

# Planificador de Eventos con JavaScript: Componentización Dinámica y Clases

---



**Duración:** 4 horas

**Puntaje Total:** 10 puntos

## Examen DWEC JavaScript Avanzado

Tu tarea es desarrollar un **Planificador de Eventos**. Este sistema debe gestionar una lista de eventos, interactuar con una API REST (JSON Server) para realizar operaciones CRUD y renderizar dinámicamente los datos en el DOM mediante componentes.

Partimos de un `index.html` que contiene únicamente un `<div id="app">`  
`</div>`. Todos los elementos del DOM deben generarse dinámicamente. Además, deberás usar clases para organizar la lógica de la aplicación.

## Ejercicio 0: Preparación del entorno y estructura

- Instalar json server en el proyecto.
- Crear las variables de entorno `URL-API` y `PORT(3000)`.

- Crear la siguiente estructura:

```
/src
├── /components
│   ├── NavBar.js           # Barra de navegación
│   ├── EventForm.js        # Formulario de eventos
│   └── EventList.js         # Lista de eventos
├── /classes
│   ├── Event.js             # Clase evento
│   └── EventManager.js      # Clase lista eventos y API
├── main.js                  # Entrada aplicación
└── style.css                # Estilos globales
```

Puntos: 0,5

## Ejercicio 1: Crear las clases **Event** y **EventManager**

**Objetivo:** Implementar las clases base que estructuran la lógica de la aplicación.

### 1. Clase **Event** :

- Representa un evento individual.
- **Propiedades:**
  - Privadas:
    - **#title** (título del evento).
    - **#date** (fecha del evento).
- **Métodos:**
  - **updateDate(newDate)** : Actualiza la fecha del evento.

### 2. Clase **EventManager** :

- Gestiona la lista de eventos.
- Al constructor se le pasa una URL de la api donde están los eventos.
- **Propiedades:**

- Privadas:
  - **#events** (array para almacenar instancias de **Event** ).
  - **#apiURL** (aquí se guarda la URL que se le pasa al constructor).
- **Métodos:**
  - **fetchEvents()** : Obtiene los eventos de la API y los convierte en instancias de **Event** .
  - **addEvent(event)** : Agrega un nuevo evento a la API y a la lista interna.
  - **deleteEvent(eventId)** : Elimina un evento de la API y de la lista interna.
  - **markAsImportant(eventId)** : Marca un evento como importante y lo guarda en LocalStorage. Así en la clave "importantEvents" guardaremos un array con los IDs de los eventos que he considerado importantes. Ejemplo:

```
{  
  "importantEvents": "[1, 3]"  
}
```

- **getImportantEvents()** : Devuelve un array de IDs que se encuentra en LocalStorage bajo la clave "importantEvents". Si no existe la clave devuelve un array vacío.
- **getEvents()** : Devuelve todos los eventos almacenados en la propiedad **#events**. Esto será útil si queremos acceder a la lista interna de eventos desde otros componentes

**Nota:** Todos los eventos que interactúan con la API deben manejar los errores.

**Puntos:** 2,75

## Ejercicio 2: Componentizar la estructura básica del DOM

**Objetivo:** Crear los componentes base de la interfaz de usuario.

### 1. Crea el componente **NavBar** :

- Muestra:
  - Muestra el texto en h2, Examen JavaScript 2024.
  - A este componente le paso como parámetro un objeto: {nombre:"tu nombre", fecha:"la fecha de hoy"}
  - Un botón "Mostrar Formulario". Al pulsarlo mostrará u ocultará el formulario para insertar los eventos que hay debajo del navBar. Recuerda que **style.display con none o block** permite cambiar entre mostrar o no un componente del DOM.
- Usa **propiedades del DOM** para generarlo dinámicamente. No está permitido **innerHTML** para la creación del navBar

**Puntos:** 1

## Ejercicio 3: Implementar el formulario dinámico (EventForm)

**Objetivo:** Crear un formulario dinámico para agregar y editar eventos.

### 1. Crea el componente **EventForm** :

- Contiene:
  - Inputs dinámicos para título, fecha y organizador.
  - Un botón "Guardar".
- Se debe mostrar u ocultar al hacer clic en "Mostrar Formulario" en el **NavBar** .

### 2. Funcionalidad del botón "Guardar":

- Crea una nueva instancia de **Event** y utiliza **addEvent** de **EventManager** para enviarlo a la API y a la lista interna.

- Actualiza la lista de eventos en el DOM sin recargar la página, es decir, se actualiza el EventList para mostrar el nuevo elemento añadido.

**Puntos:** 2

## Ejercicio 4: Crear y gestionar la lista dinámica de eventos (EventList)

**Objetivo:** Renderizar dinámicamente la lista de eventos en el DOM.

### 1. Crea el componente **EventList** :

- Es un contenedor que renderizan TODAS las instancias usando `<Div class="card">` de html. Se debe mostrar:
  - Título, fecha y organizador.
  - Botones:
    - "Eliminar": Borra el evento utilizando `deleteEvent` .
    - "Marcar como Importante": Cambia el estado del evento y lo guarda en LocalStorage.
    - El marcar importante cambia el color de la de este evento a NARANJA para así distinguirla, es decir, el fondo de la card ahora será naranja.

**Puntos:** 2,75

## Ejercicio 5: Persistencia de eventos importantes

**Objetivo:** Implementar almacenamiento y recuperación de eventos importantes con LocalStorage.

### 1. Funcionalidad requerida:

- Su ID debe guardarse en LocalStorage bajo la clave "importantEvents".
- La tarjeta del evento (EventCard) debe cambiar de color a naranja.
- El texto del botón debe alternar entre "Marcar como Importante" y "Quitar Importante".

