EXAMEN FINAL 2026 — Desarrollo Web en Entorno Cliente (React + TypeScript)

Aplicación: "IncidTech" — Gestor de Incidencias IT

> **Duración:** 3 horas
>
> **Material permitido:** Repositorio del curso (CursoReact2526-t)
>
> **Herramientas:** Visual Studio Code con perfil "Examen" (extensiones limitadas)
>
> **Prohibido:** Uso de IA (Copilot, ChatGPT, etc.), búsquedas en internet sólo para consultar la documentación oficial de React, TypeScript o las herramientas usadas.

Descripción general

Debes desarrollar una aplicación en **React + TypeScript** que permita gestionar incidencias de un departamento de IT. La aplicación consumirá un **backend ya desplegado** en la dirección que se indicará al inicio del examen.

Funcionalidades principales:

- Cualquier visitante puede **ver** la lista de incidencias (sin necesidad de estar logueado)
- Un usuario logueado puede **crear** nuevas incidencias y **eliminar** incidencias existentes
- La aplicación tendrá varias páginas con navegación mediante React Router

Lo que se te proporciona (scaffolding)

Se te entrega un proyecto Vite ya configurado con:

- `index.css` con clases de utilidad de Tailwind listas para usar
- `src/types/index.ts` con los tipos ya definidos (ver más abajo)
- `api.http` con ejemplos de peticiones al backend (para probar con REST Client o Thunder Client)

Endpoints del backend

La URL base del backend es: `http://192.168.50.120:1495/api`

MÉTODO	RUTA	AUTH REQUERIDA	DESCRIPCIÓN	BODY (JSON)	RESPUESTA
POST	`/auth/login`	No	Iniciar sesión	`{ email: string, password: string }`	`AuthResponse`
GET	`/auth/me`	Sí (Bearer)	Obtener usuario actual	—	`User`
GET	`/incidents`	No	Listar todas las incidencias	—	`IncidentsResponse`
POST	`/incidents`	Sí (Bearer)	Crear una nueva incidencia	`{ title, description, priority }`	`IncidentResponse`
DELETE	`/incidents/:id`	**Sí (Bearer)**	**Eliminar una incidencia por su ID**	—	`MessageResponse`

Cabecera de autenticación: Para los endpoints protegidos, debes enviar:

```
Authorization: Bearer <token>
```

Usuarios de prueba disponibles en el backend:

EMAIL	PASSWORD
alumno@test.com	123456
profesor@test.com	**123456**

Clases CSS disponibles en `index.css`

El fichero `index.css` incluye Tailwind y un conjunto de **clases de utilidad predefinidas** que puedes usar directamente en tus componentes. También puedes usar las clases de Tailwind directamente si lo prefieres.

Ejemplo de uso de badges con renderizado condicional:

```
{
  /* La clase del badge se construye dinámicamente con la prioridad */
}
<span className={`badge badge-${incident.priority}`}>{incident.priority}</span>;
```

APARTADOS DEL EXAMEN

APARTADO 1 — Capa de servicios API (1 punto)

Crea un fichero `src/services/api.ts` con las funciones necesarias para comunicarte con el backend.

Debes crear las siguientes funciones:

- `login(credentials: LoginDTO): Promise<AuthResponse>` — Envía email y password al endpoint de login
- `getMe(token: string): Promise<User>` — Obtiene el usuario actual usando el token
- `getIncidents(): Promise<IncidentsResponse>` — Obtiene la lista de incidencias (no requiere token)
- `createIncident(token: string, incident: CreateIncidentDTO): Promise<IncidentResponse>` — Crea una incidencia nueva (requiere token)
- `deleteIncident(token: string, id: number): Promise<MessageResponse>` — Elimina una incidencia (requiere token)

Requisitos:

- Usa `fetch` para las peticiones HTTP
- Las funciones que requieren autenticación deben recibir el `token` como parámetro y enviarlo en la cabecera `Authorization` con el formato `Bearer <token>`
- Todas las funciones deben usar `async/await`
- Usa los tipos proporcionados en `src/types/index.ts` para tipar los parámetros y los valores de retorno

APARTADO 2 — Contexto de autenticación: `AuthContext` (2 puntos)

Crea el contexto de autenticación en `src/context/AuthContext.tsx`.

