GameHub — Centro de Juegos Android

Documentación Técnica del Proyecto

Sergio

Febrero 2026

Índice

Visión General

GameHub es una aplicación Android multi-módulo que funciona como un **centro de juegos**. Consta de tres módulos Gradle:

- `app` — El Hub principal (login, perfil, desafíos, puntuaciones).
- `kaisenclicker_module` — Juego *Kaisen Clicker* + la base de datos global.
- `2048_module` — Juego *2048*.

El flujo general es: **Splash Screen** → **Login/Registro** → **Hub principal** (perfil, menú, grid de juegos) → el usuario selecciona un juego y entra a jugarlo.

Módulo App — El Hub

Todo el código se encuentra en `app/src/main/java/com/example/gamehub/`.

Splash Screen y Login

Al abrir la app, `LoginActivity` ejecuta `SplashScreen.installSplashScreen(this)` (biblioteca `androidx.core.splashscreen`), que muestra la pantalla de carga nativa de Android.

A continuación comprueba si hay sesión activa (`SessionManager.isLoggedIn()`). Si la hay, salta al Hub. Si no, muestra un formulario con campos de usuario y contraseña.

Al pulsar **«ENTRAR»**:

1. Valida que los campos no estén vacíos.
2. Llama a `UserRepository.authenticateUser(username, password)`, que busca en la tabla `users` de SQLite comparando el hash SHA-256 de la contraseña.
3. Si es correcto → crea sesión y navega al Hub.
4. Si falla → muestra error y animación *shake* en el botón.

El **registro** (`RegisterActivity`) valida: campos no vacíos, usuario ≥ 3 caracteres, contraseña ≥ 4 caracteres, confirmación coincide, usuario no existe ya. Si todo OK, inserta en la tabla `users` con la contraseña hasheada.

SessionManager

Clase en `app/.../auth/SessionManager.java`. Gestiona la sesión del usuario con `SharedPreferences` (fichero `GameHubSession.xml`).

Datos que almacena:

- `is_logged_in`, `username` — estado de sesión.
- `user_status` — estado del usuario: `."nline"`, `"playing"` o `."way"`.
- `photo_uri` — URI de la foto de perfil seleccionada de la galería.
- `total_points`, `games_played` — puntos totales y partidas jugadas.
- `member_since` — timestamp de registro.
- `game_start_time`, `total_time_played` — tracking de tiempo de juego real (se marca al entrar a un juego y se calcula al volver).

Hub Principal (MainActivity)

Pantalla que muestra:

1. **Header**: logo, título «GameHub», botón logout, bienvenida con nombre del usuario.
2. **Tarjeta de perfil**: foto circular (`ShapeableImageView`), nombre, indicador de estado (punto de color), puntos totales. Click → abre `ProfileActivity`.
3. **Menú** (4 botones): Juegos, Puntuaciones, Desafíos, Perfil.
4. **Grid de juegos**: `RecyclerView` con `GridLayoutManager(2 columnas)`. Cada tarjeta tiene icono circular, nombre y descripción. Al pulsarla: registra inicio de partida, incrementa partidas jugadas y lanza la Activity del juego con `extra_username`.

En `onResume()`, al volver de un juego, llama a `markGameEnded()` para acumular el tiempo jugado y refresca el perfil.

Los juegos se registran en un **Singleton** `GameRepository` mediante objetos `Game` (id, nombre, icono, descripción, Activity destino).

Perfil de Usuario

`ProfileActivity` muestra: foto de perfil (100×100dp circular), nombre, estado (clickable — abre AlertDialog con 3 opciones), puntos, partidas jugadas, tiempo jugado (formato `HH:MM:SS`, medido de verdad), miembro desde.

El botón **«Cambiar foto»** abre la galería con `ActivityResultContracts.GetContent(ïmage/*")`. Se obtiene permiso persistente con `takePersistableUriPermission()` y se guarda la URI en `SessionManager`.

Desafíos

`ChallengesActivity` crea 8 desafíos cuyos datos de progreso se leen de `SessionManager`:

Desafío	Objetivo	Recompensa
Primer Paso	1 partida	50 pts
Jugador Habitual	5 partidas	100 pts
Veterano	10 partidas	200 pts
Primeros Puntos	100 puntos	50 pts
Mil Puntos	1.000 puntos	150 pts
Maestro del GameHub	5.000 puntos	500 pts
Adicto al Juego	25 partidas	300 pts
Explorador	2 juegos distintos	100 pts

Cada tarjeta muestra barra de progreso, texto «X/Y» y badge verde (completado) o gris (pendiente con puntos de recompensa).

