Relevamiento de API rest

1) Por cada pantalla de su Frontend, identificar y listar los endpoints necesarios

Modificar Datos del Socio:

Funcionalidad:

- Cargar datos iniciales: cuando se abra la pantalla el frontend va a tener que obtener los datos actuales del socio para rellenar los campos
- Actualizar datos: cuando se apriete el botón “Guardar cambios”, se mandan los cambios al backend

Endpoints necesarios:

- Endpoint para obtener datos del socio
- Endpoint para actualizar datos del socio

Pagar cuotas (Socio):

Funcionalidad:

- Se visualiza el listado de cuotas del socio, las pagadas, las vencidas y las pendientes. Las vencidas tienen un botón para adjuntar el comprobante de pago y que se suba.

Endpoints necesarios:

- Endpoint para cargar la lista de cuotas
- Endpoint para enviar comprobante

HomePage del Usuario (HomePageUser):

Funcionalidad:

- Mostrar datos del usuario logueado (nombre)
- Acceder a: **Reservar cancha**, **Ver/Comprar entradas**, **Modificar perfil**, **Pagar cuota**.

Endpoints necesarios:

- Endpoint para obtener datos del socio
- Obtener estado de la cuota del socio
- Obtener accesos habilitados según el estado del socio

Iniciar Sesión:

Funcionalidad:

- Validar cuenta del socio (email/usuario y contraseña

Endpoints necesarios:

- Endpoint para validar la cuenta del socio

Registrarse:

Funcionalidad:

- Registrar nuevo socio a la Base de Datos (necesita validacion)

Endpoints necesarios:

- Endpoint para registrar socio

Reservar Cancha (Admin)

Funcionalidad:

- Elegir un deporte para luego ver los horarios disponibles de las canchas
- Seleccionar un turno y cuando se apriete el boton “Reservar” se podra completar los campos con los datos del Socio.
- Cuando se apriete el boton “Confirmar Reservar” se enviaran los datos al backend
- Se permite cancelar la reserva

Endpoints necesarios:

- Obtener turnos disponibles
- Registrar una reserva
- Cancelar una reserva

Reservar Cancha (Socio)

Funcionalidad:

- Elegir un deporte para luego ver los horarios disponibles de las canchas
- Seleccionar un turno y cuando se aprete el boton “Reservar” el frontend va a tener q obtener los datos del Socio para generar la reserva
- Cuando se apriete el boton “Confirmar Reservar” se enviaran los datos al backend
- Se permite cancelar la reserva

Endpoints necesarios:

- Obtener turnos disponibles
- Registrar una reserva
- Cancelar una reserva

Actividades (Admin)

Funcionalidad:

- Ver actividades registradas en el sistema
- Permite registrar nuevas actividades apretando el boton “Agregar”

endpoints:

- Obtener todas las actividades
- Crear una nueva actividad

Compra de Entradas (Socio)

Funcionalidad:

- Ver listado de entradas del socio (activas y pasadas).
- Ver los próximos eventos disponibles.
- Comprar entradas disponibles.
- Filtrar eventos según estado (activos/pasados).
- Ver detalles del evento (fecha, hora, lugar, ocupación, precio).

Endpoints necesarios:

- Obtener las entradas del socio autenticado.
- Obtener la lista de eventos próximos disponibles para la compra.
- Comprar una entrada para un evento.

Administración de Eventos (Admin)

Funcionalidad:

- Obtener todos los eventos (activos y completados).
- Crear nuevo evento
- Editar un evento existente
- Eliminar evento
- Registrar venta de entrada
- Obtener estadísticas generales (total de eventos, eventos activos, ventas totales y ocupación promedio)
- Filtrar por estado.
- Ver detalle de ocupación y recaudación por evento.