El contexto debe gestionar y exponer lo siguiente:

- `user: User | null` — el usuario logueado (o `null` si no hay sesión)
- `token: string | null` — el token JWT recibido del backend
- `loading: boolean` — indica si se está verificando la sesión al cargar la app
- `error: string | null` — mensaje de error si el login falla
- `login(email: string, password: string): Promise<void>` — función para iniciar sesión
- `logout(): void` — función para cerrar sesión

Requisitos obligatorios:

1. Al hacer **login** correctamente, guardar el token en `localStorage`
2. Al hacer **logout**, eliminar el token de `localStorage` y poner el usuario a `null`

3. Al **cargar la aplicación** (montaje inicial con `useEffect`), comprobar si hay un token guardado en `localStorage`. Si lo hay, llamar a `getMe(token)` para recuperar los datos del usuario. Si el token ya no es válido, eliminarlo del `localStorage`
4. Mientras se comprueba la sesión al inicio, `loading` debe estar en `true`
5. Si el login falla, guardar el mensaje de error en `error`
6. Crea un **custom hook** `useAuth()` que devuelva el contexto y lance un error si se usa fuera del `AuthProvider`

APARTADO 3 — Contexto de incidencias: `IncidentsContext` (2 puntos)

Crea el contexto de incidencias en `src/context/IncidentsContext.tsx`.

El contexto debe gestionar y exponer lo siguiente:

- `incidents: Incident[]` — la lista de incidencias
- `loading: boolean` — indica si se están cargando las incidencias
- `fetchIncidents(): Promise<void>` — obtiene la lista de incidencias del backend
- `addIncident(incident: CreateIncidentDTO): Promise<void>` — crea una nueva incidencia
- `removeIncident(id: number): Promise<void>` — elimina una incidencia

Requisitos obligatorios:

1. Las incidencias se deben cargar automáticamente al montar el componente (usa `useEffect`)
2. `addIncident` y `removeIncident` necesitan el token del usuario para comunicarse con el backend. Usa el hook `useAuth()` para obtenerlo
3. Tras crear o eliminar una incidencia, la lista debe actualizarse. Puedes volver a llamar a `fetchIncidents()` o actualizar el estado local directamente
4. Crea un **custom hook** `useIncidents()` que devuelva el contexto

APARTADO 4 — Páginas y componentes (2.5 puntos)

Crea las siguientes páginas y componentes:

4.1 — Componente `IncidentCard` (0.75 puntos)

Archivo: `src/components/IncidentCard.tsx`

Componente que muestra la información de una incidencia individual.

Debe mostrar:

- **Título** de la incidencia
- **Descripción**
- **Prioridad** con un estilo visual diferente según el valor:

  - `"alta"` → rojo (ej: texto rojo, badge rojo, borde rojo...)
  - `"media"` → amarillo/naranja
  - `"baja"` → verde
- **Estado**: mostrar visualmente si la incidencia está `"abierta"` o `"resuelta"`
- **Botón "Eliminar"** que **solo sea visible si el usuario está logueado**

Requisitos:

- El componente recibe la incidencia como **prop tipada** (`Incident`)
- El botón "Eliminar" debe llamar a `removeIncident(id)` del `IncidentsContext`
- Usa **renderizado condicional** para:
  - Mostrar/ocultar el botón según si hay usuario logueado
  - Aplicar estilos diferentes según la prioridad

4.2 — Componente `IncidentForm` (0.75 puntos)

Archivo: `src/components/IncidentForm.tsx`

Formulario para crear una nueva incidencia.

