Justificación técnica – Sistema de Inventario de TI

1. Arquitectura general

El sistema se diseñó con una arquitectura de tres capas claramente separadas:

- **Frontend**: una Single Page Application (SPA) desarrollada con React y Vite.
- **Backend**: una API RESTful implementada con .NET 8 Web API.
- **Base de datos**: SQL Server, accedida mediante Entity Framework Core como ORM.

La comunicación entre frontend y backend se realiza vía HTTP sobre JSON, utilizando endpoints REST. El backend a su vez se conecta a la base de datos SQL Server mediante EF Core, donde se modelan las entidades principales: equipos, roles, empleados, solicitudes, detalles de solicitud, necesidades por rol e historial de asignaciones.

Para facilitar la ejecución y evaluación, se definió un entorno dockerizado con tres contenedores:

- Contenedor de SQL Server.
- Contenedor del backend (.NET 8).
- Contenedor del frontend (React compilado y servido por Nginx).

La orquestación se realiza con Docker Compose, de forma que toda la solución pueda levantarse con un solo comando.

2. Tecnologías elegidas y motivos

Backend – .NET 8 Web API + Entity Framework Core + SQL Server

- **.NET 8 Web API**: ofrece alto rendimiento, soporte a largo plazo y un modelo muy claro para exponer servicios RESTful. Es una tecnología con la que tengo experiencia previa (C#, ASP.NET), lo que permite implementar más rápido y poder explicarlo con detalle en una entrevista.
- **Entity Framework Core**: simplifica el acceso a datos mediante un ORM maduro. Permite mapear entidades C# a tablas SQL, manejar relaciones, y escribir consultas de manera tipada usando LINQ. Esto reduce el riesgo de errores en SQL manual y mejora el mantenimiento.

- **SQL Server**: es un motor robusto, alineado con el ecosistema .NET. Permite trabajar cómodamente con SQL Server Management Studio para crear y ajustar el esquema de base de datos, así como revisar datos durante las pruebas.

Frontend – React + Vite

- **React**: facilita la construcción de SPAs con componentes reutilizables, manejo de estado y renderizado eficiente. Es una tecnología muy usada en la industria y encaja bien con el requisito de construir una SPA que consuma la API.
- **Vite**: se utilizó como bundler y herramienta de desarrollo por su rapidez en el arranque, recarga y build optimizado para producción.
- **CSS sencillo y controles nativos**: se eligió un diseño limpio, utilizando CSS propio y elementos HTML nativos (tablas, formularios, selects), con un tema basado en blanco y verde similar a una interfaz de gobierno. Esto mantiene el enfoque en la funcionalidad y la claridad.

Orquestación – Docker + Docker Compose

- **Docker** permite empaquetar el backend, frontend y base de datos en contenedores independientes, asegurando que el evaluador pueda ejecutar el sistema sin problemas de configuración local.
- **Docker Compose** coordina el levantamiento de los tres servicios con redes internas y variables de entorno para cadenas de conexión y URL de API.

