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: 09/10/25

TEMA 2

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 Reserva de Turnos en Tiempo Real

Desarrollar una aplicación de reserva de turnos médicos 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`: Pregunta Teórica sobre CLSX

1a) ¿Cuál es la ventaja de usar CLSX en lugar de concatenar strings para clases CSS?
Responda esta pregunta como comentario en la página `/registro` del proyecto

`Ejercicio 2`: Página de Registro con Router

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

Requisitos

1. **Formulario de ingreso**:
   - Input para el nombre del paciente (`username`) (mínimo 4 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_turnos` como id de sala.
   - Botón "Ir a Turnos"
2. **Validación y estados**:
   - Mostrar mensaje de error si el nombre tiene menos de 4 caracteres
   - El mensaje de error debe mostrarse usando **Conditional Rendering**
3. **Navegación con Router**:
   - Al hacer clic en "Ir a Turnos", redirigir a `/turnos`
   - Pasar `username` y `alumnoId` como parámetros de ruta o query string
     - URL de ejemplo: `http://localhost:3000/turnos?username=pepito&alumnoId=23`

`Ejercicio 3`: Creación de componentes Turno y AgendarNuevoTurno

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

1. Reciba por props: `especialidad` (string), `turnoActual` (number), `cantidadReservas` (number) y `pacienteActual` (string).
2. **Mostrar información**:
   - Especialidad médica actual
   - Número de turno actual
   - Nombre del paciente actual
   - Cantidad de turnos reservados

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

1. Reciba por props: `onChangeNumeroTurno` (function), `onClickRealizarReservaDeTurno` (function)
2. **El componente debe tener:**
   - Un título "Agendar nuevo turno"
   - Un Input numérico para ingresar el número de turno a reservar
   - Un Botón "Reservar Turno" para enviar el turno por el socket

---

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

Crear una página de turnos en `/turnos` 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 ENVIAS al servidor:

Evento	Descripción	Datos
`join_turnos`	Unirse a tu sala de turnos	`{ alumnoId }`
`realizar_reserva`	Reservar un turno	`{ paciente, numeroTurno }`

Eventos que RECIBIS del servidor:

Evento	Descripción
`joined_OK_turnos`	Confirmación de ingreso + datos iniciales de turnos
`nueva_reserva`	Actualización cuando hay una nueva reserva
`turnos_completos`	Los turnos se completaron (después de 5 reservas) + datos para reiniciar

---

Estructura del objeto Turno

```
{
    id: number,
    especialidad: string,
    turnoActual: number,
    pacienteActual: string,
    cantidadReservas: 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 `username` y `alumnoId` desde los parámetros de ruta que vienen desde `/registro`

---

2. Unirse a la sala de turnos

- **Botón** "Unirse a la sala de turnos"
- Al hacer clic, emitir evento `join_turnos` con `{ alumnoId }`
- Escuchar evento `joined_OK_turnos` que trae el turno inicial
- Guardar los datos del turno en un estado y mostrarlos con el componente `Turno`

---

3. Reservar turnos (Componente `AgendarNuevoTurno`)

- **Input numérico** para ingresar el número de turno a reservar
- **Botón "Reservar Turno"**
- **Validación**: El número de turno debe ser mayor al `turnoActual`, sino mostrar `alert` con mensaje de error
- Si es válido, emitir evento `realizar_reserva` con `{ paciente, numeroTurno }`

---

4. Escuchar actualizaciones en tiempo real

Evento `nueva_reserva`:

- Actualizar el componente `Turno` con los nuevos datos recibidos
- Esto permite que ambos clientes vean las reservas del otro en tiempo real

Evento `turnos_completos`:

- Recibir datos del nuevo turno para reiniciar
- Reiniciar los estados necesarios
- Mostrar mensaje de finalización

---

5. Conditional Rendering

Mostrar:

- **"Los turnos del día se han completado. Por favor presione en Unirse a sala nuevamente para comenzar con nuevos turnos."** cuando llegue el evento `turnos_completos`
- **"Número de sala: {alumnoId}"** cuando se haya conectado exitosamente
- Deshabilitar botón "Reservar Turno" si no hay turnos activos

---

`Ejercicio 5`: Historial de Turnos

Agregar funcionalidad para mostrar el historial de los últimos 5 turnos realizados.

Requisitos

1. Crear un estado `historial` (array) que almacene los turnos
2. Usar `useEffect` para actualizar el historial cada vez que llega una nueva reserva
3. Mostrar los últimos 5 turnos con: paciente, número de turno y timestamp
4. Si no hay turnos en el historial, mostrar: "No hay turnos reservados."