Instituto Pío IX Casa Salesiana

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 Parking/Estacionamiento en Tiempo Real

Desarrollar una aplicación de gestión de estacionamiento donde múltiples empleados registran entrada y salida de vehículos 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 empleado (`empleado`) (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 Parking"
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 Parking", redirigir a `/parking`
   - Pasar `empleado` y `alumnoId` como query params
     - URL: `http://localhost:3000/parking?empleado=Juan&alumnoId=4`

`Ejercicio 2`: Creación de componentes

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

1. Reciba por props: `numero` (number), `tipo` (string), `estado` (string), `patente` (string o null)
2. **Conditional Rendering**:
   - Si estado es "libre": mostrar " LIBRE"
   - Si estado es "ocupado": mostrar " OCUPADO - Patente: {patente}"
   - Si tipo es auto o moto: mostrar "Tipo: 🚗" para auto, "Tipo: 🏍️" para moto
3. Mostrar Espacio N°: {numero}

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

1. Reciba por props: `onClickEntrada` (function), `onClickSalida` (function), `onChangePatente` (function), `onChangeSelectTipo` (function)
2. **El componente debe tener:**
   - Sección "Registrar Entrada":
     - Input para patente
     - Select para tipo (auto/moto)
     - Botón "Registrar Entrada"
   - Sección "Registrar Salida":
     - Input para patente
     - Botón "Registrar Salida"

`Ejercicio 3`: Página de Parking con Socket.IO - Frontend

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

Backend Provisto

Servidor: `http://10.1.5.137:4000`

Eventos Socket.IO

Eventos que ENVIÁS:

Evento	Datos
`join_parking`	`{ alumnoId }`
`registrar_entrada`	`{ empleado, patente, tipo }`
`registrar_salida`	`{ patente }`

Eventos que RECIBÍS:

Evento	Datos
`joined_OK_parking`	`{ room, parking }`
`parking_actualizado`	`{ parking }`
`error_parking`	`{ mensaje }`

```
// Datos que recibis:
{
  room: number,
  parking: {
    espacios: [
      {
        numero: number,
        tipo: "auto" | "moto",
        estado: "libre" | "ocupado",
        patente: string | null
      },
      ...
    ],
    totalEntradas: number,
    totalSalidas: number,
    espaciosLibres: number
  }
}
```

Requisitos Frontend

1. Conexión con Socket.IO

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

2. Unirse al parking

- Botón "Conectarse"
- Emitir `join_parking` con `{ alumnoId }`
- Escuchar `joined_OK_parking` y guardar espacios

3. Registrar entrada (Componente `FormularioVehiculo`)

- Input para patente (validar que no esté vacío)
- Select para tipo (auto/moto)
- Emitir `registrar_entrada` con `{ empleado, patente, tipo }`

4. Registrar salida (Componente `FormularioVehiculo`)

- Input para patente
- Emitir `registrar_salida` con `{ patente }`

5. Escuchar actualizaciones

- Evento `parking_actualizado`: Mostrar y actualizar la lista de espacios con el componente `EspacioParking`
- Evento `error_parking`: Mostrar alert con el error

6. Conditional Rendering

- Mostrar "Sala: {alumnoId}" cuando conectado
- Mostrar "Empleado: {empleado}"
- Mostrar "Espacios libres: X/10"
- 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_parking`, completa la línea para enviar el estado inicial **SOLO al cliente que se conectó**:

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

  // ... código de inicialización ...

  // Emitir 'joined_OK_parking' SOLO a este socket
  // Debe enviar: { room: ROOM, parking: parkingSalas[ROOM] }
  // COMPLETAR CON EL EVENTO ACÁ ABAJO:



});
```

---

b. Notificar actualización a toda la sala

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

```
socket.on("registrar_entrada", ({ empleado, patente, tipo }) => {
  const ROOM = socket.data.room;

  // ... validaciones y lógica ...

  // Emitir 'parking_actualizado' a TODA LA SALA
  // Debe enviar: { parking: parkingSalas[ROOM] }
  // COMPLETAR CON EL EVENTO ACÁ ABAJO:



});
```

---

c. Emitir error solo al cliente

En el evento `registrar_salida`, completa la línea para enviar un error **SOLO al empleado que intentó registrar la salida**:

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

  // Buscar vehículo
  const espacioOcupado = parking.espacios.find(
    e => e.patente === patente
  );

  if (!espacioOcupado) {
    // Emitir 'error_parking' SOLO a este socket
    // Debe enviar: { mensaje: "Vehículo no encontrado" }
    // COMPLETAR CON EL EVENTO ACÁ ABAJO:



    return;
  }

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