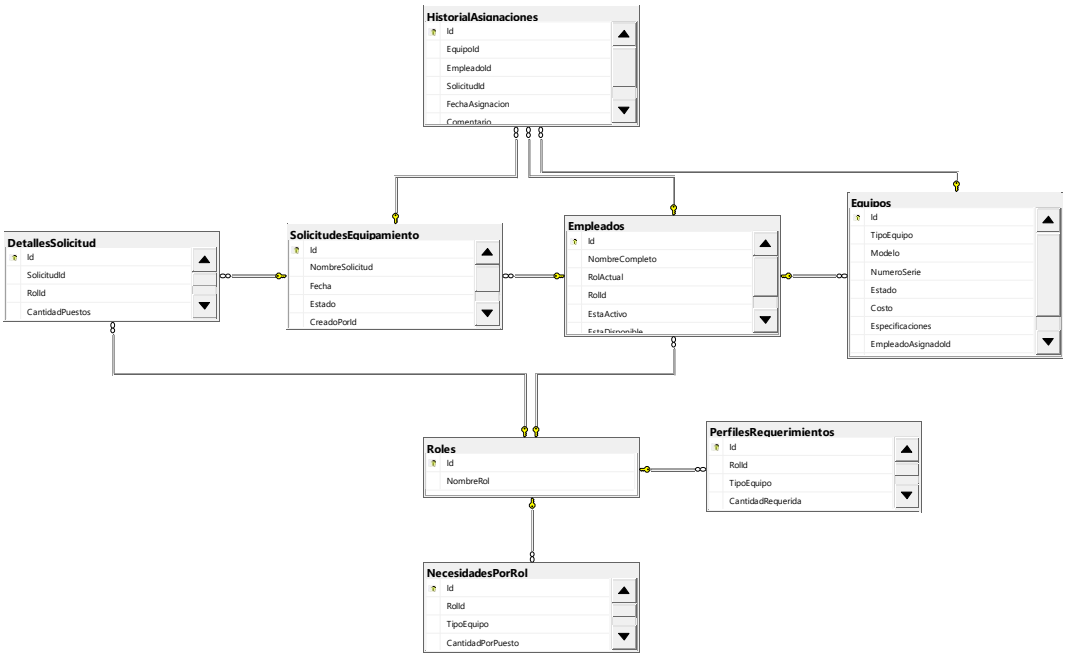
3. Esquema de base de datos

La base de datos modela las entidades necesarias para cumplir con los requisitos de inventario, solicitudes y trazabilidad de asignaciones:

- **Roles**: define los distintos tipos de rol (Diseñador, Desarrollador, Soporte, Analista, Project Manager, etc.).
- **Empleados**: almacena el nombre completo, el rol asociado y el estado (activo/disponible).
- **Equipos**: representa cada equipo físico del inventario, con tipo de equipo (Laptop, Monitor, Desktop, etc.), modelo, número de serie, costo, estado (disponible/asignado) y, opcionalmente, el empleado asignado.
- **SolicitudesEquipamiento**: cabecera de cada solicitud, con nombre, fecha y estado (pendiente/resuelta).

- **DetallesSolicitud**: relación entre cada solicitud y los roles requeridos, incluyendo cuántos puestos de cada rol se necesitan.
- **NecesidadesPorRol**: reglas de negocio que indican cuántos y qué tipos de equipos requiere cada rol por puesto (por ejemplo, un Desarrollador necesita 1 Laptop y 2 Monitores).
- **HistorialAsignacion**: registro inmutable de asignaciones realizadas cuando una solicitud se marca como resuelta (equipo, empleado, solicitud, fecha, comentario).

Diagrama base de datos



Las principales relaciones son:

- Empleados.RolId → Roles.Id (muchos empleados por rol).
- Equipos.EmpleadoAsignadoId → Empleados.Id (un equipo puede estar asignado a un empleado).
- DetallesSolicitud.SolicitudId → SolicitudesEquipamiento.Id (una solicitud tiene muchos detalles).
- DetallesSolicitud.RolId → Roles.Id (cada detalle está asociado a un rol).
- NecesidadesPorRol.RolId → Roles.Id (configuración por rol).
- HistorialAsignacion relaciona Equipo, Empleado y Solicitud.

4. Algoritmo de optimización

4.1 Criterio de “optimalidad”

El criterio de optimalidad elegido es:

1. **Cubrir tantos requerimientos como sea posible** de acuerdo con las necesidades por rol y el inventario disponible.
2. **Minimizar el costo total estimado**, seleccionando siempre los equipos más económicos dentro de cada tipo.
3. **Reportar claramente:**
    - Las asignaciones propuestas (por rol y por puesto).
    - Los faltantes por rol y tipo de equipo cuando el inventario no es suficiente.

En otras palabras, el algoritmo es un enfoque **greedy de menor costo**, sujeto a las restricciones del inventario disponible.

4.2 Funcionamiento paso a paso

De forma simplificada, el flujo es:

1. **Cargar la solicitud**:
    - Se obtiene la solicitud por id, incluyendo sus detalles (rol + cantidad de puestos).
2. **Obtener los perfiles de necesidades por rol**:
    - Para todos los roles presentes en la solicitud, se consultan sus registros en la tabla NecesidadesPorRol (por ejemplo, Desarrollador → Laptop x1, Monitor x2).
3. **Obtener inventario disponible**:
    - Se cargan todos los equipos con estado “disponible”.
    - Se agrupan por tipo de equipo (Laptop, Monitor, Desktop, etc.).
    - Dentro de cada tipo, se ordenan por costo ascendente (equipos más baratos primero).
4. **Recorrer los detalles de la solicitud**:
    - Para cada detalle (rol + cantidad de puestos), se procesa cada puesto de forma individual (por ejemplo, 3 desarrolladores → puesto 1, 2 y 3).
5. **Asignar equipos por puesto y tipo requerido**:
    - Para cada puesto, se revisan las necesidades del rol en NecesidadesPorRol.

