Proyecto Final Programado

Nombre del Proyecto: CV-MAQ

Nombre del Estudiante/Desarrollador: Cristian Valerio Pereira

Profesor/Curso: Estefania Boza Villalobos / (ISB-32) Programación Avanzada

Fecha: 22/11/2025

Tabla de Contenido

Índice de Tablas

Índice de Figuras

Parte I: Información General del Proyecto

En la actualidad, las empresas que gestionan maquinaria pesada, tractores, camiones y diversos activos, requieren llevar un control eficiente del estado y mantenimiento de sus equipos. Sin embargo, muchas veces este proceso se realiza de manera manual o dispersa, dificultando la trazabilidad y el acceso rápido a la información. Este proyecto propone el desarrollo de un sitio web que facilite la gestión diaria del mantenimiento, permitiendo el registro, consulta y seguimiento de inspecciones preventivas, optimizando así la toma de decisiones y la vida útil de los equipos

Objetivo General

Desarrollar un sistema de información web que permita registrar, consultar y dar seguimiento al mantenimiento preventivo de maquinaria y activos, facilitando la trazabilidad y el control por parte de operadores y gerentes de la empresa.

Objetivos Específicos

1. Diseñar la arquitectura del sistema para garantizar una interacción eficiente entre usuarios, base de datos y servicios internos.

2. Implementar una base de datos relacional normalizada que asegure integridad y seguridad de la información de los mantenimientos realizados.

3. Desarrollar el backend y las funcionalidades esenciales del sistema (MVP): registro de usuarios, inicio de sesión, checklist diario y consulta de registros históricos.

4. Crear una interfaz gráfica intuitiva y responsiva que mejore la experiencia de usuario para operadores y gerentes.

Justificación del Proyecto

Este proyecto surge como respuesta a la necesidad de las empresas de contar con una herramienta tecnológica que les permita sistematizar el control y seguimiento de los mantenimientos preventivos de su maquinaria. La digitalización de estos procesos mejora la organización, reduce errores, asegura la trazabilidad histórica y agiliza la toma de decisiones oportunas, minimizando así el riesgo de fallos y extendiendo la vida útil de los activos.

Alcances Esperados

- El sistema permitirá a los operadores registrar el estado de sus equipos diariamente mediante un checklist personalizado según el tipo de unidad.
- Los gerentes o supervisores podrán consultar y dar seguimiento a los registros históricos de cada activo en tiempo real.
- Se habilitará la gestión básica de usuarios y roles (operador, gerente).
- El sistema dejará la base para futuras mejoras, como la exportación de reportes en PDF/Excel y la integración de dashboards visuales.

Requerimientos del Proyecto

- Registro y autenticación de usuarios.
- Asignación de roles (operador, gerente).
- Registro de checklist diario para cada unidad/máquina.
- Consulta y trazabilidad de registros históricos por fecha, usuario y activo.
- Gestión básica de usuarios y activos.

Requerimientos no funcionales:

- Seguridad en la autenticación y manejo de datos.
- Interfaz responsiva y fácil de usar.
- Escalabilidad para futuras funcionalidades.

Infraestructura mínima:

- Servidor web
- Base de datos relacional (MySQL).
- Navegadores web modernos para acceso de usuarios.

Parte II: Diseño del Sistema (Pre-diseño visual)

El presente documento corresponde al prediseño visual del sistema web CV-Maq (Control y Valoración de Maquinarias). Su objetivo es mostrar, de manera preliminar, la estructura, apariencia y navegación general de la aplicación antes de su desarrollo final.

Elementos de Diseño

Paleta de Colores y Tipografía Colores principales: Turquesa (#009688): transmite modernidad y enfoque tecnológico. Gris claro (#f7f7f7): aporta simplicidad y limpieza visual. Blanco para fondos y áreas destacadas. Rojo para botones de acciones críticas como “Salir” o “Eliminar”.

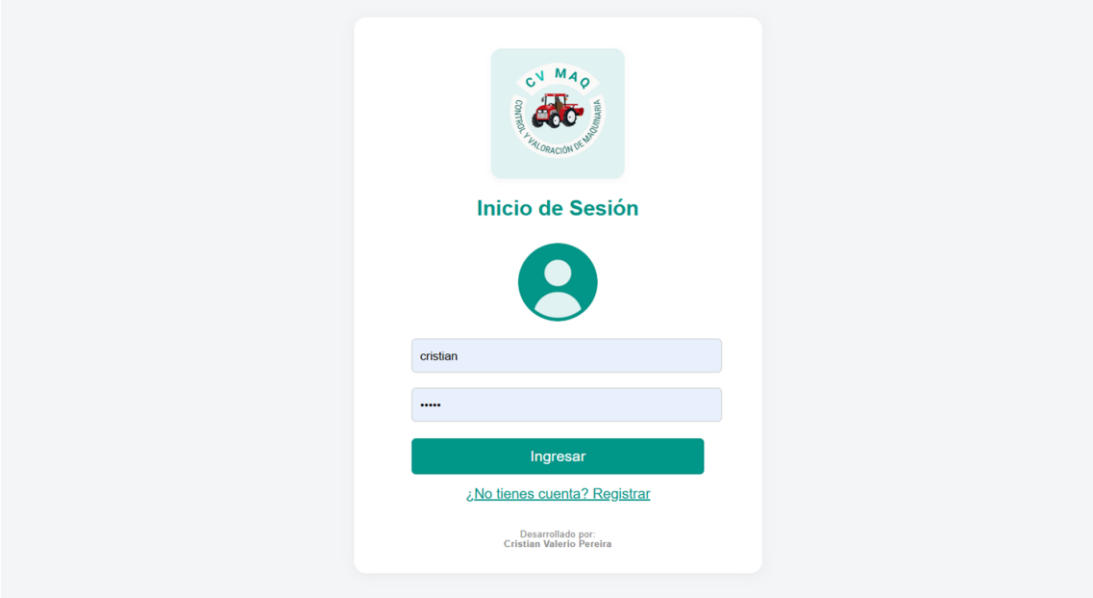
Tipografía: Se utiliza Arial (Roboto y Segoe UI), por su modernidad y excelente legibilidad en interfaces digitales.

