**INSTRUCCIONES GENERALES DEL PROYECTO**

**Requisito 1**: Crear una aplicación completa con frontend en React!

**Validación:**

El proyecto tiene una carpeta frontend con estructura de React.

Se usan componentes, rutas, contexto global y formularios.

El diseño y la lógica están implementados en React.

✅ Cumplido: El frontend está desarrollado completamente en React.

**Requisito 2:** Crear un backend y una base de datos llamada gestiones para operaciones CRUD!

**Validación:**

Existe backend en .NET Core con controladores para Solicitudes y TiposSolicitud.

El contexto GestionesDbContext define la base de datos gestiones con tablas para solicitudes y tipos de solicitud.

Los controladores implementan métodos CRUD: GET, POST, PUT, DELETE.

Se usa Entity Framework Core y migraciones para persistencia real.

✅ Cumplido: El backend y la base de datos gestiones permiten operaciones CRUD completas.

**Requisito 3:** Utilizar React hooks (useContext, useEffect, useState) y React Hook Form para formularios y validación!

**Validación:**

Se usan useContext, useEffect y useState en componentes clave (GestionList.js, TipoSolicitudList.js, AppContext.js, App.js, etc.).

Los formularios (GestionForm.js, TipoSolicitudForm.js, EditTipoModal.js, EditSolicitudModal.js) usan useForm de React Hook Form para gestión y validación.

La validación de campos y el manejo de errores se realiza con React Hook Form.

✅ Cumplido: Los hooks y React Hook Form se usan correctamente para formularios y lógica de estado.

**Requisito 4:** Utilizar Bootstrap para estilizar las vistas y hacerlas más atractivas

**Validación:**

**Home**

Usa Container, Row, Col, Card, Button de Bootstrap.

Iconos de Bootstrap, colores personalizados, estructura response.

Listado de funcionalidades con iconos y colores.

Diseño moderno y visualmente atractivo.

**Solicitudes**

Estructura con Container, Row, Col, Card.

Títulos con iconos, alertas visuales (alert-success, alert-danger), spinner de carga.

Formulario y listado integrados en un card Bootstrap.

Uso de clases de Bootstrap para estilo.

**Tipos de Solicitud**

Estructura con Container, Row, Col, Card.

Título con icono, alertas visuales, spinner de carga.

Formulario y listado integrados en card Bootstrap.

Colores y clases de Bootstrap para experiencia visual.

**Conclusión:**

Las vistas principales cumplen el requisito de utilizar Bootstrap para el estilo y hacerlas más atractivas. Se emplean componentes, clases y utilidades de Bootstrap, junto con iconos y colores personalizados, logrando una interfaz moderna.

