Desarrollo de Aplicaciones Informáticas

Examen - NextJS con Sockets

Docente: Ing. Pablo Morandi
Ayudante: Matías Marchesi
Curso: 5to año - Informática
Duración: 2 horas cátedra (80 min)
Fecha: Diciembre 2025

EXAMEN RECUPERATORIO - PERÍODO DICIEMBRE

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_DAI_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 Reservas de Mesas en Tiempo Real

Desarrollar una aplicación de gestión de mesas de restaurante donde múltiples mozos reservan y liberan mesas en tiempo real usando **Socket.IO**, **Router** para navegación, **Conditional Rendering** para mostrar diferentes estados y **useState** y **useEffect** para manejo de datos.

`Ejercicio 1`: 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 mozo (`mozo`) (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.
   - Botón "Acceder al Restaurante"
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 "Acceder al Restaurante", redirigir a `/restaurante`
   - Pasar `mozo` y `alumnoId` como query params
     - URL: `http://localhost:3000/restaurante?mozo=Juan&alumnoId=4`

`Ejercicio 2`: Creación de componentes

`2a)` Crear un componente `MesaRestaurante.js` que:

1. Reciba por props: `numeroMesa` (number), `capacidad` (number), `estado` (string), `cliente` (string o null)
2. **Conditional Rendering**:
   - Si estado es "libre": mostrar "LIBRE"
   - Si estado es "reservada": mostrar "RESERVADA - Cliente: {cliente}"
   - Si capacidad es 2: mostrar ícono 👫
   - Si capacidad es 4: mostrar ícono 👨‍👩‍👧‍👦
   - Si capacidad es 6: mostrar ícono 👨‍👩‍👧‍👦👫
3. Mostrar Mesa N°: {numeroMesa}

`2b)` Crear un componente `FormularioReserva.js` que:

1. Reciba por props: `onClickReservarMesa` (function), `onClickLiberarMesa` (function), `onChangeInputMesa` (function), `onSelectCapacidad` (function), `onChangeInputCliente` (function)
2. **El componente debe tener:**
   - Sección "Reservar Mesa":
     - Input para nombre del cliente
     - Select para capacidad (2/4/6 personas)
     - Botón "Reservar Mesa"
   - Sección "Liberar Mesa":
     - Input para número de mesa
     - Botón "Liberar Mesa"

`Ejercicio 3`: Página de Restaurante con Socket.IO

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

Backend provisto

Servidor: `http://10.1.5.137:4000`

Eventos que ENVIÁS:

Evento	Datos
`join_restaurante`	`{ alumnoId }`
`reservar_mesa`	`{ mozo, cliente, capacidad }`
`liberar_mesa`	`{ numeroMesa }`

Eventos que RECIBÍS:

Evento	Datos
`joined_OK_restaurante`	`{ room, restaurante }`
`restaurante_actualizado`	`{ restaurante }`
`error_restaurante`	`{ mensaje }`

```
// Datos que recibis:
{
  room: number,
  restaurante: {
    mesas: [
      {
        numero: number,
        capacidad: 2 | 4 | 6,
        estado: "libre" | "reservada",
        cliente: string | null
      },
      ...
    ],
    totalReservas: number,
    mesasLibres: number
  }
}
```

Requisitos Frontend

1. Conexión con Socket.IO

- Conectar a `http://10.1.5.137:4000`
- Obtener `mozo` y `alumnoId` desde query params

2. Unirse al restaurante

- Botón "Conectarse"
- Emitir `join_restaurante` con `{ alumnoId }`
- Escuchar `joined_OK_restaurante` y guardar mesas

3. Reservar mesa (Componente `FormularioReserva`)

- Input para nombre del cliente (validar que no esté vacío)
- Select para capacidad (2/4/6 personas)
- Emitir `reservar_mesa` con `{ mozo, cliente, capacidad }`

4. Liberar mesa (Componente `FormularioReserva`)

- Input para número de mesa
- Emitir `liberar_mesa` con `{ numeroMesa }`

5. Escuchar actualizaciones

- Evento `restaurante_actualizado`: Mostrar y actualizar la lista de mesas con el componente `MesaRestaurante`
- Evento `error_restaurante`: Mostrar alert con el error

6. Conditional Rendering

- Mostrar "Sala: {alumnoId}" cuando conectado
- Mostrar "Mozo: {mozo}"
- Mostrar "Mesas libres: X/12"
- Deshabilitar botones si no está conectado

`Ejercicio 4`: Completar el Backend - Socket.IO

Completar **líneas** relacionadas con Socket.IO. Crear un archivo `backend.js` y complete las siguientes porciones de código:

---

a. Emitir estado inicial al conectarse

En el evento `join_restaurante`, completa la línea para enviar el estado inicial **SOLO al cliente que se conectó**:

```
socket.on("join_restaurante", ({ alumnoId }) => {
  const ROOM = Number(alumnoId);
  socket.join(ROOM);
  socket.data.room = ROOM;

  // ... código de inicialización ...

  // Emitir 'joined_OK_restaurante' SOLO a este socket
  // Debe enviar: { room: ROOM, restaurante: restauranteSalas[ROOM] }
  // COMPLETAR CON EL EVENTO ACÁ ABAJO:



});
```

---

b. Notificar actualización a toda la sala

En el evento `reservar_mesa`, completar la línea para notificar a **TODOS los clientes de la sala**:

```
socket.on("reservar_mesa", ({ mozo, cliente, capacidad }) => {
  const ROOM = socket.data.room;

  // ... validaciones y lógica ...

  // Emitir 'restaurante_actualizado' a TODA LA SALA
  // Debe enviar: { restaurante: restauranteSalas[ROOM] }
  // COMPLETAR CON EL EVENTO ACÁ ABAJO:



});
```

---

c. Emitir error solo al cliente

En el evento `liberar_mesa`, completa la línea para enviar un error **SOLO al mozo que intentó liberar la mesa**:

```
socket.on("liberar_mesa", ({ numeroMesa }) => {
  const ROOM = socket.data.room;

  // Buscar mesa
  const mesa = restaurante.mesas.find(
    m => m.numero === Number(numeroMesa)
  );

  if (!mesa) {
    // Emitir 'error_restaurante' SOLO a este socket
    // Debe enviar: { mensaje: "Mesa no encontrada" }
    // COMPLETAR CON EL EVENTO ACÁ ABAJO:



    return;
  }

  // ... resto del código ...
});
```