Campos del formulario:

- **Título** (input de tipo text, obligatorio)
- **Descripción** (textarea, obligatorio)
- **Prioridad** (select con opciones: `"alta"`, `"media"`, `"baja"`)

Requisitos:

- Usa `useState` para manejar el estado del formulario (puedes usar un estado por campo o un objeto)
- Al enviar el formulario (`onSubmit`), llama a `addIncident()` del `IncidentsContext`
- Tras crear la incidencia exitosamente, **limpia el formulario** (resetea los campos)
- El título y la descripción no pueden estar vacíos: haz una **validación básica** antes de enviar (si están vacíos, no envíes la petición)

4.3 — Página `HomePage` (0.25 puntos)

Archivo: `src/pages/HomePage.tsx`

Página pública de inicio.

- Muestra un título de bienvenida a la aplicación (ej: "Bienvenido a IncidTech y el nombre del alumno que hace el examen")
- Incluye un enlace o botón para navegar a la lista de incidencias

4.4 — Página `IncidentsPage` (0.5 puntos)

Archivo: `src/pages/IncidentsPage.tsx`

Página pública que muestra la lista de incidencias.

Requisitos:

- Usa el hook `useIncidents()` para obtener la lista y el estado de carga
- Mientras `loading` sea `true`, muestra un mensaje de "Cargando incidencias..."
- Renderiza un componente `IncidentCard` por cada incidencia de la lista
- Si la lista está vacía (y no está cargando), muestra un mensaje como "No hay incidencias registradas"

4.5 — Página `LoginPage` (0.25 puntos)

Archivo: `src/pages/LoginPage.tsx`

Página con formulario de login.

Requisitos:

- Dos campos: **email** y **password** controlados con `useState`
- Al enviar el formulario, llama a `login()` del `AuthContext`
- Si hay un **error**, muéstralo en pantalla (ej: "Credenciales incorrectas")
- Si el login es **correcto**, redirige al usuario a la página de incidencias (usa `useNavigate` de React Router)

APARTADO 5 — Routing y navegación (1.5 puntos)

Configura React Router en `App.tsx`.

Rutas que debes crear:

RUTA	PÁGINA/COMPONENTE	ACCESO
`/`	`HomePage`	Público
`/incidents`	`IncidentsPage`	Público
`/new-incident`	`NewIncidentPage`	Protegido
`/login`	`LoginPage`	Público
`*`	**Página 404**	**Público**

5.1 — Componente `Navbar` (0.5 puntos)

Archivo: `src/components/Navbar.tsx`

Barra de navegación visible en todas las páginas.

Requisitos:

- Muestra enlaces a: **Inicio** (`/`) e **Incidencias** (`/incidents`)
- Si el usuario **NO** está logueado: mostrar un enlace a **"Iniciar sesión"** (`/login`)

- Si el usuario **SÍ** está logueado: mostrar el **nombre del usuario** y un botón **"Cerrar sesión"** que llame a `logout()`
- El enlace a **"Nueva incidencia"** (`/new-incident`) solo debe ser visible para usuarios logueados

5.2 — Componente `ProtectedRoute` (0.5 puntos)

Archivo: `src/components/ProtectedRoute.tsx`

Componente que protege rutas que requieren autenticación.

Requisitos:

- Si `loading` es `true` (se está verificando la sesión), muestra un mensaje de carga
- Si el usuario **no** está logueado (`user` es `null` y `loading` es `false`), redirige a `/login` usando el componente `Navigate` de React Router
- Si el usuario **sí** está logueado, renderiza el contenido hijo (`children`)

5.3 — Página `NewIncidentPage` (0.25 puntos)

Archivo: `src/pages/NewIncidentPage.tsx`

Página protegida que contiene el componente `IncidentForm`.