Estructura de Pantallas:

1-Login

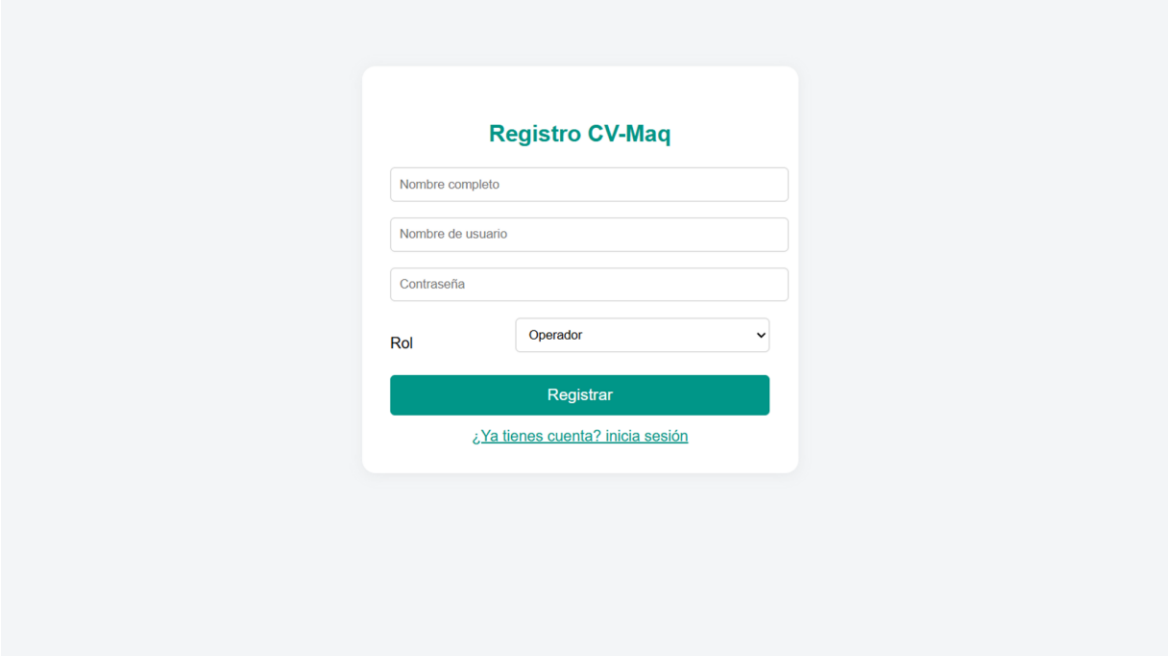
Pantalla de acceso seguro al sistema, con el logo del programa y un icono de usuario destacado. Incluye opción para registro de nuevos usuarios y un pie de página con el nombre del desarrollador.

Figura no: 1 Pantalla de Login



2-Registro de Usuario: Formulario sencillo para el registro de nuevos usuarios. Permite elegir el rol (por defecto Operador) si se desea elegir el Rol administrador, deberá ingresar una contraseña maestra)

Figura no: 2 Pantalla de Registro de Usuario

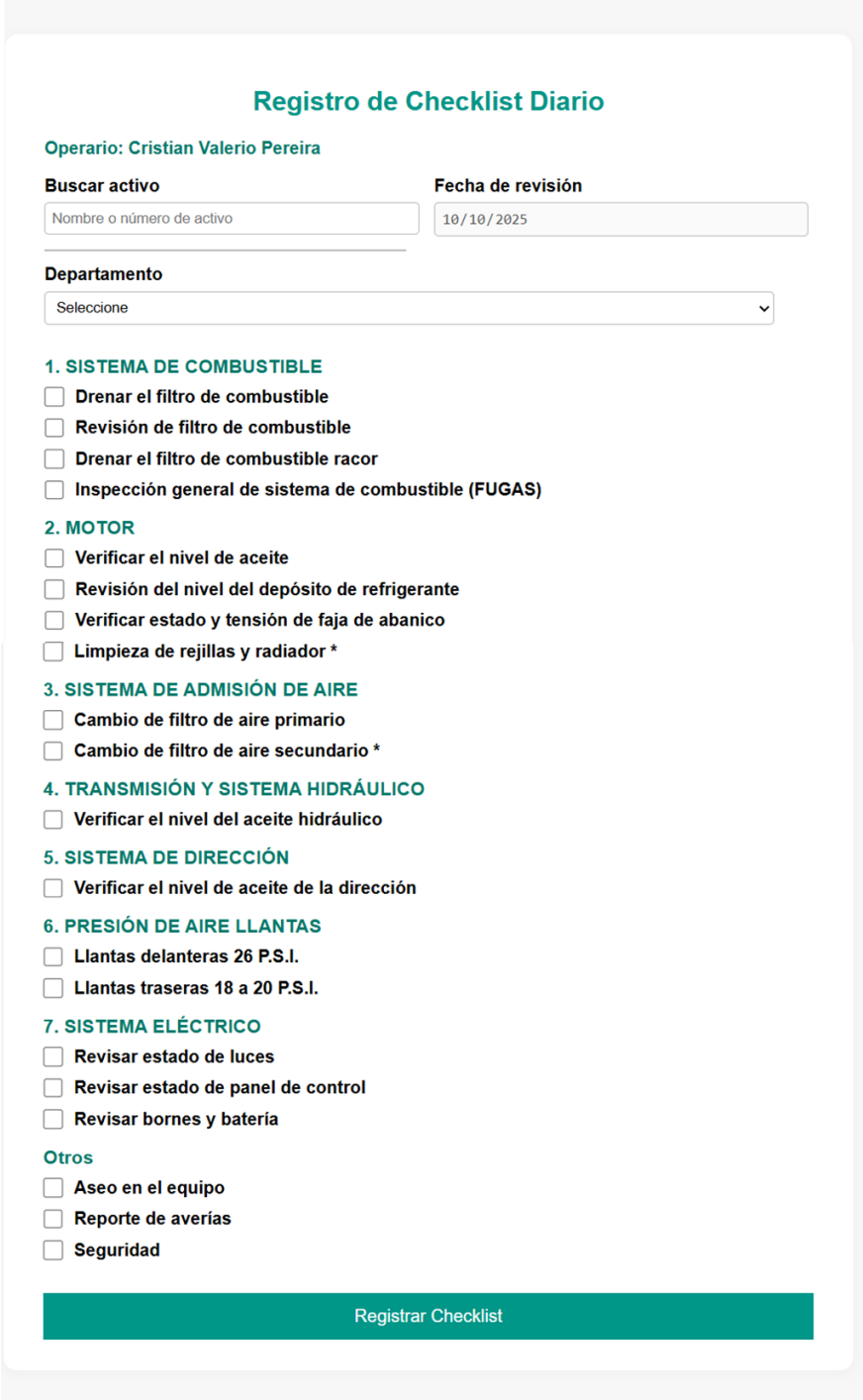


3-Dashboard o Panel Principal: Menú principal de navegación con acceso rápido a los módulos: registrar check, ver activos y gestión de usuarios (solo para administradores). El diseño es profesional, con iconos visuales y una bienvenida personalizada.

