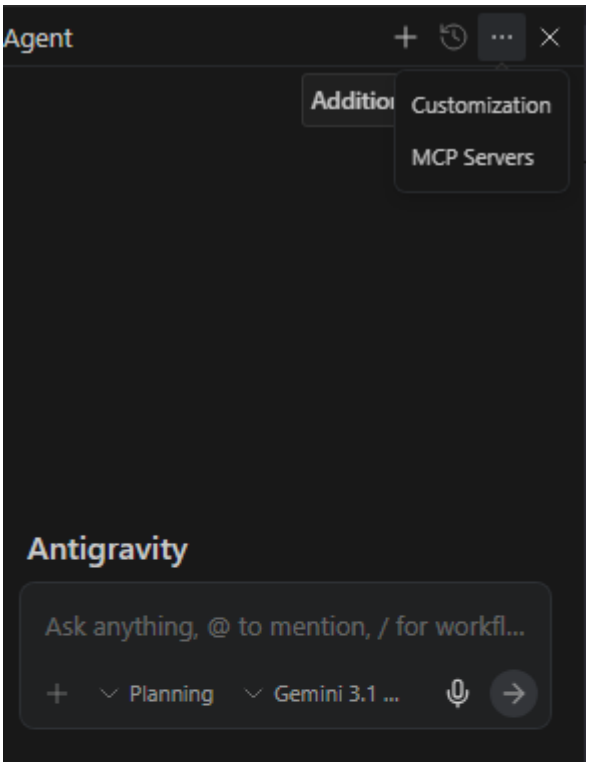
Arquitectura de la app

Idea de la app

Aplicación móvil híbrida de colección y estrategia en tiempo real. Combina la **exploración física** (Geolocalización tipo Pokémon GO para avistamiento y recolección de recursos) con **batallas estratégicas de cartas** (tipo Magic/Hearthstone) basadas en ornitología real. El sistema incluye un **Marketplace en tiempo real** y funciones sociales avanzadas, soportado por una arquitectura backend **Reactiva y No Bloqueante**.

Cómo lo vamos a estar trabajando

Usaremos el editor Antigravity. En este en los 3 puntos de la zona del chat (arriba) le damos a habilitar los MCPs.

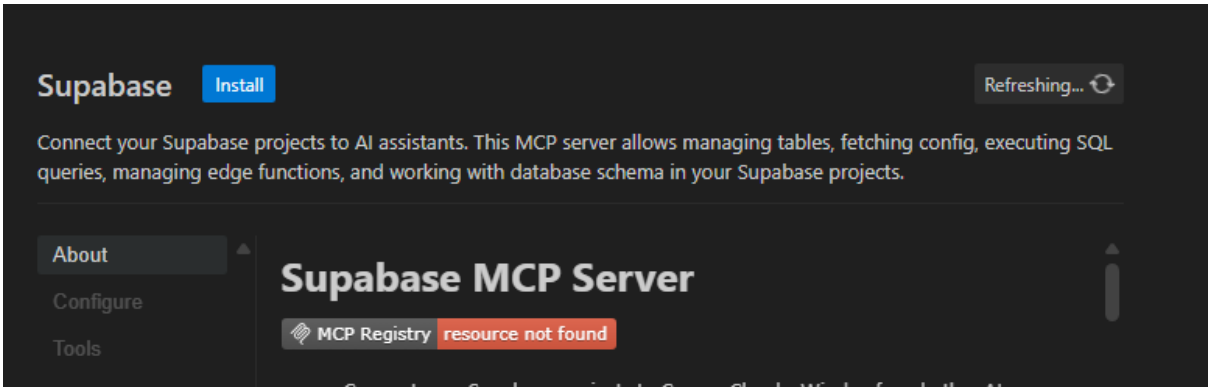


Un Mcp es una forma de darle herramientas externas y configuradas a la IA de internet (hecho por ingenieros de IA) pero en local, de manera que le permite a la IA ser más eficiente a la hora de trabajar (menos tokens). También vamos a usar unas customizaciones, que es como una configuración dedicada para el proyecto con un contexto con el que va a estar trabajando el agente puesto. Previamente necesitaremos tener instalado npx https://nodejs.org/en/download.