Endpoints necesarios:

- Obtener todos los eventos (activos y completados).
- Crear nuevo evento

- Editar un evento existente
- Eliminar evento
- Registrar venta de entrada
- Obtener estadísticas generales (total de eventos, eventos activos, ventas totales y ocupación promedio)

2)Por cada endpoint, definir:

a)Método HTTP, URL, reglas de negocio

b)Identificar qué endpoints necesitan autenticación

c)Definir interfaces TypeScript para request/response

Modificar Datos del Socio:

Endpoint 1: Obtener datos del socio

a)

- **Método HTTP:** GET
- **URL:** /api/socios/{id}
- **Reglas de negocio**:
  - Devolver solo los datos del socio especificado
  - Verificar que la fecha de nacimiento esté escrita como dd/mm/aaaa
  - Si no existe el socio, devolver error

b) **Si, necesita autenticación**: solo el socio o el admin puede ver o editar datos personales del socio

c)GetSocioRequest : es un método get que pide los datos del socio al back

```
interface GetSocioRequest {

}
```

GetSocioResponse: devuelve los datos del socio

```
interface ActualizarSocioRequest {
  nombre?: string;
  apellido?: string;
```

```
  dni?: string;
  email?: string;
  fechaNacimiento?: string;  // en formato 'dd/mm/aaaa'
  pais?: 'Argentina' | 'Bolivia' | 'Brasil' | 'Chile' | 'Colombia' |
'Costa Rica' | 'Cuba' | 'República                    Dominicana' |
'Ecuador' | 'El Salvador' | 'Guatemala' | 'Honduras' | 'México' |
'Nicaragua' | 'Panamá' | 'Paraguay' | 'Perú' | 'Uruguay' | 'Venezuela';
  sexo?: 'masculino' | 'femenino' | 'otro';
  fotoCarnet?: File;
}
```

Endpoint 2: Actualizar datos del socio

a)

- **Método HTTP**: PATCH, para actualización parcial, si solo se cambian algunos campos
- **URL**: /api/socios/{id}
- **Reglas de negocio:**
    - Validar campos: DNI único y válido (formato numérico), mail válido, fecha de nacimiento en el pasado, país y sexo de listas predefinidas
    - Manejar subida de foto: Como es un archivo, usaremos multipart/form-data; almacenar en servidor/cloud y guardar URL
    - Se actualizará solo si hay cambios
    - Por seguridad se deberá saber quién editó qué
    - Retornar el socio actualizado o confirmación de que se ha editado el socio
    - Aviso de errores, como por ejemplo que el email ingresado ya está en uso"

b) **Sí, necesita autenticación.** Solo el socio o admin pueden modificar.

c) ActualizarSocioRequest: Los datos editados

```
interface ActualizarSocioRequest {
  nombre?: string;
  apellido?: string;
  dni?: string;
  email?: string;
  fechaNacimiento?: string;
  pais?: 'Argentina' | 'Bolivia' | 'Brasil' | 'Chile' | 'Colombia' | 'Costa
Rica' | 'Cuba' | 'República Dominicana' | 'Ecuador' | 'El Salvador' |
'Guatemala' | 'Honduras' | 'México' | 'Nicaragua' | 'Panamá' | 'Paraguay' |
'Perú' | 'Uruguay' | 'Venezuela';
  sexo?: 'masculino' | 'femenino' | 'otro';
  fotoCarnet?: File;
```

```
}
```

ActualizarSocioResponse: Confirmación y muestra los datos actualizados cuando el success sea true

```
interface ActualizarSocioResponse {
  success: boolean;
  socio?: {
    id: number;
    nombre: string;
    apellido: string;
    dni: string;
    email: string;
    fechaNacimiento: string;
    pais: 'Argentina' | 'Bolivia' | 'Brasil' | 'Chile' | 'Colombia' | 'Costa
Rica' | 'Cuba' | 'República Dominicana' | 'Ecuador' | 'El Salvador' |
'Guatemala' | 'Honduras' | 'México' | 'Nicaragua' | 'Panamá' | 'Paraguay' |
'Perú' | 'Uruguay' | 'Venezuela';
    sexo: 'masculino' | 'femenino' | 'otro';
    fotoCarnet?: string;  // URL de la foto
  };
  message?: string;
}
```

Pagar cuotas (Socio):

Endpoint 1: cargar la lista de cuotas

a)

- **Método HTTP**: GET
- **URL**: /api/socios/{id}/cuotas
- **Reglas de negocio**:
    - Retornar cuotas del socio
    - Ordenar por fecha
    - Mostrar estados

b)Sí, necesita autenticación. Solo el socio y puede ver su listado de cuotas en esta pantalla.

c) GetCuotasRequest: pide al back una lista de cuotas

```
interface GetCuotasRequest {
  mes?: string;
}
```