Figura no: 3 Pantalla de Panel Principal

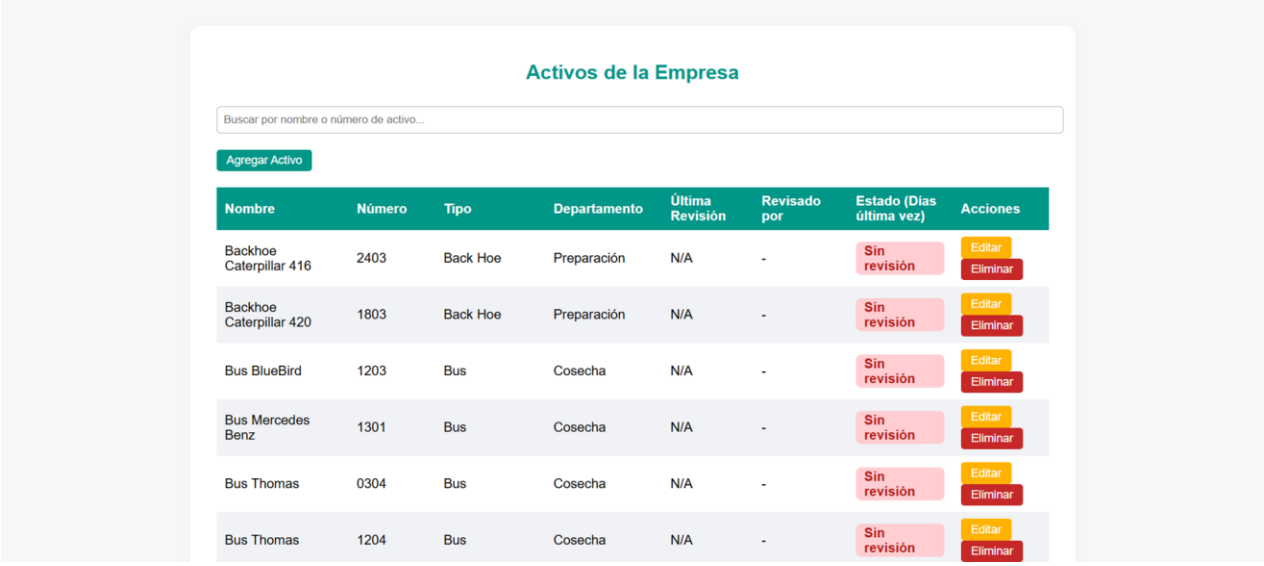
4-Registro de Checklist Diario: Formulario para el registro de mantenimientos preventivos de cada activo. Incluye buscador inteligente, autocompletado y checkboxes alineados con los textos para facilidad de uso.

Figura no: 4 Pantalla de CheckList



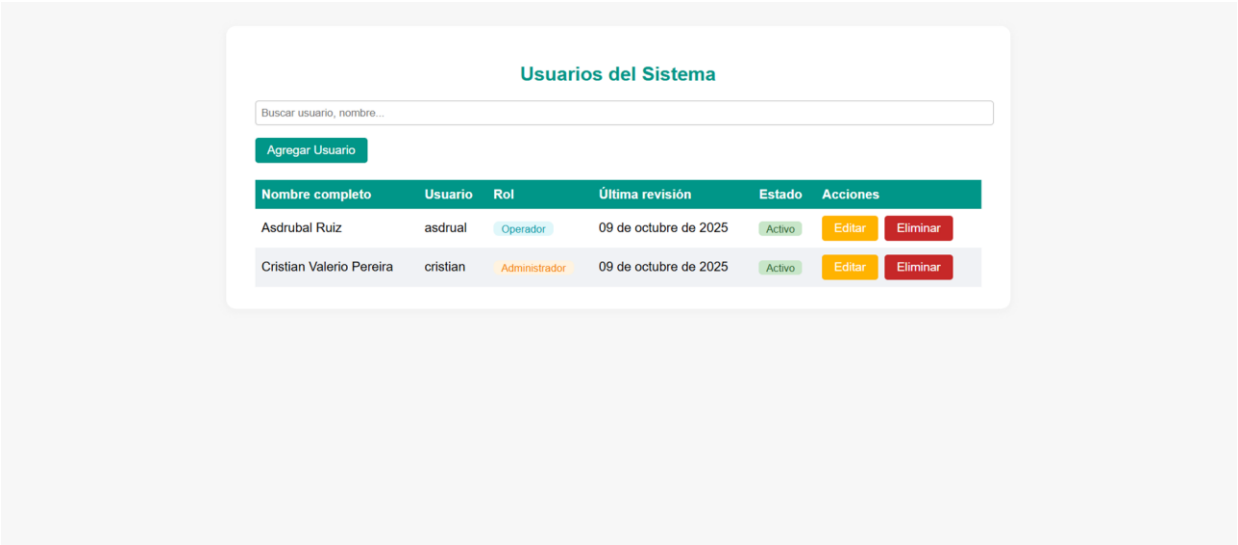
5-Ver Activos: Lista de activos de la empresa, con buscador, filtros y visualización clara del estado de revisión de cada equipo. Los botones de editar/eliminar solo están habilitados para administradores.

Figura no: 5 Pantalla de activos



6-Gestión de Usuarios: Pantalla exclusiva para administradores, donde pueden visualizar, agregar, editar o eliminar usuarios, además de ver la última actividad de cada uno.

Figura no: 6 Pantalla Gestión de usuarios



Iconografía e Imágenes:

Uso de iconos personalizados: Los botones principales y el menú incluyen íconos propios en blanco sobre fondo turquesa, manteniendo coherencia visual y facilitando la identificación rápida de cada módulo.

Figura no: 7 Logo

El Sistema presenta su propio logo, diseñado para reflejar la identidad y propósito de la aplicación. Imágenes ilustrativas: Algunas pantallas incluyen fotografías reales de maquinaria agrícola, reforzando la temática del sistema

Explicación

Este pre-diseño visual funciona como guía inicial para el desarrollo del sistema. Permite identificar y validar la estructura, navegación y estética. Además, facilita la retroalimentación temprana y asegura que el producto final cumpla con las expectativas del usuario y las necesidades funcionales del negocio.

Parte III: Base de Datos

Definición

Tipo de base de datos:
Relacional (MySQL)

Objetivo:
La base de datos del sistema CV-MAQ tiene como propósito centralizar, almacenar y gestionar de forma eficiente toda la información relacionada con los activos de la empresa, los usuarios, los registros diarios de mantenimiento (checklists) y las relaciones entre estos. Se garantiza la integridad de los datos y el correcto funcionamiento de las funcionalidades del sistema web.

Motor elegido:
MySQL Community Server 8.x (este lo elegí por ser gratuito, ampliamente soportado y compatible con Node.js/Express).

Elementos

1.Entidades y atributos:

Tabla 1 Entidades y atributos

Entidad	Campo	Tipo de dato	Justificación
usuarios	id	INT, PK, AI	Identificador único, este es autoincremental
usuarios	nombre_completo	VARCHAR(100)	almacena nombres completos
usuarios	nombre_usuario	VARCHAR(30)	Identificador de login, es único
usuarios	contrasena	VARCHAR(100)	Encriptable, texto protegido
usuarios	rol_id	INT, FK	Relaciona con la tabla roles
roles	id	INT, PK, AI	Identificador único, es autoincremental
roles	nombre	VARCHAR(30)	Nombre del rol (Operador, Administrador)
activos	id	INT, PK, AI	Identificador único, es autoincremental
activos	nombre	VARCHAR(100)	Nombre descriptivo del activo
activos	tipo	VARCHAR(30)	Tipo de activo (Ejemplo un Tractor, Bus, etc.)
activos	numero_activo	VARCHAR(20)	Número de placa o código interno
activos	anio	VARCHAR(10)	Año del activo
activos	placa	VARCHAR(20)	Placa física