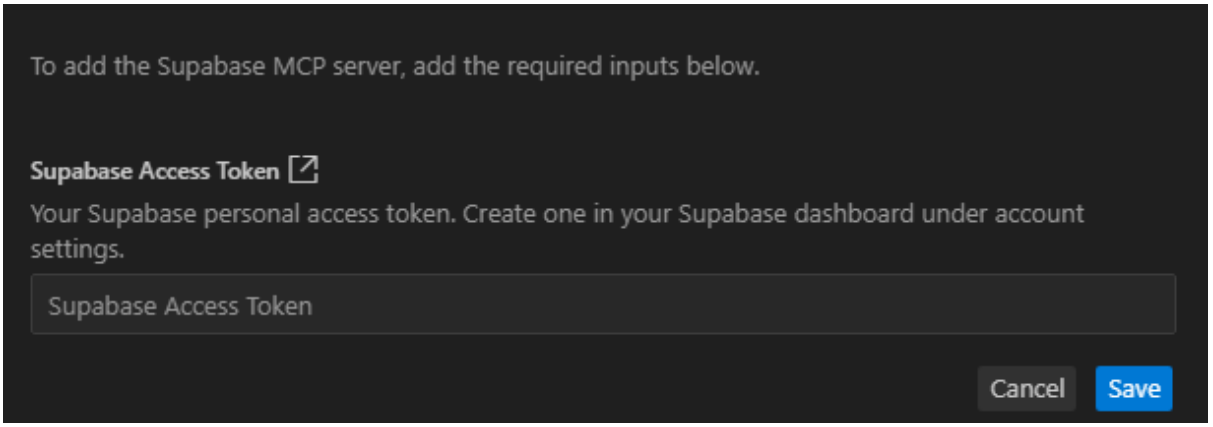
MCPs:

