

# Examen DWEC Dual- Gestión de Eventos



Desarrollo de una aplicación web de gestión de eventos utilizando React y una API REST para la gestión de eventos. La aplicación permitirá ver, filtrar, crear, editar y eliminar eventos, implementando un sistema de autenticación para proteger ciertas operaciones.

## Ejercicios y Evaluación

### 1. 🗝️ Inicio (/)

- Deberemos implementar un esquema web como el siguiente:

- Las card muestran información de los eventos que tenemos en nuestro backend.

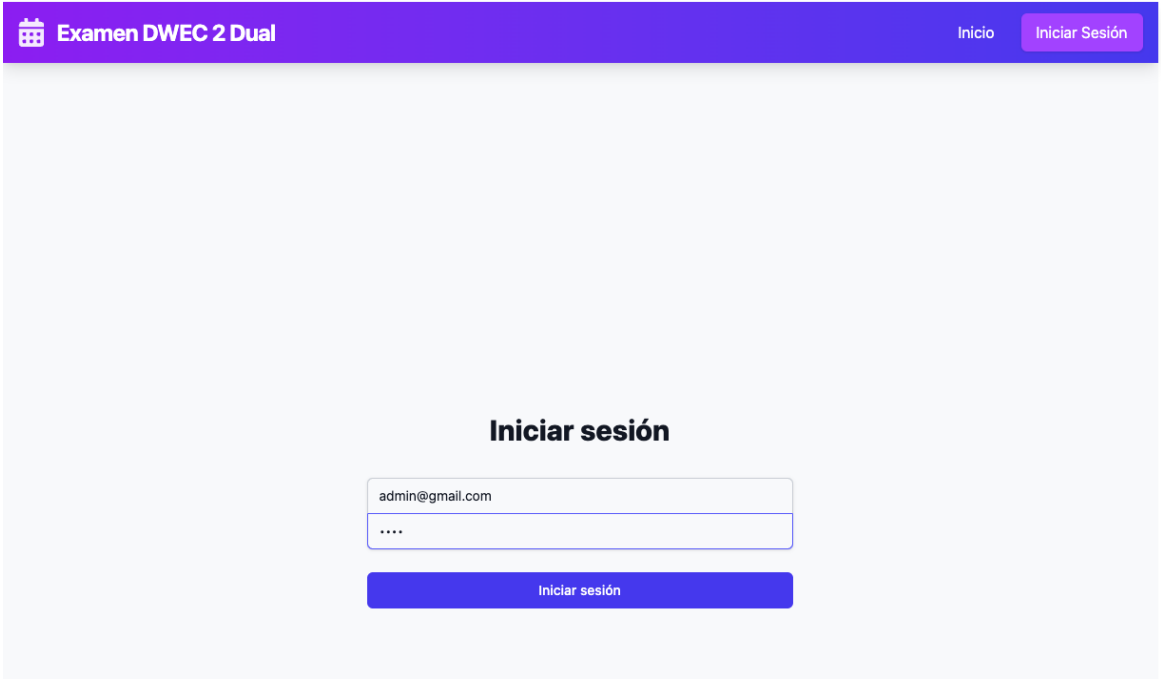
### 2. 🗝️ Configuración Inicial y Autenticación (0,5 puntos)

#### Objetivos

- Implementar sistema de autenticación completo (sólo Login)
- Gestionar estado global de autenticación

- Manejar persistencia de sesión a través del localStorage almacenando el JWT obtenido en el localStorage

Resultado Esperado



Página de login con validación y mensajes de error

2. 🛣️ Sistema de Rutas (1 puntos)

Rutas y Componentes Requeridos:

Ruta	Componente	Descripción	Protegida
/	Home	Página principal con listado de eventos	No
/login	Login	Página de inicio de sesión	No
/events/new	EventForm	Crear nuevo evento	Sí
/events/edit/:id	EventForm	Editar evento existente	Sí

Componentes de Enrutamiento:

- Implementar ProtectedRoute para rutas privadas
- RootLayout con:
  - Header que muestre el título y el icono de eventos
  - Navegación condicional según estado de autenticación