activos	departamento	VARCHAR(50)	Departamento responsable
checklist	id	INT, PK, AI	Identificador único, es autoincremental
checklist	activo_id	INT, FK	Relaciona con activos
checklist	usuario_id	INT, FK	Relaciona con usuarios
checklist	fecha	DATE	Fecha del registro
checklist	departamento	VARCHAR(50)	Departamento seleccionado
checklist	combustible1	TINYINT(1)	Estado (checkbox) del punto 1 de combustible
checklist	combustible2	TINYINT(1)	Estado (checkbox) del punto 2 de combustible
checklist	combustible3	TINYINT(1)	Estado (checkbox) del punto 3 de combustible
checklist	combustible4	TINYINT(1)	Estado (checkbox) del punto 4 de combustible
checklist	motor1	TINYINT(1)	Estado del punto 1 de motor
checklist	motor2	TINYINT(1)	Estado del punto 2 de motor
checklist	motor3	TINYINT(1)	Estado del punto 3 de motor
checklist	motor4	TINYINT(1)	Estado del punto 4 de motor
checklist	aire1	TINYINT(1)	Estado del punto 1 de admisión de aire
checklist	aire2	TINYINT(1)	Estado del punto 2 de admisión de aire
checklist	transmision1	TINYINT(1)	Estado del sistema de transmisión/hidráulico
checklist	direccion1	TINYINT(1)	Estado del sistema de dirección
checklist	llantas1	TINYINT(1)	Estado de llantas delanteras
checklist	llantas2	TINYINT(1)	Estado de llantas traseras
checklist	electrico1	TINYINT(1)	Estado de luces
checklist	electrico2	TINYINT(1)	Estado de panel de control
checklist	electrico3	TINYINT(1)	Estado de batería y bornes
checklist	aseo	TINYINT(1)	Aseo del equipo
checklist	reporte	TINYINT(1)	Reporte de averías
checklist	seguridad	TINYINT(1)	Seguridad

Justificación de los tipos de datos utilizados

1. **INT, PK, AI (Clave primaria, autoincremental):**
   Se utiliza para los campos identificadores (id) de cada tabla. Esto garantiza que cada registro tenga un identificador único, esto nos facilita las búsquedas y relaciones entre tablas, y el autoincremento asegura que el valor siempre será diferente para cada nuevo registro.

2. **VARCHAR(N):**
   Se utiliza para campos de texto de longitud variable, como nombre_completo, nombre_usuario, nombre, tipo, numero_activo, placa y departamento. Esto permite almacenar cadenas de diferentes tamaños optimizando el espacio utilizado, y se puede ajustar el tamaño según las necesidades específicas del campo.
3. **TINYINT(1):**
   Se utiliza para los campos que representan estados de checklist (checkbox), ya que solo necesitan almacenar valores binarios (0 o 1, es decir, desmarcado o marcado). Esto optimiza el almacenamiento y es lo mejor para respuestas tipo Sí/No.
4. **DATE:**
   Se emplea en el campo fecha para almacenar fechas exactas de los registros, permitiendo fácilmente búsquedas, filtros y operaciones con fechas.
5. **Foreign Keys (FK):**
   Los campos que hacen referencia a otras tablas, como rol_id, activo_id, usuario_id, se definen como INT y **Foreign Key** para asegurar la integridad referencial. Así, solo se podrán ingresar valores que existan en las tablas relacionadas.

2.Llaves primarias y foráneas

Todas las tablas poseen una llave primaria (id).

Tabla 2 Llaves Primarias y foráneas

Tabla	Llave primaria	Llaves foráneas
roles	id	Ninguna
usuarios	id	rol_id → roles(id)
activos	id	Ninguna
checklist	id	activo_id → activos(id), usuario_id → usuarios(id)

Relaciones:

- checklist.activo_id → activos.id
- checklist.usuario_id → usuarios.id
- usuarios.rol_id → roles.id

Ejemplos de normalización en mi proyecto:

Primera Forma Normal (1FN):

- Cada campo almacena un solo dato.
- No hay listas ni datos repetidos en una sola columna.

Segunda Forma Normal (2FN):

- Todas las columnas dependen de la clave primaria.
- Se han separado los roles en su propia tabla y los registros de checklists hacen referencia a activos y usuarios.

Tercera Forma Normal (3FN):

- No hay dependencias transitivas.
- Cada columna almacena información directamente dependiente de la clave primaria.

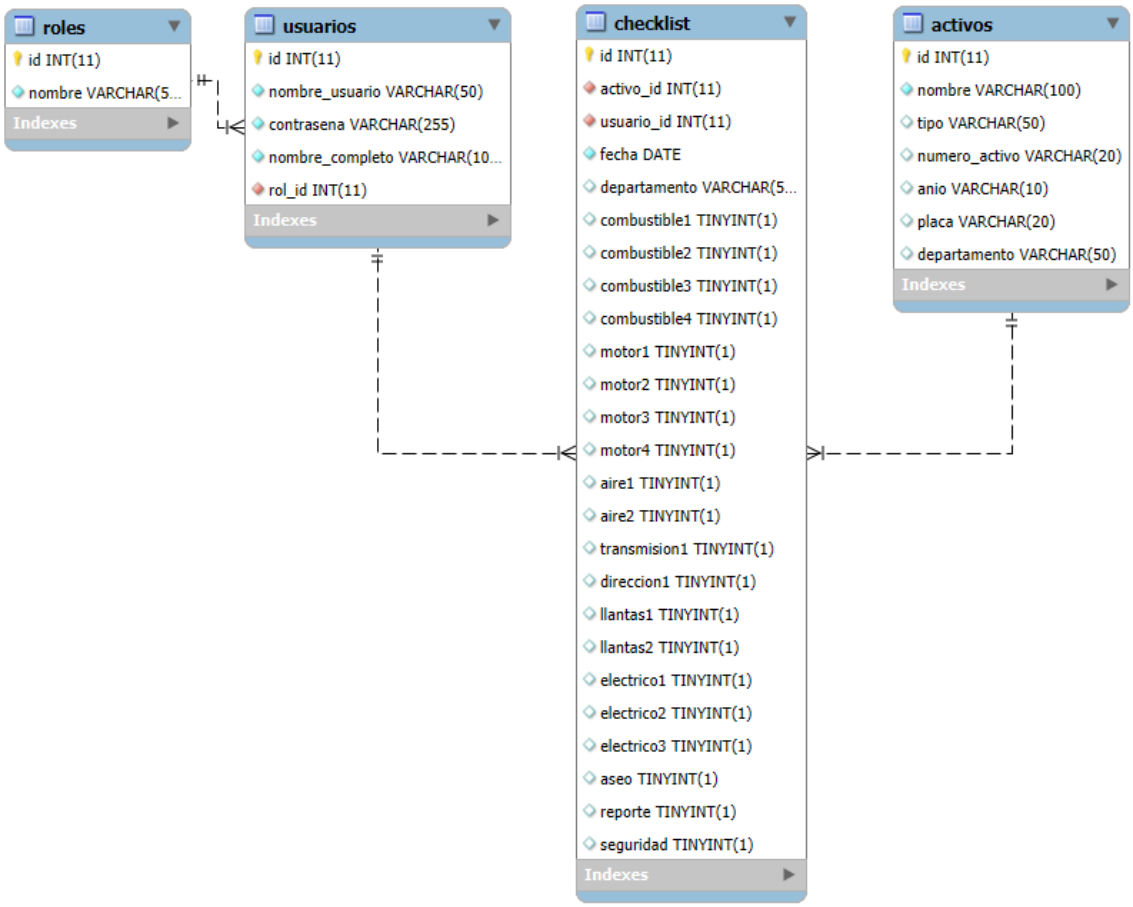
Resultado:
La base de datos está normalizada hasta 3FN, eliminando redundancias y facilitando mantenibilidad.