Puntuaciones (Leaderboard)

`LeaderboardActivity` tiene 2 tabs:

- **Kaisen Clicker**: lee de `GameDataManager` — nivel del enemigo, clicks totales, daño total, enemigos/bosses derrotados, energía maldita, nivel del personaje, tiempo jugado.
- **2048**: lee de `SqlRepository` vía `kv_store` — puntuación actual, mejor puntuación, movimientos, tiempo.

Historial de Puntuaciones (ScoresListActivity)

Pantalla con:

- Búsqueda por nombre (`LIKE`) y por valor de puntuación con operadores (=, >, <, ≥, ≤).
- 3 botones de ordenación: Nombre, Puntuación, Fecha.
- **Swipe to delete**: deslizar una tarjeta la borra (`ItemTouchHelper`).
- Click en tarjeta → `ScoreDetailActivity` con nombre, juego, puntuación y fecha.

El adapter (`ScoresCursorAdapter`) usa un **Cursor** de SQLite con `RecyclerView` + `CardView`. El método `swapCursor()` cierra el cursor anterior y carga el nuevo.

Módulo Kaisen Clicker

Código en `kaisenclicker_module/src/main/java/com/example/kaisenclicker/`.

Activity Principal

`ui/activities/MainActivity.java`: lee el `extra_username`, crea un `GameDataManager` y carga los datos guardados.

Navegación inferior: 5 botones circulares (`MaterialCardView`):

Pos	Icono	Archivo	Abre
1	Flecha	`ic_arrow_up` (vector)	ShopFragment
2	Cofre	`chest.png` (PNG)	ChestFragment
3	Espadas	`battle_icon.png` (PNG)	CampaignFragment
4	Personaje	`character_menu.png` (PNG)	CharacterInventoryFragment
5	Trofeo	`ic_trophy` (vector)	StatisticsFragment

El botón seleccionado se agranda (60dp → 72dp), cambia borde a dorado y tiene animación de rebote (`OvershootInterpolator`).

Combate (CampaignFragment)

Pantalla principal del juego. Fondo: `shibuya.webp`. Elementos:

- Barra de vida (`HpBarComponent`): degradado dinámico verde/amarillo/rojo.
- Imagen del enemigo: cambia según el nivel (normal, boss, fases).
- Popup de daño: texto rojo flotante con fade-out.
- Display de energía maldita (esquina superior derecha).
- 4 botones de habilidades (`SkillButtonView`) con cooldown visual.

Enemigos (imágenes en `res/drawable/`): `yusepe.png` (básico), `choso_boss.webp`, `mahito.png`, `mahoraga_boss.png`, versiones dañadas y transformadas.

Tienda (ShopFragment)

4 mejoras comprables con energía maldita:

Mejora	Icono	Efecto
Tap Damage	`clicks.png`	+daño por click
Auto Clicker	`autoclicker.png`	clicks automáticos
Black Flash	`blackflash.png`	+críticos
Energy Boost	`energy_boost.png`	+energía ganada

Cofres (ChestFragment)

Al abrir un cofre: 30 % probabilidad de desbloquear un personaje → muestra `RareSummonDialogFragment` con animación. Si no: se gana energía maldita escalada.

Inventario (CharacterInventoryFragment)

2 personajes: **Ryomen Sukuna** (id=1) y **Satoru Gojo** (id=2). Cada uno con 4 habilidades con nivel mejorable. Imágenes: `sukunapfp.jpg`, `gojo_character.png`, más imágenes de cada habilidad.

Sistema de GIFs Animados

Las habilidades especiales reproducen **GIFs animados a pantalla completa** como efecto visual. Los GIFs se cargan con la biblioteca **Glide** (versión 4.16.0).

Cómo funciona

1. Existe un `ImageView` oculto (`skillGifOverlay`) que ocupa toda la pantalla del fragment.
2. Al usar una habilidad que tiene GIF, se llama a `showSkillGifAnimation(gifResId, durationMs, onComplete)`.
3. Este método:
   a) Hace visible el overlay con alpha = 0.
   b) Carga el GIF con `Glide.with(this).asGif().load(gifResId).into(overlay)`.
   c) Aplica fade-in (200ms).
   d) Tras `durationMs`, aplica fade-out (300ms).
   e) Al terminar, oculta el overlay, limpia Glide con `Glide.with(this).clear(overlay)`, y ejecuta el callback `onComplete` (que normalmente aplica el daño).

