GameHub

Documentación Técnica Breve

Proyecto Android Multi-Módulo

Lenguaje: Java | Build System: Gradle (Kotlin DSL)

Min SDK: 26 | Target SDK: 36

Módulos:

- `app` Hub principal (login, perfil, menús)
- `kaisenclicker_module` Juego: Kaisen Clicker
- `2048_module` Juego: 2048

Febrero 2026

Índice

Visión General del Proyecto

GameHub es una aplicación Android multi-módulo que funciona como un **centro de juegos**. El módulo principal (`app`) contiene el sistema de autenticación, gestión de perfil, puntuaciones y desafíos. Los juegos se implementan como módulos independientes que se integran en el hub.

- **Hub (`app`)**: Login/Registro, pantalla principal con grid de juegos, perfil de usuario, leaderboard, sistema de desafíos.
- **Kaisen Clicker (`kaisenclicker_module`)**: Juego clicker con temática de Jujutsu Kaisen. Incluye campaña, tienda, cofres, inventario de personajes y estadísticas.
- **2048 (`2048_module`)**: Implementación del juego clásico 2048 con modos de juego (normal, blitz) y temas visuales.

Base de Datos — Arquitectura de Persistencia

La persistencia del proyecto se basa en **SQLite nativo** (sin Room) gestionado a través de `SQLiteOpenHelper`. Existen dos bases de datos principales y un sistema auxiliar de `SharedPreferences`.

Base de Datos Principal: `kaisen_clicker.db`

Gestionada por `AppDatabaseHelper` (versión 4). Es la BD **global** de toda la aplicación. Almacena datos tanto de Kaisen Clicker como de 2048.

Ruta: `kaisenclicker_module/src/main/java/com/example/kaisenclicker/persistence/save/AppDataba`

Tablas de la BD principal

Tabla	Columnas principales	Descripción
`users`	`id`, `username` (UNIQUE), `password_hash`, `created_at`	Usuarios registrados. Contraseñas almacenadas con hash SHA-256.
`kv_store`	`k` (PK), `value_text`, `value_int`, `value_long`, `value_real`	Almacén clave-valor genérico. Guarda energía maldita, niveles de mejora, progreso del 2048, etc.
`characters`	`id` (PK), `unlocked`, `level`, `xp`	Personajes jugables. IDs predeterminados: 1 = Ryomen Sukuna, 2 = Satoru Gojo.
`upgrades`	`id` (TEXT PK), `level`, `purchased`	Mejoras compradas en la tienda (daño, auto-clicker, etc.).
`skills`	`id` (TEXT PK), `character_id`, `unlocked`, `level`	Habilidades de los personajes (Cleave, Dismantle, Fuga, Domain Expansion, etc.).
`enemies`	`id`, `enemy_level`, `defeated_count`	Estado del progreso de enemigos (nivel actual y derrotados).
`scores`	`id`, `user_id`, `player_name`, `game_name`, `score_value`, `created_at`, `extra`	Tabla de puntuaciones compartida entre todos los juegos.

Aislamiento por usuario

Los datos se aíslan por usuario mediante **bases de datos separadas**: cada usuario obtiene su propia BD con nombre `kaisen_clicker_<username>.db`. El constructor de `GameDataManager` recibe el username y genera el nombre de fichero correspondiente.

```
// GameDataManager.java
String dbName = (username != null && !username.isEmpty())
        ? "kaisen_clicker_" + username + ".db"
        : AppDatabaseHelper.DATABASE_NAME;
repository = new SqlRepository(context, dbName);
```

Base de Datos de Estadísticas: `kaisenclicker_stats.db`

Gestionada por `StatsDatabaseHelper` (versión 1). Base de datos independiente y específica para las estadísticas del módulo Kaisen Clicker.

