📘 Manual de Arquitectura y Desarrollo: La Polla Virtual

Versión: 1.0 (Post-Limpieza)

Fecha: Febrero 2026

Estado: Producción (Modo Pagos Manuales)

---

1. 🏗️ Visión General (El Mapa del Edificio)

La aplicación utiliza una arquitectura moderna llamada **Monorepo**. Imagina una casa con dos habitaciones principales que comparten el mismo terreno:

1. **El Frontend (`apps/web`):** Es la "Cara Bonita". Lo que ven los usuarios en sus celulares.
    - **Tecnología:** Next.js (React).
    - **Dónde vive:** Vercel.
2. **El Backend (`apps/api`):** Es el "Cerebro". Donde se hacen los cálculos, se guardan datos y se validan reglas.
    - **Tecnología:** NestJS (Node.js).
    - **Dónde vive:** Railway.
3. **La Base de Datos:** Es la "Memoria".
    - **Tecnología:** PostgreSQL.
    - **Dónde vive:** Railway.

---

2. 🛠️ Tecnologías y Servicios Externos (APIs)

Para que la app no tenga que reinventar la rueda, nos conectamos con servicios expertos en tareas específicas. Aquí está la lista de "Proveedores":

Servicio	¿Para qué sirve?	Ubicación en Código
Wompi	Pasarela de pagos (Tarjetas/PSE). *Actualmente en*	`apps/api/src/payments`

	modo espera, código listo.	
Cloudinary	Almacenamiento de imágenes. Aquí se guardan las fotos de los comprobantes de pago y avatares.	`apps/api/src/upload`
Google OAuth	Permite iniciar sesión con Gmail sin crear contraseña.	`apps/api/src/auth/strategies/google.strategy.ts`
SMTP (Email)	Envío de correos (Recuperar contraseña, Bienvenida).	`apps/api/src/mail`
PostgreSQL	Base de datos relacional robusta.	Nube (Railway)

---

3. 🧠 El Cerebro: Backend (`apps/api`)

El backend está organizado por **Módulos**. Piensa en cada módulo como un departamento de una empresa.

Carpetas Clave en `apps/api/src`:

- `auth/` **(Departamento de Seguridad):**
  - **Qué hace:** Revisa si el usuario y contraseña son correctos. Genera un "Token" (JWT) que es como el carnet de identificación digital del usuario.
- `leagues/` **(Gestión de Ligas):**
  - **Qué hace:** Crea las pollas, genera códigos de acceso, y maneja quién pertenece a qué grupo.
- `matches/` **(Resultados Reales):**
  - **Qué hace:** Aquí el Admin registra "Colombia 2 - 0 Argentina".

    - **Magia:** Cuando un partido se marca como FINISHED, este módulo avisa al sistema de scoring para recalcular puntos.
- **predictions/ (Apuestas):**
    - **Qué hace:** Guarda lo que el usuario cree que va a pasar. Tiene un "candado" (time-lock.guard.ts) que impide apostar si el partido ya empezó.
- **scoring/ (La Calculadora):**
    - **Qué hace:** Compara matches (realidad) vs predictions (usuario) y asigna puntos (3 por exacto, 1 por ganador, etc.).
- **payments/ (La Caja):**
    - **Qué hace:** Maneja la lógica de cobro. Ahora mismo procesa la aprobación manual de los administradores.
- **database/entities/**: Son los planos de las tablas. Si quieres saber qué datos guardamos de un usuario, miras user.entity.ts.

---

4. 💅 La Cara: Frontend (apps/web)

El frontend usa la estructura de **App Router** de Next.js. Esto significa que las carpetas definen la URL de la página.

Carpetas Clave en apps/web/src:

- **app/ (Las Páginas):**
    - app/login/page.tsx → lapollavirtual.com/login
    - app/dashboard/page.tsx → Pantalla principal.
    - app/admin/... → Panel de control (Solo visible si eres Admin).
- **components/ (Piezas de LEGO):**
    - Aquí están los trozos reutilizables.
    - **Ejemplo:** MatchCard.tsx es la tarjetita que muestra los escudos de los equipos. Se usa cientos de veces, pero se programa una sola vez.
- **lib/ (Utilidades):**
    - Funciones aburridas pero útiles: formatear fechas, calcular porcentajes, conectarse a la API.

---

5. 🔄 Flujos Críticos (¿Cómo funciona por dentro?)

Si un desarrollador nuevo entra, esto es lo primero que debe entender:

A. El Flujo de Puntuación (Scoring)

1. Admin entra al Panel y pone el marcador final de un partido.
2. El Backend recibe el dato y busca **todas** las predicciones de ese partido.
3. Calcula puntos para cada usuario en segundo plano.
4. Actualiza la tabla league_participants con el nuevo puntaje total.
5. El usuario refresca su app y ve su nueva posición en la tabla.

B. El Flujo de Pago Manual (Actual)

1. Usuario sube foto del comprobante.
2. Frontend sube la foto a **Cloudinary** y obtiene una URL (link).
3. Frontend envía esa URL al Backend.
4. Backend crea una `Transaction` con estado `PENDING`.
5. Admin ve la foto en su panel y da clic en "Aprobar".
6. Backend cambia estado a `APPROVED` y activa al usuario en la liga.

---

6. 🛡️ Diccionario para Principiantes

Términos "raros" que encontrarás en el código y qué significan:

- **DTO (Data Transfer Object):** Es como un formulario de validación. Asegura que si el usuario envía su email, sea realmente un email y no un número.
- **Entity (Entidad):** Representación de una tabla de la base de datos en código (TypeScript).
- **Guard (Guardia):** Un portero de discoteca. Se pone antes de una función para decir: "¿Tienes permiso de Admin para entrar aquí?".
- **Migration (Migración):** Un archivo histórico de cambios en la base de datos. Si agregamos una columna nueva, se crea una migración para que la base de datos sepa cómo actualizarse.
- **Cron Job:** Tareas automáticas que se ejecutan solas cada cierto tiempo (ej: revisar partidos pendientes).

---

7. 🚀 Cómo arrancar el proyecto (Para Devs)

Si contratas a alguien mañana, solo tiene que hacer esto:

1. **Clonar:** Descargar el código.
2. **Instalar:** Ejecutar `npm install` (instala las librerías de `node_modules`).
3. **Variables:** Crear el archivo `.env` con las llaves (Database, Cloudinary, etc.).
4. **Correr:**
    - `npm run dev:api` (Prende el cerebro).
    - `npm run dev:web` (Prende la cara).

Nota Final: Este proyecto está limpio y estructurado profesionalmente. La lógica de negocio está separada de la visual, lo que lo hace escalable y seguro. ¡Tienes una base tecnológica muy sólida! 🏆