Los 4 GIFs y cuándo se usan

Todos están en `kaisenclicker_module/src/main/res/drawable/`:

Archivo	Habilidad	Personaje	Duración
`fuga_animation.gif`	Fuga (Habilidad 3)	Sukuna	1.5s
`hollow_purple_animation.gif`	Fuga / Vacío Púrpura (Hab. 3)	Gojo	1.5s
`sukuna_domain_animation.gif`	Expansión de Dominio (Ultimate)	Sukuna	2s
`gojo_domain_animation.gif`	Expansión de Dominio (Ultimate)	Gojo	2s

Qué pasa después de cada GIF

- **Fuga (Sukuna)**: tras el GIF, aplica un golpe devastador + quemadura DoT (daño por segundo durante varios segundos).
- **Hollow Purple (Gojo)**: mismo efecto que Fuga pero para Gojo.
- **Dominio de Sukuna**: tras el GIF, cambia el fondo a `shrine_background.jpg`, activa ráfaga de 12 golpes alternando Cleave/Dismantle durante 5 segundos + sangrado continuo.
- **Dominio de Gojo**: tras el GIF, cambia el fondo a `gojo_domain_background.jpg`, activa 5 segundos de habilidades sin cooldown.

Dependencia Glide

En `kaisenclicker_module/build.gradle.kts`:

```
implementation("com.github.bumptech.glide:glide:4.16.0")
annotationProcessor("com.github.bumptech.glide:compiler:4.16.0")
```

Android no soporta GIF animados de forma nativa en `ImageView`. Glide decodifica los frames del GIF y los reproduce automáticamente cuando se usa `Glide.with(ctx).asGif().load(R.drawable.xxx).in`

Base de Datos

Ubicación del código

Todo el código de la base de datos está en `kaisenclicker_module`:

`kaisenclicker_module/src/main/java/com/example/kaisenclicker/persistence/save/`

Archivo	Función
`AppDatabaseHelper.java`	Crea y actualiza las tablas SQLite
`SqlRepository.java`	Operaciones CRUD (lectura/escritura)
`GameDataManager.java`	Capa de alto nivel (SharedPreferences + SQL)
`UserRepository.java`	Registro y login de usuarios

Aunque vive en el módulo Kaisen Clicker, es la **base de datos global** de toda la app. El Hub y el 2048 importan estas clases porque `app/build.gradle.kts` incluye `implementation(project(":kaisenclic`

Ubicación de los ficheros en el dispositivo

Los ficheros `.db` se crean automáticamente en:

`/data/data/com.example.gamehub/databases/`

- `kaisen_clicker.db` — tabla `users` (compartida, para login/registro).
- `kaisen_clicker_<usuario>.db` — todo el progreso del usuario (una BD por usuario).

Esta carpeta es **privada**: solo la app puede acceder. Al desinstalar se pierde todo.
Los ficheros SharedPreferences están en `/data/data/com.example.gamehub/shared_prefs/`.
Para inspeccionar durante el desarrollo: **Android Studio → App Inspection → Database Inspector**.

Tablas de la base de datos

`AppDatabaseHelper` (versión 4) crea 7 tablas:

1. Tabla `users`

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,           -- SHA-256, nunca texto plano
    created_at INTEGER DEFAULT (strftime('%s','now'))
);
```

2. Tabla `kv_store` (clave-valor genérico)

```
CREATE TABLE kv_store (
    k TEXT PRIMARY KEY,    -- 'k' en vez de 'key' (palabra reservada)
    value_text TEXT,
    value_int INTEGER,
    value_long INTEGER,
    value_real REAL
);
```

Almacena cualquier dato simple de ambos juegos. Ejemplos de claves:

- Kaisen: `cursed_energy`, `enemy_level`, `total_clicks`, `total_damage`, `enemies_defeated`, `bosses_defeated`, `character_level`, `total_play_seconds`...
- 2048: `2048_score`, `2048_best_score`, `2048_moves`, `2048_seconds`.

3. Tabla `characters`

```
CREATE TABLE characters (
    id INTEGER PRIMARY KEY, unlocked INTEGER DEFAULT 0,
    level INTEGER DEFAULT 1, xp INTEGER DEFAULT 0
);
-- 2 filas por defecto: Sukuna (id=1), Gojo (id=2), ambos bloqueados
```

4. Tabla `upgrades`

```
CREATE TABLE upgrades (
    id TEXT PRIMARY KEY, level INTEGER DEFAULT 0, purchased INTEGER
       DEFAULT 0
);
```

IDs: `tap_damage_level`, `auto_clicker_level`, `critical_damage_level`, `energy_boost_level`.

5. Tabla `skills`