Ruta: `kaisenclicker_module/src/main/java/com/example/kaisenclicker/StatsDatabaseHelper.java`

Columna	Descripción
`user_id`	ID del usuario
`total_damage`	Daño total infligido
`dps`	Daño por segundo (REAL)
`enemies`	Enemigos normales derrotados
`bosses`	Bosses derrotados
`clicks`	Clicks totales
`unlocked`	Personajes desbloqueados
`next_progress`	Progreso al siguiente objetivo (0–100)

Capa de Acceso a Datos: `SqlRepository`

Clase central de acceso a la BD (**811 líneas**). Provee operaciones CRUD tipadas sobre el almacén clave-valor y las tablas específicas.

Ruta: `kaisenclicker_module/.../persistence/save/SqlRepository.java`

- **KV Store**: `putInt/getInt`, `putLong/getLong`, `putString/getString` — con reintentos automáticos ante fallos.
- **Characters**: `upsertCharacter()`, `getAllCharacters()`.
- **Enemies**: `setEnemyLevel()`, `getCurrentEnemyLevel()`, `setCurrentEnemyState()`.
- **Scores**: Lectura/escritura de puntuaciones para el leaderboard.

Persistencia Dual: SharedPreferences + SQLite

`GameDataManager` (**508 líneas**) implementa una estrategia de **persistencia dual**:

1. Los datos se escriben **siempre** en `SharedPreferences` (escritura rápida).
2. Simultáneamente se escriben en `SQLite` vía `SqlRepository`.
3. Al leer, si la migración a SQL está completa, se prefiere SQLite; de lo contrario, se usa SharedPreferences como fallback.
4. La migración automática se ejecuta en `migratePrefsToSqlIfNeeded()`.

Autenticación de Usuarios: `UserRepository`

Ruta: `kaisenclicker_module/.../persistence/save/UserRepository.java`

- Usa la BD global (`kaisen_clicker.db`), **no** la per-usuario.
- `registerUser()`: Inserta en tabla `users` con contraseña hasheada (SHA-256).
- `authenticateUser()`: Verifica credenciales comparando hashes.
- `userExists()`: Comprueba si un username ya está registrado.

Sesión de Usuario: `SessionManager`

Ruta: `app/.../auth/SessionManager.java`

Gestiona la sesión activa del usuario mediante `SharedPreferences` (`GameHubSession`):

- Login/logout, username, estado (online/playing/away).
- Foto de perfil (URI), puntos totales, partidas jugadas.
- **Tracking de tiempo**: `markGameStarted()` / `markGameEnded()` para acumular segundos jugados con protección anti-corrupción (máximo 24h por sesión).

GIFs Animados — Animaciones de Habilidades

El módulo Kaisen Clicker utiliza **GIFs animados** como efectos visuales a pantalla completa cuando se activan habilidades especiales de los personajes. Los GIFs se cargan mediante la librería **Glide**.

Archivos GIF

Fichero	Uso
`fuga_animation.gif`	Habilidad "Fuga" de Ryomen Sukuna
`hollow_purple_animation.gif`	Habilidad "Vacío Púrpura" de Satoru Gojo
`gojo_domain_animation.gif`	Expansión de Dominio de Gojo (Infinite Void)
`sukuna_domain_animation.gif`	Expansión de Dominio de Sukuna (Malevolent Shrine)

Ubicación: `kaisenclicker_module/src/main/res/drawable/`

Mecanismo de Reproducción

El método `showSkillGifAnimation()` en `CampaignFragment` gestiona la reproducción:

```
private void showSkillGifAnimation(int gifResId, int durationMs,
                                   Runnable onComplete) {
    // 1. Hacer visible el overlay (ImageView a pantalla completa)
    skillGifOverlay.setVisibility(View.VISIBLE);

    // 2. Cargar GIF con Glide
    Glide.with(this).asGif().load(gifResId).into(skillGifOverlay);

    // 3. Fade-in rapido (200ms)
```

```
    skillGifOverlay.animate().alpha(1f).setDuration(200).start();

    // 4. Tras la duracion, fade-out y ejecutar callback de dano
    mainHandler.postDelayed(() -> {
        skillGifOverlay.animate().alpha(0f).setDuration(300)
            .withEndAction(() -> {
                skillGifOverlay.setVisibility(View.GONE);
                Glide.with(this).clear(skillGifOverlay);
                if (onComplete != null) onComplete.run();
            }).start();
    }, durationMs);
}
```

- **Duración típica**: 1500ms (habilidades normales), 2000ms (dominios).
- **Fallback**: Si Glide falla, se carga como imagen estática con `setImageResource()`.
- **Liberación de memoria**: Se llama a `Glide.clear()` al terminar la animación.
- El daño se aplica **después** de la animación (en el callback `onComplete`).