Funciones, vistas o triggers

- Actualmente no he implementado triggers personalizados, pero la integridad se asegura mediante claves foráneas y validaciones desde la aplicación.

3.Esquema y Diagrama Entidad-Relación

Figura no: 8 Esquema y Diagrama Entidad-Relación



Descripción del Modelo Entidad-Relación del Sistema CV-MAQ

El modelo entidad-relación del sistema CV-MAQ tiene 4 tablas principales:

- **roles:** Permite definir los distintos tipos de usuario en el sistema (por ejemplo, Administrador y Operador), utilizando una clave primaria id y el campo nombre para distinguir cada rol.

- **usuarios:** Almacena los datos de los usuarios, como nombre de usuario, contraseña, nombre completo y referencia al rol que desempeñan en el sistema (rol_id). Cada usuario está asociado a un único rol, lo que se modela mediante una clave foránea hacia la tabla roles.

- **activos:** Registra los activos sujetos a inspección, almacenando información relevante como el nombre del activo, tipo, número de identificación, año, placa y departamento al que pertenece.

- **checklist:** Registra cada revisión diaria realizada a los activos. Cada registro de checklist está vinculado tanto a un usuario (quien fue que realizó la revisión)

como a un activo (el equipo revisado), por medio de claves foráneas (usuario_id y activo_id). Además, almacena la fecha y el resultado de cada uno de los puntos de verificación definidos en el proceso de mantenimiento preventivo.

Las relaciones entre tablas están definidas de la siguiente forma:

- Un rol puede estar asociado a varios usuarios.
- Un usuario puede registrar múltiples checklists.

4.Conexión con la Aplicación

Tecnología empleada:

- Node.js con Express y el paquete mysql2 (para conexión a MySQL).
- Backend configurado con variables de entorno para seguridad.

Figura no: 9 Fragmento de código de conexión con Base de datos:

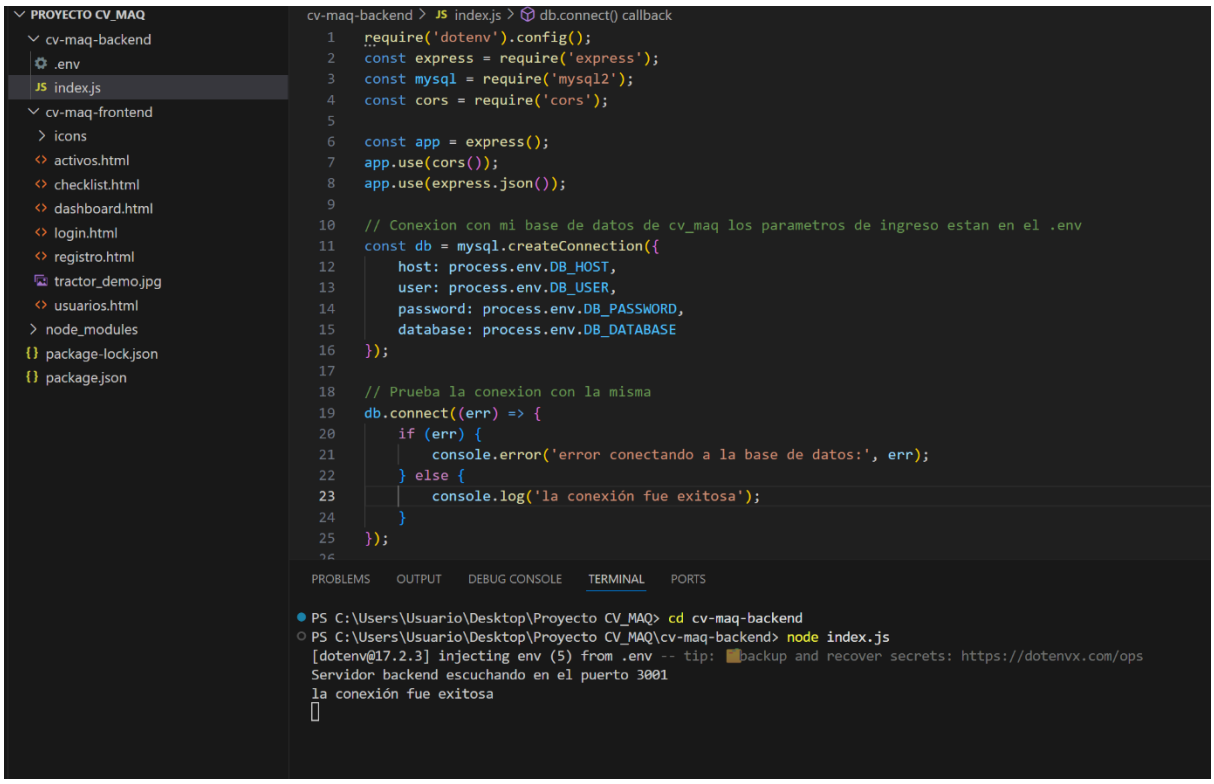
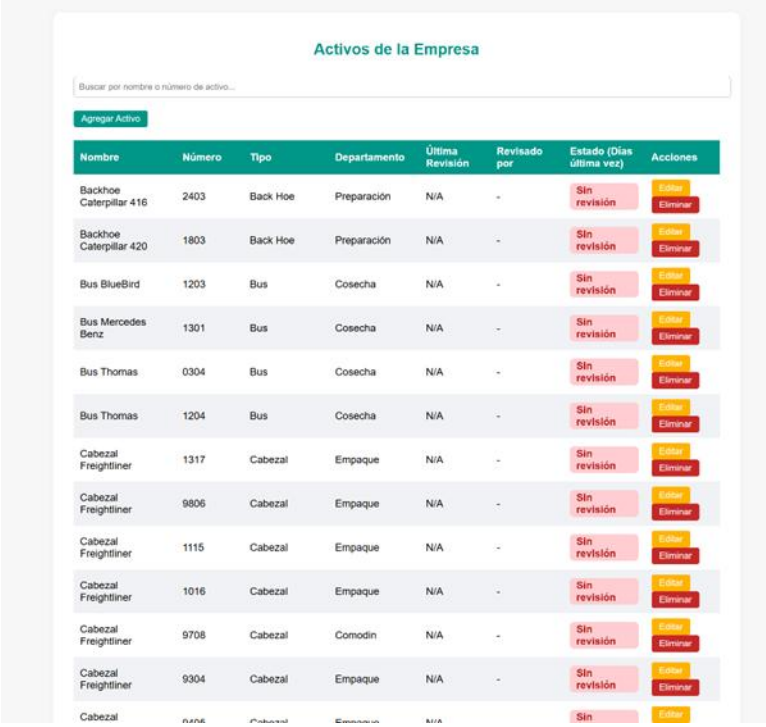


Figura no: 10 Tabla de activos cargada desde la base de datos



Parte IV: Desarrollo del Backend

1. Arquitectura del Backend

Para el backend de CV-Maq se utilizó una **arquitectura monolítica con organización por capas lógicas**:

- **Monolítica** porque toda la aplicación del lado del servidor se ejecuta en un único proyecto Node.js (un solo proceso de Express). Esta decisión es adecuada para:
    - El tamaño actual del sistema.
    - La cantidad de usuarios concurrentes esperada.
    - La facilidad de despliegue en un entorno académico y/o en un servidor interno de la empresa.