**DETALLES DEL PROYECTO**

﻿**Frontend (React)**

|  |  |  |
| --- | --- | --- |
| Generales | Requerimiento | Validación |
| **Formulario de Solicitud** | ﻿Crear un formulario para agregar y actualizar solicitudes utilizando  React Hook Forms. Validar que ningún campo obligatorio quede  vacío | El formulario para agregar y actualizar solicitudes (GestionForm.js) utiliza React Hook Form y valida que ningún campo obligatorio quede vacío:  Campos validados como requeridos:  Nombre (required: true)  Fecha (required: true y no permite fechas futuras)  Tipo de Solicitud (required: true)  Mensajes de error:  Se muestran mensajes claros si algún campo obligatorio está vacío.  Integración con React Hook Form:  Uso de register, handleSubmit, formState.errors.  El formulario es reutilizable para agregar y actualizar. |
| ﻿Mostrar una lista de solicitudes obtenidas desde una API.  Implementar búsqueda y filtrado con filtero map. Incluir opciones  para editar y "eliminar" (cambiar estado a inactivo) las  solicitudes | Mostrar una lista de solicitudes obtenidas desde una API:  El contexto (AppContext.js) obtiene las solicitudes desde la API (getSolicitudes) y las expone vía solicitudes.  El componente GestionList.js consume solicitudes desde el contexto y las muestra en una tabla Bootstrap.  Implementar búsqueda y filtrado con filter o map  En GestionList.js, se usa .filter() para búsqueda por nombre/descripcion y por estado (activo/inactivo/todos).  Se usa .map() para renderizar la lista paginada de solicitudes.  Opciones para editar y "eliminar" (cambiar estado a inactivo)  Botón "Editar" abre un modal con el formulario para actualizar la solicitud.  Botón "Eliminar" llama a deleteSolicitud, que actualiza el estado de la solicitud a "inactivo" en la API y en el estado local. |
| **﻿Catálogo de Tipo de Solicitudes** | ﻿Crear un formulario para agregar y actualizar tipos de solicitudes | El componente TipoSolicitudForm.js utiliza React Hook Form.  Valida el campo "nombre" como obligatorio.  Permite agregar y actualizar tipos de solicitud (usando defaultValues para edición) |
| ﻿Mostrar una lista de tipos de solicitudes obtenidas desde una API, con opciones para editar y eliminar tipos | El componente TipoSolicitudList.js consume los tipos desde el contexto (tipos), que se obtienen vía API.  Muestra la lista en una tabla Bootstrap, con paginación y filtrado por nombre/estad.  Botón "Editar" abre un modal con el formulario para actualizar el tipo.  Botón "Eliminar" ejecuta la función onDelete, que elimina el tipo (o lo marca como inactivo según la lógica del contexto). |
| **﻿Uso de Hooks y Context** | ﻿Utilizar useContext para manejar el estado global de la aplicación y  useEffectpara obtener la lista de solicitudes y tipos de solicitudes al  cargar la página | Se utiliza useContext en los componentes principales (Solicitudes, TiposSolicitud, GestionList, TipoSolicitudList) para acceder al estado global (solicitudes, tipos, y funciones CRUD) definido en AppContext.  En AppContext.js, se emplea useEffect para obtener la lista de solicitudes y tipos de solicitudes desde la API al cargar la página, y se actualizan los estados globales correspondientes. |
| **﻿Estilización con Bootstrap** | ﻿Utilizar Bootstrap para estilizar los formularios y las listas, asegurando vistas responsivas y atractivas. | Formularios (GestionForm.js, TipoSolicitudForm.js):  Se usan clases Bootstrap: form-control, form-label, form-select, btn, alert.  Estructura responsiva con filas y columnas (row, col-md-6).  Mensajes de validación y alertas visuales.  Listas (GestionList.js, TipoSolicitudList.js):  Tablas responsivas con clases table, table-bordered, table-responsive.  Paginación con clases pagination, page-item, page-link.  Botones de acción con estilos Bootstrap (btn-warning, btn-danger, btn-sm, me-2).  Uso de modales Bootstrap para edición. |

**﻿**

**﻿Backend y Base de Datos**

|  |  |  |
| --- | --- | --- |
| Generales | Requerimiento | Validación |
| **﻿API de Solicitudes y Tipos de Solicitudes** | ﻿ Crear un backend que exponga las APIs necesarias para realizar  operaciones CRUD con las solicitudes y los tipos de solicitudes. | El backend expone las APIs necesarias para operaciones CRUD de solicitudes y tipos de solicitudes:   * **SolicitudesController**:   + GET: Listar todas las solicitudes.   + GET/{id}: Obtener una solicitud por ID.   + POST: Crear una nueva solicitud.   + PUT/{id}: Actualizar una solicitud existente.   + DELETE/{id}: Eliminar una solicitud. * **TiposSolicitudController**:   + GET: Listar todos los tipos de solicitud.   + GET/{id}: Obtener un tipo por ID.   + POST: Crear un nuevo tipo de solicitud.   + PUT/{id}: Actualizar un tipo existente.   + DELETE/{id}: Eliminar un tipo de solicitud.   Ambos controladores usan Entity Framework y exponen endpoints RESTful, cumpliendo el requisito de backend CRUD para ambos modelos. |
| **﻿** **﻿Base de Datos** | ﻿﻿Crear una base de datos llamada gestiones con dos tablas: Solicitudesy  TipoDeSolicitudes | **Tabla Solicitudes (Solicitud.cs):**  Campos: Id, Nombre, Descripcion, Fecha, Estado, TipoSolicitudId.  **Tabla TipoDeSolicitudes** (TipoSolicitud.cs):  Campos: Id, Nombre, Descripcion, Estado. |

**﻿**

.