1.Identificar en el frontend los endpoints ya implementados en el backend y que están siendo utilizados en las distintas pantallas del sistema.

Mapeo de pantallas con los **endpoints** correspondientes:

- CrearEvento.jsx → **/api/eventos**

- SongVoting.jsx → **/api/canciones**, **/api/votos**

- FiltrosBusqueda.jsx → **/api/eventos**

- SelectorDeRol.jsx → **/api/usuarios**

- PresentCard.jsx → **/api/eventos/:id**

- FooterEmpresa.jsx → **/api/empresas/:id**

- FooterUsuario.jsx → **/api/usuarios/:id**

- Header.jsx → **/api/usuarios/me**

2.Para cada endpoint listado en el punto anterior, detallar el método HTTP, reglas de negocio, si requiere autenticación y las interfaces TypeScript de request y response.

Pantalla CrearEvento.jsx:

Endpoint: **/api/eventos**

Métodos y Reglas de Negocio:

- POST /api/eventos → Crear un nuevo evento (requiere datos básicos como nombre, fecha, lugar, empresa).
- GET /api/eventos → Listar todos los eventos disponibles.

Autenticación: POST requiere autenticación (empresa logueada). GET puede ser público.

Interfaces TypeScript:

```
// Request para crear evento
interface CreateEventoRequest {
  nombre: string;
  fecha: string;   // ISO date
  lugar: string;
  empresaId: number;
}
```

```
// Response de crear evento
interface CreateEventoResponse {
  id: number;
  nombre: string;
  fecha: string;
  lugar: string;
  empresaId: number;
}
```

Pantalla SongVoting.jsx:

Endpoint: **/api/canciones**

Métodos y Reglas de Negocio:

- GET /api/canciones → Listar canciones disponibles para votar.

Autenticación: Pública.

Interfaces TypeScript:

```
interface Cancion {
  id: number;
  titulo: string;
  artista: string;
}

type GetCancionesResponse = Cancion[];
```

Endpoint: **/api/votos**

Métodos y Reglas de Negocio:

- POST /api/votos → Registrar un voto de un usuario sobre una canción.

Autenticación: Requiere usuario logueado.

Interfaces TypeScript:

```
interface CreateVotoRequest {
  usuarioId: number;
  cancionId: number;
  eventoId: number;
}
```

```
interface CreateVotoResponse {
  id: number;
  usuarioId: number;
  cancionId: number;
  eventoId: number;
  fecha: string;
}
```

Pantalla FiltrosBusqueda.jsx:

Endpoint: **/api/eventos**

Métodos y Reglas de Negocio:

- GET /api/eventos?filtro=... → Filtrar eventos por criterios (ej: fecha, nombre, lugar).

Autenticación: Pública.

Interfaces TypeScript:

```
interface FiltroEventosRequest {
  nombre?: string;
  fecha?: string;
  lugar?: string;
}

type FiltroEventosResponse = CreateEventoResponse[];
```

Pantalla SelectorDeRol.jsx:

Endpoint: **/api/usuarios**

Métodos y Reglas de Negocio:

- POST /api/usuarios/login → Autenticación del usuario (selección de rol: empresa o usuario final).

Autenticación: Requiere credenciales.

Interfaces TypeScript:

```
interface LoginRequest {
  email: string;
```

```
  password: string;
}

interface LoginResponse {
  id: number;
  nombre: string;
  rol: "empresa" | "usuario";
  token: string;
}
```

Pantalla PresentCard.jsx:

Endpoint: **/api/eventos/:id**

Métodos y Reglas de Negocio:

- GET /api/eventos/:id → Obtener detalles de un evento específico.

Autenticación: Pública.

Interfaces TypeScript:

```
type GetEventoResponse = CreateEventoResponse & {
  canciones?: Cancion[];
  votos?: CreateVotoResponse[];
};
```

Componentes FooterEmpresa.jsx / FooterUsuario.jsx:

Endpoint: **/api/empresas/:id**

Métodos y Reglas de Negocio:

- GET /api/empresas/:id → Obtener datos de la empresa logueada.

Autenticación: Requiere autenticación (rol empresa).

Interfaces TypeScript:

```
interface Empresa {
  id: number;
  nombre: string;
  email: string;
}
```

```
type GetEmpresaResponse = Empresa;
```

Endpoint: **/api/usuarios/:id**

Métodos y Reglas de Negocio:

- GET /api/usuarios/:id → Obtener datos del usuario logueado.

Autenticación: Requiere autenticación (rol usuario).

Interfaces TypeScript:

```
interface Usuario {
  id: number;
  nombre: string;
  email: string;
}

type GetUsuarioResponse = Usuario;
```

Componente Header.jsx

Endpoint: **/api/usuarios/me**

Métodos y Reglas de Negocio:

- GET /api/usuarios/me → Verificar sesión activa y obtener datos del usuario logueado.

Autenticación: Requiere token.

Interfaces TypeScript:

```
type GetUsuarioMeResponse = Usuario & { rol: "empresa" | "usuario" };
```