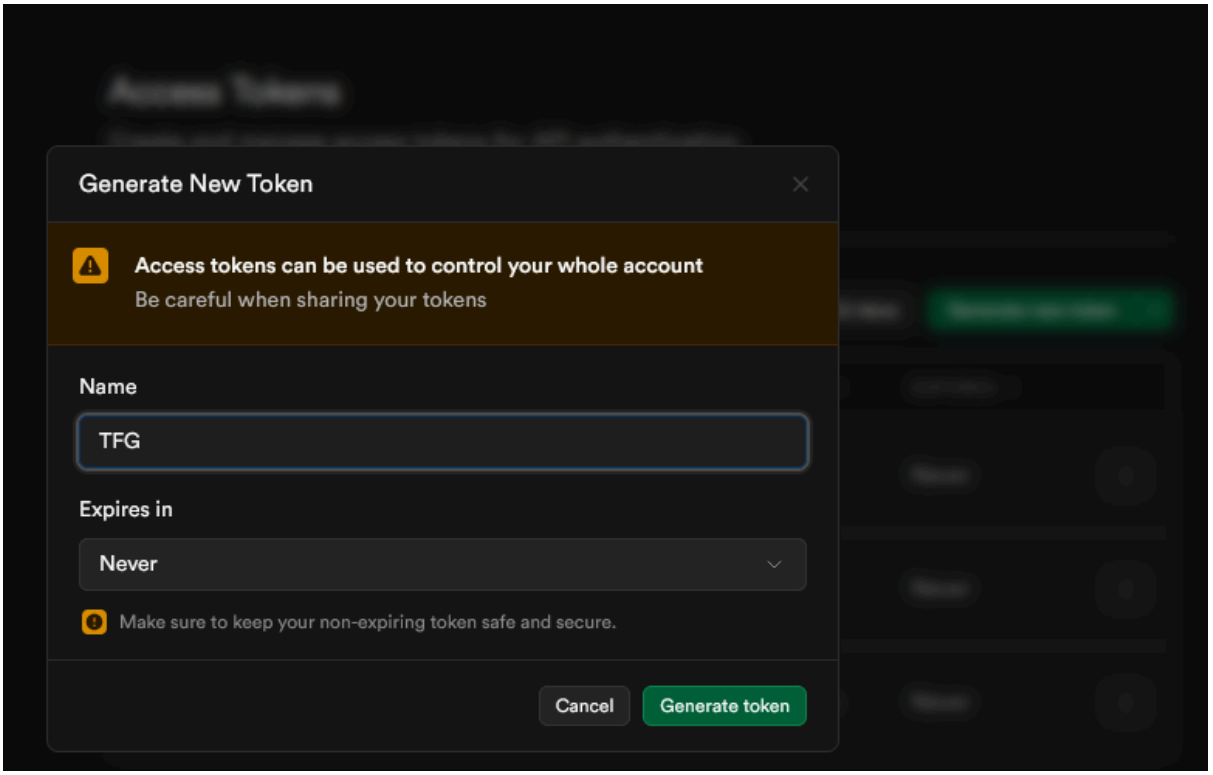
- Supabase

- Github

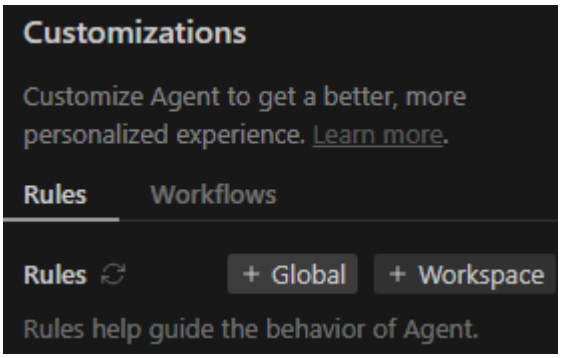


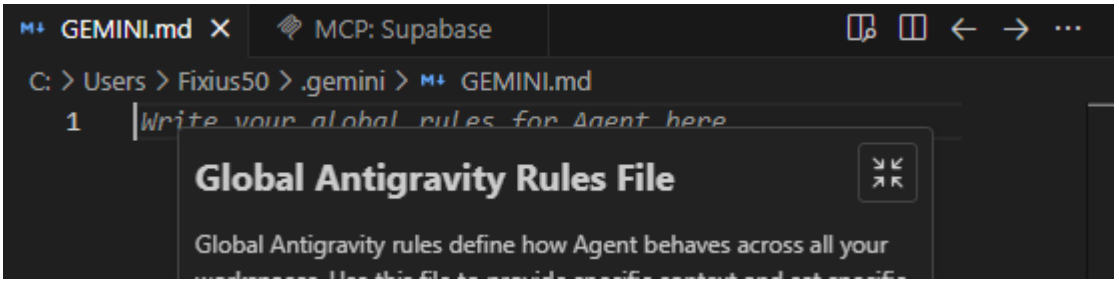
Le damos a instalar; nos pide en algunas el token de supabase, donde se encuentra en el perfil de nuestro usuario (suele proporcionarse mediante un link que te sale).





Customizaciones: están en el drive. Se pone dándole a “Global” y metiendo ahí el rol de “cerebro.txt”.





Jugabilidad

Va a consistir en que fabriquemos nuestras cartas mediante materiales (como si fueran jaulas) con distintos materiales, formas, y tamaños. Una vez hechas vamos a cazar pájaros, cada uno según con lo que creemos que pueden atraer al pájaro según los datos de la vida real (algunos con comida, otros con sonidos, etc…).

1. El Ciclo de Vida del Jugador (UX Unificada)

El juego se organiza en un ciclo diario que transforma recursos físicos en logros competitivos:

- **Mañana (Expedición):** Recolección activa de materiales brutos (Madera, Bayas) mediante minijuegos de "Enfoque".
- **Mediodía (Crafting):** Construcción de la **Estación de Reclamo** en el Taller combinando materiales.
- **Tarde (Notificación):** El sistema sincroniza datos reales (clima/probabilidad) y atrae a un ave. Tomas la foto y generas la carta.
- **Noche (Certamen):** Utilizas tus nuevas cartas en la Arena para ganar **Reputación** y materiales de alta calidad (como Metal).

2. Fase de Adquisición: El Taller del Naturalista

En lugar de comprar sobres, el jugador **construye la oportunidad** de obtener cartas específicas mediante la personalización de "Estaciones de Reclamo".

A. Recolección de Materiales

Se obtienen materiales mediante expediciones pasivas o minijuegos activos:

- **Madera:** Para atraer aves de bosque.
- **Fibras/Hierbas:** Funcionan como camuflaje; sin ellas, las aves tímidas no se acercarán.
- **Metal/Restos:** Para estructuras urbanas o resistentes.
- **Cebos:** Semillas, fruta o insectos que definen la probabilidad de la especie.

