dominio (ej.: `AuthError.InvalidCredentials`).
- **Wrappers**: resultados tipados (p. ej., `Either<Failure, T>` o `Result<T>`), con metadata (retryable, code).
- **Interfaces**: contratos claros (`AuthRepository`, `TeamRepository`, `MatchRepository`).

**ViewModel (MVVM/MVI)**:
- Mantiene **State** inmutable (sealed data class), expone **Events** (UI → VM) y **Effects** (one-shot: toasts, navegación).
- **Corrutinas**: scope de VM, `StateFlow` / `MutableStateFlow`.
- **Reducción de estado**: funciones puras `reduce(old, action)` para trazabilidad.

### 3.2 Ejemplo práctico (simplificado)

```
// domain
sealed interface AuthError { object Network: AuthError; object
InvalidCredentials: AuthError }

data class User(val id: String, val email: String?, val displayName: String?)

interface AuthRepository { suspend fun signInWithGoogle(idToken: String):
Result<User> }

class SignInWithGoogleUseCase(private val repo: AuthRepository) {
  suspend operator fun invoke(idToken: String) =
repo.signInWithGoogle(idToken)
```

```
}

// data
data class UserDto(val uid: String, val email: String?, val name: String?)
fun UserDto.toDomain() = User(uid, email, name)

class FirebaseAuthRepository(/* deps */): AuthRepository {
  override suspend fun signInWithGoogle(idToken: String): Result<User> = try
{
    // firebase auth logic here → returns UserDto
    val dto = /* ... */ UserDto("uid","mail","name")
    Result.success(dto.toDomain())
  } catch (t: Throwable) {
    Result.failure(mapAuthError(t))
  }
}

// presentation
sealed interface AuthEvent { data class GoogleToken(val idToken: String):
AuthEvent }

data class AuthState(val isLoading: Boolean = false, val user: User? = null,
val error: AuthError? = null)

class AuthViewModel(private val signIn: SignInWithGoogleUseCase):
ViewModel() {
  private val _state = MutableStateFlow(AuthState())
  val state: StateFlow<AuthState> = _state

  fun onEvent(e: AuthEvent) = when(e) {
    is AuthEvent.GoogleToken -> launch { login(e.idToken) }
  }

  private fun launch(block: suspend () -> Unit) = viewModelScope.launch {
block() }

  private suspend fun login(idToken: String) {
    _state.update { it.copy(isLoading = true, error = null) }
    when (val r = signIn(idToken)) {
      is Result.Success -> _state.update { it.copy(isLoading = false, user =
r.getOrNull()) }
      else -> _state.update { it.copy(isLoading = false, error =
AuthError.InvalidCredentials) }
    }
  }
}
```

### 3.3 Ejercicio propuesto

- Agrega `ErrorMapper` por proveedor (Google/Apple/Facebook/Email) y tests de `mapAuthError(Throwable)`.

- Implementa un `AuthLocalDataSource` para caché de sesión (Keychain/Keystore).

### 3.4 Solución (pistas)

- Usa una sealed class por código de error; agrupa por **retryable** vs **fatal**.
- Mockea fuentes con `kotlinx.coroutines-test` y `turbine` para flows.

---

## 4) Geolocalización y geohash

### 4.1 Diseño

- **Geohash** por **equipo** (y opcional por usuario si deseas granularidad para "pickup games").
- Guardar `{lat, lng, geohash, updatedAt}` en el documento del equipo.
- Índices por `geohash` y por prefijo (consulta por *range* de prefijos).

### 4.2 Matching (MVP)

1) Usuario fija **radio** y **nivel** aproximado.
2) Query por **prefijos de geohash** compatibles (ej.: 5–7 chars).
3) Filtrar por nivel/disponibilidad/horario.
4) Enviar **propuesta de partido** con ventana de respuesta + notificaciones.

### 4.3 Cloud Functions/Tasks (borrador)

- **Function** `proposeMatch(teamA, constraints)` guarda propuesta y dispara notificaciones a `teamB`.
- **Task** de expiración (p. ej., 2h) que marca `expired` si no hay respuesta.
- **Function** `acceptMatch(proposalId)` cierra la negociación y crea `match`.

