# Guía para Conectar el Backend a la Aplicación PetFlow

Hola equipo de backend,

Esta es una guía para construir el "cerebro" de nuestra aplicación PetFlow. La aplicación que ven (el frontend) ya está lista para conectarse y mostrar los datos reales que ustedes van a gestionar.

La idea es que ustedes construyan una API (un conjunto de direcciones web a las que nuestra app pueda hacerle peticiones) que reemplace todos los datos de prueba que tenemos actualmente.

Aquí les explicamos qué necesita la aplicación, pantalla por pantalla:

### 1. El Login de Usuario

\*   **\*\*Qué ve el usuario\*\***: La pantalla de inicio de sesión.

\*   **\*\*Archivo del frontend involucrado\*\***: `views/LoginView.tsx`.

\*   **\*\*Qué necesita hacer el backend\*\***: Crear una dirección web (un endpoint), por ejemplo `/api/login`, que reciba un email y una contraseña. Si son correctos, debe devolvernos una "llave de acceso" (un token JWT) para que sepamos que el usuario ha iniciado sesión.

### 2. El Dashboard Principal (la pantalla más importante)

\*   **\*\*Qué ve el usuario\*\***: La pantalla principal con el nivel de comida y agua, los horarios y el historial.

\*   **\*\*Archivo del frontend involucrado\*\***: `views/DashboardView.tsx`.

\*   **\*\*Qué necesita hacer el backend\*\***: Crear una dirección web, por ejemplo `/api/dashboard`, que cuando la app la consulte, le devuelva toda la información principal de una sola vez:

    \*   Los niveles actuales de los dispensadores (comida y agua, ej: 73% y 85%).

    \*   La lista de horarios programados (hora, cantidad, sustancia, si está activado o no).

    \*   El historial de las últimas veces que se dispensó comida o agua.

\*   **\*\*Nota\*\***: Actualmente, todos estos datos están como "datos falsos" en nuestro archivo `App.tsx`. El objetivo es que el backend sea la única fuente real de esta información.

### 3. Dispensar Comida y Agua Manualmente

\*   **\*\*Qué ve el usuario\*\***: El panel de "Dispensación Manual" en el Dashboard y en su propia pantalla.

\*   **\*\*Archivos del frontend involucrados\*\***: `components/ControlPanel.tsx` y `views/ManualDispenseView.tsx`.

\*   **\*\*Qué necesita hacer el backend\*\***: Crear una dirección, por ejemplo `/api/dispense`, que acepte el tipo de sustancia ('Food' o 'Water'), la cantidad y para qué mascota/grupo es. Cuando la app la "llame", el backend debe:

    1.  Guardar que se ha dispensado una nueva ración en la base de datos.

    2.  Restar esa cantidad del nivel total del contenedor correspondiente.

    3.  Devolverle a la app los nuevos niveles de comida y agua para que se actualicen las barritas de progreso.

### 4. Gestionar Horarios (Añadir, Ver y Activar/Desactivar)

\*   **\*\*Qué ve el usuario\*\***: La pantalla "Horarios" y los interruptores (switches) al lado de cada horario en el Dashboard.

\*   **\*\*Archivos del frontend involucrados\*\***: `views/SchedulesView.tsx` y `components/ScheduleList.tsx`.

\*   **\*\*Qué necesita hacer el backend\*\***:

    \*   **\*\*`GET /api/schedules`\*\***: Debe devolver una lista completa de todos los horarios del usuario.

    \*   **\*\*`POST /api/schedules`\*\***: Debe permitir crear un nuevo horario, recibiendo la hora, cantidad, mascota y tipo (comida o agua).

    \*   **\*\*`PATCH /api/schedules/:id/toggle`\*\***: Cuando el usuario mueva el interruptor, la app llamará a esta dirección con el ID del horario. El backend deberá cambiar el estado de ese horario (de activado a desactivado, o viceversa) en la base de datos.

### 5. El Planificador de Comidas con IA (Muy Importante)

\*   **\*\*Qué ve el usuario\*\***: La pantalla "Planificador IA" donde introduce los datos de su mascota.

\*   **\*\*Archivos del frontend involucrados\*\***: `views/MealPlannerView.tsx` y `services/geminiService.ts`.

\*   **\*\*Qué necesita hacer el backend\*\***: Por seguridad, la clave secreta de la IA de Google (API Key) no puede estar en nuestra aplicación visible. Por eso, el backend debe actuar como un intermediario seguro.

    1.  Crear una dirección, por ejemplo `/api/generate-plan`.

    2.  Nuestra app le enviará los datos de la mascota (peso, edad, etc.) a esa dirección.

    3.  El backend (que tendrá la clave secreta guardada de forma segura) le preguntará a la IA de Google.

    4.  Finalmente, el backend le enviará la respuesta completa de la IA (que incluye porciones de comida y recomendación de agua) a nuestra app.

\*   **\*\*Nota\*\***: La lógica que está ahora en `services/geminiService.ts` (el prompt, el schema de la respuesta, etc.) debe ser replicada en el backend.

### 6. Perfiles de las Mascotas

\*   **\*\*Qué ve el usuario\*\***: La pantalla "Perfiles", que muestra una lista de sus mascotas y permite añadir nuevas.

\*   **\*\*Archivo del frontend involucrado\*\***: `views/PetProfilesView.tsx`.

\*   **\*\*Qué necesita hacer el backend\*\***:

    \*   **\*\*`GET /api/pets`\*\***: Debe devolver la lista de todas las mascotas que le pertenecen al usuario que ha iniciado sesión.

    \*   **\*\*`POST /api/pets`\*\***: Debe permitir registrar una nueva mascota en la base de datos.

---

En resumen: Cada acción en la aplicación necesita "hablar" con una dirección web que ustedes crearán para leer o modificar los datos. El frontend ya está preparado para consumir esta API.