```
CREATE TABLE skills (
    id TEXT PRIMARY KEY, character_id INTEGER,
    unlocked INTEGER DEFAULT 0, level INTEGER DEFAULT 0
);
```

IDs: `cleave`, `dismantle`, `fuga`, `domain`, `amplificacion_azul`, `ritual_inverso_rojo`, `vacio_purpura`.

6. Tabla `enemies`

```
CREATE TABLE enemies (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    enemy_level INTEGER DEFAULT 1, defeated_count INTEGER DEFAULT 0
);
-- 1 fila por defecto (id=1, nivel 1, 0 derrotados)
```

7. Tabla `scores` (puntuaciones de TODOS los juegos)

```
CREATE TABLE scores (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    player_name TEXT NOT NULL,
    game_name TEXT,            -- "2048" o "Kaisen Clicker"
    score_value INTEGER NOT NULL,
    created_at INTEGER DEFAULT (strftime('%s','now')),
    extra TEXT                 -- JSON libre, ej: {"game":"2048"}
);
```

SqlRepository — Operaciones principales

- **kv_store**: `putInt/getInt`, `putLong/getLong`, `putString/getString`. Usa `INSERT OR REPLACE`. Método `incrementIntKV(key, delta)` usa transacciones atómicas.
- **scores**: `insertScore()`, `deleteScoreById()`, `getScoresCursor(nameFilter, scoreOp, scoreValue, orderBy)` (query dinámica con filtros), `getScoreById()`.
- **enemies**: `getEnemyLevel()`, `setEnemyLevel()`, `incrementEnemiesDefeated()` (transacción atómica).
- **characters**: `upsertCharacter()`, `getAllCharacters()`.
- **upgrades**: `setUpgradeLevel()`, `getUpgradeLevel()`.
- **skills**: `upsertSkill()`, `getSkillLevel(skillId, characterId)`.

Todos los métodos de kv_store tienen un **sistema de retry**: si falla, ejecuta `CREATE TABLE IF NOT EXISTS` y reintenta una vez.

GameDataManager — Doble escritura

Es la interfaz que usan los fragments. Combina **SharedPreferences** + **SQLite**.

Al escribir — siempre escribe en ambos:

```
public void saveCursedEnergy(int energy) {
    prefs.edit().putInt("cursed_energy", energy).apply();  //
        SharedPreferences
    repository.putInt("cursed_energy", energy);            // SQLite
}
```

Al leer — si la migración a SQL está completa, lee de SQLite; si no, de SharedPreferences:

```
public int getCursedEnergy() {
    if (useSql()) return repository.getInt("cursed_energy",
                         prefs.getInt("cursed_energy", 0));
    return prefs.getInt("cursed_energy", 0);
}
```

La primera vez se ejecuta una **migración automática** que copia todos los valores de SharedPreferences a SQLite.

UserRepository — Autenticación

Usa siempre `kaisen_clicker.db` (sin usuario). Las contraseñas se hashean con **SHA-256**:

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hash = digest.digest(password.getBytes(UTF_8));
// Se convierte a hexadecimal y se guarda en password_hash
```

Métodos: `registerUser()`, `authenticateUser()`, `userExists()`.

Módulo 2048 (resumen)

- **MainActivity**: tablero 4×4 (`GridLayout`), puntuación, temporizador, gestos swipe, modos Normal/Blitz (5 min).
- **GameEngine**: motor lógico puro (matriz `int[4][4]`). Movimientos, fusión de fichas, spawn aleatorio (90 % un 2, 10 % un 4).
- Guarda datos en SharedPreferences **Y** en la BD global (`kaisen_clicker_<user>.db`) con claves `2048_*`.
- Cuando hay nuevo récord, inserta en la tabla `scores` con `game_name = "2048"`.

Resumen de tecnologías

Tecnología	Uso
Java	Lenguaje principal
Android SDK (compileSdk 36, minSdk 26)	Plataforma
Gradle (Kotlin DSL)	Build multi-módulo
SQLite (`SQLiteOpenHelper`)	Base de datos local
SharedPreferences	Sesión y configuración rápida
RecyclerView + CardView + Cursor	Listas de puntuaciones
ItemTouchHelper	Swipe-to-delete
Material Design 3	Componentes UI
Glide 4.16.0	Carga y reproducción de GIFs animados
SplashScreen API	Splash nativo
SHA-256	Hash de contraseñas
DataBinding	Enlace de vistas (StatisticsFragment)