Enunciado de la Prueba de desempeño

Next JS

1. Caso de uso:

HelpDeskPro es una empresa que presta servicios de soporte técnico a clientes internos y externos. Actualmente gestionan los requerimientos de soporte mediante correos sueltos, chats y hojas de cálculo, lo que ha generado varios problemas:

- No hay un registro centralizado de los tickets.
- Se pierden correos o se responden tarde, generando mala experiencia del cliente.
- No existe un seguimiento claro del estado de cada ticket (abierto, en progreso, resuelto, cerrado).
- Los agentes no tienen prioridad ni recordatorios sobre los tickets sin respuesta.
- La gerencia no puede medir tiempos de respuesta ni estados globales del soporte.

La dirección decide construir una **aplicación web interna con Next.js y TypeScript** que permita gestionar de forma eficiente los tickets, los usuarios (clientes y agentes), las respuestas, las notificaciones por correo y tareas programadas de recordatorio.

2. Objetivo:

Construir una aplicación con React + TypeScript + Next.js, utilizando Hooks, Context API, formularios tipados, Axios y MongoDB (Mongoose), para digitalizar y optimizar la gestión de tickets de soporte de HelpDeskPro, garantizando trazabilidad, tiempos de respuesta claros y comunicación efectiva por correo.

El modelado de interfaces (User, Ticket, Comment), la autenticación con roles, la API CRUD de tickets y la automatización mediante cron jobs forman parte del alcance del ejercicio.

El sistema deberá:

- Centralizar los tickets, usuarios y comentarios en un mismo sistema.
- Facilitar la gestión de tickets: crear, responder, actualizar estado y cerrar.
- Separar vistas y permisos entre usuarios cliente y usuarios agente.
- Enviar correos electrónicos automáticos ante eventos clave (creación, respuesta, cierre).
- Ejecutar al menos un cron job para detectar tickets sin respuesta y enviar recordatorios a los agentes, y opcionalmente encuestas posteriores al cierre.
- Aplicar principios de tipado fuerte y componentización reutilizable (Button, Badge, Card) para garantizar consistencia.

3. Funcionalidades principales:
Para alcanzar un resultado óptimo en esta prueba, deberás cumplir cada uno de los siguientes requisitos y funcionalidades:

3.1) Gestión de Tickets de Soporte:

- Title
- Description
- createdBy (referencia al usuario cliente)
- assignedTo (opcional, referencia al agente)
- status (por ejemplo: "open" | "in_progress" | "resolved" | "closed")
- priority (por ejemplo: "low" | "medium" | "high")
- createdAt, updatedAt

Requisitos:

- Crear nuevos tickets desde el panel de usuario cliente.
- Editar/actualizar información relevante del ticket (status, priority, agente asignado) desde el panel de agente.
- Cerrar un ticket y cambiar su estado a closed.
- Listar tickets:
  - Para el cliente: solo sus propios tickets.
  - Para el agente: todos los tickets, con filtros por estado y prioridad.
- Formularios de creación/edición controlados y tipados en TypeScript.

3.2) Gestión de Usuarios y Autenticación con Roles
Requisitos:

- Módulo de autenticación (login) con validación de credenciales.
- Al menos dos roles:
  - client (usuario que crea tickets)
  - agent (usuario que gestiona/responde tickets)
- Tras login exitoso:
  - Redirigir al panel de usuario si el rol es client.
  - Redirigir al panel de agente si el rol es agent.
- Mantener el estado de autenticación con Context API (usuario autenticado, rol, token o sesión).
- Proteger rutas:
  - Panel de cliente accesible solo para client.
  - Panel de agente accesible solo para agent.

3.3) Comentarios y Respuestas en Tickets
Cada ticket debe tener un hilo de comentarios/respuestas.
Requisitos:

- Modelo Comment o estructura equivalente con:

- ticketId
  - author (usuario que comenta)
  - message
  - createdAt
- Desde el panel de usuario:
  - El cliente puede agregar comentarios (por ejemplo, adjuntar más información o responder a la solución propuesta).
- Desde el panel de agente:
  - El agente puede responder los tickets con comentarios.
- Mostrar los comentarios en orden cronológico en la vista de detalle de un ticket.

3.4) UI Reutilizable

Construir componentes UI reutilizables con props tipadas:

- Button (variantes y tamaños).
- Badge (por ejemplo para estado o prioridad del ticket).
- Card (para mostrar resumen de ticket).

