

Sistema de Mantenimiento Predictivo con IA

Guía completa para desarrollar tu proyecto final paso a

paso Instructor: **Iván Malavé Fierro**

SENA Mosquera - Centro de Biotecnología Agropecuaria

(CBA) Tecnología en Análisis y Desarrollo de Software

¿Qué aprenderás en esta guía?

Esta guía te enseñará a crear un sistema completo de mantenimiento predictivo usando PHP, Laravel y tecnologías de Inteligencia Artificial. Es como construir un "doctor para máquinas" que puede predecir cuándo se van a dañar antes de que suceda.

Al finalizar, tendrás un proyecto profesional que demuestra tus habilidades técnicas y creatividad, cumpliendo todos los requisitos del instructor **Iván Malavé Fierro** para obtener una excelente calificación.

Antes de Empezar

comandos

Requisitos Previos

- Conocimientos básicos de PHP
- Comprensión de HTML y CSS
- Manejo básico de bases de datos
- MySQL
- Saber usar la terminal o línea de

Herramientas

Necesarias

- XAMPP o WAMP instalado
- Composer (gestor de dependencias de PHP)
- Visual Studio Code o tu editor favorito
- Git para control de versiones

Tiempo Estimado

Este proyecto requiere aproximadamente: 40-60 horas de desarrollo
15 horas de documentación
10 horas de pruebas
5 horas para preparar la presentación

Mapa del Proyecto

01

Configuración del Entorno

Instalación de Laravel, configuración de base de datos y estructura inicial del proyecto

03

Sistema de Sensores IoT

Simulación de sensores, protocolo MQTT y almacenamiento de datos en tiempo real

05

Frontend y Visualización

Dashboard interactivo con Vue.js, gráficos en tiempo real y alertas

02

Backend con PHP/Laravel

Creación de modelos, controladores, rutas API y sistema de autenticación

04

Inteligencia Artificial

¿Qué es un Sistema de Mantenimiento

Implementación de modelos predictivos usando Python y integración con PHP

06

Documentación y Pruebas

Manuales técnicos, pruebas de calidad y preparación de la presentación final

Predictivo?

Imagina que pudieras saber exactamente cuándo tu carro necesita mantenimiento antes de que se dañe. Un sistema de mantenimiento predictivo hace exactamente eso, pero para máquinas industriales.

Es como tener un "médico digital" que constantemente revisa la "salud" de las máquinas usando sensores que miden temperatura, vibración, presión y otros datos importantes.

Para qué sirve este sistema



Beneficios Principales

Evita paradas inesperadas: Predice fallos antes de que ocurran

Ahorra dinero: Reduce costos de reparaciones de emergencia

Aumenta productividad: Las máquinas trabajan más tiempo sin interrupciones

Mejora la seguridad: Previene accidentes por fallos de equipos

Optimiza recursos: Programa mantenimiento cuando es

realmente necesario

Paso 1: Configuración del Entorno de Desarrollo

1.1 ¿Qué es Laravel?

Laravel es un framework de PHP que facilita crear aplicaciones web. Es como tener una "caja de herramientas" pre-construida que nos ahorra mucho tiempo de programación.

1.2 Para qué sirve

Laravel nos permite crear aplicaciones web profesionales más rápido, con menos código y siguiendo las mejores prácticas de programación.

1.3 Cómo se instala

Usaremos Composer (el gestor de paquetes de PHP) para instalar Laravel automáticamente.

Paso a Paso: Instalación de Laravel

1 Verificar PHP y Composer

Abre la terminal y ejecuta estos comandos para verificar que tienes las herramientas instaladas:

```
php --version  
composer --version
```

```
composer create-project laravel/laravel sistema-mantenimiento
```

3 Configurar la base de datos

Crea una base de datos en phpMyAdmin llamada "mantenimiento_predictivo"

2 Crear el proyecto Laravel

En la terminal, navega a tu carpeta de proyectos y ejecuta:

4 Configurar variables de entorno

Edita el archivo .env en la raíz del proyecto con los datos de tu base de datos

Ejemplo Mínimo Reproducible: Primer Laravel

Copia exactamente este código en tu archivo .env:

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=mantenimiento_predictivo  
DB_USERNAME=root  
DB_PASSWORD=
```

Salida Esperada

Al ejecutar `php artisan serve` en la terminal, deberías ver:

```
Laravel development server started: http://127.0.0.1:8000
```

TIP: Si ves este mensaje, ¡tu instalación fue exitosa! Abre tu navegador en esa dirección para ver la página de bienvenida de Laravel.

Errores Comunes en la Instalación

Error: "composer command not found"

Causa: Composer no está instalado o no está en el PATH del sistema

Solución: Descarga Composer desde getcomposer.org e instálalo siguiendo

las instrucciones para tu sistema operativo

Error: "SQLSTATE[HY000] [1049] Unknown database"

Causa: La base de datos especificada en .env no existe

Solución: Crea la base de datos "mantenimiento_predictivo" en

phpMyAdmin antes de ejecutar las migraciones

Error: "Port 8000 already in use"

Causa: Ya hay otra aplicación usando el puerto 8000

Solución: Usa otro puerto: `php artisan serve --port=8001`

Comprueba tu Comprensión

Mini-tarea de verificación:

Abre tu navegador en <http://127.0.0.1:8000>.

¿Puedes ver la página de bienvenida de Laravel?2.

Ejecuta `php artisan --version` en la terminal3.

¿Te muestra la versión de Laravel instalada?4.

¡Perfecto! Si ambas respuestas son "Sí", ya tienes Laravel funcionando correctamente. Puedes continuar con el siguiente paso.

Paso 2: Estructura de la Base de Datos

2.1 ¿Qué son las migraciones?

Las migraciones son como "planos" de construcción para tu base de datos. Te permiten crear y modificar tablas usando código PHP en lugar de hacerlo manualmente en phpMyAdmin.

2.2 Para qué sirven

Las migraciones mantienen sincronizada la estructura de tu base de datos entre diferentes entornos (desarrollo, pruebas, producción) y permiten trabajar en equipo sin problemas.

Creando las Tablas del Sistema

01

Tabla de Equipos

Almacena información de cada máquina: nombre, ubicación, tipo, fecha de instalación