GetCuotasResponse: devuelve una lista de cuotas

```
interface GetCuotasResponse {
  cuotas: {
    nroCuota: number;
    mes: string;
    vencimiento: string;
    monto: number;
    estado: 'Aprobada' | 'Vencida' | 'En revisión';
  }[];
  message?: string;
}
```

Endpoint 2: enviar comprobante

a)

- **Método HTTP**: POST
- **URL**: /api/cuotas/{cuotaId}/comprobante
- **Reglas de negocio**:
    - Validar archivo (debe ser PDF/JPG/PNG y pesar menos de 5MB)
    - Usar multipart/form-data
    - Se debe actualizar el estado a "En revisión" (admin aprueba después para "Aprobada")
    - Almacenar URL del comprobante.

b) **Sí, necesita autenticación.** Solo el socio puede enviar comprobante

c) EnviarComprobanteRequest: Se envía al back el archivo del comprobante

```
interface EnviarComprobanteRequest {
  comprobante: File;
}
```

EnviarComprobanteResponse: El back devuelve una confirmación de éxito o error.

```
interface EnviarComprobanteResponse {
  success: boolean;
  message?: string;     //opcional porque solo salta mensaje cuando no se haya podido cargar
}
```

HomePage del Usuario (HomePageUser):

Endpoint para obtener datos del socio

a)

- **Método HTTP:** GET
- **URL:** /api/socios/{id}
- **Reglas de negocio**:
  - Devolver solo los datos del socio especificado
  - Verificar que la fecha de nacimiento esté escrita como dd/mm/aaaa
  - Si no existe el socio, devolver error

b) **Si, necesita autenticación**: solo el socio puede ver o editar datos personales del socio

c)

```
interface GetSocioRequest {
  // No hay body, solo ID en la URL
}
interface GetSocioResponse {
  id: number;
  nombre: string;
  apellido: string;
  email: string;
  fechaNacimiento: string; // 'dd/mm/aaaa'
  pais: string;
  sexo: 'masculino' | 'femenino' | 'otro';
  cuotaAlDia: boolean;
  message?: string; // Para errores
}
```

Obtener estado de la cuota del socio

a)

- **Método HTTP:** GET
- **URL:** /api/socios/{id}/estado-cuota
- **Reglas de negocio**:
  - Retornar si el socio tiene la cuota al día.
  - Si la cuota está vencida, incluir fecha de vencimiento y monto adeudado.
  - Si tiene pagos vencidos, no puede reservar cancha, inscribirse o participar en eventos

b) **Si, necesita autenticación**: solo el socio o el administrador puede consultar este estado.

c)

```
interface GetEstadoCuotaRequest {
  // Solo ID en la URL
}

interface GetEstadoCuotaResponse {
```

```
  cuotaAlDia: boolean;
  fechaVencimiento?: string; // 'dd/mm/aaaa' si está vencida
  montoAdeudado?: number; // en ARS
  message?: string; // Para errores
}
```

Obtener accesos habilitados según el estado del socio

a)
- **Método HTTP:** GET
- **URL:** /api/socios/{id}/accesos
- **Reglas de negocio**:
  - Retornar un array con las acciones habilitadas según el estado del socio.

b) **Si, necesita autenticación**: Sí, solo el socio o el administrador.

c)
```
interface GetAccesosRequest {}
interface GetAccesosResponse {
  accesos: {
    nombre: string; // Ej: 'Reservar Cancha'
    habilitado: boolean;
    motivo?: string; // Ej: 'Cuota vencida'
  }[];
}
```

ADMIN:

Cuotas pagadas por el Socio comprobadas por el Admin (CuotasAdminPage):

Funcionalidad:

- Ver el listado de cuotas de todos los socios.
- Filtrar por estado (**Todas**, **Aprobada**, **Pendiente**, **Rechazada**).
- Buscar cuotas por nombre del socio.
- Ver el comprobante de pago asociado a una cuota.

- Cambiar el estado de un comprobante (aprobar o rechazar).