---

## 5) Modelo de datos (Firestore/Auth/Storage)

### 5.1 Esquema principal (Firestore)

```
/users/{uid}
  displayName: string
  email: string
  photoUrl: string
  createdAt: timestamp
  teamId: string? // referencia
  roles: { captain: bool, admin: bool }

/teams/{teamId}
  name: string
  logoPath: string (Storage)
  level: int (1=casual .. 5=competitivo)
  homeLocation: { lat: double, lng: double, geohash: string }
  radiusKm: number
  members: array<uid>
```

```
  createdAt: timestamp

/matches/{matchId}
  teamAId: ref
  teamBId: ref
  status: string (proposed|accepted|played|canceled|expired)
  when: timestamp
  where: { name: string?, lat: double?, lng: double? }
  chatId: ref? // opcional

/proposals/{proposalId}
  fromTeamId: ref
  toTeamId: ref
  constraints: { dateRange, levelRange }
  status: string (sent|accepted|declined|expired)
  ttl: timestamp // para Cloud Task
```

### 5.2 Reglas de seguridad (borrador)

- **users**: un usuario puede leer su propio doc y los de miembros de su equipo (mínimos campos públicos), escribir sólo el propio.
- **teams**: sólo **capitán/admin** puede editar; lectura pública limitada (nombre, nivel, zona).
- **proposals/matches**: lectura restringida a equipos involucrados.
- **storage**: `teams/{teamId}/logo.png` con validación de rol.

  Implementar **custom claims** (captain/admin) desde Functions para simplificar reglas.

---

## 6) Autenticación y flujos in-app (incluye email verificación/reset)

### 6.1 Proveedores

- **Google** (Android/iOS), **Apple** (iOS), **Email/Contraseña**; **Facebook** opcional.
- Gestionar tokens en `AuthService` (expect/actual) + repositorio.

### 6.2 Email verification & password reset in-app

- Usar **Email Action Handler** de Firebase con **deep links** propios (Android App Links / iOS Universal Links).
- Interceptar `intent/url` en app → `Auth.applyActionCode(oobCode)` → feedback UI.
- **Ventajas**: menor fricción, control de UX, evita saltar a navegador.

### 6.3 Anti-abuso

- **Rate limiting** de envíos de correo, **reCAPTCHA** en WebView fallback.
- Auditar intentos fallidos y bloquear por IP/UID tras N intentos.

---

## 7) UI/UX: lineamientos

- **Paleta**: base oscura/clara con **accent lime** (Paleta B).
- **Mascota cabra**: estilo duolingo-like, amistosa, con camiseta; micronarrativas en onboarding.
- **Rive**: animaciones pequeñas (éxito de match, pull-to-refresh, empty states).
- **Accesibilidad**: contrastes AA, tamaños dinámicos, toques ≥ 44pt, etiquetas para lectores.

---

## 8) Observabilidad, analítica, calidad

- **Firebase Analytics**: eventos de funnel (signup_start/success, team_create, proposal_send, match_accept).
- **Crashlytics** + **Logs estructurados** (tag por feature).
- **A/B testing**: Remote Config para probar variantes de onboarding.
- **Métricas de rendimiento**: cold start, TTI, scroll jank, tamaño APK/IPA.

**QA/Testing**
- **Unit tests** dominio/data, **instrumented** Android, **iOS** via KMP tests + snapshots de UI (opcional).
- **Contract tests** para mappers y reglas de seguridad (emuladores Firebase).
- **Kover** para cobertura, **Detekt/ktlint** para estilo.

---

## 9) CI/CD

- **GitHub Actions**: jobs para `lint → test → build`, matrices `android/ios`.
- **Artefactos**: `.apk` / `.aab`, frameworks iOS (`XCFramework`), símbolos de crash.
- **Distribución interna**: Firebase App Distribution / TestFlight.
- **Versionado**: SemVer + `-beta` para builds de prueba.

---

## 10) Costos y escalabilidad (Firebase)

- **Firestore**: optimizar con **prefijos de geohash** y documentos compactos (evitar subcolecciones hiperactivas).
- **Storage**: imágenes optimizadas (WebP/HEIF), límites de tamaño, borrado de huérfanos.
- **Functions**: programar tareas (TTL proposals), usar regiones cercanas al usuario.
- **Cálculo**: el costo crece con lecturas; cache local + `get()` condicional por `updatedAt`.

  Monitorear en GA4: usuarios activos, tasa de conversión onboarding, frecuencia de propuestas, aceptación, retención D7/D30.

---

## 11) Cumplimiento legal y privacidad (orientativo, no legal)

- **Política de Privacidad** y **Términos** públicos; base legal para tratamiento de ubicación.
- **Menores**: si se incluyen features de control parental o seguimiento, se requieren **consentimientos explícitos** y mecanismos de revocación; revisar ley arg. 25.326 y equivalentes internacionales.
- **Retención**: tiempos claros para datos (ubicación, logs, imágenes).

- **Seguridad**: encriptación en tránsito, protección de credenciales, rotación de claves.