- La persistencia debe garantizar que, al recargar la página, los eventos importantes mantengan su color y estado.
  - Al cargar la aplicación, los eventos importantes deben recuperarse y mostrarse correctamente.
2. Modifica el contador de eventos importantes en el **NavBar** para reflejar el estado actual.

Puntos: 1

## Resultado

EXAMEN JAVASCRIPT 2024

Isaías FL11/12/2024MOSTRAR FORMULARIO

FORMULARIO Eventos

Titulo del Evento

Fecha del Evento

Organizador

GUARDAR

Lista de Eventos

Titulo Evento

fecha del Evento

Organizador

★🗑

Titulo Evento

fecha del Evento

Organizador

★🗑

Titulo Evento

fecha del Evento

Organizador

★🗑

Titulo Evento

fecha del Evento

Organizador

★🗑

Titulo Evento

fecha del Evento

Organizador

★🗑

Titulo Evento

fecha del Evento

Organizador

★🗑

## Rúbrica de Calificación

- Es **OBLIGATORIO** la documentación y el establecer **TU NOMBRE EN TODOS LOS FICHEROS QUE ENTREGUES**.
- Este examen evalúa todos los RA de este primer trimestre y que corresponden con la parte de JavaScript VANILLA.

### Ejercicio 0: Preparación del entorno y estructura

Puntos: 0,5

Criterio	Puntos
Instalación correcta de JSON Server	0,2
Uso correcto de las variables de entorno ( <code>URL-API</code> y <code>PORT</code> )	0,2
Creación de la estructura de carpetas especificada	0,1

### Ejercicio 1: Clases `Event` y `EventManager`

Puntos: 2,75

Criterio	Puntos
Clase <code>Event</code> : Implementación correcta de las propiedades privadas	0,5
Clase <code>Event</code> : Método <code>updateDate</code> funcional	0,25
Clase <code>EventManager</code> : Uso de propiedades privadas ( <code>#events</code> y <code>#apiURL</code> )	0,5
Clase <code>EventManager</code> : Método <code>fetchEvents</code> : Convierte datos de la API en instancias de <code>Event</code>	0,5
Clase <code>EventManager</code> : Método <code>addEvent</code> : Agrega correctamente un evento a la API y lista interna	0,5
Clase <code>EventManager</code> : Método <code>deleteEvent</code> : Borra correctamente un evento de la API y la lista interna	0,25
Clase <code>EventManager</code> : Método <code>markAsImportant</code> : Maneja LocalStorage correctamente	0,5

Criterio	Puntos
Clase <code>EventManager</code> : Método <code>getImportantEvents</code> funcional	0,25
Clase <code>EventManager</code> : Método <code>getEvents</code> funcional	0,25

## Ejercicio 2: Componentizar la estructura básica del DOM

Puntos: 1

Criterio	Puntos
<code>NavBar</code> : Botón "Mostrar Formulario" funcional	0,25
<code>NavBar</code> : Sin uso de <code>innerHTML</code> para la creación	0,25
<code>Footer</code> : Renderiza el objeto pasado como parámetro	0,5

## Ejercicio 3: Implementar el formulario dinámico (EventForm)

Puntos: 2

Criterio	Puntos
Formulario ( <code>EventForm</code> ) : Inputs y botón "Guardar" correctamente creados	0,75
Función del botón "Guardar":	
- Crea una instancia de <code>Event</code> correctamente	0,5
- Agrega el evento a la API y actualiza el DOM sin recargar la página	0,75

## Ejercicio 4: Crear y gestionar la lista dinámica de eventos (EventList)

Puntos: 2,75

Criterio	Puntos
----------	--------



Criterio	Puntos
<b>Componente <code>EventList</code></b> : Contenedor renderiza correctamente las tarjetas	1
<b>Botones en <code>EventCard</code></b> : Funcionalidad implementada	
- Botón "Eliminar": Borra el evento de la API y el DOM	1
- Botón "Marcar como Importante": Cambia el estado, color y texto del botón	0,75

## Ejercicio 5: Persistencia de eventos importantes

**Puntos: 1**

Criterio	Puntos
<b>Guardado en <code>LocalStorage</code></b> : IDs de eventos importantes se almacenan correctamente bajo la clave "importantEvents"	0,5
<b>Persistencia visual:</b>	
- Los eventos importantes mantienen su color naranja al recargar la página	0,25
- El texto del botón alterna entre "Marcar como Importante" y "Quitar Importante" según el estado	0,25

## Criterios Generales

Además de los criterios específicos por ejercicio, considera los siguientes aspectos generales:

Aspecto	Puntos
Código organizado, modular y comentado	+0,25
Uso adecuado de buenas prácticas (nombres claros, separación de responsabilidades)	+0,25

## Resumen

Ejercicio	Puntos Máximos
-----------	----------------

Ejercicio	Puntos Máximos
Ejercicio 0: Preparación del entorno	0,5
Ejercicio 1: Clases <b>Event</b> y <b>EventManager</b>	2,75
Ejercicio 2: Componentizar el DOM	1
Ejercicio 3: Formulario dinámico	2
Ejercicio 4: Lista dinámica de eventos	2,75
Ejercicio 5: Persistencia de eventos importantes	1
<b>Total</b>	<b>10 puntos</b>