```
php artisan make:migration create_equipos_table
```

03

Tabla de Lecturas

Guarda todos los datos que envían los sensores: valor, timestamp, estado

```
php artisan make:migration create_lecturas_table
```

02

Tabla de Sensores

Registra cada sensor asociado a un equipo: tipo de sensor, rango de operación, límites de alerta

```
php artisan make:migration create_sensores_table
```

04

Tabla de Alertas

Almacena alertas generadas por el sistema de IA: nivel de criticidad, tipo de fallo predicho

```
php artisan make:migration create_alertas_table
```

Ejemplo: Migración de Equipos

Abre el archivo de migración que acabas de crear y reemplaza el contenido con este código:

```
id();  
$table->string('nombre', 100);  
$table->string('tipo', 50);  
$table->string('ubicacion', 100);  
$table->text('descripcion')->nullable();  
$table->date('fecha_instalacion');  
$table->enum('estado', ['activo', 'mantenimiento', 'inactivo']);  
$table->timestamps();  
});  
}  
  
public function down()  
{  
Schema::dropIfExists('equipos');  
}  
};
```

Salida Esperada de las Migraciones

Ejecuta las migraciones con el comando:

```
php artisan migrate
```

Deberías ver algo como esto:

```
Migrating: 2024_01_15_000001_create_equipos_table
Migrated: 2024_01_15_000001_create_equipos_table (25.43ms)
Migrating: 2024_01_15_000002_create_sensores_table
Migrated: 2024_01_15_000002_create_sensores_table (18.92ms)
```

CUIDADO: Si ves errores de sintaxis, revisa que hayas copiado el código exactamente como se muestra, incluyendo punto y coma al final.

Paso 3: Modelos Eloquent

3.1 ¿Qué es un modelo Eloquent?

Un modelo Eloquent es como un "traductor" entre tu código PHP y la base de datos. Cada modelo representa una tabla y te permite trabajar con los datos de manera más fácil y natural.

3.2 Para qué sirve

Los modelos te permiten hacer consultas a la base de datos usando código PHP limpio, en lugar de escribir SQL directamente. Es como hablar en español en lugar de en código binario.

3.3 Cómo se crean

Laravel incluye un comando artisan que genera automáticamente la estructura básica del modelo.

Creando los Modelos del Sistema

1

Equipo

Modelo Equipo

Representa cada máquina

industrial php artisan make:model

2

3

Modelo Lectura

Representa datos de sensores php artisan make:model Sensor

artisan make:model Lectura

Representa cada sensor IoT php

Representa alertas del sistema php

artisan make:model Alerta

4

Modelo Sensor

Modelo Alerta

Ejemplo: Modelo Equipo Completo

Edita el archivo app/Models/Equipo.php con este código:

```
hasMany(Sensor::class);
}

// Relación: Un equipo tiene muchas alertas
public function alertas()
{
    return $this->hasMany(Alerta::class);
}
```

Paso 4: Controladores API

4.1 ¿Qué es un controlador?

Un controlador es como el "cerebro" de tu aplicación. Recibe las peticiones del usuario, procesa la información y decide qué respuesta enviar de vuelta.

4.2 Para qué sirve

Los controladores organizan la lógica de tu aplicación de manera ordenada. Cada acción (crear, leer, actualizar, eliminar) tiene su propio método dentro del controlador.

4.3 Cómo se crean

Laravel puede generar controladores automáticamente con todas las funciones básicas incluidas.

Creando los Controladores

Controlador de Equipos

```
php artisan make:controller  
Api/EquipoController --api
```

Manejará todas las operaciones CRUD
de equipos

Controlador de Sensores

IoT

```
php artisan make:controller  
Api/SensorController --api
```

Gestionará la configuración de sensores

Controlador de Dashboard

```
php artisan make:controller  
Api/DashboardController
```

Proporcionará datos para la visualización en tiempo real

TIP: El flag --api crea un controlador optimizado para APIs, sin los métodos create() y edit() que solo se usan en aplicaciones web tradicionales.

Ejemplo: Controlador de Equipos

Edita `app/Http/Controllers/Api/EquipoController.php`:

```
get();  
return response()->json($equipos);  
  
// Crear un nuevo equipo  
public function store(Request $request)  
{  
    $equipo = Equipo::create($request->all());  
    return response()->json($equipo, 201);
```

```
}

// Mostrar un equipo específico
public function show($id)
{
    $equipo = Equipo::with('sensores', 'alertas')->find($id);
    if (!$equipo) {
        return response()->json(['error' => 'Equipo no encontrado'], 404);
    }
    return response()->json($equipo);
}
```

Configurando las Rutas API

Las rutas definen qué URLs estarán disponibles en tu aplicación. Edita el archivo `routes/api.php`:

¡Excelente! Ahora tienes una API REST completamente funcional. Puedes probarla usando Postman o cualquier cliente HTTP.

Paso 5: Simulación de Sensores IoT

5.1 ¿Qué son los sensores IoT?

Los sensores IoT son dispositivos pequeños que miden condiciones físicas (temperatura, vibración, presión) y envían estos datos a través de internet. Es como tener "ojos y oídos" digitales en cada máquina.

5.2 Para qué sirve la simulación

Como no tenemos sensores físicos reales, crearemos un "simulador" que genere datos realistas para probar nuestro sistema.

Creando el Simulador de Sensores

Crea un nuevo comando artisan que simule sensores enviando datos: `php artisan make:command SimularSensores`

Edita el archivo `app\Console\Commands\SimularSensores.php`:

```
generarValor($sensor->tipo);

// Crear nueva lectura
Lectura::create([
'sensor_id' => $sensor->id,
'velor' => $valor,
'timestamp' => Carbon::now(),
'estado' => $this->evaluarEstado($valor, $sensor)
]);
}

$this->info('Datos de sensores simulados exitosamente');
}

private function generarValor($tipo)
{
switch ($tipo) {
case 'temperatura':
return rand(20, 80); // Grados Celsius
case 'vibracion':
return rand(0, 100); // mm/s
case 'presion':
return rand(1, 10); // Bar
```

```
default:  
    return rand(0, 100);  
}  
}  
}
```

Ejecutando la Simulación

Para ejecutar el simulador de sensores, usa este comando en la terminal:

```
php artisan sensores:simular
```

Salida Esperada:

Datos de sensores simulados exitosamente

Para automatizar la simulación cada minuto, agrega esta línea al archivo `app\Console\Kernel.php` en el método

```
schedule(): $schedule->command('sensores:simular')->everyMinute();
```

TIP: Puedes verificar que los datos se están guardando consultando la tabla 'lecturas' en phpMyAdmin.

Paso 6: Integración con Inteligencia Artificial

Artificial 6.1 ¿Qué es la IA Predictiva?

La Inteligencia Artificial predictiva es como tener un "oráculo" que analiza patrones en los datos históricos para predecir qué va a pasar en el futuro.

6.2 Para qué sirve en mantenimiento

La IA puede detectar patrones sutiles que indican que una máquina está a punto de fallar, incluso antes de que los técnicos humanos lo

noten. 6.3 Cómo funciona

Entrenaremos un modelo con datos históricos de máquinas que han fallado antes, para que aprenda a reconocer las "señales de alarma".

Configurando Python para IA Crear entorno virtual

En la carpeta de tu proyecto Laravel:

pip install numpy pandas scikit-learn
instálalo. Verifica la instalación: python -m venv ia_env # En

Instalar librerías de IA Con el entorno activado, instala:

Instalar Python y pip

Descarga Python 3.9+ desde python.org e

macOS/Linux:

```
python --version pip --version  
ia_env\Scripts\activate # En
```

Windows:

matplotlib seaborn fastapi unicorn

Creando el Modelo Predictivo

Crea una carpeta `ia/` en tu proyecto y dentro un archivo `modelo_predictivo.py`:

```
# ia/modelo_predictivo.py  
import pandas as pd  
import numpy as np  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import accuracy_score  
import joblib
```

```
class ModeloMantenimiento:  
    def __init__(self):  
        self.modelo = RandomForestClassifier(n_estimators=100, random_state=42)  
        self.caracteristicas = ['temperatura', 'vibracion', 'presion', 'horas_operacion']  
  
    def entrenar(self, datos_historicos):  
        """  
        Entrena el modelo con datos históricos  
        datos_historicos: DataFrame con columnas ['temperatura', 'vibracion', 'presion', 'horas_operacion', 'fallo']  
        X = datos_historicos[self.caracteristicas]  
        y = datos_historicos['fallo'] # 1 = fallo, 0 = normal  
  
        # Dividir datos en entrenamiento y prueba  
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
        # Entrenar modelo  
        self.modelo.fit(X_train, y_train)  
  
        # Evaluar precisión  
        predicciones = self.modelo.predict(X_test)  
        precision = accuracy_score(y_test, predicciones)  
  
        print(f"Precisión del modelo: {precision:.2%}")
```

```
# Guardar modelo entrenado
joblib.dump(self.modelo, 'modelo_mantenimiento.pkl')

return precision

def predecir(self, temperatura, vibracion, presion, horas_operacion):
"""
Predice la probabilidad de fallo
"""

datos = np.array([[temperatura, vibracion, presion, horas_operacion]])
probabilidad = self.modelo.predict_proba(datos)[0][1] # Probabilidad de fallo

return probabilidad
```

API de IA con FastAPI

Crea el archivo `ia/api_ia.py` para exponer el modelo como API:

```
# ia/api_ia.py
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
import numpy as np
```

```
app = FastAPI()

# Cargar modelo entrenado
try:
    modelo = joblib.load('modelo_mantenimiento.pkl')
    print("Modelo cargado exitosamente")
except:
    print("No se pudo cargar el modelo. Asegúrate de entrenarlo primero.")
    modelo = None

class DatosSensor(BaseModel):
    temperatura: float
    vibracion: float
    presion: float
    horas_operacion: float

@app.post("/predecir")
def predecir_fallo(datos: DatosSensor):
    if modelo is None:
        return {"error": "Modelo no disponible"}

    # Hacer predicción
    entrada = np.array([[datos.temperatura, datos.vibracion, datos.presion, datos.horas_operacion]])  probabilidad =
```

```
modelo.predict_proba(entrada)[0][1]

# Clasificar riesgo
if probabilidad > 0.8:
    riesgo = "crítico"
elif probabilidad > 0.6:
    riesgo = "alto"
elif probabilidad > 0.4:
    riesgo = "moderado"
else:
    riesgo = "bajo"

return {
    "probabilidad_fallo": round(probabilidad * 100, 2),
    "nivel_riesgo": riesgo,
    "recomendacion": obtener_recomendacion(riesgo)
}

def obtener_recomendacion(riesgo):
    recomendaciones = {
        "crítico": "Detener equipo inmediatamente y realizar mantenimiento",
        "alto": "Programar mantenimiento en las próximas 24 horas",
        "moderado": "Programar inspección en la próxima semana",
        "bajo": "Continuar operación normal"
    }
```

```
return recomendaciones[riesgo]

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8001)
```

Conectando PHP con la IA

Crea un servicio en Laravel para comunicarse con la API de Python. Archivo app/Services/IAService.php:

```
apiUrl . '/predecir', [
    'temperatura' => $temperatura,
    'vibracion' => $vibracion,
    'presion' => $presion,
    'horas_operacion' => $horasOperacion
]);

if ($response->successful()) {
    return $response->json();
}

return ['error' => 'No se pudo conectar con el servicio de IA'];

} catch (\Exception $e) {
```

```
return ['error' => 'Error de comunicación: ' . $e->getMessage()];
}
}
}
```

TIP: Para que funcione la comunicación, primero inicia la API de Python con: python ia/api_ia.py

Paso 7: Frontend con

Vue.js 7.1 ¿Qué es Vue.js?

Vue.js es un framework de JavaScript que hace que crear interfaces de usuario sea tan fácil como armar bloques de Lego. Cada componente es un bloque reutilizable.

7.2 Para qué sirve en nuestro proyecto

Vue.js nos permitirá crear un dashboard interactivo donde los usuarios puedan ver datos en tiempo real, gráficos dinámicos y recibir alertas.

Configurando Vue.js con Laravel

01

Instalar Node.js y npm

Descarga Node.js desde nodejs.org. Verifica la instalación:

node --version

npm --version

02

Configurar Laravel Mix

Creando el Componente

Dashboard

Crea el archivo resources/js/components/Dashboard.vue:

En la carpeta de tu proyecto Laravel:

npm install

npm install vue@next @vitejs/plugin-vue
03

Configurar Vite

Edita el archivo vite.config.js:

```
import { defineConfig } from 'vite';
import laravel from
```

Estilos CSS para el

```
'laravel-vite-plugin'; import vue from
'@vitejs/plugin-vue';
```

```
export default defineConfig({
  plugins: [
    laravel(['resources/js/app.js']),
    vue(),
  ],
});
```

Dashboard

Agrega estos estilos al final del componente

Dashboard.vue:

Configurando el Router de Vue

Edita el archivo resources/js/app.js:

```
import { createApp } from 'vue'  
import Dashboard from './components/Dashboard.vue'  
  
const app = createApp({  
  components: {  
    Dashboard  
  }  
})  
  
app.mount('#app')
```

Y modifica la plantilla principal resources/views/welcome.blade.php:

```
@vite(['resources/js/app.js'])
```

¡Genial! Ahora ejecuta `npm run dev` para compilar los assets de Vue.js

Paso 8: Sistema de Alertas en Tiempo Real

8.1 ¿Qué son las alertas en tiempo real?

Las alertas en tiempo real son notificaciones instantáneas que se envían cuando el sistema detecta una situación que requiere atención inmediata. Es como tener un "sistema de alarma" para las máquinas.

8.2 Para qué sirven

Permiten responder rápidamente a problemas potenciales antes de que se conviertan en fallas costosas o peligrosas.

Implementando WebSockets

Instala Laravel WebSockets para comunicación en tiempo real:

```
composer require beyondcode/laravel-websockets  
php artisan vendor:publish --provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider" --tag="migrations" php artisan migrate  
php artisan vendor:publish --provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider" --tag="config" Configura el archivo .env:
```

```
BROADCAST_DRIVER=pusher  
PUSHER_APP_ID=local  
PUSHER_APP_KEY=local  
PUSHER_APP_SECRET=local  
PUSHER_APP_CLUSTER=mt1
```

En config/broadcasting.php, actualiza la configuración de Pusher:

```
'pusher' => [
```

```
'driver' => 'pusher',
'key' => env('PUSHER_APP_KEY'),
'secret' => env('PUSHER_APP_SECRET'),
'app_id' => env('PUSHER_APP_ID'),
'options' => [
  'cluster' => env('PUSHER_APP_CLUSTER'),
  'host' => '127.0.0.1',
  'port' => 6001,
  'scheme' => 'http',
  'encrypted' => false,
  'useTLS' => false,
],
],
```

Creando Eventos de Alerta

Genera un evento para las alertas:

```
php artisan make:event AlertaGenerada
```

Edita app/Events/AlertaGenerada.php:

```
alerta = $alerta;
}
```

```
public function broadcastOn()
{
    return new Channel('alertas');
}

public function broadcastAs()
{
    return 'nueva-alerta';
}

public function broadcastWith()
{
    return [
        'id' => $this->alerta->id,
        'equipo' => $this->alerta->equipo->nombre,
        'mensaje' => $this->alerta->mensaje,
        'nivel' => $this->alerta->nivel,
        'timestamp' => $this->alerta->created_at->format('Y-m-d H:i:s') ];
}
```

Job para Análisis Predictivo

Crea un job que analice constantemente los datos y genere alertas: php artisan make:job AnalizarDatosPredictivos

Edita `app/Jobs/AnalizarDatosPredictivos.php`:

```
get();

foreach ($equipos as $equipo) {
    // Obtener últimas lecturas de sensores
    $ultimasLecturas = $this->obtenerUltimasLecturas($equipo);

    if (count($ultimasLecturas) >= 4) {
        // Hacer predicción con IA
        $prediccion = $iaService->predecirFallo(
            $ultimasLecturas['temperatura'],
            $ultimasLecturas['vibracion'],
            $ultimasLecturas['presion'],
            $equipo->horas_operacion ?? 1000
        );

        // Si el riesgo es alto, crear alerta
        if (isset($prediccion['probabilidad_fallo']) && $prediccion['probabilidad_fallo'] > 70) { $alerta = Alerta::create([
            'equipo_id' => $equipo->id,
            'tipo' => 'predictivo',
        ]);
    }
}
```

```
'nivel' => $prediccion['nivel_riesgo'],
'mensaje' => "Riesgo de fallo del {$prediccion['probabilidad_fallo']}%", 'recomendacion' => $prediccion['recomendacion'],
'datos_sensor' => json_encode($ultimasLecturas)
]);


// Emitir evento en tiempo real
broadcast(new AlertaGenerada($alerta));
}
}
}
}
```

```
private function obtenerUltimasLecturas($equipo)
{
$lecturas = [];
foreach ($equipo->sensores as $sensor) {
$ultimaLectura = Lectura::where('sensor_id', $sensor->id)
->latest()
->first();
if ($ultimaLectura) {
$lecturas[$sensor->tipo] = $ultimaLectura->valor;
}
}
return $lecturas;
```

}

}

Paso 9: Pruebas de Calidad del Software

9.1 ¿Qué son las pruebas de software?

Las pruebas de software son como "exámenes" que le hacemos a nuestro código para asegurar que funcione correctamente en diferentes situaciones.

9.2 Tipos de pruebas importantes

Pruebas Unitarias: Verifican que cada función individual funcione

correctamente **Pruebas de Integración:** Verifican que diferentes partes del sistema trabajen juntas

Pruebas de API: Verifican que las APIs respondan correctamente

Pruebas de Rendimiento: Verifican que el sistema sea rápido bajo carga

Configurando PHPUnit para Pruebas

Laravel incluye PHPUnit por

defecto. Primero configura la base de datos de pruebas en `phpunit.xml`:

Crea una prueba para el modelo Equipo:

```
php artisan make:test EquipoTest --unit
```

Edita `tests/Unit/EquipoTest.php`:

```
'Bomba Principal',  
'tipo' => 'Bomba Centrífuga',  
'ubicacion' => 'Planta Norte',  
'descripcion' => 'Bomba de agua principal',  
'fecha_instalacion' => '2023-01-15',  
'estado' => 'activo'
```