- Para cada tipo de equipo requerido (ej. Laptop, Monitor):
    - Se intenta tomar de la lista de equipos disponibles del tipo correspondiente.
    - Siempre se toma el primer elemento, que es el más barato, y se elimina de la lista (para no reutilizarlo).
    - Se acumula el costo y se agrega el equipo a la lista de equipos asignados a ese puesto.

6. **Control de faltantes**:
    - Si en algún momento no hay suficientes equipos de un tipo (por ejemplo, ya no hay laptops), se registra un faltante en una estructura de agregación por (Rol, TipoEquipo), incrementando un contador de *cantidad faltante*.

7. **Construcción del resultado**:
    - Se arma un objeto de respuesta que incluye:
        - La lista de asignaciones por rol y puesto (qué equipos se asignan a cada uno).
        - El costo total estimado.
        - Una lista de faltantes por rol y tipo de equipo.
        - Un mensaje final indicando si la propuesta se puede cubrir completa o solo parcialmente.
8. **Marcado como resuelta**:
    - Si no hay faltantes, desde la vista de detalle se permite marcar la solicitud como “resuelta”.
    - En ese momento se actualizan:
        - El estado de los equipos (a “asignado”).
        - El empleado asociado (según el rol).
        - El registro en HistorialAsignacion.

4.3 Complejidad computacional aproximada

Sea:

- E = número total de equipos disponibles.
- T = número de tipos de equipo.
- D = número de detalles de solicitud (rol + cantidad).
- P = número total de puestos requeridos (suma de cantidades).
- R = número de roles distintos involucrados.

Las operaciones principales son:

1. Ordenar los equipos por costo dentro de cada tipo:
    - En el peor caso, se puede ver como O(E log E), aunque en la práctica se hace por grupos de tipo.
2. Recorrer cada puesto de cada rol y asignar equipos requeridos:
    - Aproximadamente O(P * k), donde k es la cantidad de tipos de equipo que requiere cada rol (normalmente pequeño).
3. Construcción de faltantes y agrupaciones:
    - O(P * k), lineal respecto al número de requerimientos.

En conjunto, la complejidad dominante es **O(E log E + P * k)**, lo que es perfectamente manejable para el tamaño de inventario típico y en muchos entornos reales.

4.4 Posibles mejoras o alternativas

En un entorno real, se podrían considerar mejoras como:

- **Priorización por empleados o proyectos**: incluir prioridad de ciertas solicitudes o roles, y resolver primero las más críticas.
- **Optimización avanzada**:
    - Modelar el problema como un flujo máximo con costo mínimo.
    - Utilizar algoritmos de programación lineal o entera (por ejemplo, con un solver tipo ILP) si el tamaño del problema lo amerita.
- **Reserve/hold de equipos**: permitir reservar equipos temporalmente para propuestas sin marcarlos todavía como asignados hasta la aprobación final.
- **Escalabilidad**:
    - Índices específicos en la base de datos para acelerar queries sobre equipos disponibles.
    - Paginación en listados de solicitudes e inventario para bases más grandes.

