Instituto Pío IX Casa Salesiana

Desarrollo de Aplicaciones Informáticas

Examen de NextJS con Sockets

Docente: Ing. Pablo Morandi
Ayudante: Matías Marchesi
Curso: 5to año - Informática
Duración: 2 horas cátedra (80 min)
Fecha: 20/11/25

TEMA 5

Leer con atención antes de empezar el examen:

`Material permitido`: Apuntes de clase, documentación oficial, búsquedas en Google SIN IA. El uso de la IA implica que **el examen sea anulado automáticamente** sin excepción y con nota 1(uno).

`Formato de entrega`: Subir carpeta completa del proyecto en un archivo .RAR y con el formato "5x_NEXT_APELLIDO" a Google Classroom.
POR FAVOR, respetar el formato de entrega.

`IMPORTANTE:` la carpeta **/node_modules** y la carpeta **./next NO DEBEN SER ENTREGADAS**. De ser asi, **se restará 1 (uno) punto** de la nota final del examen sin excepción.

Sistema de Comentarios en Tiempo Real

Desarrollar una aplicación de muro de comentarios que utilice **Socket.IO** para comunicación en tiempo real, **Router** para navegación, **Conditional Rendering** para mostrar diferentes estados de la interfaz y **useState** y **useEffect** para manejo de datos.
A modo de recordatorio, para crear un proyecto de NextJS utilizamos `npx create-next-app@latest` o en su defecto pueden utilizar un proyecto ya existente teniendo en consideración de borrar todo lo que no esté relacionado a la evaluación al momento de la entrega.

`Ejercicio 1`: Preguntas Teóricas

1a) ¿Qué son las props en React? ¿Cuál es la diferencia principal entre props y state? Proporcione un ejemplo de cuándo usar cada uno.
1b) ¿Qué ventajas tiene usar web sockets antes que otra forma de comunicación entre back y front, por ejemplo un pedido HTTP?

Responda estas preguntas como comentarios en la página `/inicio` del proyecto

---

`Ejercicio 2`: Página de Inicio con Router

Crear una página de inicio en `/inicio` que cumpla con los siguientes requisitos:

Requisitos

1. **Formulario de ingreso**:
    - Input para el nombre del usuario (`usuario`) (mínimo 3 caracteres)
    - Input para el ID de alumno (`alumnoId`). Este ID es el nro que a usted le corresponde de la lista del curso ordenada alfabéticamente.
        - Esta variable se utilizará luego para el evento `join_muro` como id de sala.
    - Botón "Ir al Muro"
2. **Validación y estados**:
    - Mostrar mensaje de error si el nombre tiene menos de 3 caracteres
    - El mensaje de error debe mostrarse usando **Conditional Rendering**
3. **Navegación con Router**:
    - Al hacer clic en "Ir al Muro", redirigir a `/muro`
    - Pasar `usuario` y `alumnoId` como parámetros de ruta o query string
        - URL de ejemplo: `http://localhost:3000/muro?usuario=pepito&alumnoId=4`

`Ejercicio 3`: Creación de componentes Comentario y FormularioComentario

`3a)` Crear un componente `Comentario.js` que:

1. Reciba por props: `usuario` (string), `texto` (string), `categoria` (string), `timestamp` (string)
2. **Conditional Rendering** para sumar un emoji dependiendo de la categoría:
   - Si categoria es "Info": muestre "ℹ️ Información ℹ️"
   - Si categoria es "Pregunta": muestre "⁉️ Pregunta ⁉️"
   - Si categoria es "Respuesta": muestre " ☑️ Respuesta ☑️"
3. Mostrar la hora del comentario formateada (solo hora:minutos)

`3b)` Crear un componente `FormularioComentario.js` que:

1. Reciba por props: `onChangeTexto` (function), `onChangeCategoria` (function), `onClickPublicar` (function)
2. **El componente debe tener:**
   - Un título "Publicar Comentario"
   - Un input para ingresar el texto del comentario
   - Un select para elegir la categoría (Info, Pregunta, Respuesta)
   - Un contador de caracteres (ej: "45/200")
   - Un Botón "Publicar" para enviar el comentario por el socket.

