Examen Parcial C8288

Reglas de la evaluación

- Queda terminantemente prohibido el uso de herramientas como ChatGPT, WhatsApp, o cualquier herramienta similar durante la realización de esta prueba. El uso de estas herramientas, por cualquier motivo, resultará en la anulación inmediata de la evaluación.
- Solo se permite usar las laptops del aula virtual. No deben usar sus laptops personales.
- Presenten código desarrollado en clases, técnicas, herramientas o sintaxis ajena a lo desarrollado en el curso no está permitida.
- Las respuestas deben presentarse con una explicación detallada, utilizando términos técnicos apropiados. La mera descripción sin el uso de terminología técnica, especialmente términos discutidos en clase, se considerará insuficiente y podrá resultar en que la respuesta sea marcada como incorrecta.
- Utiliza imágenes para explicar o complementar las respuestas, siempre y cuando estas sean de creación propia y relevantes para la respuesta, como es el caso de la verificación de los pasos de pruebas.
- Cada estudiante debe presentar su propio trabajo. Los códigos iguales o muy parecidos entre sí serán considerados como una violación a la integridad académica, implicando una copia, y serán sancionados de acuerdo con las políticas de la universidad.
- La claridad, orden, y presentación general de las evaluaciones serán tomadas en cuenta en la calificación final.

Instrucciones de entrega para la prueba calificada

Para asegurar una entrega adecuada y organizada de su prueba calificada, sigan cuidadosamente las siguientes instrucciones. El incumplimiento de estas pautas resultará en que su prueba no sea revisada, sin importar el escenario.

- Presenta un archivo comprimido con todas las respuestas y soluciones documentadas, completas y presentadas en archivos Markdown (.md) junto con el codigo del lenguaje trabajado.dentro de las subcarpetas correspondientes a cada ejercicio.
- No se aceptarán entregas en formatos como documentos de Word (.docx) o archivos PDF. La entrega debe cumplir estrictamente con el formato Markdown (.md) y las extensiones del código del lenguaje desarrollado.
- Es crucial seguir todas las instrucciones proporcionadas para la entrega de su prueba calificada. Cualquier desviación de estas pautas resultará en que su prueba no sea considerada para revisión.
- Presenta en un archivo .txt la dirección de tu repositorio con todas tus respuestas.

Ejercicio 1: Simulación del event loop en Node.js con TypeScript (6 puntos)

Enunciado del problema:

Desarrolla un script en TypeScript que simule y demuestre detalladamente el funcionamiento de las fases del Event Loop en Node.js, incluyendo el manejo de microtareas y macrotareas. El objetivo es ilustrar cómo las tareas se encolan y ejecutan en las diferentes fases, y cómo las microtareas tienen prioridad sobre las macrotareas.

Además, el script debe utilizar características avanzadas de TypeScript, como:

- Tipos avanzados (tipos de unión e intersección).
- Utility types (Pick, Partial, Omit, Readonly).
- Interfaces y tipos de funciones.
- Generics para crear clases y funciones reutilizables.
- Programación Orientada a Objetos (POO) aplicando herencia, polimorfismo, abstracción y encapsulación.
- Manejo de null y undefined con narrowing.

El script debe incluir:

1. Varias funciones que se ejecuten en diferentes fases del Event Loop (setTimeout, setImmediate, process.nextTick, Promises).
2. Clases e interfaces que encapsulen la lógica de las tareas asíncronas.
3. Uso de Promises y async/await para manejar código asíncrono.
4. Ejemplos de callbacks para demostrar su funcionamiento.
5. Implementación de utility types para manipular y restringir tipos.

Entrada:

No se requiere entrada del usuario. El script se ejecuta y simula las operaciones internamente.

Salida:

El script debe imprimir en la consola mensajes que indiquen:

- El orden de ejecución de las tareas.
- En qué fase del Event Loop se está ejecutando cada tarea.
- Cómo interactúan las microtareas y macrotareas.
- Ejemplos de polimorfismo y herencia en acción.

Restricciones:

- No se permite el uso de librerías externas; solo se debe utilizar Node.js y TypeScript.
- Todo el código debe estar en **un único archivo**.

- No se debe proporcionar código de solución ni ejemplos que puedan ser copiados directamente.
- Se debe hacer énfasis en los conceptos avanzados mencionados y su correcta implementación.

Ejercicio 2: Aplicación React con Hooks poersonalizados y manejo del estado (7 puntos)

Problema:

Crea un archivo que contenga una aplicación React funcional, enfocándote en el manejo avanzado del estado y efectos colaterales utilizando Hooks. La aplicación debe simular un tablero de control en tiempo real que muestra datos actualizados cada cierto intervalo, permitiendo al usuario filtrar y manipular la información presentada.

Entrada:

- Datos simulados que representen información en tiempo real (puedes generar datos aleatorios dentro del código).
- Interacciones del usuario para filtrar y actualizar la información.

Salida:

- Interfaz de usuario que muestra los datos actualizados.
- Respuestas a las interacciones del usuario en tiempo real.
- Logs en la consola que muestren el ciclo de vida de los componentes y el manejo del estado.

