**Manual Técnico**

1. **Roles/Integrantes**

* Elsa Valentina Trigo Maldonado – Team Leade
* Joel Israel Lopez Ticlla – Git Master
* Angela Nicole Sánchez Leaño - Bata base

1. **Introducción:**

El presente proyecto tiene como objetivo el desarrollo de una plataforma web destinada a la gestión de información médica para el **Servicio Departamental de Salud (SEDES)** en Cochabamba, Bolivia. Esta plataforma tiene como propósito principal mejorar la organización y acceso a los datos relativos a **hospitales, servicios médicos** y **especialidades** dentro del departamento. Al facilitar el acceso rápido y sencillo a esta información clave, se busca mejorar la administración de los recursos de salud, optimizando la eficiencia en la toma de decisiones.

1. **Descripción del proyecto:**

El proyecto consiste en el desarrollo de una **página web interactiva** que permita al **SEDES Cochabamba** gestionar de manera centralizada y eficiente los registros de hospitales, servicios médicos y especialidades dentro del departamento. La plataforma permitirá registrar, modificar, eliminar y consultar información de manera intuitiva, proporcionando un sistema de fácil acceso y navegación para los usuarios.

La página tendrá tres funcionalidades principales:

1. **Gestión de Hospitales**: Registro, modificación y eliminación de hospitales en el sistema, con la posibilidad de visualizar una lista completa de hospitales registrados.
2. **Gestión de Cartera de Servicios**: Registro, modificación y eliminación de servicios médicos ofrecidos por los hospitales, permitiendo la consulta de la lista de servicios.
3. **Gestión de Especialidades**: Registro, modificación y eliminación de especialidades médicas disponibles en los hospitales, con opción para consultar una lista actualizada de especialidades.

Además, se incluirá un **sistema de autenticación.**

1. **Link al Video demostrativo YouTube (5 minutos máximo)**

[**https://youtu.be/cRx85ogFTWo**](https://youtu.be/cRx85ogFTWo)

1. **Listado de los Requisitos Funcionales del Sistema**

**Gestión de Hospitales**:

* Registro de hospitales con información básica.
* Modificación y eliminación de registros hospitalarios.
* Visualización de la lista de hospitales registrados.

**Gestión de Cartera de Servicios**:

* Registro de Servicios ofrecidos.
* Modificación y eliminación de registros de la Cartera de Servicios.
* Visualización de la lista de los Servicios registrados.

**Gestión de Especialidades**:

* Registro de Especialidades.
* Modificación y eliminación de registros de Especialidades.
* Visualización de la lista de los Especialidades.

**Acceso de Usuarios**:

* Sistema de autenticación y roles.

1. **Arquitectura del software**

La arquitectura de nuestro proyecto está diseñada con una estructura clara y organizada, dividiéndose en dos partes principales: el **frontend** y el **backend**, cada uno con herramientas y tecnologías específicas que nos permiten mantener el orden, facilitar el desarrollo y asegurar un buen rendimiento.

* **Frontend**

Para el desarrollo del frontend, utilizamos **React** junto con **Vite** para optimizar el entorno de desarrollo y **TypeScript** para añadir tipado estático, lo que ayuda a detectar errores de manera temprana y mejora la seguridad en el desarrollo. La estilización de la aplicación se llevó a cabo con **Tailwind CSS**, una herramienta que nos permite aplicar estilos de manera rápida y eficiente mediante clases predefinidas, haciendo el diseño más consistente y adaptable.

El frontend se encarga de manejar la interfaz de usuario y las interacciones, como clics y formularios, comunicándose con el backend a través de solicitudes HTTP mediante una **API REST**. Esto asegura una comunicación fluida y ordenada entre las diferentes capas del sistema.

* **Backend**

El backend fue desarrollado usando **NestJS**, un framework de Node.js que nos permite mantener una estructura modular y organizada. Aquí se maneja toda la lógica del negocio, las reglas y la interacción con la base de datos. Para gestionar esta última, usamos **MySQL** como base de datos y un **ORM** (Object-Relational Mapper) para facilitar la interacción con la base de datos mediante objetos y modelos, evitando el uso excesivo de consultas SQL directas.