B. Construcción y Sincronización (Backend Logic)

El usuario diseña la estación eligiendo **Base** (Madera/Metal), **Tamaño** (Pequeña para gorriones, Grande para rapaces) y **Cebo**. El servidor calcula qué pájaro aparece basándose en:

1. **Estructura:** Atrae familias específicas (ej. Madera $\rightarrow$ Pájaro Carpintero).
2. **Cebo:** Define la dieta (ej. Insectos $\rightarrow$ Insectívoros).
3. **Clima (API):** Si llueve, aumenta la probabilidad de aves buscando refugio.

> **Resultado:** Tras un tiempo, la estación se gasta o pierde durabilidad, obligando a cerrar el ciclo de consumo y construir una nueva.

3. Fase de Batalla: El Certamen ("Magic Simplificado")

Un duelo 1vs1 por turnos que utiliza un sistema de **Posturas** y gestión de energía.

El Tablero y Recursos

- **Zona de Juego:** 3 huecos para pájaros por jugador.
- **Mana (Semillas):** Recurso que aumenta progresivamente (Turno 1 = 1 Semilla, Turno 2 = 2 Semillas, etc.) para invocar aves más poderosas.
- **Estructura de Carta:** Incluye el Coste (Semillas), Postura Predilecta y Habilidad Pasiva (ej. "Si llueve, gana +1 en Vuelo").

Confrontación y Duelo de Posturas

Si un pájaro enfrenta un hueco vacío, otorga **Puntos Directos (Reputación)**. Si enfrenta a otro pájaro, se inicia un **Duelo de Posturas** basado en el Triángulo de Poder:

Postura	**Vence a...**	**Lógica Narrativa**
Canto (Rojo)	Plumaje	El grito asusta a la belleza.
Plumaje (Verde)	Vuelo	La belleza distrae al movimiento.

Vuelo (Azul)	Canto	La velocidad escapa del ruido.

Resolución del Duelo:

- **Ganas:** El pájaro rival huye (se elimina de la mesa).
- **Empatas:** Ambos se quedan "cansados" en el tablero.
- **Pierdes:** Tu pájaro se retira del combate.

4. Gestión y Progresión

- **Álbum (Colección):** Las cartas tienen una **Cara A** (Juego/Stats) y una **Cara B** (Educativa con datos científicos y audio real vía Nuthatch API).
- **Marketplace:** Las cartas crafteadas o repetidas pueden venderse o subastarse en tiempo real utilizando la arquitectura reactiva del servidor (WebFlux + Redis).

Para completar el ecosistema de **AVIS**, la dimensión social actúa como el tejido que une la exploración, el crafteo y el combate, fomentando tanto la competitividad en el mercado como la cooperación en el cuidado de las aves.

Aquí tienes la integración del **Módulo Social** con el resto de la jugabilidad:

1. Bandadas (Sindicatos de Naturalistas)

La interacción principal se organiza a través de grupos de usuarios llamados **Bandadas**.

- **Chat en Tiempo Real:** Comunicación fluida mediante chats privados y de clan, implementados sobre la arquitectura **RSocket** para garantizar baja latencia.
- **Eventos de Comunidad:** Participación en eventos especiales que son fundamentales para el progreso del grupo.
- **Estrategia Compartida:** Los rivales pueden ofrecer consejos automáticos tras las batallas; por ejemplo, sugerir buscar especies de montaña si detectan debilidades en tu equipo.

2. Marketplace Reactivo (Trading)

El sistema permite una economía viva donde los usuarios no dependen del azar, sino del intercambio.

- **Compra y Venta:** Los usuarios pueden vender cartas que han crafteado en el Taller o que tienen repetidas.
- **Subastas en Tiempo Real:** Sistema de pujas dinámico soportado por **WebFlux**, permitiendo actualizaciones instantáneas sin recargar la app.
- **Seguridad y Rapidez:** Uso de **Redis** para búsquedas en submilisegundos y **Redisson** para evitar que dos personas compren la misma carta simultáneamente (bloqueos distribuidos).

3. Socialización Cooperativa y Mantenimiento

La app premia activamente la interacción positiva entre los naturalistas para mejorar la experiencia general.

- **Mantenimiento de Puestos:** Los puestos de avistamiento tienen un tiempo máximo de 18 horas de efectividad.
- **Bonus por Ayuda:** Si un puesto ajeno queda inhabilitado, cualquier usuario puede limpiarlo o mantenerlo, recibiendo a cambio un **bonus de recompensa**.
- **Santuarios Visitables:** La filosofía de "Mejorar tu Santuario" se extiende a mostrar tus logros y aves raras a otros miembros de tu Bandada.