Endpoints necesarios:

- Endpoint para obtener lista de cuotas de todos los socios
- Endpoint para obtener comprobante de una cuota específica
- Endpoint para actualizar estado de una cuota

Obtener lista de cuotas de todos los socios

a)

- **Método HTTP:** GET
- **URL:** /api/cuotas
- **Reglas de negocio**:
  - Retornar cuotas con información: ID, socio, estado, URL de comprobante (si existe).
  - Permitir parámetros de filtrado por estado y búsqueda por nombre.
  - Ordenar por fecha de vencimiento o por estado (opcional).

b) **Si, necesita autenticación**: solo el administrador puede acceder a esta informacion

c)

```
interface GetCuotasAdminRequest {
  estado?: 'Aprobada' | 'Pendiente' | 'Rechazada';
  nombre?: string; // filtro de búsqueda
}

interface GetCuotasAdminResponse {
  cuotas: {
    id: number;
    socioNombre: string;
    estado: 'Aprobada' | 'Pendiente' | 'Rechazada';
    comprobanteUrl?: string;
    avatar?: boolean;
 }[];
}
```

Obtener comprobante de una cuota específica

a)

- **Método HTTP:** GET
- **URL:** /api/cuotas/{id}/comprobante
- **Reglas de negocio**:
  - Verificar que la cuota tenga comprobante cargado.

- Devolver el archivo en formato PDF/JPG/PNG o la URL del recurso.
- Solo el administrador puede acceder a cualquier comprobante.

b) **Si, necesita autenticación**: solo el administrador puede acceder a esta información

c)

```
interface GetComprobanteRequest {
  id: number;
}

interface GetComprobanteResponse {
  comprobanteUrl: string;
  formato: 'PDF' | 'JPG' | 'PNG';
  fechaCarga: string; // 'dd/mm/aaaa'
}
```

Obtener comprobante de una cuota específica

a)

- **Método HTTP:** PATCH
- **URL:** /api/cuotas/{id}
- **Reglas de negocio**:
  - Solo el administrador puede cambiar el estado.
  - Los estados posibles son: **Aprobada**, **Rechazada**.
  - Guardar fecha y usuario que realizó el cambio.
  - Si se rechaza, permitir incluir un mensaje de motivo.
  - Notificar al socio el cambio de estado.

b) **Si, necesita autenticación**: solo el administrador puede acceder a esta información

c)

```
interface UpdateEstadoCuotaRequest {
  estado: 'Aprobada' | 'Rechazada';
  motivo?: string;
}

interface UpdateEstadoCuotaResponse {
  id: number;
  estado: 'Aprobada' | 'Rechazada';
  fechaCambio: string; // 'dd/mm/aaaa'
  cambiadoPor: string; // nombre del admin
```

```
}
```

Cuotas pagadas por el Socio comprobadas por el Admin (CuotasAdminPage):

Funcionalidad:

- Visualizar el comprobante subido por un socio para el pago de una cuota.
- Ver detalles asociados: nombre del socio, mes correspondiente, monto.
- Aprobar o rechazar el comprobante en cualquier momento (no solo si está pendiente).

Endpoints necesarios:

- Endpoint para obtener datos y comprobante de una cuota específica
- Endpoint para actualizar estado de la cuota (aprobado/rechazado)

Obtener datos y comprobante de una cuota específica

a)

- **Método HTTP:** GET
- **URL:** /api/cuotas/{id}
- **Reglas de negocio**:
  - Retornar información de la cuota: socio, mes, monto, estado, URL del comprobante.
  - El comprobante puede visualizarse en cualquier estado (Pendiente, Aprobada o Rechazada).
  - Si no existe el comprobante, devolver un mensaje indicando que no fue cargado.

b) **Si, necesita autenticación**: solo el administrador puede acceder a esta información

c)

```
interface GetComprobanteDetalleRequest {
  id: number;
}

interface GetComprobanteDetalleResponse {
  id: number;
  socioNombre: string;
  mes: string; // Ej: 'Junio'
```

```
  monto: number;
  estado: 'Aprobada' | 'Pendiente' | 'Rechazada';
  comprobanteUrl?: string;
  fechaCarga?: string; // 'dd/mm/aaaa'
  message?: string; // Para errores
}
```

Obtener datos y comprobante de una cuota específica

a)