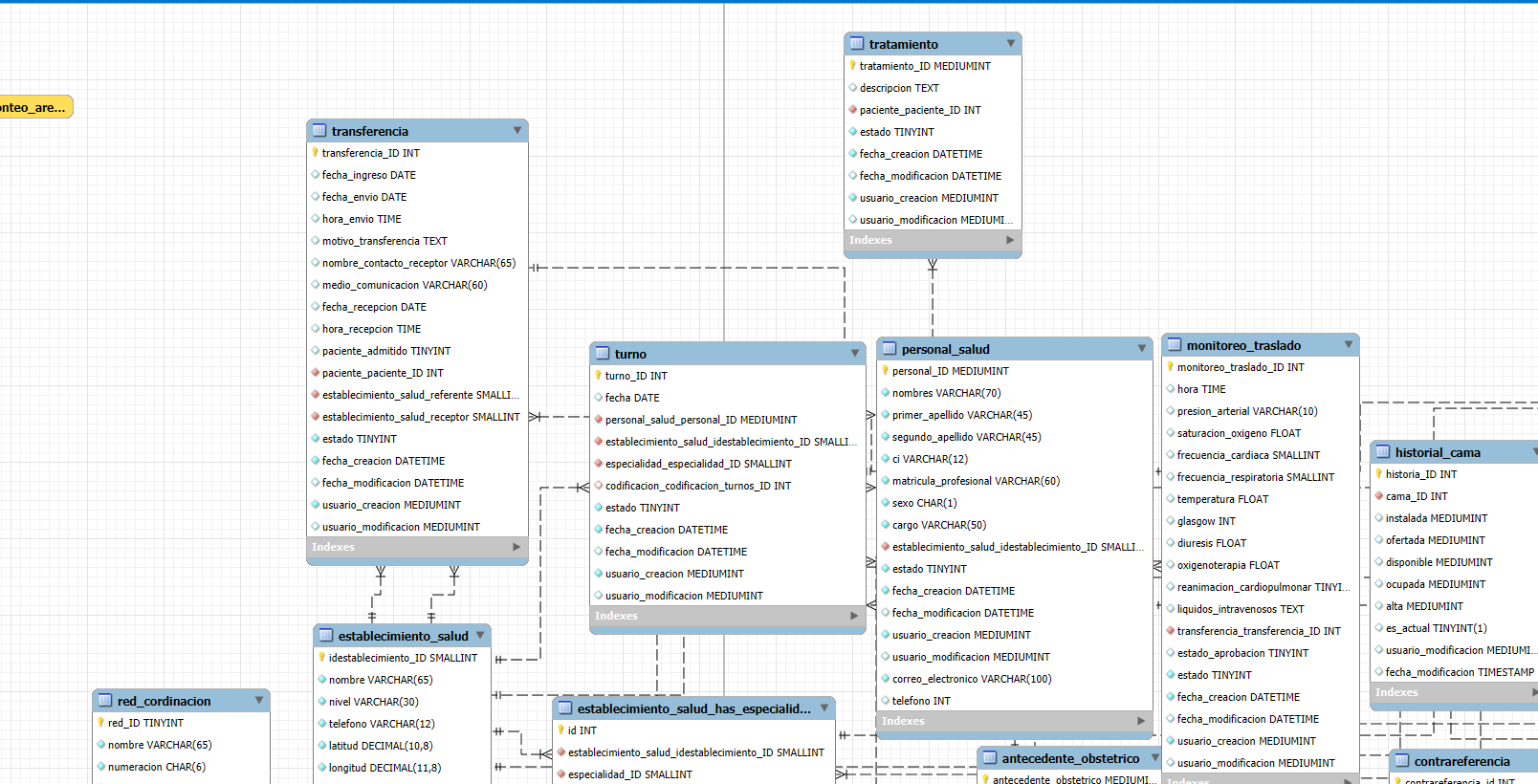
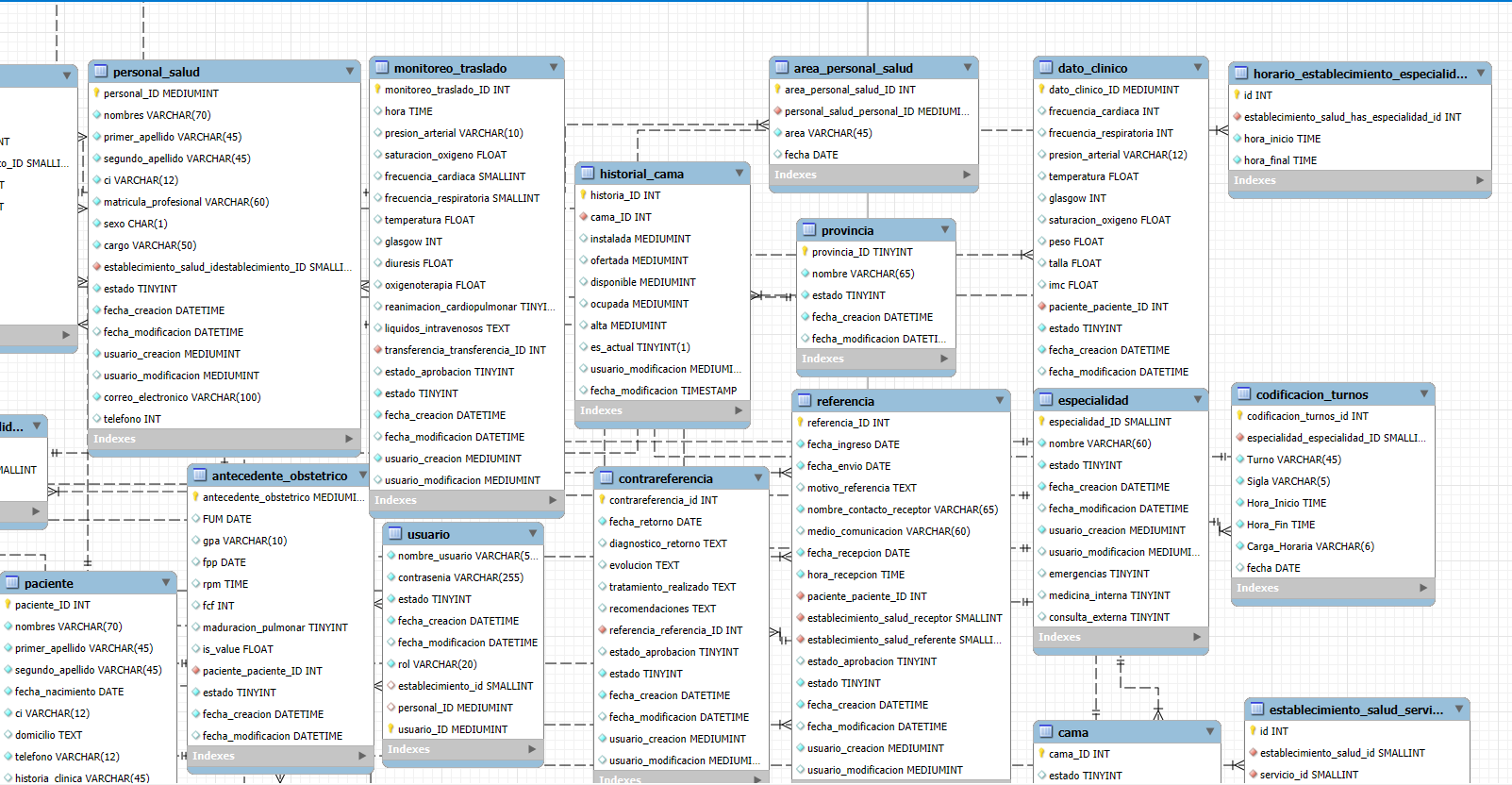
* **Interacción entre Frontend y Backend**