Requisitos:

- Integrar al menos un Badge y un Button dentro de cada Card del listado de tickets.
- Mostrar un grid o lista de Cards con:
  - Título
  - Estado
  - Prioridad
  - Fecha de creación
  - Botón para ver detalle

3.5) API, Servicios y Dashboard con Next.js + MongoDB

Requisitos:

- Definir modelos Mongoose mínimos:
  - User
  - Ticket
  - Comment
- Configurar una conexión estable a MongoDB.
- Implementar API en Next.js (por ejemplo /api/tickets, /api/comments, /api/auth/login, etc.) con operaciones:
  - Tickets:
    - GET (listar, filtrar por usuario o estado)
    - POST (crear)
    - PUT/PATCH (actualizar estado, prioridad o agente)
    - DELETE (opcional, según tus reglas de negocio)
  - Comments:
    - GET por ticket
    - POST para agregar comentarios

- Crear servicios Axios (por ejemplo: getTickets, createTicket, updateTicket, getCommentsByTicket, createComment) y consumirlos en los paneles (usuario/ agente).
- Construir un **Dashboard de agente** con:
    - Listado de tickets
    - Filtros por estado
    - Acciones para responder y cambiar estado

3.6) Notificaciones por Correo

El sistema debe enviar correos electrónicos al usuario cliente cuando:

- Se crea un ticket.
- Un agente agrega una respuesta/comentario al ticket.
- El ticket se cierra.

Requisitos:

- Implementar una capa de servicio de correo (usar librería de envío de emails).
- Desde la API de Next.js, disparar el envío de correo en los eventos anteriores.
- Es recomendable centralizar la lógica de envío en un helper o servicio reutilizable.

3.7) Manejo de Errores y Validaciones

Requisitos:

- Captura de errores con try/catch en:
    - Servicios Axios.
    - Rutas de API.
- Mostrar mensajes de error y éxito claros al usuario:
    - Ejemplo: “Ticket creado correctamente”, “No se pudo actualizar el ticket”, etc.
- Validaciones mínimas:
    - title y description obligatorios.
    - No permitir crear tickets sin usuario autenticado.
    - Roles respetados en cada operación (por ejemplo, solo agentes pueden cambiar el estado a resolved o closed).

4. <u>Criterios de aceptación</u>:

El sistema se considerará **aprobado** si cumple con los siguientes criterios:

4.1) Gestión de Tickets

- Se puede registrar un ticket con todos los datos obligatorios.
- Se puede editar el estado, prioridad y agente asignado del ticket desde el panel de agente.
- Se puede cerrar un ticket marcándolo como closed.
- Se pueden listar y filtrar tickets:

- Por usuario (cliente ve solo los suyos).
- Por estado y/o prioridad (en el panel de agente).

-

4.2) Gestión de Usuarios, Roles y Autenticación

- Existe un login funcional.
- La app redirecciona correctamente según el rol (client o agent).
- Las rutas están protegidas según el rol.
- El estado de sesión se maneja de forma centralizada (Context API u otra solución de estado global).

4.3) Comentarios y UI Reutilizable

- Cada ticket tiene un hilo de comentarios visible en su detalle.
- Tanto clientes como agentes pueden agregar comentarios según sus permisos.
- Las Cards de tickets se muestran con:
  - Badge(s) representando estado/ prioridad.
  - Button(s) para ver detalle o cambiar estado.
- Las props de componentes reutilizables están tipadas y tienen variantes/tamaños donde aplique.

4.4) API, Servicios y Dashboard

- La API responde correctamente a las operaciones definidas para tickets y comentarios (GET/POST/PUT/DELETE).
- Los servicios Axios consumen la API y el Dashboard permite:
  - Listar tickets.
  - Crear nuevos tickets (panel cliente).
  - Editar estado/prioridad/asignación y responder (panel agente).
- La app ejecuta sin errores con npm run dev (o el comando especificado en el README).

4.5) Notificaciones por Correo

- Al crear un ticket, se genera el envío de un correo al cliente.
- Cuando el agente responde un ticket, se dispara el envío de correo al cliente.
- Al cerrar un ticket, se envía correo de cierre al cliente.

4.6) Manejo de Errores y Validaciones

- Errores se muestran con mensajes claros al usuario (no solo en consola).
- Las validaciones de negocio se respetan:
  - No se crean tickets con campos obligatorios vacíos.

    - No se permiten acciones protegidas a usuarios sin rol adecuado.
- La aplicación no se rompe ante errores de red o de API (se manejan mediante try/catch y estados de error).

4.7) **Documentación**

El repositorio incluye un **README.md** con:

- Descripción general del sistema de soporte / tickets.
- Requisitos previos (MongoDB, variables de entorno para conexión y correo).
- Pasos para clonar, configurar y ejecutar el proyecto.
- Capturas de pantalla o gifs cortos del flujo principal:
    - Creación de ticket (cliente).
    - Gestión de ticket (agente).
    - Vista de comentarios.
- Datos del Coder:
    - Nombre
    - Clan
    - Correo
    - Documento de identidad

5. <u>Entregables</u>:

1. **Enlace al repositorio GitHub** (público).
2. **Proyecto comprimido (.zip)**.
3. **README** con instrucciones claras y datos del Coder (Nombre, Clan, correo, documento).