- **Capas lógicas dentro del monolito**:
    - **Capa de Presentación (Frontend):** páginas HTML/CSS/JS estáticas (login, dashboard, checklist, activos, usuarios). No se renderizan en el servidor, solo consumen la API.
    - **Capa de API / Controladores (Negocio):** implementada con **Express**. Define los endpoints REST (/api/login, /api/checklist, /api/activos, /api/usuarios, etc.), valida datos y aplica reglas de negocio básicas (por ejemplo: no duplicar números de activo, validar que exista usuario/activo al registrar un checklist, etc.).
    - **Capa de Datos:** conexión a **MySQL** usando la librería mysql2. Todas las operaciones de lectura y escritura se realizan mediante sentencias SQL parametrizadas.

Esta arquitectura permite que, en el futuro, el monolito se pueda ir modularizando (por ejemplo, moviendo la lógica de activos o usuarios a módulos independientes e incluso a microservicios) sin tener que reescribir todo el sistema.

2. Componentes y estructura del código

El backend está implementado principalmente en el archivo index.js (o server.js) del proyecto Node.js. A nivel lógico, se distinguen los siguientes componentes:

1. **Servidor Express y configuración inicial**
    - Carga variables de entorno con dotenv.
    - Configura CORS y el middleware express.json() para recibir cuerpos JSON.
    - Inicializa la conexión a MySQL.
2. **Módulo de autenticación (login)**
    - Endpoint POST /api/login.
    - Recibe nombre_usuario y contrasena, los valida contra la tabla usuarios y devuelve un objeto con:

        - id, nombre_usuario, nombre_completo, rol_id.
    - Esta información se guarda en localStorage en el frontend para controlar acceso y rol.

3. **Módulo de checklist**

    - Endpoint POST /api/checklist.
    - Registra una revisión para un activo y un usuario en la tabla checklist, incluyendo:
        - Fecha, departamento, y todos los campos de revisión (combustible, motor, aire, transmisión, llantas, eléctrico, aseo, reporte y seguridad).
    - Antes de insertar, valida que exista activo_id y usuario_id.

4. **Módulo de usuarios**

    Endpoint GET /api/usuarios
    Devuelve id, nombre_usuario y nombre_completo (para mostrar en combos).

    Endpoint GET /api/usuarios-completo
    Devuelve información con el último checklist registrado por cada usuario (ultima_revision), utilizado en el módulo de administración de usuarios.

    Endpoint POST /api/registrar-usuario
    Registra nuevos usuarios con:

    - nombre_completo, nombre_usuario, contrasena, rol_id.
    - Valida que no exista otro usuario con el mismo nombre_usuario.

5. **Módulo de activos**
    - Endpoint GET /api/activos
      Devuelve todos los activos sin información de checklist.
    - Endpoint GET /api/activos-completo
      Devuelve los activos junto con:
        - Fecha de última revisión.
        - Nombre del usuario que hizo esa revisión.

- Días desde la última revisión (dias_desde_ultima), calculados en SQL.

- Endpoint GET /api/activos/:id/ultima-revision
  Trae el detalle completo del último checklist de un activo (para el modal "Ver detalle").
- Endpoint POST /api/activos
  Crea un activo nuevo, validando:
  - Campos obligatorios (nombre, tipo, numero_activo, departamento).
  - Que el numero_activo no esté duplicado.
- Endpoint PUT /api/activos/:id
  Actualiza un activo existente, asegurando que no se repita el numero_activo en otro registro.
- Endpoint DELETE /api/activos/:id
  Elimina un activo, controlando el error de llaves foráneas (si tiene checklists asociados).

6. **Middleware de manejo de errores**
   - Al final del archivo se define un middleware Express:

Figura no: 11 Middleware de manejo de errores

```
});


//  MIDDLEWARE GLOBAL DE MANEJO DE ERRORES

app.use((err, req, res, next) => {
  console.error('Unhandled error:', err);
  if (res.headersSent) return next(err);
  res.status(500).json({ error: err.message || 'Error interno del servidor' });
});
```

- Centraliza el manejo de errores no controlados y devuelve un mensaje JSON estándar.

Diagrama de interacción (descripción textual)

- El **frontend** (HTML+JS) realiza llamadas fetch a la API REST:
  - login.html → POST /api/login
  - checklist.html → GET /api/usuarios, GET /api/activos, POST /api/checklist
  - activos.html → GET /api/activos-completo, POST/PUT/DELETE /api/activos, GET /api/activos/:id/ultima-revision
  - usuarios.html → GET /api/usuarios-completo
- El **servidor Express** recibe las peticiones, aplica la lógica de negocio y ejecuta las consultas MySQL.
- **MySQL** almacena toda la información persistente del sistema.

3. Elementos Clave del Backend

3.1 Seguridad

En esta etapa del proyecto se implementaron los siguientes mecanismos básicos de seguridad:

- **Autenticación de usuarios**
  - Endpoint POST /api/login que valida usuario y contraseña en la tabla usuarios.
  - Si las credenciales son correctas, se devuelve un objeto JSON con la información pública del usuario.
- **Autorización basada en rol (frontend)**
  - El objeto usuario se guarda en localStorage.
  - En las páginas:
    - activos.html y usuarios.html se verifica rol_id.
    - Solo se muestran los botones de **Agregar**, **Editar** y **Eliminar** si el rol_id es igual a 2 (Administrador).
  - Si un usuario no administrador intenta entrar directamente a usuarios.html, se le redirige al dashboard.html.

- **Validaciones de datos**
    - En el backend se valida que los campos obligatorios no lleguen vacíos y se devuelven errores HTTP adecuados (400, 401, 409, 500).
    - En el frontend, antes de enviar los formularios, se verifica que se completen todos los campos requeridos.

Nota: En esta versión las contraseñas se almacenan en texto plano por simplicidad académica. Como trabajo futuro se propone utilizar hashing con bcrypt y manejo de tokens (JWT) para una seguridad más robusta.

3.2 Gestión de errores

- Uso de códigos de estado HTTP:
    - 400 – Petición inválida (faltan campos).
    - 401 – Error de autenticación.
    - 404 – Recurso no encontrado (por ejemplo, activo a eliminar que no existe).
    - 409 – Conflictos de integridad (número de activo duplicado, eliminar activo con checklists asociados).
    - 500 – Errores internos del servidor o de la base de datos.
- Mensajes de error personalizados enviados en formato JSON, por ejemplo:

Figura no: 12 Ejemplo de manejo de errores

- ```
  { error: 'El número de activo ya existe.' });
  ```
- En el frontend se muestran estos mensajes en el formulario (div msgActivo, loginMsg, registroMsg) o con alert() para operaciones como eliminar.