El frontend y el backend se comunican mediante **API REST**, donde el frontend envía solicitudes al backend para obtener o enviar datos. Por ejemplo, cuando el usuario llena un formulario, esa información es enviada al backend, que la procesa, accede a la base de datos si es necesario, y responde al frontend, mostrando un resultado o mensaje al usuario.

**Patrones de Diseño Utilizados**

* **Separación de Responsabilidades**: Mantenemos una separación clara entre el frontend, que gestiona la presentación e interacción, y el backend, que se encarga de la lógica y los datos.
* **Inyección de Dependencias**: En el backend, **NestJS** utiliza este patrón para hacer que los componentes sean más fáciles de gestionar y probar.
* **Componentes en React**: Nuestro frontend está diseñado con componentes reutilizables, facilitando la creación de una interfaz consistente y modular.

1. **Base de datos**
   1. **Diagrama completo y actual**



* 1. **En el GIT una carpeta con la base de datos con script de generación e inserción de datos de ejemplo utilizados**
  2. **Script simple (copiado y pegado en este documento)**

1. **Listado de Roles más sus credenciales de todos los Admin / Users del sistema**

* **Admin Sedes: A**cceso total del Sistema menos a la sección de "Mi hospital"
* **Admin Hospital:** Acceso a especialidades y servicios y establecer cuáles de estas tiene u ofrece su hospital.
* **Doctor: A**cceso a la vista (solo visualización) de especialidades y servicios de todos Los hospitales y también a los de El hospital al que pertenece.