---

## 12) Plan de implementación (8 semanas)

**Semana 1–2**
- Setup KMP/Compose, módulos, DI Koin, navegación base.
- Auth Google/Apple/Email, verificación y reset in-app con deep links.

**Semana 3–4**
- Perfil y equipos, subida de logos (Storage), reglas iniciales.
- LocationService + persistencia de `{lat,lng,geohash}`.

**Semana 5–6**
- Descubrir equipos (query por geohash), UI de filtros.
- Propuestas de partido + notificaciones + expiración (Tasks).

**Semana 7**
- Analítica/end-to-end funnel, Crashlytics, QA de reglas, pruebas en emuladores.

**Semana 8**
- Pulido de UX, accesibilidad, micro-animaciones Rive, preparación de publicación.

---

## 13) KPIs y métricas

| Métrica | Definición | Objetivo MVP |
|---|---|---|
| Conversión onboarding | signup_success / signup_start | ≥ 60% |
| Creación de equipo | usuarios con teamId / usuarios | ≥ 40% |
| Propuestas enviadas/usuario/semana | promedio | ≥ 1.5 |
| Tasa de aceptación | propuestas aceptadas / enviadas | ≥ 35% |
| Retención D7 | % de usuarios activos al día 7 | ≥ 25% |

---

## 14) Riesgos y mitigaciones

- **Ubicación inexacta/permisos denegados** → fallback manual de ubicación, selección de zona.
- **Abuso/Spam de propuestas** → límites por día, reputación de equipo.
- **Costos Firestore** → cache + batch reads, reducir lecturas reactivas, usar `where` por prefijo geohash.
- **Fricción en iOS con proveedores** → guías claras para “Sign in with Apple”, fallback email.
- **Cumplimiento menores** → posponer feature parental al Roadmap 2 hasta contar con asesoría legal.

---

## 15) Checklist de publicación

- [ ] Icono y splash (sin hard gates; recomienda animación ligera).
- [ ] Política de privacidad/TyC hospedados y vinculados en tienda y en app.
- [ ] Reglas Firestore/Storage auditadas con emuladores.
- [ ] Deep links verificados (Android App Links / iOS Universal Links).
- [ ] Pruebas en dispositivos reales con ubicación.
- [ ] Analytics revisada y panel de métricas preparado.

---

## 16) Anexos

### A) Esqueleto de reglas Firestore (orientativo)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    function isSignedIn() { return request.auth != null }
    function uid() { return request.auth.uid }

    match /users/{userId} {
      allow read: if isSignedIn() && (userId == uid());
      allow update, delete: if userId == uid();
      allow create: if isSignedIn();
    }

    match /teams/{teamId} {
      allow read: if true; // limitar campos públicos en el cliente
      allow write: if isSignedIn() && isCaptain(teamId);
    }

    match /proposals/{proposalId} {
      allow read, write: if isSignedIn() && isInvolved(proposalId);
    }

    // helper functions isCaptain, isInvolved resueltas vía custom claims /
refs
  }
}
```

### B) DTOs/Entities (muestra)

```
data class TeamDto(
  val name: String = "",
  val level: Int = 1,
  val homeLocation: GeoDto? = null,
  val radiusKm: Double = 5.0,
  val members: List<String> = emptyList(),
```

```
)

data class Team(
  val id: String,
  val name: String,
  val level: Int,
  val geo: Geo?,
  val radiusKm: Double,
  val members: List<String>,
)

fun TeamDto.toDomain(id: String) = Team(
  id, name, level, homeLocation?.toDomain(), radiusKm, members
)
```

### C) Algoritmo simple de prefijos geohash

1. Codifica lat/lng a geohash.
2. Toma prefijo de 5–7 chars según densidad urbana.
3. Consulta `where(geohash >= low && geohash <= high)` para cada caja de vecinos (8).
4. Mergea resultados, deduplica y filtra por nivel/distancia real (haversine).

### D) Prompts útiles (diseño)

- *"Diseña una cabra mascota estilo flat 2D, divertida y deportiva, con camiseta verde lima, pensada para micro-animaciones en Rive. Exportable a SVG, capas limpias, outline suave."*

---

## 17) Próximos pasos sugeridos

1. Asegurar deep links de verificación/reset **in-app** en Android/iOS.
2. Implementar `LocationService` y escritura de `{lat,lng,geohash}` de equipo.
3. Construir lista Discover con consultas por prefijo de geohash + filtros.
4. Propuestas de partido con expiración y notificaciones.
5. Cerrar el circuito de analítica del funnel.

---

> Este informe consolida visión, arquitectura, datos, seguridad, UX y plan de entrega. Sirve como guía viva: ajústalo sprint a sprint y registra decisiones (ADR) por cambios estructurales.