- **Método HTTP:** PATCH
- **URL:** /api/cuotas/{id}
- **Reglas de negocio**:
  - Estados posibles: **Aprobada**, **Rechazada**.
  - Guardar fecha y usuario (admin) que realizó el cambio.
  - No hay restricción de que el comprobante esté en estado pendiente para modificarlo.
  - Si se rechaza, se puede incluir un motivo.
  - Notificar al socio del cambio vía email o sistema de notificaciones.

b) **Si, necesita autenticación**: solo el administrador puede acceder a esta información

c)

```
interface UpdateEstadoComprobanteRequest {
  estado: 'Aprobada' | 'Rechazada';
  motivo?: string;
}

interface UpdateEstadoComprobanteResponse {
  id: number;
  estado: 'Aprobada' | 'Rechazada';
  fechaCambio: string; // 'dd/mm/aaaa'
  cambiadoPor: string; // nombre del admin
}
```

Registrarse:

Endpoint para registrar socio

a)
- **Método HTTP:** POST
- **URL:** /api/registro
- **Reglas de negocio**:
  - Cada cuenta tiene su dni y email únicos

b) **Si, necesita autenticación**: No debe existir una cuenta con el mismo dni ni email

c)
```
interface RegistroRequest{
    nombre: string
    apellido: string
    dni: number
    password: string
}

interface RegistroResponse{
    estadoIngreso: 'ingresoExitoso' | 'ingresoFallido'
}
```

Iniciar sesión:

Endpoint para validar la cuenta del socio

a)
- **Método HTTP:** POST
- **URL:** /api/
- **Reglas de negocio**:
  - El usuario debe existir

b) **Si, necesita autenticación**: La contraseña de la cuenta ingresada debe coincidir con la guardada en la base de datos

c)
```
interface LoginRequest {
  email: string;
  password: string;
}


interface LoginResponse {
```

```
  rol: 'socio' | 'admin';
}
```

Endpoint para Obtener turnos disponibles (Admin)

a)

- **Método HTTP:** GET
- **URL:** /api/turnos
- **Reglas de negocio**:
  - Fechas a partir del dia actual
  - El deporte debe estar registrado en el sistema
  - Se devuelve el listado completo de turnos para esa fecha (de 10:00 a 22:00), con estado de disponibilidad.

b) Si, necesita autenticacion. Solo el admin puede acceder a esta pantalla

c)

```
interface Turno {
  hora: string;
  disponible: boolean;
}

interface ObtenerTurnosResponse {
  fecha: string;
  deporte: string;
  turnos: Turno[];
}
```

Endpoint para Registrar una reserva (Admin)

a)

- **Método HTTP:** POST
- **URL:** /api/turnos/reservar
- **Reglas de negocio**:
  - No se pueden reservar turnos en fechas pasadas
  - No se puede reservar un turno ya ocupado
  - El dni debe tener formato numérico válido
  - El nombre y apellido no pueden estar vacíos.
  - Se registra la reserva con estado “activa” y se marca el turno como ocupado.

b) Si, necesita autenticacion. Solo el admin puede acceder a esta pantalla.

c)

```
interface ReservaRequest {
```

```
  fecha: string;
  hora: string;
  deporte: string;
  socio: {
    nombre: string;
    apellido: string;
    dni: string;
  };
}

interface ReservaResponse {
  reservaId: string;
  mensaje: string;
}
```

Endpoint para Cancelar una reserva (Admin)

a)

- **Método HTTP:** DELETE
- **URL:** /api/turnos/reservar/{id}
- **Reglas de negocio**:
    - Solo pueden cancelarse futuras reservas
    - Al cancelar, el turno vuelve a marcarse como “disponible”

b) Si, necesita autenticación. Solo el admin puede acceder a esta pantalla.
c)

```
type CancelarReservaParams = {
  id: string; // Id resreva
};

interface CancelarReservaResponse {
  mensaje: string;
  reservaCancelada: {
    id: string;
    fecha: string;
    hora: string;
    deporte: string;
    socioId?: string;
  };
}
```

Endpoint para Obtener turnos disponibles por fecha y deporte (Socio)

a)