Arquitectura de Módulos y Navegación

Registro Dinámico de Juegos

`GameRepository` (Singleton) gestiona los juegos disponibles en el hub. Cada módulo se registra en `MainActivity.registerGames()`:

```
repo.registerGame(new Game(
    "kaisen_clicker",
    "Kaisen Clicker",
    com.example.kaisenclicker.R.drawable.kaisen_icon,
    "Haz clic para derrotar maldiciones!",
    com.example.kaisenclicker.ui.activities.MainActivity.class
));

repo.registerGame(new Game(
    "2048", "2048",
    com.example.a2048.R.drawable.icon,
    "Desliza y combina numeros!",
    com.example.a2048.MainActivity.class
));
```

El modelo `Game` encapsula: ID, nombre, icono, descripción y la `Activity.class` destino.

Flujo de Navegación Principal

1. **LoginActivity** → Autenticación vía `UserRepository`.
2. **MainActivity (Hub)** → Grid de juegos, perfil, menú.
3. Al seleccionar un juego → Se lanza la `Activity` del módulo correspondiente con `EXTRA_USERNAME`.
4. **Al volver** → `SessionManager.markGameEnded()` acumula tiempo jugado.

BaseActivity — Modo Inmersivo

Todas las Activities del hub extienden `BaseActivity`, que oculta automáticamente la barra de navegación del sistema para ofrecer una experiencia a pantalla completa (API 30+: `WindowInsetsController`; versiones anteriores: flags `SYSTEM_UI`).

Sistema de Juego: Kaisen Clicker

Fragmentos Principales

Fragment	Función
`CampaignFragment`	Fragmento principal del combate (**2676 líneas**). Gestiona taps, habilidades, enemigos, bosses, DPS, ultimate y efectos visuales.
`ShopFragment`	Tienda de mejoras: Daño de Tap, Auto Clicker, Black Flash, Energy Boost. Costes escalados por nivel.
`ChestFragment`	Sistema de cofres con probabilidad de desbloquear personajes (30 %) y recompensas de energía maldita escaladas por nivel.
`CharacterInventoryFragment`	Inventario de personajes desbloqueados.
`StatisticsFragment`	Pantalla de estadísticas del jugador.

Sistema de Combate

- **Enemigos**: HP escalado exponencialmente: $HP = 45 \times 1{,}11^{(nivel-1)}$, con tope en 5×10^8.
- **Bosses**: Choso (dos fases), Mahito (transformación), Mahoraga (adaptación progresiva).
- **Auto Clicker**: Clicks automáticos cuya frecuencia depende del nivel comprado.
- **DPS en tiempo real**: Ventana deslizante de 3 segundos, recalculada cada 500ms.
- **Global ultimate**: Carga de 0 a 100 %, permite usar la Expansión de Dominio.

Sistema de Habilidades

Gestionado por `CharacterSkillManager`. Cada habilidad tiene tipo, cooldown, nivel máximo y parámetros de sangrado (DoT):

Habilidad	Tipo	Cooldown	Efecto
Cleave	NORMAL_1	2s	100 % + 20 %/nivel + sangrado (5s, ticks cada 1s)
Dismantle	NORMAL_2	3s	% de vida actual del enemigo (12 % base + 3 %/nivel)
Fuga	NORMAL_3	4s	120 % + 35 %/nivel, golpe devastador
Domain Expansion	ULTIMATE	8s	300 % + 60 %/nivel, ráfaga masiva

Los cooldowns se reducen un 10 % por cada nivel adicional de la habilidad.