3.3 Servicios externos

El backend **no consume servicios externos** en esta versión. Todo se maneja con recursos internos (servidor Express y base de datos MySQL). Esto simplifica el despliegue y es suficiente para el alcance del proyecto.
  ```

3.4 Pruebas

Se realizaron principalmente **pruebas funcionales manuales**, verificando:

- Inicio de sesión con credenciales válidas e inválidas.
- Registro de usuarios nuevos, incluyendo:
  - Validación de usuario repetido.
  - Restricción de rol administrador con contraseña maestra.
- Registro de checklists:
  - Selección de activo y usuario.
  - Verificación de que los checklists se reflejen en el historial y en los indicadores de “Última revisión”.
- CRUD de activos:
  - Creación de activos nuevos.
  - Edición de activos existentes.
  - Manejo de errores al intentar crear/editar con un número de activo duplicado.
  - Eliminación de activos sin checklists.
  - Mensaje de error al intentar eliminar activos con checklists asociados.
- Comportamiento de control de acceso:
  - Botones de administración visibles solo para usuarios con rol_id = 2.

Como **trabajo futuro**, se propone agregar pruebas automatizadas con Jest + Supertest para los endpoints más críticos (login, checklist y activos).

3.5 Escalabilidad

Aunque el sistema es un monolito simple, se tomaron decisiones que facilitan su crecimiento:

- El backend está diseñado como **API REST**, por lo que el frontend podría migrarse a un framework moderno (React, Vue, etc.) o incluso reemplazarse por una app móvil, sin cambiar la lógica del servidor.
- El uso de MySQL permite escalar verticalmente (mejores recursos) y, si fuera necesario, horizontalmente mediante replicación.

- La división por módulos lógicos (login, usuarios, activos, checklist) facilita separar, en el futuro, servicios independientes (por ejemplo, un microservicio de reportes).

---

4. Conexión con la base de datos y el frontend

4.1 Conexión con la base de datos

La conexión se realiza desde Node.js usando mysql2 y variables de entorno (.env) para no exponer credenciales en el código: Figura no: 13 Conexión y configuración mysql

```
//  CARGA DE VARIABLES DE ENTORNO Y DEPENDENCIAS

require('dotenv').config();
const express = require('express');
const mysql   = require('mysql2');
const cors    = require('cors');


//  CONFIGURACIÓN BASICA DEL EXPRESS

const app = express();

// Habilitar CORS (para que el front pueda llamar al backend)
app.use(cors());

// Permitir json en el body de las peticiones
app.use(express.json());


//  CONEXIÓN A BASE DE DATOS MySQL (cv_maq)
//  Los parámetros vienen del archivo .env

const db = mysql.createConnection({
  host    : process.env.DB_HOST,
  user    : process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_DATABASE
});

// Probar la conexión al iniciar el servidor
db.connect((err) => {
  if (err) {
    console.error('Error conectando a la base de datos:', err);
  } else {
    console.log('La conexión a la base de datos fue exitosa');
  }
});
```

La base de datos reutiliza el diseño de la Parte III (tablas usuarios, activos, checklist, etc.), respetando las llaves foráneas establecidas (por ejemplo, checklist.activo_id → activos.id).

4.2 Comunicación con el frontend

El frontend está compuesto por archivos HTML estáticos (login.html, dashboard.html, checklist.html, activos.html, usuarios.html) que consumen la API mediante fetch:

- **Login:**

Figura no: 14 Comunicacón con frontend Login

```
// Llamada al backend (/api/login)
fetch('http://localhost:3001/api/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ nombre_usuario, contrasena })
})
```

- **Activos (carga inicial):**

Figura no: 15 Comunicacón con frontend Activos

```
function cargarActivos() {
  return fetch("http://localhost:3001/api/activos-completo")
    .then((res) => res.json())
    .then((activos) => {
```

- **Checklists (registro):**

Figura no: 16 Comunicacón con frontend Checklist

```
fetch('http://localhost:3001/api/checklist', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(checklist)
```

Esta separación clara entre frontend y backend permite que ambos evolucionen de manera independiente.

5. Fragmentos de código representativos

5.1 Endpoint de login

Figura no: 17 Endpoint de login

```
app.post('/api/login', (req, res) => {
  const { nombre_usuario, contrasena } = req.body;

  const query = `
    SELECT *
    FROM usuarios
    WHERE nombre_usuario = ? AND contrasena = ?
  `;

  db.query(query, [nombre_usuario, contrasena], (err, results) => {
    if (err) {
      console.error('Error en login:', err);
      return res.status(500).json({ error: 'Error en el servidor' });
    }

    if (results.length === 0) {
      return res.status(401).json({ error: 'Usuario o contraseña incorrectos' });
    }

    const usuario = results[0];
    res.json({
      id             : usuario.id,
      nombre_usuario : usuario.nombre_usuario,
      nombre_completo: usuario.nombre_completo,
      rol_id         : usuario.rol_id
    });
  });
});
```

valida las credenciales y devuelve solo la información necesaria, sin exponer la contraseña.

5.2 Registro de checklist

Figura no: 18 Registro de checklist

```
app.post('/api/checklist', (req, res) => {
  const c = req.body;

  // Validación mínima
  if (!c.activo_id || !c.usuario_id) {
    return res.status(400).json({ error: 'Falta seleccionar activo o usuario.' });
  }

  const insertQuery = `
    INSERT INTO checklist (
      activo_id, usuario_id, fecha, departamento,
      combustible1, combustible2, combustible3, combustible4,
      motor1, motor2, motor3, motor4,
      aire1, aire2, transmision1, direccion1,
      llantas1, llantas2,
      electrico1, electrico2, electrico3,
      aseo, reporte, seguridad
    ) VALUES (?, ?, ?, ?,
              ?, ?, ?, ?,
              ?, ?, ?, ?,
              ?, ?, ?, ?,
              ?, ?,
              ?, ?, ?,
              ?, ?, ?)
  `;

  const values = [
    c.activo_id,
    c.usuario_id,
    c.fecha,
    c.departamento,
    c.combustible1, c.combustible2, c.combustible3, c.combustible4,
    c.motor1,       c.motor2,       c.motor3,       c.motor4,
    c.aire1,        c.aire2,
    c.transmision1,
    c.direccion1,
    c.llantas1,     c.llantas2,
    c.electrico1,   c.electrico2,   c.electrico3,
    c.aseo,         c.reporte,      c.seguridad
  ];

  db.query(insertQuery, values, (err) => {
    if (err) {
      console.error('Error al registrar checklist:', err);
      return res.status(500).json({ error: 'Error al registrar el checklist' });
    }
    res.json({ mensaje: 'Checklist registrado correctamente' });
  });
});
```

5.3 CRUD de activos (ejemplo: creación)

Figura no: 19 CRUD de activos ejemplo de creación