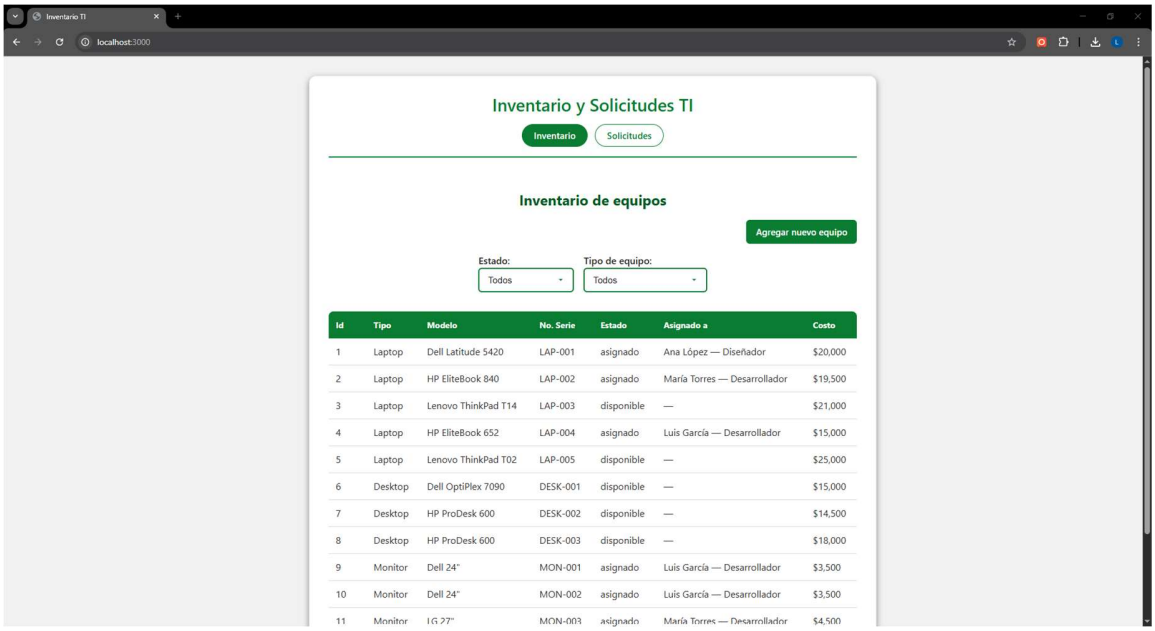
5. Decisiones de diseño

Algunas decisiones de diseño relevantes fueron:

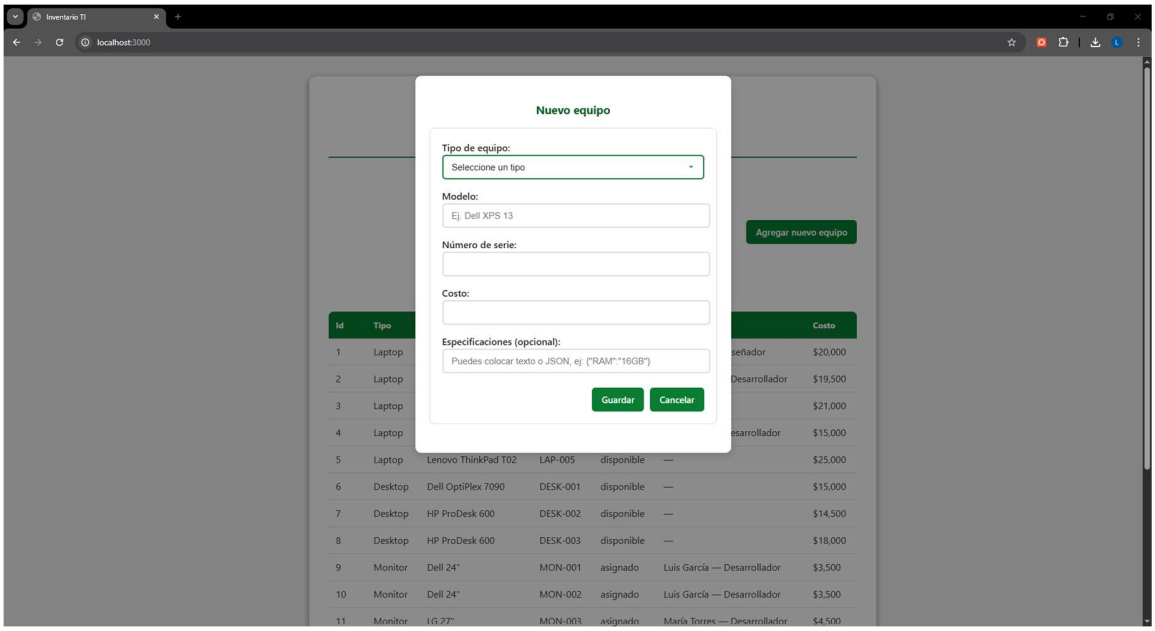
- **Separación de entidades y DTOs**: el backend utiliza entidades de dominio para la persistencia y DTOs específicos para exponer información al frontend. Esto evita exponer directamente el modelo interno y facilita cambios futuros.
- **Validaciones en backend**:
    - En creación de equipos se valida que el costo sea mayor a cero.
    - En creación de solicitudes se valida que haya al menos un rol con cantidad positiva.
    - Se retorna BadRequest con mensajes claros cuando faltan datos requeridos.
- **Manejo de errores en frontend**:
    - Se muestran mensajes de error cuando fallan las llamadas a la API (por ejemplo, problemas de red o errores del servidor).
    - Se deshabilita el botón de “Marcar como resuelta” cuando la propuesta presenta faltantes, para evitar estados inconsistentes.
- **Historial de asignaciones**:
    - En lugar de solo cambiar el estado del equipo, se registra cada asignación en la tabla HistorialAsignacion. Esto crea un log inmutable útil para auditorías y seguimiento.
- **CORS y comunicación entre servicios**:
    - Se habilitó CORS en el backend para permitir llamadas desde el origen del frontend (http://localhost:3000 y el puerto de desarrollo). Sin esto, el navegador bloquearía las peticiones.
- **Paginación y filtros**:
    - En inventario se implementan filtros por estado y tipo de equipo desde backend.
    - Para el tamaño esperado en la prueba no fue estrictamente necesario paginar, pero la estructura del controlador y de la API permite extenderlo fácilmente si creciera el volumen de datos.