- Footer con copyright y nombre del autor
- Área principal para el contenido (Outlet)

**Resultado Esperado** cuando estemos logueados en el sistema

*Navegación con usuario autenticado*

### 3. 🌐 Gestión del Estado Global (2 puntos)

**EventContext** debe proporcionar:

- Estado de eventos: array de eventos
- Métodos CRUD completos:

```
fetchEvents(): Promise<void> // Obtener todos los eventos
addEvent(event: Event): Promise<Event> // Crear evento
updateEvent(id: string, event: Event): Promise<Event> //
Actualizar evento
deleteEvent(id: string): Promise<void> // Eliminar evento
```

- Manejo de headers de autenticación
- Persistencia del token

- Manejo de errores centralizado

#### 4. 📅 Listado de Eventos (1 puntos)

##### Componente EventCard:

- Mostrar nombre, fecha, tipo y valoración
- Formato de fecha en español
- Botones de edición/eliminación condicionales si estamos logueados o no

##### Página Home:

- Grid responsive de eventos
- Sección de filtros
- Mensaje cuando no hay eventos
- Botón de "Nuevo Evento" si está autenticado
- Botón "Iniciar sesión para editar" si no está autenticado

#### 5. 🔍 Sistema de Filtrado (2 puntos)

##### Hook useEventFilters debe implementar al menos 3 filtros:

El hook debe manejar un objeto filter con la siguiente estructura:

```
// Estado del filtro
{
  name: "",           // Texto para filtrar por nombre
  type: "",           // Tipo de evento (conferencia, concierto, taller)
  date: "",           // Fecha en formato YYYY-MM-DD
  sortBy: "newest"    // "newest" o "oldest" para ordenar por fecha
}
```

El hook debe retornar:

```
{
  filteredEvents, // Array de eventos filtrados
}
```

```
filter, // Estado actual del filtro
setFilter, // Función para actualizar los filtros
clearFilters; // Función para resetear los filtros
}
```

### Funcionalidades Requeridas:

- Filtrado por nombre (case insensitive)
- Filtrado por tipo de evento
- Filtrado por fecha exacta
- Ordenamiento por fecha (más reciente/más antiguo)
- Filtros combinados (todos los filtros deben funcionar simultáneamente)
- Función para resetear filtros

**Lista de Eventos** Iniciar sesión para editar

Buscar por nombre:

Tipo de evento: 

- ✓ Todos
- Conferencia
- Concierto
- Taller

Fecha:

Ordenar por fecha:

Limpiar filtros

-----Nuevo ISAIAS ----- ★ 2/5

Conferencia de Educación ★ 3/5

Conferencia de Educación ★ 3/5

### 6. Formulario creación de un Evento NUEVO (1 puntos)

### Crear Nuevo Evento

Nombre:

Fecha:

Tipo:

Descripción:

Valoración (1-5):

#### Estructura de datos del formulario:


```
{  
  name: "",           // Requerido  
  date: "",           // Requerido, formato ISO  
  type: "",           // Requerido: "conferencia", "concierto" o  
  "taller"  
  description: "",    // Opcional  
  valoracion: 1       // Requerido, número del 1 al 5  
}
```

#### Funcionalidades Requeridas:

- Validación de campos:
  - Nombre: obligatorio
  - Fecha: obligatoria

- Tipo: debe ser uno de los tres tipos permitidos (conferencia", "concierto" o "taller")
- Valoración: número entre 1 y 5
- Modo edición:
  - Cargar datos del evento existente
  - Actualizar solo los campos modificados
- Manejo de errores:
  - Mostrar errores de validación
  - Mostrar errores de la API
- Navegación:
  - Redirección tras éxito
  - Botón de cancelar

## 7 Formulario Edición y eliminación de un Evento (1 puntos)

 **Examen DWEC 2 Dual**

InicioAñadir EventoCerrar Sesión

### Editar Evento

Nombre:

Fecha:

Tipo:

Descripción:

Valoración (1-5):