- **Método HTTP:** GET
- **URL:** /api/turnos
- **Reglas de negocio**:
    - Fechas a partir del dia actual
    - El deporte debe estar registrado en el sistema
    - Se devuelve el listado completo de turnos para esa fecha (de 10:00 a 22:00), con estado de disponibilidad
    - Con respecto a los turnos ya reservados, el sistema no muestra los datos de quien lo reservo

b) Si, necesita autenticacion. Solo el socio puede acceder a esta pantalla

c)

```
interface Turno {
  hora: string;
  disponible: boolean;
}

interface ObtenerTurnosResponse {
  fecha: string;
  deporte: string;
  turnos: Turno[];
}
```

<u>Endpoint para Registrar una reserva (Socio)</u>

a)

- **Método HTTP:** POST
- **URL:** /api/turnos/reservar
- **Reglas de negocio**:
    - No se pueden reservar turnos en fechas pasadas
    - No se puede reservar un turno ya ocupado
    - Se registra la reserva con estado “activa” y se marca el turno como ocupado.
    - El backend debe registrar la reserva y vincularla al respectivo socio.

b) Si, necesita autenticacion. Solo el Socio puede acceder a esta pantalla

c)

```
interface ReservaTurnoRequest {
  deporte: string;
  fecha: string;
  hora: string;
}
```

```
interface ReservaTurnoResponse {
  reservaId: string;
  mensaje: string;
}
```

Endpoint para Cancelar una reserva (Socio)

a)

- **Método HTTP:** DELETE
- **URL:** /api/turnos/reservar/{id}
- **Reglas de negocio**:
  - Solo pueden cancelarse futuras reservas
  - Solo pueden cancelarse reservas propias del socio
  - Al cancelar, el turno vuelve a marcarse como “disponible”

b) Si, necesita autenticación. Solo el socio puede acceder a esta pantalla.
c)

```
type CancelarReservaParams = {
  id: string;
};

interface ReservaCancelada {
  id: string;
  canchaId: string;
  socioId: string;
  fecha: string;
  hora: string;
  estado: 'cancelada';
}

interface CancelarReservaResponse {
  mensaje: string;
  reserva: ReservaCancelada;
}
```

Endpoint para Obtener todas las actividades

- **Método HTTP:** GET
- **URL:** /api/actividades
- **Reglas de negocio**:
  - Devuelve una lista de actividades

b) Si necesita autenticacion, solo el admin puede acceder a esta pantalla

c)

```
interface Actividad {
  id: number;
  nombre: string;
}

type ObtenerActividadesResponse = Actividad[];
```

Endpoint para Crear una nueva actividad

- **Método HTTP:** POST
- **URL:** /api/actividades
- **Reglas de negocio**:
    - El campo no puede estar vacio
    - El campo debe ser valido (alfabetico)
    - El nombre no debe repetirse

b) Si, necesita autenticacion, solo el admin puede realizar esta accion

c)

```
interface CrearActividadRequest {
  nombre: string;
}

interface CrearActividadResponse {
  id: number;
  nombre: string;
}
```

Compra de Entradas (Socio):

Endpoint Obtener las entradas del socio.

a)

- **Método HTTP:** GET
- **URL:** /api/entradas/
- **Reglas de negocio:**
    - Solo se deben devolver las entradas del socio autenticado.
    - Puede incluir un filtro opcional por estado (activa, usada, cancelada).
    - El frontend las muestra en tarjetas según el estado y fecha.

b) Necesita autenticación. Solo el socio puede acceder a esta pantalla

c)

```
interface Entrada {
  id: string;
  eventoId: string;
  nombreEvento: string;
  fecha: string;
  hora: string;
  cantidad: number;
  precioUnitario: number;
  total: number;
  estado: 'activa' | 'usada' | 'cancelada';
  fechaCompra: string;
  comprador: Comprador;
  categoria: string;
  ubicacion: string;
  codigoEntrada: string;
}

interface Comprador {
  nombre: string;
  apellido: string;
  dni: string;
  email?: string;
  telefono?:string;
}

para el filtro:
interface ObtenerEntradasRequest {
  estado?: 'activa' | 'usada' | 'cancelada';
}

type ObtenerEntradasResponse = Entrada[];
```

<u>Endpoint Obtener la lista de eventos próximos disponibles para la compra</u>
a)

- **Método HTTP:** GET
- **URL:** /api/entradas/proximos
- **Reglas de negocio:**
    - Mostrar solo eventos con fecha futura y capacidad disponible (entradasVendidas < capacidad).
    - Ordenar por fecha ascendente.