4. Integración Técnica Social

Para que esta experiencia sea fluida, el backend utiliza herramientas avanzadas:

- **Validación de Identidad: Spring Security Reactive** con tokens JWT asegura que cada transacción y mensaje sea auténtico y seguro.
- **Desacoplamiento de Recompensas:** Tras una interacción social o batalla, un gestor de eventos (**RabbitMQ o Kafka**) procesa las recompensas en segundo plano para no interrumpir la navegación del usuario.
- **Accesibilidad:** Los botones sociales incluyen etiquetas descriptivas (Semantics) para que cualquier persona, independientemente de su nivel técnico, pueda interactuar con la comunidad.

Frontend

- Funcionalidad simple para que cualquier persona entienda bien ell funcionamiento de la aplicación da igual la edad y el nivel técnico.

Tecnologías:

- **React Native**

BORRADOR

GUÍA DE ESTILO Y UX (Design System)

Antes de pintar pantallas, definimos las reglas visuales.

1. La Metáfora Visual: "El Cuaderno de Campo Vivo"

La aplicación debe sentirse como si tuvieras un diario de naturalista en las manos, pero mágico.

- **Paleta de Colores (Naturaleza Soft):**

    - ***Primario:*** **Verde Salvia (`#7C9A92`) - Para acciones principales.**
    - ***Secundario:*** **Terracota Suave (`#D9A08B`) - Para alertas o combate.**
    - ***Fondo:*** **Papel Crema/Hueso (`#FDFBF7`) - Evita el blanco puro, cansa la vista.**
    - ***Texto:*** **Gris Carbón (`#2C3E50`) - Alto contraste pero menos agresivo que el negro.**
- **Tipografía:**
    - ***Títulos:*** **Merriweather o Lora (Serif). Elegancia clásica.**
    - ***Cuerpo/Botones:*** **Nunito o Quicksand (Sans Serif redondeada). Muy amigable y legible.**
- **Formas:**
    - **Glassmorphism: Paneles semitransparentes con desenfoque (`blur`) para superponer menús sobre los fondos de naturaleza.**
    - **Bordes: Todo muy redondeado (`border-radius: 20px`). Nada de esquinas afiladas.**

📱 ESQUEMA DE PANTALLAS (WIREFRAMES & FLUJO)

La navegación se basa en una Barra Inferior (Bottom Bar) persistente con 4 iconos grandes.

1. PANTALLA: EL SANTUARIO (HOME)

Objetivo: Relajación y estado general.

- **Fondo (Dinámico):**
    - **Ocupa toda la pantalla. Es una ilustración vectorial (SVG) o Lottie que cambia según la API del tiempo (Lluvia, Sol, Noche).**
- **Zona Central (El Árbol):**
    - **Un árbol ilustrado donde se posan aleatoriamente 3-5 pájaros de tu colección.**
    - ***Interacción:*** **Si tocas un pájaro, hace su sonido (Audio) y suelta una pequeña animación de corazones.**
- **Panel Superior (Resumen):**
    - **Pequeña tarjeta de cristal: "🌤️ Madrid: Despejado | 🌡️ 18ºC".**
    - **Contador de recursos: "🌰 Semillas: 150".**

2. PANTALLA: EXPEDICIÓN (Juego Pasivo/Activo)

Objetivo: Conseguir "Fotos" (Material base).

- **Selector de Bioma (Carrusel):**
    - **Deslizar izquierda/derecha para elegir: Bosque, Costa, Montaña.**
    - ***Indicador Visual:* Si llueve en el juego, el icono del Bosque tiene gotitas.**
- **Estado A: Iniciar Expedición**
    - **Botón grande "Enviar Observador".**
    - **Selector de Cebo (Iconos grandes: Gusano, Fruta, Pez).**
- **Estado B: Esperando (El Minijuego)**
    - **Mientras corre el tiempo (ej: 01:59:00), aparece un botón vibrante: "Realizar Avistamiento Rápido".**
    - **Overlay del Minijuego "Enfoque":**
        - **Fondo: Una foto de Pexels muy borrosa.**
        - **Control: Un slider horizontal en la parte baja.**
        - **Acción: Mover el slider hasta que la foto esté nítida. Al soltar en el punto exacto -> ¡Flash! -> Ganas "Notas de Campo".**

3. PANTALLA: TALLER (CRAFTING)

Objetivo: Crear las cartas finales.