- Solo accesible para usuarios logueados (envuelta en `ProtectedRoute`)
- Muestra un título como "Nueva incidencia" y debajo el formulario

5.4 — Configuración de rutas en `App.tsx` (0.25 puntos)

Requisitos:

- Configura `BrowserRouter` con todas las rutas indicadas en la tabla
- Envuelve la aplicación con los providers necesarios: `AuthProvider` y `IncidentsProvider`
- La ruta `/new-incident` debe estar protegida con el componente `ProtectedRoute`
- La `Navbar` debe ser visible en todas las páginas

APARTADO 6 — Funcionamiento completo e integración (1 punto)

Este apartado evalúa que **todo funcione correctamente de forma integrada**:

- **(0.25 pts)** La aplicación arranca sin errores en la consola del navegador
- **(0.25 pts)** El flujo de login funciona correctamente: el usuario se loguea, se persiste la sesión en `localStorage`, y al recargar la página el usuario sigue logueado
- **(0.25 pts)** El CRUD funciona contra el backend: se listan, crean y eliminan incidencias correctamente
- **(0.25 pts)** La interfaz es usable: se entiende qué hacer en cada pantalla, los botones funcionan, hay feedback visual (mensajes de carga, errores, lista vacía)

RESUMEN DE LA RÚBRICA

APARTADO	DESCRIPCIÓN	PUNTUACIÓN
1	Capa de servicios API (`services/api.ts`)	1.0
2	AuthContext + useAuth + persistencia en localStorage	2.0
3	IncidentsContext + useIncidents + CRUD	2.0
4	Páginas y componentes (cards, formulario, páginas)	2.5
5	Routing, Navbar, ProtectedRoute y App.tsx	1.5
6	Funcionamiento completo e integración	1.0
TOTAL		**10.0**

Estructura de archivos esperada (orientativa)

```
src/
├── components/
│   ├── Navbar.tsx
│   ├── ProtectedRoute.tsx
│   ├── IncidentCard.tsx
│   └── IncidentForm.tsx
├── context/
│   ├── AuthContext.tsx          (incluye useAuth)
│   └── IncidentsContext.tsx     (incluye useIncidents)
├── pages/
│   ├── HomePage.tsx
│   ├── IncidentsPage.tsx
│   ├── LoginPage.tsx
│   └── NewIncidentPage.tsx
├── services/
│   └── api.ts
├── types/
│   └── index.ts                 (proporcionado)
├── App.tsx
├── main.tsx
└── index.css                    (proporcionado)
```

> **Nota:** La estructura de carpetas es orientativa. Puedes organizar los ficheros como prefieras siempre que la funcionalidad sea correcta. Los custom hooks (`useAuth`, `useIncidents`) pueden estar dentro de los ficheros de contexto o en ficheros separados en una carpeta `hooks/`.

Notas importantes

- **No necesitas implementar registro de usuarios.** Solo login con los usuarios de prueba proporcionados
- **No necesitas implementar edición de incidencias.** Solo crear, listar y eliminar
- Puedes usar `Outlet` de React Router si lo deseas, pero **no es obligatorio**
- No te preocupes por hacer la aplicación responsive ni por un diseño visual elaborado. Lo importante es que sea **funcional y comprensible**
- Usa los tipos proporcionados en `src/types/index.ts`. No necesitas crear tipos adicionales salvo los que necesites para tipar el estado de tus contextos
- Si tienes dudas sobre el formato de las respuestas del backend, consulta los tipos `AuthResponse`, `IncidentsResponse`, etc.
- Puedes usar el fichero `api.http` proporcionado para probar los endpoints del backend con REST Client o Thunder Client antes de implementar las funciones en código

---

Referencia visual — Así debería verse la aplicación

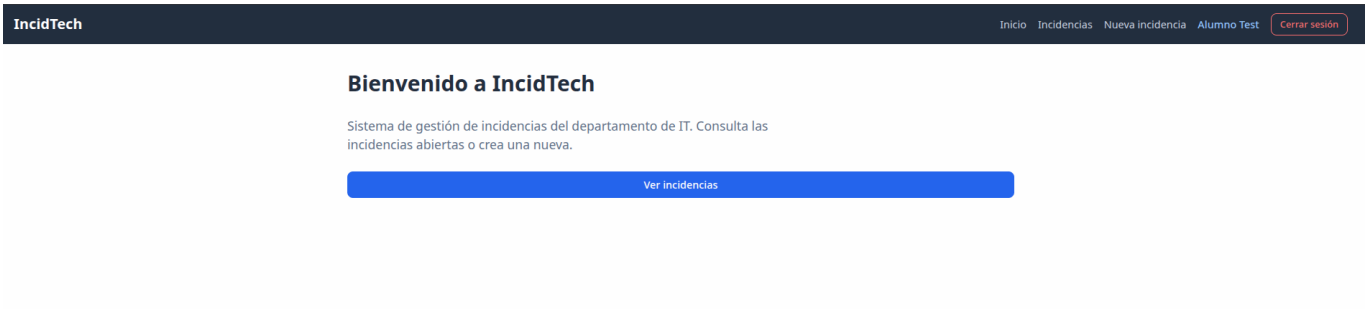
A continuación se muestran capturas de pantalla orientativas del resultado esperado. El diseño no tiene que ser idéntico, pero sí debe incluir los elementos y la estructura que se describen en cada apartado.