- Mostrar descripción, ocupación actual, y precio.

b) Necesita autenticación. Solo los socios logueados pueden acceder a la sección de entradas

c)

```
interface Evento {
  id: number;
  nombre: string;
  fecha: string;
  horaInicio: string;
  horaFin: string;
  capacidad: number;
  precioEntrada: number;
  entradasVendidas: number;
  categoria: string;
  ubicacion: string;
  descripcion: string;
}
```

Endpoint Comprar una entrada para un evento

a)

- **Método HTTP:** POST
- **URL:** /api/entradas/comprar
- **Reglas de negocio:**
  - Validar que haya disponibilidad suficiente antes de confirmar la compra.
  - Generar automáticamente codigoEntrada y fechaCompra.
  - Calcular total como cantidad * precioEntrada.
  - Estado inicial: 'activa'.

b)Necesita autenticación.

c)

```
interface CompraRequest {
  eventoId: number;
  cantidad: number;
  comprador: {
    apellido: string;
```

```
    nombre: string;
    dni: string;
    email?: string;
    telefono?: string;
  };
}

interface CompraResponse {
  mensaje: string;
  entrada: Entrada;
}
```

<u>Administración de Eventos (Admin)</u>

Endpoint Obtener todos los eventos (activos y completados).
a)

- **Método HTTP:** GET
- **URL:** /api/admi/eventos
- **Regla de negocio:**

b) Necesita autenticación.

c)

```
interface Evento {
  id: number;
  nro: string;
  nombre: string;
  fecha: string;
  horaInicio: string;
  horaFin: string;
  capacidad: number;
  precioEntrada: number;
  entradasVendidas: number;
  montoTotal: number;
  ubicacion: string;
  descripcion: string;
  estado: 'activo' | 'completado';
}

type ObtenerEventosResponse = Evento[];
```

Endpoint Crear nuevo evento

a)

- **Método HTTP:** POST
- **URL:** /api/admin/eventos
- **Regla de negocio:**
  - Validar que la fecha no sea pasada.
  - Validar que capacidad y precioEntrada sean positivos.
  - El campo nro debe ser único.

b) Necesita autenticación

c)

```
interface NuevoEventoRequest {
  nro: string;
  nombre: string;
  fecha: string;
  horaInicio: string;
  horaFin: string;
  capacidad: number;
  precioEntrada: number;
  ubicacion?: string;
  descripcion?: string;
}

interface EventoResponse extends Evento {}
```

Endpoint Editar un evento existente

a)

- **Método HTTP:** PUT
- **URL**: /api/admin/eventos/:id
- **Regla de negocio:**
  - No se puede modificar un evento con estado 'completado'.
  - Validar que la fecha no sea pasada.
  - Validar que capacidad y precioEntrada sean positivos.

b) Necesita autenticación.

c)

```
interface EditarEventoRequest extends NuevoEventoRequest {}
```

Endpoint Eliminar evento

a)

- **Método HTTP:** DELETE
- **URL:** /api/admin/eventos/:id
- **Regla de negocio:**
  - No se puede eliminar un evento si ya tiene entradas vendidas.
  - Solo eventos en estado 'activo' pueden eliminarse.

b) Necesita autenticación.

Endpoint Registrar venta de entrada

a)

- **Método HTTP:** POST
- **URL:** /api/admin/eventos/:id/ventas
- **Regla de negocio:**
  - Validar que haya capacidad suficiente antes de registrar venta.
  - Actualizar entradasVendidas y montoTotal.

b) Necesita autenticación.

c)
```
interface VentaManualRequest {
  cantidad: number;
  comprador: {
    apellido: string;
    nombre: string;
    dni: string;
    email?: string;
    telefono?: string;
  };
}

interface VentaManualResponse {
  mensaje: string;
  entradasVendidas: number;
  montoTotal: number;
}
```

Endpoint Obtener estadísticas generales

a)

- **Método HTTP:** GET

- **URL:** /api/admin/eventos/estadísticas
- **Regla de negocio:**

b) Necesita autenticación

c)
```
interface EstadisticasEventos {
  totalEventos: number;
  eventosActivos: number;
  eventosCompletados: number;
  totalVentas: number;
  totalEntradasVendidas: number;
  promedioOcupacion: number;
}
```