- **Diseño: Parece una mesa de madera de escritorio.**
- **Zona de Drop (El Tapete):**
    - **Tres huecos vacíos con formas: [Foto] + [Pluma] + [Notas].**
- **Inventario Inferior:**
    - **Lista horizontal arrastrable con tus objetos conseguidos en expediciones.**
- **Feedback:**
    - **Al arrastrar los 3 objetos correctos, el centro brilla.**
    - **Botón "Registrar Ave" se activa.**
    - **Animación de Recompensa: La carta aparece en blanco, se dibuja el contorno y luego se rellena de color (efecto acuarela).**

4. PANTALLA: CERTAMEN (BATALLA)

Objetivo: Competir (Piedra-Papel-Tijera Estratégico).

- **Layout Dividido:**
    - **Arriba: Pájaro del Rival (Avatar + Nombre).**
    - **Abajo: Tu Pájaro (Avatar + Stats).**
- **Zona Central (La Acción):**
    - **3 Botones Circulares Grandes (Iconos):**
        1. **🎤 Canto (Rojo)**
        2. **💨 Vuelo (Azul)**
        3. **🪶 Plumaje (Verde)**
- **Indicadores de Ayuda (Accesibilidad):**
    - **Flechas pequeñas entre los botones que indican quién gana a quién (Canto > Plumaje).**
- **El Factor Clima:**

    - **Si hay VIENTO (API), el botón de Vuelo tiene un icono de "Prohibido" o un candado semitransparente que indica "Cuesta más energía".**

5. PANTALLA: EL ÁLBUM (COLECCIÓN)

Objetivo: Gestión y Consulta.

- **Vista de Rejilla: Cards rectangulares verticales.**
- **Filtros: Pestañas simples: "Todos", "Agua", "Bosque".**
- **Detalle de Carta (Modal a pantalla completa):**
    - **Cara A (Juego): Foto grande, Nombre Común, Iconos de Stats (1-10).**
    - **Botón Girar: Icono de flecha en la esquina.**
    - **Cara B (Educativa): Fondo estilo papel antiguo. Nombre científico, Mapa de distribución, Texto de curiosidad, Botón de "Escuchar Canto".**

🛠️ COMPONENTES TÉCNICOS (FLUTTER/REACT)

Para programar esto ordenadamente en el Frontend:

A. Widgets Reutilizables (Components)

1. **`WeatherBackground`: Un componente que recibe el estado del clima ("RAIN", "SUN") y renderiza el fondo animado correspondiente.**
2. **`BirdCard`: El componente visual de la carta. Debe aceptar parámetros para mostrarse en "Modo Mini" (Inventario) o "Modo Full" (Álbum).**
3. **`ResourceCounter`: Pill (pastilla) visual con icono y número animado.**

B. Gestión de Estado (Logic)

Necesitamos un gestor de estado global (Provider, Bloc o Riverpod en Flutter).

- **`WeatherProvider`: Consulta a tu Backend al abrir la app y guarda el clima. Todos los componentes se suscriben a esto para saber si "llueve".**
- **`UserProvider`: Guarda las monedas (Semillas, Notas) y el Inventario.**

C. Accesibilidad (El Toque Profesional)

- **Semantics (Flutter): Etiquetar cada botón para lectores de pantalla.**
    - ***Mal:*** **Botón con icono de micrófono.**
    - ***Bien:*** **Etiqueta "Usar Canto. Fuerte contra Plumaje".**
- **Haptic Feedback: Vibración suave del móvil cuando:**

- **Enfocas bien la foto en el minijuego.**
- **Ganas una ronda en el Certamen.**

💡 DETALLE DE "JUGABILIDAD INTERCONECTADA" EN LA UI

¿Cómo le mostramos al usuario que todo está conectado? Con Pistas Visuales.