- **UI centrada en claridad**:
    - Se optó por una interfaz simple, con pestañas para separar inventario y solicitudes, tablas claras y un modal para detalle y propuesta óptima. Esto ayuda a seguir el flujo de negocio sin sobrecargar la pantalla.

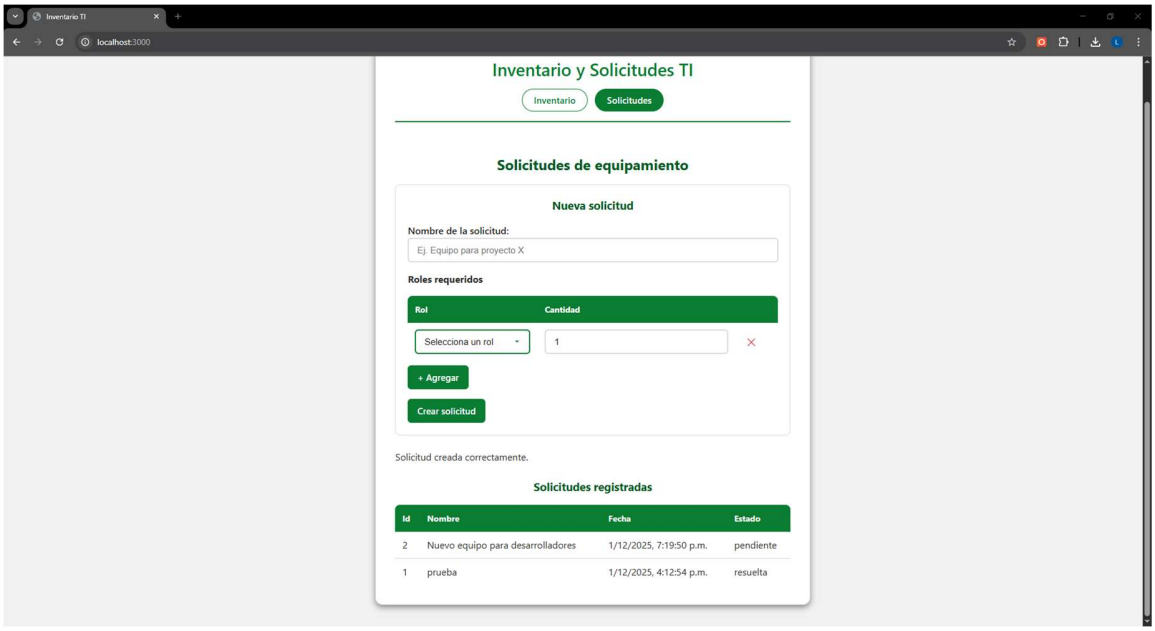
Dashboard de inventario



Agregar nuevo equipo



Detalle de solicitud y propuesta óptima



Detalle de solicitud y propuesta óptima