1. **Requisitos del sistema:**

* Requerimientos de Hardware (mínimo): (cliente)

Este apartado se refiere a los equipos donde se utilizará la aplicación web (frontend):

* **Procesador**: Intel Core i3 (o equivalente).
* **Memoria RAM**: 4 GB.
* **Espacio en disco**: 200 MB libres para navegador y caché.
* **Resolución de pantalla**: 1366x768 píxeles o superior.
* Requerimientos de Software: (cliente)
  + **Navegador compatible**:
    - Google Chrome (versión 95 o superior).
    - Mozilla Firefox (versión 90 o superior).
    - Microsoft Edge (versión 95 o superior).
  + **Sistema operativo**:
    - Windows 10 o superior, MacOS 11.0 (Big Sur) o superior, o Linux con soporte de navegadores modernos.
* Requerimientos de Hardware (server/ hosting/BD)
* Requerimientos de Software (server/ hosting/BD)

1. **Instalación y configuración:** *Instrucciones detalladas sobre cómo instalar el software, configurar los componentes necesarios y establecer la conexión con otros sistemas o bases de datos:*

**Frontend**

1. Clonar el repositorio del proyecto:

* git clone <URL\_REPOSITORIO\_FRONTEND>
* cd <CARPETA\_FRONTEND>

2. Instalar dependencias:

* npm install

3. Configurar el archivo .env:

* - Crear un archivo .env en la raíz del proyecto con la variable VITE\_API\_BASE\_URL apuntando al backend:
* VITE\_API\_BASE\_URL=http://<IP\_SERVIDOR>:3000

4. Ejecutar en modo desarrollo:

* npm run dev

5. Compilar para producción:

* + - npm run build

**Backend**

1. Clonar el repositorio del proyecto:

* + - git clone <URL\_REPOSITORIO\_BACKEND>
    - cd <CARPETA\_BACKEND>

2. Instalar dependencias:

* + - npm install

3. Configurar el archivo .env:

* + - - Crear un archivo .env en la raíz del proyecto con las credenciales de la base de datos:
    - DB\_HOST=localhost  
       DB\_PORT=3306  
       DB\_USERNAME=root  
       DB\_PASSWORD=tu\_contraseña  
       DB\_DATABASE=nombre\_base\_datos

4. Ejecutar en modo desarrollo:

* + - npm run start:dev

5. Compilar para producción:

* + - npm run build

**Base de Datos**

1. Configurar el servidor MySQL:

* - Instalar MySQL en el servidor o entorno deseado.
* Copia la estructura proporcionada en los archivos .sql en la rama master del repositorio. Disponibles 1 con solo estructura y otro con estructura y datos

1. **PROCEDIMIENTO DE HOSTEADO / HOSTING (configuración)**

* **Sitio Web.**
* **B.D.**
* **API / servicios Web**
* **Otros** *(firebase, etc.)*

*Detalle DETALLADO paso a paso de la puesta en marcha en hosting, tanto para el sitio Web, API, B.D., etc.etc. (incluir scripts BD, Credenciales de acceso server, root BD, Admin, users clientes etc.)*

## **Sitio Web (Frontend)**

1. Subir el frontend a un servicio de hosting (por ejemplo, Vercel, Netlify o un servidor propio).  
 2. Configuración de URL Base:  
 - Asegúrate de configurar el archivo `.env.production` con la URL de tu backend en producción.

## **API / Servicios Web (Backend)**

1. Subir el backend a un servidor:  
 - Puedes usar servicios como AWS EC2, Heroku o servidores dedicados.  
 2. Configuración de Variables de Entorno:  
 - Asegúrate de que las variables `.env` en producción tengan las credenciales correctas.  
 3. Correr el backend en producción:  
 - Usa un manejador de procesos como PM2:  
 ```bash  
 pm2 start dist/main.js --name backend  
 ```

## **Base de Datos**

1. Asegúrate de que MySQL esté configurado correctamente en el servidor.  
 2. Configura los permisos de acceso para el backend:  
 - Crear un usuario específico con acceso limitado:  
 ```sql  
 CREATE USER 'api\_user'@'%' IDENTIFIED BY 'api\_password';  
 GRANT ALL PRIVILEGES ON nombre\_base\_datos.\* TO 'api\_user'@'%';  
 FLUSH PRIVILEGES;  
 ```

## **Detalle Paso a Paso**

1. Configuración de Frontend:  
 - Sube los archivos compilados (`dist/`) a un servicio como Vercel o a un servidor Nginx.  
 - Asegúrate de que la URL del backend esté configurada correctamente.  
   