1. **En la Expedición:**
   - **Si tienes "Notas de Campo" (conseguidas en el minijuego), el botón de "Enviar Expedición" brilla en dorado, indicando que tendrás más suerte.**
2. **En el Taller:**
   - **Si te falta una "Pluma" para completar un pájaro, al pulsar el hueco vacío, sale un popup: *"¡Gana certámenes para conseguir plumas!"*. (Te redirige al juego).**
3. **En el Certamen:**
   - **Si pierdes una batalla, el rival te dice: *"Tu pájaro es lento. Busca especies de Montaña en las Expediciones"*.**

TERMINA BORRADOR

Backend

APIS RECOMENDADAS POR EL PROFE

APIs útiles para el proyecto de DAM
Unsplash API y Pexels API son opciones populares para descargar fotos gratuitas de alta calidad mediante solicitudes programáticas.
Unsplash API
Proporciona acceso a millones de imágenes gratuitas para uso comercial sin atribución obligatoria. Regístrate en unsplash.com/developers para obtener una clave API gratuita, con límites iniciales de 50 solicitudes por hora (ampliables). Usa endpoints como /photos/random para fotos aleatorias o /search/photos para búsquedas específicas.
Pexels API
Ofrece fotos y videos gratuitos con licencia CC0, ideal para proyectos web. Crea una cuenta en pexels.com/api para tu clave API gratuita (200 solicitudes por hora, 20,000 mensuales). Endpoints clave incluyen /v1/photos/popular y /v1/search para curación y búsquedas.
Otras alternativas simples

Lorem Picsum: API sin registro para imágenes placeholder aleatorias (ej. https://picsum.photos/800/600), perfecta para pruebas rápidas sin límites estrictos.
Pixabay API: Miles de fotos gratuitas con clave gratuita, soporta búsquedas por categoría y alta resolución.
Nuthatch API — Es una API gratuita orientada a aves; su catálogo incluye fotos (compiladas con imágenes de fuentes libres como Unsplash + contribuciones privadas).
Vecteezy API — Permite acceder a una biblioteca enorme de fotos, vectores y recursos gráficos; útil si no necesitas fotos específicamente ornitológicas, pero sí imágenes "libres" de aves o naturaleza.

APIs Externas (Gestión de Recursos)

- **Nuthatch API:** Fuente de verdad para datos taxonómicos (nombre científico, familia).
- **Unsplash / Pexels API:** Fondos de cartas (hábitats) y eventos.
- **Vecteezy:** Iconografía vectorial para la UI (plumas, nidos, ataque).

Alojamiento de Datos

Supabase con JSON
SQLite en local

Manejo de Datos (Java + Spring Framework)

Se centrará en usar Java + Spring para el manejo de los datos y la información.

Aquí tienes el diagrama conceptual de lo que vamos a construir, seguido de la tabla definitiva que resume tu **Stack Reactivo y Asíncrono**.

El Stack 100% Asíncrono (WebFlux / Reactor)

Módulo / Tecnología	¿Por qué lo usamos? (La Ventaja)	¿Cómo se implementa? (La Práctica)
Spring WebFlux *(Servidor Netty)*	Es el motor central no bloqueante. Permite que un solo servidor maneje miles de partidas a la vez sin quedarse sin memoria RAM.	Se usa en lugar de Spring MVC (Tomcat). En tu código Java, en vez de devolver Carta, devuelves Mono<Carta> o Flux<Carta>.
RSocket *(Comunicación)*	Protocolo bidireccional más rápido que WebSockets. Soporta *Backpressure*: si el móvil va lento, ajusta el envío de datos para no colgar la app React.	Se habilita con spring-boot-starter-rsocket. Usas la anotación @MessageMapping para escuchar los movimientos (ej: "Atacar") del cliente.
Spring Data R2DBC *(Conexión DB)*	Es el driver asíncrono para PostgreSQL (Supabase). Evita que el servidor se quede "congelado"	Reemplaza a Hibernate/JPA. Creas repositorios reactivos (ReactiveCrudRepository) que devuelven Mono/Flux al interactuar con las tablas.