Restricciones:

- Utiliza los Hooks de React: useState, useEffect, useContext, useReducer, useMemo, useCallback y useRef.
- Implementa **Hooks Personalizados** para reutilizar lógica entre componentes.
- Gestiona un **estado global** utilizando useContext y useReducer.
- Optimiza el rendimiento de la aplicación utilizando useMemo y useCallback para evitar renders innecesarios.
- Manipula referencias directas al DOM con useRef y preserva valores entre renders.
- No utilices librerías de manejo de estado como Redux; todo debe ser con Hooks.
- El código debe estar escrito en TypeScript, utilizando **Interfaces**, **Tipos Avanzados** y **Generics** donde sea apropiado.
- Debes manejar errores y casos de borde, asegurando que la aplicación no se rompa ante entradas inesperadas.

Ejercicio 3: Sistema de Gestión de Eventos con React y JavaScript (7 puntos)

Problema

Desarrolla una aplicación de gestión de eventos utilizando React con JavaScript. La aplicación debe permitir a los usuarios crear, editar y eliminar eventos, así como visualizar una lista de eventos existentes. Para lograr esto, deberás implementar diversas características avanzadas que abarcan conceptos de programación orientada a objetos (OOP), manejo de estado en React y gestión avanzada de promesas.

Entrada

La aplicación no requiere una entrada directa por parte del usuario en formato de consola. Sin embargo, debe manejar los siguientes tipos de interacciones:

1. **Crear evento**: Formulario para ingresar detalles del evento (título, descripción, fecha, ubicación).
2. **Editar evento**: Opción para modificar los detalles de un evento existente.
3. **Eliminar evento**: Acción para eliminar un evento de la lista.
4. **Visualizar eventos**: Interfaz que muestra una lista de todos los eventos creados.

Salida

La aplicación debe mostrar una interfaz de usuario dinámica que refleje las operaciones realizadas:

1. **Lista de eventos**: Tabla o lista que muestra todos los eventos con sus detalles.
2. **Confirmaciones**: Mensajes de confirmación al crear, editar o eliminar un evento.
3. **Errores**: Manejo y visualización de errores en caso de operaciones fallidas.

Restricciones de respuesta

Para resolver este ejercicio, debes cumplir con las siguientes restricciones y requisitos:

1. **Programación orientada a objetos (OOP)**:
    a. Utiliza clases en JavaScript para definir la estructura de los eventos.
    b. Implementa principios de encapsulación, herencia y polimorfismo donde sea aplicable.
    c. Aplica abstracción para separar la lógica de negocio de la interfaz de usuario.
2. **Promises y cadenas de promesas**:
    a. Simula operaciones asíncronas (como llamadas a una API) utilizando Promesas.
    b. Implementa una cadena de promesas para manejar secuencias de operaciones dependientes.
3. **React Hooks** :
    a. **useState**: Gestiona el estado de los eventos y los formularios de entrada.

   b. **useEffect**: Crea efectos secundarios para cargar eventos iniciales y actualizar la interfaz tras operaciones.
   c. **useContext**: Maneja el estado global de la aplicación para compartir datos entre componentes.
   d. **useReducer**: Implementa un gestor de estado más complejo para manejar múltiples acciones relacionadas con eventos.
   e. **useMemo**: Optimiza el rendimiento memorizando cálculos costosos, como el filtrado de eventos.
   f. **useCallback**: Evita la recreación innecesaria de funciones que se pasan como props.
   g. **useRef**: Accede directamente a elementos del DOM si es necesario y persiste valores entre renders.
   h. **Hooks Personalizados**: Crea hooks personalizados para reutilizar lógica relacionada con la gestión de eventos.
4. **Encapsulación y modularidad**:
   a. Organiza el código en módulos separados para componentes, servicios y utilidades.
   b. Asegura que cada módulo tenga una única responsabilidad y esté adecuadamente encapsulado.
5. **Restricciones adicionales**:
   a. No utilices librerías externas para la gestión del estado más allá de React.
   b. La aplicación debe estar escrita en JavaScript y aprovechar las características modernas de ECMAScript.
   c. Evita el uso de ejemplos relacionados con inventarios, gestión de usuarios o productos; enfócate únicamente en la gestión de eventos.

Notas

- Asegúrate de que el código sea limpio, bien documentado y siga buenas prácticas de desarrollo.
- La solución debe consistir en archivos únicos que integren todas las funcionalidades mencionadas.
- No se requiere crear un proyecto completo, pero el archivo o archivos deben ser suficientemente detallados y avanzados.
- **No se requiere incluir tipos explícitos** como en TypeScript, pero se recomienda utilizar comentarios JSDoc para mejorar la legibilidad y mantenimiento del código.
- **Evita el uso de características exclusivas de TypeScript** como interfaces o tipos genéricos; en su lugar, utiliza la flexibilidad de JavaScript para lograr una arquitectura robusta.
- **Enfócate en la reutilización de componentes y lógica**, aprovechando al máximo los hooks personalizados para mantener el código modular y limpio.