2. Configuración del Backend:  
 - Instala Node.js y las dependencias necesarias en el servidor.  
 - Configura las variables de entorno con las credenciales de la base de datos.  
 - Inicia el backend usando PM2 o un servicio similar.  
   
3. Configuración de la Base de Datos:  
 - Instala MySQL en el servidor.  
 - Importa el script SQL con la estructura inicial de las tablas.  
 - Configura un usuario con permisos limitados para el backend.  
   
4. Pruebas Finales:  
 - Asegúrate de que la conexión entre frontend, backend y base de datos funciona correctamente.  
 - Prueba los endpoints del backend usando herramientas como Postman.  
   
5. Entrega de Credenciales:  
 - Documenta las credenciales de acceso al servidor, base de datos y API.  
 - Usa herramientas seguras (como un gestor de contraseñas) para entregar esta información.

1. **GIT :**

• Versión final entregada del proyecto.

Rama Integracion(esto se cambiara a master)

• Entrega compilados ejecutables

1. **Frontend:**

* Ejecutar el comando estando en terminal en el directorio raíz del Frontend(carpeta espeficica):

“npm run build”

* El resultado se encuentra en la carpeta dist y contiene los archivos estáticos listos para despliegue.

1. **Backend:**

* Ejecutar el comando estando en terminal en el directorio raíz del Backend(carpeta especifica):

“npm run build”

* Los archivos de salida se generan en la carpeta dist que poseen cada proyecto.

1. **Dockerizado** *Del Sitio WEB, de la Base de Datos*
   1. **Proceso de dokerizado, Configuración**
   2. **Como hacer Correr, Acceso credenciales:** 
      1. **base datos**
      2. **Roles Admin, User, etc**
      3. **Base de datos con datos válidos y legibles.**
2. **Personalización y configuración:**

Una vez descargado el código fuente de Github. Abrir terminal para cada proyecto y ubicarse en la raíz de ambos. Estando en la raíz en ambos proyectos ejecutar en ambos “npm install” para instalar todas las dependencias de ambso proyectos

**Frontend:**

1. Configuración de API Base:

* Cada página o componente que consume datos del backend define la constante API\_BASE\_URL con el valor <http://localhost:3000>.
* Para entornos de producción o cambio a otro entorno diferente al localhost, el desarrollador debe actualizar esta constante manualmente en el archivo “.env” ubicado en la raíz del proyecto frontend:

mientras sea localhost

VITE\_API\_BASE\_URL=http://localhost:3000

usar esto en las pagina en vez de poner directa,ente la URL del servidor en otro entorno de produccion

VITE\_API\_BASE\_URL=https://api.miapp.com

1. Estilos:

* Configuración de Tailwind CSS en el archivo tailwind.config.js.
* Los estilos globales se encuentran en src/index.css.

**Backend:**

1. Conexión a la Base de Datos:

* Las credenciales de la base de datos y otras configuraciones críticas están definidas en el archivo .env (crearlo en la raíz del proyecto backend):

DB\_HOST=localhost

DB\_PORT=3306

DB\_USERNAME=root

DB\_PASSWORD=tu\_contraseña

DB\_DATABASE=nombre\_base\_datos

* Estas variables son consumidas en el módulo TypeOrmModule en AppModule.

1. Configuraciones Clave:

* El backend está configurado para correr en el puerto 3000. Esto se define en el archivo .env . Puede modificarlo según la ruta del servidor

1. Parámetros Adicionales:

* La sincronización de esquemas con la base de datos (synchronize) está deshabilitada en producción para evitar cambios accidentales en las tablas.

1. **Seguridad**:

**Frontend:**

1. Políticas CORS:

* En el backend, CORS está habilitado para aceptar solicitudes desde cualquier origen durante el desarrollo. En producción, esto debe restringirse a dominios confiables.

1. Variables Sensibles:

* Evitar incluir datos sensibles o rutas críticas directamente en el código fuente del frontend. En su lugar, usa variables de entorno y configuración centralizada.Esto se configura desde el archivo .env para evitar exponer la dirección del servidor

**Backend:**

1. Protección de Base de Datos:

* Las credenciales de conexión se manejan a través del archivo .env y no están expuestas en el código fuente.

