Planificador de Viajes con Clases y Funciones Dinámicas

Duración: 4 horas
Puntaje Total: 10 puntos

Examen DWEC JavaScript Avanzado

Tu tarea es desarrollar un **Planificador de Viajes**. Este sistema permitirá gestionar una lista de destinos, interactuar con una API REST (JSON Server) para realizar operaciones CRUD, y renderizar dinámicamente los datos en el DOM mediante funciones que creen la interfaz de usuario.

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 estructurar la lógica de la aplicación.

Ejercicio 0: Preparación del entorno y estructura

1. **Configuración inicial:**
    - Instalar `json-server` en el proyecto.
    - Configurar las variables de entorno `URL_API` y `PORT(3500)`.
    - Crear un script llamado "examen" que permita levantar la api

2. **Crear la estructura básica del proyecto:**

```
/src
 ├── /classes
 |    ├── Destination.js         # Clase un destino
 |     ├── TravelPlanner.js       # Clase lista de destinos y la
API
 ├── /database
 |    ├── db.json
 |
 ├── main.js                     # Punto de entrada
 ├── style.css                   # Estilos globales
```

Puntos: 0,5

Ejercicio 1: Crear las clases `Destination` y `TravelPlanner`

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

1. **Clase `Destination`:**
    - Representa un destino de viaje.
    - **Propiedades:**
        - Privadas:
            - `#name` (nombre del destino).
            - `#date` (fecha del viaje en formato `YYYY-MM-DD`).
            - `#tripCost` (presupuesto estimado).
    - **Métodos:**
        - `updateTripCost(newTripCost)`: Actualiza el presupuesto del destino.
2. **Clase `TravelPlanner`:**
    - Gestiona la lista de destinos y la interacción con la API.
    - Al constructor se le pasa una URL
    - **Propiedades:**

        - Privadas:
            - `#destinations` (array para almacenar instancias de `Destination`).
            - `#apiURL` (URL base de la API: `http://localhost:3500/destinations`).
    - **Métodos:**
        - `fetchDestinations()`: Obtiene los destinos de la API y los convierte en instancias de `Destination`.
        - `addDestination(destination)`: Agrega un nuevo destino a la API y a la lista interna.
        - `deleteDestination(destinationId)`: Elimina un destino de la API y de la lista interna.
        - `getDestinations()`: Devuelve todos los destinos almacenados en la propiedad `#destinations`.

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

Puntos: 2,75

Ejercicio 2: Crear la estructura básica del DOM

Objetivo: Crear funciones para generar las secciones principales de la interfaz de usuario.

1. **Función `createNavBar`:**
    - Genera dinámicamente una barra de navegación con:
        - Un contador de destinos totales (inicia en 0).
        - Un botón `"Agregar Destino"`.
2. **Función `createFooter`:**
    - Se le pasa como parámetro un objeto: `{name:"tu nombre", date:"generada automáticamente"}`
    - Al Renderizarse, genera dinámicamente un pie de página que incluye:
        - El nombre del estudiante.
        - La fecha del examen actual.

Puntos: 1

Ejercicio 3: Implementar el formulario dinámico

Objetivo: Crear un formulario dinámico para agregar nuevos destinos.

1. **Función `createDestinationForm`:**
    - Genera un formulario **usando los métodos del DOM** (sin usar innerHTML) con los siguientes campos:
        - Input para el nombre del destino.
        - Input para la fecha del viaje. (tipo date)
        - Input para el presupuesto. (tipo número)
        - Botón "Guardar".
        - Todos los campos han de estar rellenos obligatoriamente y validados por JavaScript.
    - Mostrar u ocultar el formulario al hacer clic en "Agregar Destino" desde la barra de navegación.
2. **Funcionalidad del botón "Guardar":**
    - Crea una nueva instancia de `Destination` y utiliza `addDestination` de `TravelPlanner` para enviarlo a la API y agregarlo a la lista interna.
    - Actualiza la lista de destinos en el DOM sin recargar la página.

Puntos: 2

Ejercicio 4: Crear y gestionar la lista dinámica de destinos

Objetivo: Renderizar dinámicamente la lista de destinos en el DOM.

1. **Función `renderDestinationList`:**
    - Genera un contenedor para mostrar en **forma de Tarjetas** todos los destinos.
2. Cada tarjeta de destino debe incluir:
    - Nombre, fecha y presupuesto.
    - Botones:

- "Eliminar": Borra el destino utilizando `deleteDestination`.
- "Editar Presupuesto": Permite modificar el presupuesto mediante un `prompt` y actualiza la API.

3. Actualiza automáticamente la lista y la API al editar o eliminar destinos.

Puntos: 2,75

Ejercicio 5: Persistencia del contador de destinos

Objetivo: Implementar un contador dinámico y persistente de destinos totales.

1. **Funcionalidad requerida:**
   - El contador en la barra de navegación debe actualizarse automáticamente al agregar o eliminar destinos.
   - La persistencia en el **LocalStorage** debe garantizar que, al recargar la página, el contador refleje el estado actual.

Puntos: 1

Rúbrica de Calificación

- Es **OBLIGATORIO** documentar el código y colocar **TU NOMBRE EN TODOS LOS FICHEROS QUE ENTREGUES.**
- Este examen evalúa los RA de este primer trimestre relacionados con JavaScript Vanilla.

Ejercicio	Puntos Máximos
Ejercicio 0: Preparación del entorno	0,5
Ejercicio 1: Clases `Destination` y `TravelPlanner`	2,75
Ejercicio 2: Crear estructura básica del DOM	1
Ejercicio 3: Formulario dinámico	2
Ejercicio 4: Lista dinámica de destinos	2,75
Ejercicio 5: Persistencia del contador	1
Total	**10 puntos**