Página de inicio (usuario no logueado)

La página principal muestra un mensaje de bienvenida y un botón para acceder a las incidencias. En la barra de navegación se ve el enlace de **Iniciar sesión**.

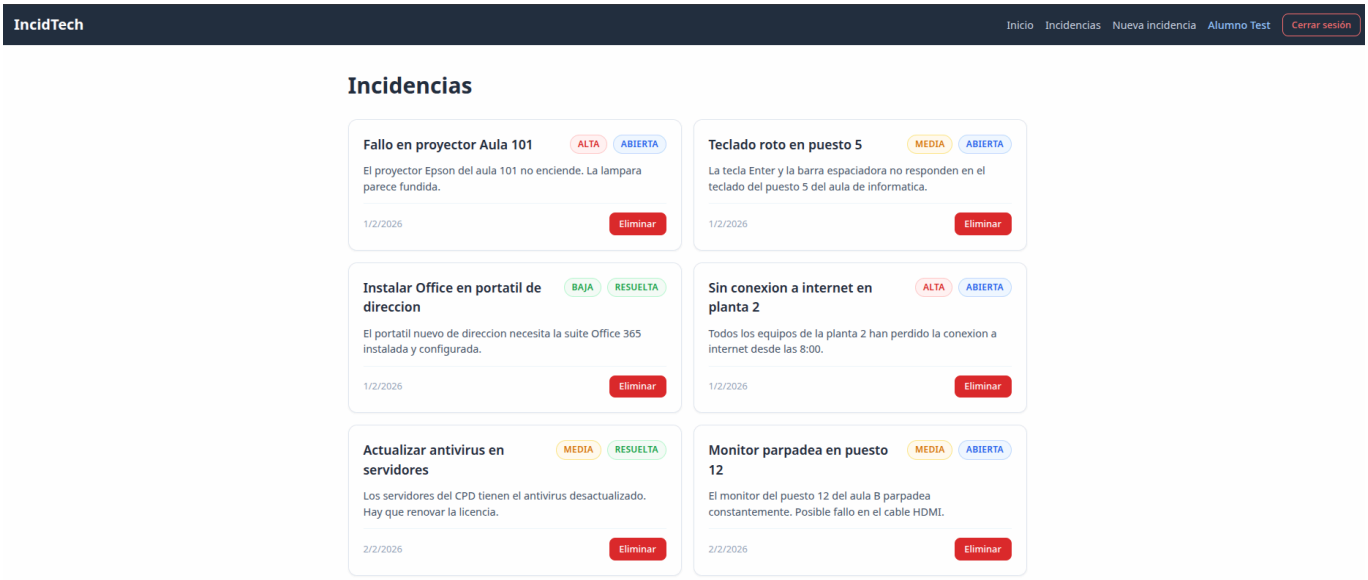
Página de inicio (usuario logueado)

Tras iniciar sesión, la barra de navegación muestra el **nombre del usuario**, el enlace a **Nueva incidencia** y el botón de **Cerrar sesión**.



Lista de incidencias

Se muestran las incidencias en forma de tarjetas (cards) con su título, descripción, badges de **prioridad** y **estado**, y el botón **Eliminar** (visible solo para usuarios logueados).



Formulario de nueva incidencia

La página protegida permite crear una nueva incidencia con los campos de título, descripción y prioridad.