Actualizar EventoCancelar

- Se debe de poder Editar un evento cuando se pulse en Editar
- Se debe de poder eliminar un evento al pulsar el botón de Eliminar

## 8. Gestión de Estados Visuales (0,5 punto)

**Estados a Implementar:**

- Loading:
  - Durante la carga inicial de eventos
  - Durante operaciones CRUD
  - Durante el login
- Errores:
  - Mensajes de error en formularios
  - Errores de autenticación
  - Errores en operaciones CRUD
- Confirmaciones:
  - Confirmación para eliminar
  - Mensajes de éxito tras operaciones
  - Feedback visual en acciones importantes

**9. ✨ Punto Extra por Funcionamiento Completo (1 punto)**

Se otorgará un punto adicional si la aplicación cumple con todos estos requisitos:

- Todas las funcionalidades básicas funcionan correctamente
- No hay errores en la consola
- La navegación es fluida y sin errores
- Los datos persisten correctamente
- La autenticación funciona sin problemas
- Los filtros funcionan correctamente

**Código Base y Estructura**

Se proporciona un scaffolding inicial con la siguiente estructura:

```
frontend/  
  src/  
    components/  
    context/  
    hooks/
```



```
layout/  
pages/  
router/
```

Documentación de la API 🛡️

Servicio de Autenticación

- Base URL: `http://localhost:3000/api/auth`

Endpoint	Método	Body	Descripción	Respuesta
<code>/register</code>	POST	<pre>{ name: string, email: string, password: string }</pre>	Registra un nuevo usuario	<pre>Success (201): { token: string, user: Object } Error (400): { message: "User already exists" } Error (500): { message: "Error creating user" }</pre>
<code>/login</code>	POST	<pre>{ email: string, password: string }</pre>	Inicia sesión de usuario	<pre>Success (200): { token: string, user: Object } Error (401): { message: "Invalid credentials" } Error (500): { message: "Error logging in" }</pre>

Servicio de Eventos

- Base URL: `http://localhost:3000/api/events`

Endpoint	Método	Body/Params	Descripción	Respuesta
----------	--------	-------------	-------------	-----------

Endpoint	Método	Body/Params	Descripción	Respuesta
/	GET	-	Obtiene todos los eventos	Success (200): [Event] Error (500): { message: string, error: string }
/:id	GET	id en URL	Obtiene un evento específico	Success (200): Event Error (404): { message: "Event not found" } Error (500): { message: string, error: string }
/	POST	{ name: string, date: string, type: string, ...eventData }	Crea un nuevo evento	Success (201): Event Error (400): { message: "Missing required fields" }
/:id	PUT	id en URL, body con campos a actualizar	Actualiza un evento existente	Success (200): Event Error (404): { message: "Event not found" } Error (400): { message: string, error: string }
/:id	DELETE	id en URL	Elimina un evento	Success (200): { message: "Event deleted successfully" } Error (404): { message: "Event not found" }

Modelos de Datos

Event

```
{
  name: String,           // Nombre del evento (requerido)
  date: Date,             // Fecha del evento (requerido)
  type: String,           // Tipo de evento (requerido, enum: ["conferencia", "concierto", "taller"])
  description: String,    // Descripción del evento (opcional)
  valoracion: Number,     // Valoración del evento (opcional,
```

```
rango: 1-5)
    createdAt: Date,      // Fecha de creación (automático, lo
                           genera el backend)
    updatedAt: Date      // Fecha de última actualización
                           (automático, lo genera el backend)
}
```

## User

```
{
  name: String,          // Nombre del usuario
  email: String,         // Email del usuario (único)
  password: String      // Contraseña (se almacena hasheada)
}
```

## Consejos para el Examen 💡

### Antes de Empezar

1. Lee todo el enunciado detenidamente
2. Revisa la estructura del proyecto proporcionado
3. Identifica los componentes que necesitarás crear
4. Planifica el orden de implementación
5. **Testing Manual** ✅ Prueba el login/logout ✅ Verifica el CRUD completo ✅ Comprueba los filtros ✅ Revisa la persistencia de datos ✅ Graba un vídeo independiente mostrando el funcionamiento de tu web