```
app.post('/api/activos', (req, res) => {
  let { nombre, tipo, numero_activo, anio, placa, departamento } = req.body;

  // Normalizar datos
  nombre        = (nombre || '').trim();
  tipo          = (tipo || '').trim();
  numero_activo = (numero_activo || '').trim();
  placa         = (placa || '').trim();
  departamento  = (departamento || '').trim();
  anio          = (anio === '' || anio === undefined) ? null : String(anio).trim();

  // Validaciones mínimas
  if (!nombre || !tipo || !numero_activo || !departamento) {
    return res.status(400).json({ error: 'Faltan campos obligatorios.' });
  }

  // Validar número de activo único
  db.query(
    'SELECT id FROM activos WHERE numero_activo = ?',
    [numero_activo],
    (err, rows) => {
      if (err) {
        console.error('SELECT duplicados error:', err);
        return res.status(500).json({ error: 'Error en el servidor.' });
      }

      if (rows.length > 0) {
        return res.status(409).json({ error: 'El número de activo ya existe.' });
      }

      const sql = `
        INSERT INTO activos (nombre, tipo, numero_activo, anio, placa, departamento)
        VALUES (?, ?, ?, ?, ?, ?)
      `;
      db.query(sql, [nombre, tipo, numero_activo, anio, placa || null, departamento], (err2, result) => {
        if (err2) {
          console.error('INSERT activos error:', err2.code, err2.sqlMessage);
          return res
            .status(400)
            .json({ error: `${err2.code}: ${err2.sqlMessage}` });
        }

        res.json({ id: result.insertId, mensaje: 'Activo agregado correctamente' });
      });
    }
  );
});
```

El backend de **CV-Maq** cumple con los requerimientos planteados para el proyecto:

- Implementa una **API REST** clara y organizada, adaptada al tamaño y complejidad del sistema.
- Mantiene una estructura lógica por módulos (autenticación, usuarios, activos, checklist) que favorece la **modularidad y el mantenimiento**.
- Integra adecuadamente la **base de datos MySQL** definida en la Parte III, utilizando consultas parametrizadas para reducir riesgos de inyección SQL.
- Incluye elementos básicos de **seguridad** (autenticación, control de rol, validaciones) y de **gestión de errores**, con mensajes claros hacia el usuario.
- Se ha verificado el correcto funcionamiento a través de **pruebas funcionales** en las principales funcionalidades del sistema.

Como líneas de mejora y trabajo futuro se proponen:

- Implementar **cifrado de contraseñas** y autenticación basada en tokens (JWT).
- Agregar **pruebas automatizadas** (unitarias y de integración) con Jest y Supertest.
- Separar el código en archivos de rutas y controladores independientes para una mayor limpieza.
- Exponer nuevos endpoints de reportes y estadísticas, y eventualmente evolucionar hacia una arquitectura de servicios más desacoplada.

Conclusión

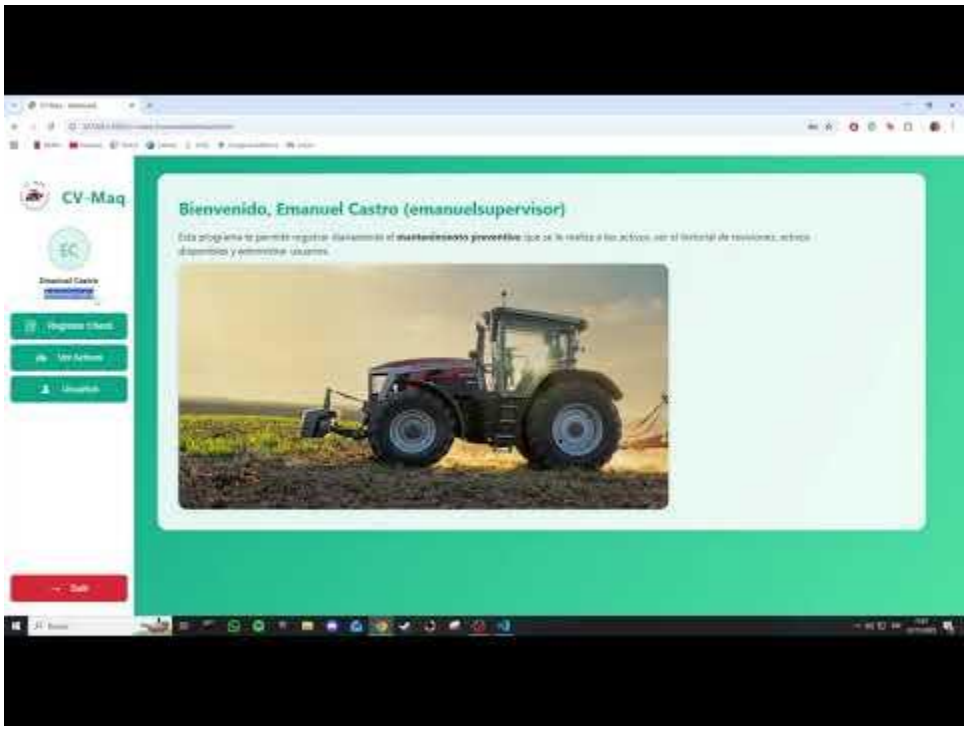
El desarrollo del sistema **CV-Maq** permitió digitalizar por completo el control de activos y las revisiones preventivas, integrando una base de datos sólida, un backend seguro en Node.js y una interfaz web intuitiva. El sistema cumple los objetivos principales: gestionar activos, registrar checklists, controlar accesos por rol y mantener un historial claro de mantenimiento.

Además, se implementaron mejoras importantes como modales personalizados, validaciones, eliminación con control de datos asociados y una experiencia de usuario más moderna y eficiente. El proyecto deja lista una base tecnológica estable, escalable y preparada para futuras funciones como reportes avanzados, estadísticas o integración móvil.

En conclusión, CV-Maq representa un avance significativo en la automatización y organización interna de cualquier empresa que utilice registros en papel, demostrando que una solución diseñada adecuadamente puede optimizar procesos y facilitar la toma de decisiones en la empresa.

Evidencia en Video

- **Link del video:**



https://youtu.be/bzCHTNZ2kPM

Referencias

Bessa, A. (29 de Julio de 2025). *alura LATAM*. Obtenido de Node.JS: qué es, cómo funciona este entorno de ejecución de JavaScript y una guía para empezar: https://www.aluracursos.com/blog/node-js-que-es-como-funciona-este-entorno-de-ejecucion-de-javascript-y-una-guia-para-empezar

IBM. (13 de Noviembre de 2025). *IBM*. Obtenido de Creación de un modal: https://www.ibm.com/docs/es/store-engagement?topic=modals-creating-modal

Ironhack. (9 de Julio de 2023). *Ironhack*. Obtenido de Guía para principiantes sobre HTML y CSS para el desarrollo web: https://www.ironhack.com/us/blog/a-beginner-s-guide-to-html-and-css-for-web-development

Silva, I. (1 de Julio de 2024). *alura LATAM*. Obtenido de Guía de Instalación de MySQL Server y MySQL Workbench en Diferentes Sistemas Operativos": https://www.aluracursos.com/blog/descargar-mysql-serve