1. Restricciones de CORS:

* En producción, actualiza la configuración de app.enableCors en main.ts para permitir únicamente solicitudes de dominios específicos.

1. **Depuración y solución de problemas:**

**Problemas Comunes:**

1. Conexión con la Base de Datos:

* Error ECONNREFUSED: Asegúrate de que el servidor MySQL esté corriendo y que las credenciales en .env sean correctas.

1. Errores de CORS:

* Si el frontend no puede acceder al backend, revisa la configuración de CORS en el archivo main.ts.

1. Errores en la URL Base del Backend:

* Si http://localhost:3000 no es accesible, confirma que el backend esté corriendo correctamente.

**Herramientas de Depuración:**

1. Frontend:

* Usa las herramientas de desarrollo del navegador para inspeccionar errores en la consola o problemas de red.

1. Backend:

* Habilita logging más detallado para rastrear errores internos.

1. Postman:

* Verifica manualmente los endpoints del backend para asegurar que devuelvan los datos esperados.

**Soluciones Comunes:**

1. Problemas con Dependencias:

Elimina la carpeta node\_modules y ejecuta en terminal estando en la raíz del proyecto que causa problemas: “npm install” asi se volverán a instalar todas las dependencias.

Debe haber 0 vulnerabilidades; en caso de que tenga vulnerabilidades ejecutar “npm audit” o “npm audit fix” y ver que se solucionen esas viulnerabilidades

1. **Glosario de términos:**

* **API REST (Application Programming Interface - Representational State Transfer):** Estilo de arquitectura que permite que las aplicaciones se comuniquen entre sí a través de solicitudes HTTP, como GET, POST, PUT y DELETE.
* **Frontend:** Parte de una aplicación o sitio web que interactúa directamente con el usuario y muestra la interfaz.
* **Backend:** Parte de una aplicación que maneja la lógica del negocio, la base de datos y las solicitudes del usuario desde el frontend.
* **NestJS:** Framework para aplicaciones Node.js que ofrece una estructura modular para el desarrollo de aplicaciones escalables del lado del servidor.
* **React:** Biblioteca de JavaScript utilizada para construir interfaces de usuario interactivas.
* **Vite:** Herramienta rápida de desarrollo para proyectos web modernos que optimiza la creación de aplicaciones React.
* **Tailwind CSS:** Framework de CSS basado en utilidades que permite diseñar interfaces de manera rápida mediante clases predefinidas.
* **ORM (Object-Relational Mapper):** Herramienta que facilita la interacción con bases de datos mediante objetos y modelos, evitando consultas SQL directas.
* **CORS (Cross-Origin Resource Sharing):** Política de seguridad que permite o restringe las solicitudes entre diferentes dominios web.
* **Dockerización:** Proceso de empaquetar una aplicación y sus dependencias en un contenedor para asegurar su ejecución en cualquier entorno.
* **TypeScript:** Lenguaje de programación que extiende JavaScript con tipos estáticos, mejorando la detección de errores en tiempo de desarrollo.

1. **Referencias y recursos adicionales:**

* Documentación de React: <https://reactjs.org/docs/getting-started.html>
* Documentación de NestJS: <https://docs.nestjs.com>
* Guía de Tailwind CSS: <https://tailwindcss.com/docs>
* Tutorial de Vite para React: <https://vitejs.dev/guide>
* Documentación de MySQL: <https://dev.mysql.com/doc>
* Introducción a la API REST: [https://restfulapi.net](https://restfulapi.net/)
* Recursos de soporte para Node.js: <https://nodejs.org/en/community>
* Postman (para probar APIs): [https://www.postman.com](https://www.postman.com/)

1. **Herramientas de Implementación:**

**Lenguajes de programación:**

JavaScript

TypeScript

Frameworks:

Frontend: React (con Vite)

Backend: NestJS (basado en Node.js)

**APIs de terceros:**

Librerías y paquetes npm como Axios para solicitudes HTTP.

**Bases de Datos:**

MySQL (gestor de base de datos relacional)

1. **Bibliografía**

* "Node.js Design Patterns" de Mario Casciaro y Luciano Mammino.
* Documentación oficial de React: [https://reactjs.org](https://reactjs.org/)
* Documentación oficial de NestJS: [https://nestjs.com](https://nestjs.com/)
* "MySQL Reference Manual" de Oracle Corporation.
* "Building RESTful Web APIs with Node.js" por Pradeep Kumar Singh.