	esperando a que la base de datos lea el inventario.	
Jackson + R2DBC Converters *(Mapeo JSON)*	R2DBC es "crudo" y no entiende el tipo JSONB de Postgres mágicamente. Jackson traduce ese JSON a objetos Java al vuelo.	Creas una clase que implemente Converter<Json, MiCartaRecord> y usas ObjectMapper de Jackson para deserializar el string JSONB de Supabase.
Spring Data Redis Reactive *(Caché Marketplace)*	Para el Marketplace, consultar Supabase por cada búsqueda saturaría la red. Redis mantiene el catálogo en RAM para respuestas en submilisegundos.	Usas ReactiveRedisTemplate para guardar y leer las ofertas activas sin bloquear el hilo principal.
Redisson *(Distributed Locks)*	Evita el "doble gasto" o clonación de cartas en el Marketplace. Bloquea una carta a nivel de red entera si dos personas la compran a la vez.	Cuando un usuario le da a "Comprar", pides un lock (RLockReactive) a Redisson por el ID de la carta. Nadie más puede tocarla hasta que se libere.
Spring Security Reactive *(Auth)*	Valida que el usuario sea quien dice ser sin frenar el flujo de datos. Se integra directamente con el sistema Auth de Supabase.	Configuras un ReactiveJwtDecoder que verifica la firma del token enviado desde la WebView y guarda el usuario en el ReactiveSecurityContext.
Spring Kafka o **RabbitMQ** *(Event Broker)*	Saca el trabajo pesado fuera de la partida. (Ej: Repartir experiencia, guardar logs, dar recompensas al acabar un duelo).	Al acabar la partida, emites un evento PartidaTerminada. El hilo del jugador queda libre al instante, y otro microservicio procesa las recompensas.

Las Alternativas (El "Plan B")

Si durante el desarrollo veis que el modelo reactivo (Mono / Flux) hace que el código sea muy difícil de leer o testear, estas son las alternativas que sacrifican un poco de rendimiento puro por **facilidad de desarrollo**:

- **Alternativa a WebFlux (Motor): *Spring Web MVC + Virtual Threads (Java 21+).***
  - *Por qué:* Te permite escribir código tradicional paso a paso (síncrono y fácil de leer) pero con casi el mismo rendimiento masivo que WebFlux gracias a los hilos virtuales de Java.
- **Alternativa a RSocket (Comunicación): *WebSockets Clásicos (STOMP).***
  - *Por qué:* Es mucho más fácil de integrar con React y la WebView. Hay decenas de librerías listas para usar (como sockjs-client), mientras que RSocket requiere un poco más de configuración en el frontend.
- **Alternativa a R2DBC (Base de datos): *Spring Data JPA (Hibernate) + Hypersistence Utils.***
  - *Por qué:* R2DBC requiere que hagas tú mismo el mapeo de relaciones y JSONB. JPA/Hibernate te hace toda la "magia" de convertir las tablas y el JSON de Supabase en objetos Java con simples anotaciones, a costa de usar operaciones bloqueantes.

- **Alternativa a Kafka/Redis (Mercado simple): *Optimistic Locking en PostgreSQL.***
    - *Por qué:* Si no quieres pagar/mantener servidores extra de Redis o Kafka, puedes usar un simple campo `@Version` en Supabase. Si hay un conflicto de compra, la base de datos lanza una excepción, se cancela todo, y le muestras un error de "Carta ya vendida" al usuario.

Backend (Lo que usaremos)

- **Framework Core:** Spring Boot 3 (WebFlux).
- **Motor Asíncrono:** Project Reactor (Mono/Flux).
- **Comunicación Cliente-Servidor: RSocket** (para el juego en tiempo real) + REST (para configuración inicial).
- **Persistencia:** Spring Data R2DBC (Reactivo).
- **Base de Datos: Supabase (PostgreSQL)**. Uso intensivo de tipos `JSONB` para las cartas y `PostGIS` (si Supabase lo permite, o coordenadas simples) para la geolocalización.
- **Caché & Locks: Redis Reactive** (con Redisson para bloqueos distribuidos en el Marketplace).
- **Event Broker:** RabbitMQ o Kafka (para desacoplar la lógica de recompensas post-partida).
- **Seguridad:** Spring Security Reactive + JWT (validando contra Supabase Auth).

Implementación en el server

Comandos Server:

```
 2  sudo apt update && sudo apt upgrade -y

 3  sudo apt install python-is-python3 -y

 4  sudo apt install python3 python3-pip python3-venv -y

 5  python3 --version

 6  pip3 --version

 7  sudo apt install curl -y

 8  curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

 9  source ~/.bashrc

10  nvm install node

11  sudo apt install openjdk-21-jdk -y

12  sudo apt install docker.io docker-compose -y
```

```
13  sudo usermod -aG docker $USER
14  sudo usermod -aG docker $lubuntu
15  sudo apt install maven -y
```