`Ejercicio 4`: Página de Muro con Socket.IO

Crear una página de muro en `/muro` que implemente comunicación en tiempo real.

`Nota`: para probar el proyecto, deben abrirse 2 instancias con `npm run dev`, una en el puerto 3000 y otra en el puerto 3001 para poder probar todas las funcionalidades requeridas en este punto.

`Nota`: para probar el proyecto, utilice la consola del navegador con F12 para ver lo que llega de los eventos y eventuales errores.

Backend Provisto

Servidor: `http://10.1.5.137:4000`

Eventos Socket.IO

Eventos que ENVÍAS al servidor:

Evento	Descripción	Datos
`join_muro`	Unirse a tu sala de muro	`{ alumnoId }`
`publicar_comentario`	Publicar un nuevo comentario	`{ usuario, texto, categoria }`

Eventos que RECIBES del servidor:

Evento	Descripción
`joined_OK_muro`	Confirmación de ingreso + comentarios iniciales
`nuevo_comentario`	Actualización cuando hay un nuevo comentario
`muro_lleno`	Se publicaron 6 comentarios + datos para reiniciar

Estructura del objeto Comentario

```
{
    id: number,
    usuario: string,
    texto: string,
    categoria: string, // "Info" o "Pregunta" o "Respuesta"
    cantidadComentarios: number,
    timestamp: string
}
```

Requisitos

1. Conexión con Socket.IO

- Poner la ip del backend en el hook de useSocket `http://10.1.5.137:4000`
- Obtener `usuario` y `alumnoId` desde los parámetros de ruta que vienen desde `/inicio`

---

2. Unirse a la sala del muro

- **Botón** "Unirse al Muro"
- Al hacer clic, emitir evento `join_muro` con `{ alumnoId }`
- Escuchar evento `joined_OK_muro` que trae los comentarios iniciales
- Guardar los comentarios en un estado y mostrarlos con el componente `Comentario`

---

3. Publicar comentarios (Componente `FormularioComentario`)

- **Input** para ingresar el texto del comentario
- **Select** para elegir categoría (Info/Pregunta/Respuesta)
- **Contador de caracteres** que muestre cuántos caracteres lleva (ej: "45/200")
- **Botón "Publicar"**
- **Validaciones**:
  - El texto debe tener entre 10 y 200 caracteres
  - Debe seleccionar una categoría
  - Si no cumple, mostrar `alert` con mensaje de error
- Si es válido, emitir evento `publicar_comentario` con `{ usuario, texto, categoria }`

---

4. Escuchar actualizaciones en tiempo real

Evento `nuevo_comentario`:

- Actualizar la lista de comentarios con los nuevos datos recibidos
- Esto permite que ambos clientes vean los comentarios del otro en tiempo real
- Limpiar el textarea después de publicar

Evento `muro_lleno`:

- Recibir datos para reiniciar el muro
- Reiniciar los estados necesarios
- Mostrar mensaje de finalización

`Ejercicio 5`: Historial de Comentarios y estadísticas

1. Mantener un registro de todos los comentarios que se reciben en la aplicación mediante el evento `nuevo_comentario` hasta que llegue el evento `muro_lleno`.
    - Guardar un historial en un vector de todos los comentarios recibidos desde que el usuario presionó el botón "Unirse al Muro".
    - Cada vez que se reciba un evento `nuevo_comentario` agregar el comentario recibido al vector.
    - El historial se deberá reiniciar cuando se llenó el muro, o sea cuando se recibió el evento `muro_lleno`.
2. Mostrar el historial en una sección que contenga:
    - Usuario
    - Categoría
    - Texto
    - Hora
3. Mostrar estadísticas de los comentarios recibidos.
    - Mostrar:
        - Cantidad de comentarios totales recibidos
        - Cantidad de comentarios de información recibidos
        - Cantidad de comentarios de pregunta recibidos
        - Cantidad de comentarios de respuesta recibidos
    - Estas estadísticas deben actualizarse cada vez que se recibe un comentario nuevo.
4. Si no hay comentarios mostrar en la sección: "No se recibieron comentarios aún."