Personajes Jugables

Personaje	ID	Habilidades específicas
Ryomen Sukuna	1	Cleave, Dismantle, Fuga, Domain Expansion (Malevolent Shrine)
Satoru Gojo	2	Amplificación Azul, Ritual Inverso Rojo, Vacío Púrpura, Domain (Infinite Void)

Gojo: su dominio activa un estado donde las habilidades no tienen cooldown y Vacío Púrpura es gratuito.

Sistema de Juego: 2048

Motor del Juego: `GameEngine`

Implementación clásica del juego 2048 con una matriz `int[4][4]`:

- **Spawn**: Al iniciar, dos casillas aleatorias. Tras cada movimiento válido, aparece un 2 (90 %) o un 4 (10 %).
- **Movimientos**: `moveLeft()`, `moveRight()`, `moveUp()`, `moveDown()`.
- **Lógica**: `compressAndMerge()` comprime y fusiona una línea, sumando al score al fusionar.
- **Persistencia**: Score y best score se guardan en `kv_store` de la BD global con claves `2048_score`, `2048_best_score`, `2048_moves`, `2048_seconds`.
- **Detección táctil**: `OnSwipeTouchListener` detecta gestos de deslizamiento.

Sistema de Desafíos

La clase `ChallengesActivity` define logros que el usuario puede completar para ganar puntos:

ID	Nombre	Objetivo	Recompensa
`first_game`	Primer Paso	1 partida	50 puntos
`five_games`	Jugador Habitual	5 partidas	100 puntos
`ten_games`	Veterano	10 partidas	200 puntos
`hundred_points`	Primeros Puntos	100 pts	50 puntos
`thousand_points`	Mil Puntos	1.000 pts	150 puntos
`five_thousand`	Maestro del GameHub	5.000 pts	500 puntos
`twentyfive_games`	Adicto al Juego	25 partidas	300 puntos
`try_all_games`	Explorador	2 juegos	100 puntos

Cada `Challenge` calcula su progreso como porcentaje: $progreso = \text{mín}\left(100, \frac{actual \times 100}{objetivo}\right)$.

Leaderboard y Puntuaciones

- **LeaderboardActivity**: Pantalla con tabs para alternar entre las stats de Kaisen Clicker y 2048.

- **Kaisen Clicker**: Muestra nivel de enemigo, clicks totales, daño total, enemigos/bosses derrotados, energía maldita, nivel de personaje y tiempo jugado.
- **2048**: Muestra score actual, mejor score, movimientos y tiempo.
- **ScoresListActivity**: Historial de puntuaciones con búsqueda, ordenación y swipe-to-delete. Usa `Cursor` + `RecyclerView`.
- Los datos de 2048 se leen del `SqlRepository` con claves `2048_*` en la BD per-usuario.

Dependencias Principales

Dependencia	Uso
`appcompat`	Compatibilidad hacia atrás de Activities/Fragments
`material`	Componentes Material Design (CardView, Buttons, etc.)
`constraintlayout`	Layouts responsivos
`recyclerview`	Listas eficientes (juegos, puntuaciones, desafíos)
`core-splashscreen`	Pantalla de splash nativa (API 31+)
`gridlayout`	Grid para la pantalla principal
`Glide`	Carga y renderizado de GIFs animados

Resumen de Rutas Clave

Componente	Ruta relativa
BD principal	`kaisenclicker_module/.../persistence/save/AppDatabaseHelper.java`
Repositorio SQL	`kaisenclicker_module/.../persistence/save/SqlRepository.java`
Gestor de datos	`kaisenclicker_module/.../persistence/save/GameDataManager.java`
Repositorio usuarios	`kaisenclicker_module/.../persistence/save/UserRepository.java`
BD estadísticas	`kaisenclicker_module/.../StatsDatabaseHelper.java`
Sesión	`app/.../auth/SessionManager.java`
GIFs animados	`kaisenclicker_module/src/main/res/drawable/*.gif`
Motor 2048	`2048_module/.../GameEngine.java`
Hub principal	`app/.../MainActivity.java`
Desafíos	`app/.../ChallengesActivity.java`
Leaderboard	`app/.../leaderboard/LeaderboardActivity.java`