```
]);  
  
$this->assertInstanceOf(Equipo::class, $equipo);  
$this->assertEquals('Bomba Principal', $equipo->nombre);  
$this->assertEquals('activo', $equipo->estado);  
}  
  
public function test_equipo_puede_tener_sensores()  
{  
    $equipo = Equipo::factory()->create();  
    $this->assertInstanceOf('Illuminate\Database\Eloquent\Collection', $equipo->sensores); }  
}
```

Pruebas de API con Feature Tests

Crea pruebas para verificar que las APIs funcionen correctamente:

```
php artisan make:test Api/EquipoApiTest
```

Edita `tests/Feature/Api/EquipoApiTest.php`:

```
count(3)->create();  
  
// Hacer petición GET
```

```
$response = $this->getJson('/api/equipos');

// Verificar respuesta
$response->assertStatus(200)
->assertJsonCount(3);
}

public function test_puede_crear_equipo_via_api()
{
    $datosEquipo = [
        'nombre' => 'Compresor Test',
        'tipo' => 'Compresor',
        'ubicacion' => 'Área de Pruebas',
        'descripcion' => 'Equipo para testing',
        'fecha_instalacion' => '2024-01-01',
        'estado' => 'activo'
    ];

    $response = $this->postJson('/api/equipos', $datosEquipo);

    $response->assertStatus(201)
->assertJson(['nombre' => 'Compresor Test']);

    $this->assertDatabaseHas('equipos', $datosEquipo);
}
```

```
}

public function test_puede_obtener_equipo_especifico()
{
    $equipo = Equipo::factory()->create();

    $response = $this->getJson("/api/equipos/{$equipo->id}");

    $response->assertStatus(200)
        ->assertJson(['id' => $equipo->id]);
}

}
```

Ejecutando las Pruebas

Para ejecutar todas las pruebas, usa estos comandos:

```
# Ejecutar todas las pruebas
php artisan test
```

```
# Ejecutar solo pruebas unitarias
php artisan test --testsuite=Unit
```

```
# Ejecutar solo pruebas de características/API
```

```
php artisan test --testsuite=Feature
```

```
# Ejecutar con reporte de cobertura
```

```
php artisan test --coverage
```

Salida Esperada:

```
PASS Tests\Unit\EquipoTest
```

```
7 puede crear equipo
```

```
7 equipo puede tener sensores
```

```
PASS Tests\Feature\Api\EquipoApiTest
```

```
7 puede obtener lista equipos
```

```
7 puede crear equipo via api
```

```
7 puede obtener equipo especifico
```

```
Tests: 5 passed
```

```
Time: 0.84s
```

¡Excelente! Si todas las pruebas pasan, tu código está funcionando correctamente. Las pruebas automáticas te dan confianza en tu aplicación.

Paso 10: Manual Técnico ¿Qué

debe incluir el Manual Técnico?

El manual técnico es como el "manual de instrucciones" para otros desarrolladores. Debe explicar cómo funciona tu sistema internamente.

Arquitectura del

Sistema Diagrama de

componentes Tecnologías
utilizadas

Flujo de datos

Patrones de diseño aplicados

Documentación de

APIs Endpoints disponibles

Parámetros de entrada

Respuestas esperadas

Códigos de error

Instalación y

Configuración

Requisitos

del sistema

Pasos de instalación

Variables de entorno

Configuración de base de datos

Plantilla para Manual Técnico

Usa esta estructura para tu manual técnico:

MANUAL TÉCNICO

Sistema de Mantenimiento Predictivo con IA

1. INTRODUCCIÓN

- Propósito del sistema
- Audiencia objetivo
- Alcance del documento

2. ARQUITECTURA DEL SISTEMA

- Arquitectura general (diagrama)
- Componentes principales:
 - * Backend (Laravel + PHP)
 - * Frontend (Vue.js)
 - * Sistema de IA (Python + FastAPI)
 - * Base de datos (MySQL + InfluxDB)

3. TECNOLOGÍAS UTILIZADAS

- PHP 8.2+ con Laravel 10
- Vue.js 3 para frontend
- Python 3.9+ con scikit-learn
- MySQL para datos transaccionales

- WebSockets para tiempo real

4. INSTALACIÓN Y CONFIGURACIÓN

4.1 Requisitos del Sistema

- PHP 8.2 o superior
- Composer
- Node.js 18+
- MySQL 8.0+
- Python 3.9+

4.2 Pasos de Instalación

1. Clonar repositorio
2. Instalar dependencias PHP: `composer install`
3. Instalar dependencias JS: `npm install`
4. Configurar archivo .env
5. Ejecutar migraciones: `php artisan migrate`
6. Compilar assets: `npm run build`

5. ESTRUCTURA DEL CÓDIGO

- /app/Models/ - Modelos Eloquent
- /app/Http/Controllers/ - Controladores
- /resources/js/ - Componentes Vue.js
- /ia/ - Scripts de Inteligencia Artificial
- /tests/ - Pruebas automatizadas

6. APIs DISPONIBLES

Equipos

- GET /api/equipos - Listar equipos
- POST /api/equipos - Crear equipo
- GET /api/equipos/{id} - Obtener equipo específico

7. CONFIGURACIÓN DE IA

- Entrenamiento del modelo
- API de predicciones
- Integración con Laravel

8. DEPLOYMENT

- Configuración para producción
- Variables de entorno
- Optimizaciones

Manual de Usuario

¿Qué debe incluir el Manual de Usuario?

El manual de usuario está dirigido a las personas que van a usar el sistema día a

día. Debe ser simple, visual y fácil de entender.

Introducción al Sistema

¿Qué hace el sistema?

Beneficios para el usuario
Cómo acceder
Requisitos básicos

comunes
Mensajes de error
A quién contactar
Preguntas frecuentes

Guía Paso a Paso

Cómo iniciar sesión
Navegación por el dashboard
Interpretar alertas
Generar reportes

Resolución de Problemas

Problemas

Plantilla para Manual de Usuario

Estructura recomendada para tu manual de usuario:

```
# MANUAL DE USUARIO
## Sistema de Mantenimiento Predictivo

### 1. BIENVENIDO AL SISTEMA
```

¿Qué hace este sistema?

- Monitorea máquinas en tiempo real
- Predice cuando una máquina puede fallar
- Envía alertas antes de que ocurran problemas
- Ayuda a planificar el mantenimiento

2. CÓMO ACCEDER

1. Abrir navegador web
2. Ir a: <http://tu-dominio.com>
3. Ingresar usuario y contraseña
4. Hacer clic en "Iniciar Sesión"

3. USANDO EL DASHBOARD

3.1 Pantalla Principal

Al ingresar verás:

- Número de equipos activos
- Alertas críticas pendientes
- Eficiencia general del sistema

3.2 Estado de Equipos

- Verde: Funcionando normal
- Amarillo: Requiere atención pronto
- Rojo: Requiere atención inmediata

4. INTERPRETANDO ALERTAS

Niveles de Riesgo:

- Bajo (0-40%): Operación normal
- Moderado (41-60%): Programar inspección
- Alto (61-80%): Mantenimiento en 24 horas
- Crítico (80%+): Detener equipo inmediatamente

5. GENERANDO REPORTES

1. Ir a sección "Reportes"
2. Seleccionar período de tiempo
3. Elegir equipos a incluir
4. Hacer clic en "Generar Reporte"
5. Descargar PDF o Excel

6. PROBLEMAS COMUNES

- No puedo ver datos en tiempo real
 - ³ Verificar conexión a internet
- Las alertas no llegan
 - ³ Revisar configuración de email
- El sistema se ve lento
 - ³ Cerrar pestañas no utilizadas

Paso 11: Proyecto de Investigación

¿Qué es el Proyecto de Investigación?

Es un documento académico donde reflexionas sobre todo el proceso de desarrollo, justificas tus decisiones técnicas y demuestras tu crecimiento como tecnólogo.

Estructura del Proyecto de Investigación

01

Introducción y Objetivos

Presenta el problema a resolver y los objetivos del proyecto

Resultados y Análisis

02

Marco Teórico

Investiga sobre mantenimiento predictivo, IoT e Inteligencia Artificial

Conclusiones

03

Metodología de Desarrollo

Explica cómo abordaste el proyecto paso a paso

04

Presenta lo que lograste y analiza los resultados

Preparación de la

05

Reflexiona sobre aprendizajes y trabajo futuro

Presentación Estructura de la

Presentación

Tu presentación debe contar la historia de tu proyecto de manera clara y convincente:

Tecnologías (3 min)

Introducción (2 min)

Preséntate y explica el problema que resuelve
tu sistema

Explica las tecnologías usadas y por qué las elegiste

3

4

2

destaca características innovadoras

Conclusiones (2 min)

Comparte aprendizajes y posibles mejoras

Demo en Vivo (5 min)

Muestra tu sistema funcionando,

TIP: Practica tu presentación varias veces. El instructor **Iván Malavé Fierro** valorará tu dominio del tema y claridad al explicar.

Preparación para la Sustentación

Preguntas Frecuentes en Sustentación

Preguntas Técnicas

¿Por qué elegiste Laravel sobre otros frameworks?

¿Cómo funciona el algoritmo de predicción?

¿Qué medidas de seguridad implementaste?

¿Cómo escalarías el sistema para más equipos?

Preguntas de Innovación

¿Qué funcionalidades adicionales

implementaste? ¿Cómo mejoraste los

requerimientos básicos?

¿Qué impacto tendría tu sistema en una empresa real?

Preguntas de Proceso

¿Cuál fue el mayor reto del

proyecto?

¿Qué harías diferente si
empezaras de nuevo?

¿Cómo probaste la calidad del

sistema?

¿Qué recursos usaste para
aprender nuevas
tecnologías?

Funcionalidades Adicionales para Destacar

Autenticación Avanzada

Implementa autenticación de dos factores (2FA) usando Google Authenticator:

```
composer require pragmarx/google2fa-laravel
```

Agrega estos campos a la tabla users:

```
$table->text('google2fa_secret')->nullable();  
$table->timestamp('google2fa_ts')->nullable();
```

Dashboard con Realidad Aumentada

Integra AR.js para visualizar datos de equipos en realidad aumentada:

npm install three ar.js

Análisis Estadístico Avanzado

Usa librerías como Chart.js para gráficos interactivos:

npm install chart.js vue-chartjs

¡Importante! Estas funcionalidades adicionales pueden darte puntos extra según el criterio del instructor **Iván Malavé Fierro**.

Optimización y Buenas Prácticas

Comprimir imágenes y assets

Performance

- Usar caché Redis para consultas
- frecuentes Implementar lazy loading en
- Vue.js
- Optimizar consultas con Eloquent

Código Limpio

- Comentarios descriptivos en
- español Nombres de variables claros
- Funciones pequeñas y específicas

Consistencia en el estilo

Seguridad

- Validar todas las entradas de usuario
- Usar HTTPS en producción
- Implementar rate limiting
- Sanitizar datos antes de mostrar

Cobertura mínima del 80%

- Pruebas para casos críticos
- Tests de integración con APIs
- Pruebas de rendimiento

Testing

Criterios de Evaluación Detallados

Funcionalidad

Características Avanzadas

Código Fuente

Manual Técnico

Manual Usuario

Pruebas Calidad

Investigación

Presentación

0 8 16 24

El instructor **Iván Malavé Fierro** evaluará cada aspecto cuidadosamente. Asegúrate de cumplir con todos los requisitos y agregar valor con funcionalidades innovadoras.

Errores Frecuentes y Cómo Evitarlos

Documentación Incompleta

Error: Entregar manuales muy básicos o sin capturas de pantalla

Solución: Dedica tiempo suficiente a documentar. Incluye ejemplos visuales y casos de uso reales

para otros desarrolladores

Solución: Comenta líneas complejas en español, explica la lógica de negocio

Error: No incluir suficientes pruebas automatizadas

Solución: Escribe pruebas para funciones críticas, especialmente APIs y modelos

Demo que no Funciona

Código Sin Comentarios

Error: Código difícil de entender

Error: Fallos técnicos durante la presentación

Solución: Prueba tu demo múltiples veces, ten un plan B con

Plagio de Código

Falta de Pruebas

capturas de pantalla

Error: Copiar código sin entender o sin citar fuentes

Solución: Adapta código externo, entiéndelo completamente y cita las fuentes

Checklist Final del Proyecto

Verificación Completa Antes de

Entregar

Funcionalidad Básica

Sistema CRUD de equipos funciona
Simulación de sensores operativa
Dashboard muestra datos en tiempo real
Sistema de alertas funciona
Integración con IA operativa

Características Avanzadas

Autenticación implementada
WebSockets para tiempo real
Modelo de IA entrenado
APIs REST documentadas
Funcionalidades extras agregadas

Documentación

Manual técnico completo
Manual de usuario con capturas Proyecto de investigación
Código bien comentado
README.md actualizado

¡Perfecto! Si puedes marcar todos estos elementos, tu proyecto está listo para impresionar al instructor **Iván Malavé Fierro**.

Glosario de Términos Técnicos

API Interfaz de Programación de Aplicaciones - permite comunicación entre sistemas **CRUD** Create, Read, Update, Delete -

operaciones básicas de datos **Dashboard** Panel de control visual con métricas e información importante **Eloquent** ORM de Laravel para interactuar con base de datos

Framework Conjunto de herramientas y librerías para desarrollo rápido **IoT** Internet of Things - dispositivos conectados que envían datos **Laravel** Framework de PHP para desarrollo web moderno

Machine Learning Inteligencia Artificial que aprende de datos históricos **MQTT** Protocolo de comunicación para dispositivos IoT

Middleware Software que conecta diferentes aplicaciones o servicios **MVC** Modelo-Vista-Controlador - patrón de arquitectura de software **ORM** Object-Relational Mapping - mapeo entre objetos y base de datos **REST** Estilo de arquitectura para APIs web

Sensor Dispositivo que mide condiciones físicas (temperatura, vibración) **Vue.js** Framework de JavaScript para interfaces de usuario interactivas **WebSocket** Protocolo de comunicación bidireccional en tiempo real

Recursos Adicionales para Aprender Más

Documentación Oficial

Laravel: laravel.com/docs - Documentación completa y actualizada

Vue.js: vuejs.org/guide - Guías paso a paso para Vue 3

Scikit-learn: scikit-learn.org - Tutoriales de Machine Learning

Cursos en Línea

Platzi: Cursos de Laravel, Vue.js y PHP

Coursera: Especializaciones en Machine Learning

YouTube: Canales como "Código Facilito"

y "EDteam"

Comunidades

Stack Overflow: Preguntas y respuestas técnicas

GitHub: Proyectos open source para inspirarse

Laravel Colombia: Comunidad local de

desarrolladores

Mini-Quiz de Verificación

Evalúa tu Comprensión del Proyecto

Pregunta 1: ¿Cuál es el propósito principal de un sistema de mantenimiento predictivo?

- a) Reparar máquinas después de que se dañen
- b) Predecir fallos antes de que ocurran
- c) Reemplazar técnicos humanos
- d) Reducir el costo de los sensores

Pregunta 2: ¿Qué tecnología usamos para la comunicación en tiempo real?

- a) HTTP tradicional
- b) WebSockets
- c) Email
- d) SMS

Pregunta 3: ¿Cuál es la función principal de un modelo de Machine Learning en este proyecto?

- a) Crear interfaces bonitas
- b) Gestionar usuarios
- c) Analizar datos y predecir fallos
- d) Enviar emails

Pregunta 4: ¿Qué significa CRUD en programación?

- a) Crear, Revisar, Actualizar, Desarrollar
- b) Create, Read, Update, Delete
- c) Código, Rápido, Útil, Dinámico
- d) Control, Registro, Usuario, Datos

Pregunta 5: ¿Cuántos puntos máximos se pueden obtener en "Características Avanzadas"?

- a) 10 puntos
- b) 20 puntos
- c) 15 puntos
- d) 25 puntos

Respuestas del Mini-Quiz

Respuesta 1: b)

Predecir fallos antes de que ocurran - Este es el objetivo principal del mantenimiento predictivo: anticiparse a los problemas.

Create, Read, Update, Delete - Las cuatro operaciones básicas en

Respuesta 4: b)

Respuesta 2: b)

WebSockets - Permiten comunicación bidireccional en tiempo real entre el servidor y el cliente.

cualquier sistema de datos.

15 puntos - Según la tabla de criterios de evaluación del instructor **Iván Malavé Fierro**.

Respuesta 5: c)

Respuesta 3: c)

Analizar datos y predecir fallos - El modelo de IA aprende patrones de los datos históricos para hacer predicciones.

¡Excelente! Si respondiste correctamente 4 o 5 preguntas, tienes una comprensión sólida del proyecto.

¡Felicitaciones!

Has Completado la Guía del Proyecto

Ahora tienes todos los conocimientos y herramientas necesarias para desarrollar un **Sistema de Mantenimiento Predictivo con IA**

que impresionará al instructor **Iván Malavé Fierro** y te permitirá obtener una excelente calificación.

60

Pasos Completados

Desde configuración hasta presentación final

7

Tecnologías Dominadas

PHP, Laravel, Vue.js, Python, IA,
MySQL, WebSockets

100

Puntos Posibles

Conoces todos los criterios de evaluación

Recuerda: la clave del éxito está en la **práctica constante**, la **documentación detallada** y la **innovación** en las características adicionales.
¡Tu futuro como tecnólogo en Análisis y Desarrollo de Software comienza ahora!

¡Mucho éxito en tu proyecto! Con dedicación y siguiendo esta guía, estás preparado para crear algo excepcional.

Comandos Laravel Esenciales - Guía Práctica Completa

Contenido detallado sobre comandos Laravel fundamentales con ejemplos prácticos:

1. Comandos de Artisan Básicos

`php artisan list` – Muestra todos los comandos disponibles

`php artisan help migrate` - Ayuda específica para cualquier comando

`php artisan tinker` - Consola interactiva para probar código PHP

2. Comandos de Base de Datos

`php artisan migrate:status` - Ver estado de migraciones

`php artisan migrate:rollback` - Deshacer última migración

`php artisan migrate:fresh --seed` - Recrear BD con datos de prueba

`php artisan db:seed` - Ejecutar seeders para datos de prueba

3. Comandos de Caché y Optimización

`php artisan cache:clear` - Limpiar caché de aplicación

`php artisan config:cache` - Cachear configuración para producción

`php artisan route:cache` - Cachear rutas para mejor rendimiento

`php artisan view:clear` - Limpiar vistas compiladas

Ejemplo Práctico:

```
# Secuencia típica de desarrollo
```

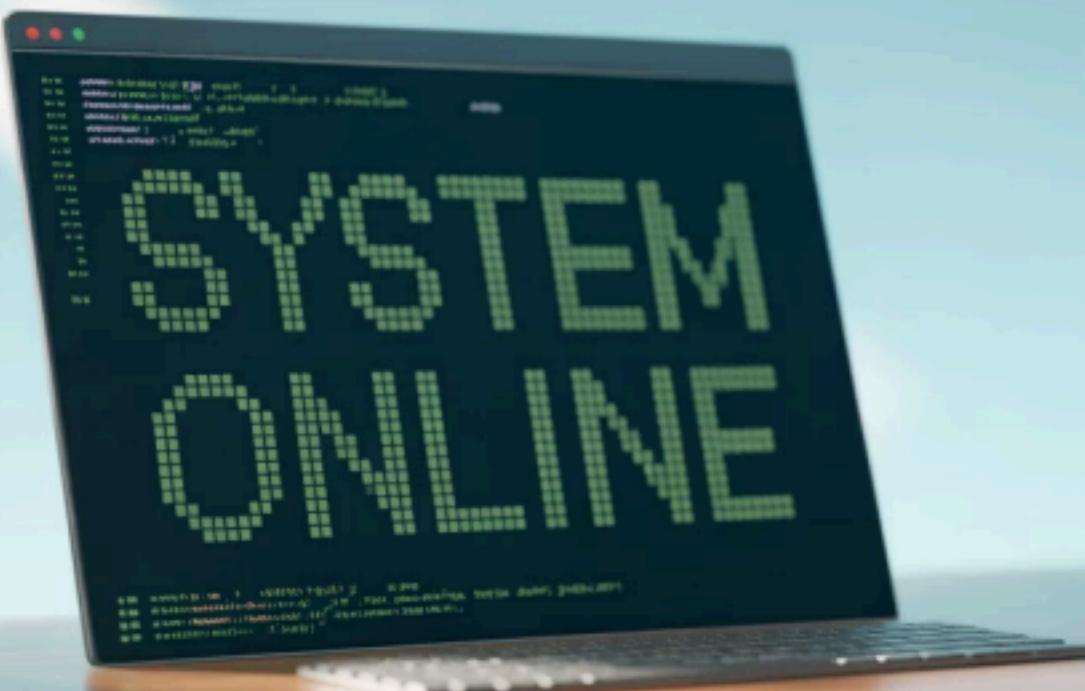
```
php artisan make:model Sensor -mcr
```

```
php artisan migrate
```

```
php artisan tinker
```

```
>>> Sensor::factory()->count(5)->create()
```

SYSTEM
ONLINE



Factories y Seeders - Datos de Prueba Realistas

Contenido sobre cómo crear datos de prueba profesionales:

¿Qué son los Factories?

Los factories son "fábricas" que generan datos falsos pero realistas para probar tu aplicación. Es como tener un robot que crea equipos, sensores y lecturas automáticamente.

Creando Factory para Equipos:

```
// database/factories/EquipoFactory.php
public function definition()
{
    return [
        'nombre' => $this->faker->randomElement([
            'Bomba Centrífuga #1', 'Compresor Principal',
            'Motor Eléctrico A-001', 'Turbina Generadora'
        ]),
        'tipo' => $this->faker->randomElement([
            'Bomba', 'Compresor', 'Motor', 'Turbina', 'Ventilador'
        ]),
        'ubicacion' => $this->faker->randomElement([
            'Sala de Control', 'Caja de Compresión',
            'Planta de Tratamiento', 'Almacén de Suministros',
            'Caja de Seguridad', 'Sala de Mantenimiento'
        ])
    ];
}
```

```
'Planta Norte', 'Área de Producción', 'Sala de Máquinas',
'Sector Industrial', 'Zona de Compresores'
]),
'descripcion' => $this->faker->sentence(10),
'fecha_instalacion' => $this->faker->dateTimeBetween('-5 years', '-1 year'),
'estado' => $this->faker->randomElement(['activo', 'mantenimiento', 'inactivo']),
'horas_operacion' => $this->faker->numberBetween(1000, 50000)
];
}
```

Comandos Prácticos:

```
php artisan make:factory EquipoFactory --model=Equipo
php artisan make:seeder EquipoSeeder
php artisan db:seed --class=EquipoSeeder
```

Ejemplo de Uso en Tinker:

```
// Crear 10 equipos con sensores
Equipo::factory()
->count(10)
->has(Sensor::factory()->count(3))
->create();
```



Eloquent Avanzado - Relaciones y Consultas Poderosas

Contenido detallado sobre cómo aprovechar al máximo las relaciones y consultas en Laravel Eloquent:

Relaciones Complejas Explicadas Paso a Paso:

1. Relación Uno a Muchos (hasMany)

Un equipo tiene muchos sensores:

```
// En el modelo Equipo
public function sensores()
{
    return $this->hasMany(Sensor::class);
}
```

Uso práctico:

```
$equipo = Equipo::find(1);
$sensores = $equipo->sensores; // Obtiene todos los sensores del equipo
```

2. Relación Muchos a Muchos (belongsToMany)

Un técnico puede mantener varios equipos, y un equipo puede ser mantenido por varios técnicos:

```
// En el modelo Equipo
public function tecnicos()
{
    return $this->belongsToMany(Tecnico::class, 'equipo_tecnico')
        ->withPivot('fecha_asignacion', 'tipo_mantenimiento')
        ->withTimestamps();
}
```

3. Consultas con Eager Loading (Carga Anticipada)

Evita el problema N+1 cargando relaciones de una vez:

MALO - Hace muchas consultas

```
$equipos = Equipo::all();
foreach($equipos as $equipo) {
    echo $equipo->sensores->count(); // Nueva consulta por cada equipo
}
```

BUENO - Una sola consulta optimizada

```
$equipos = Equipo::with(['sensores', 'alertas'])->get();
```

4. Consultas Avanzadas con Where

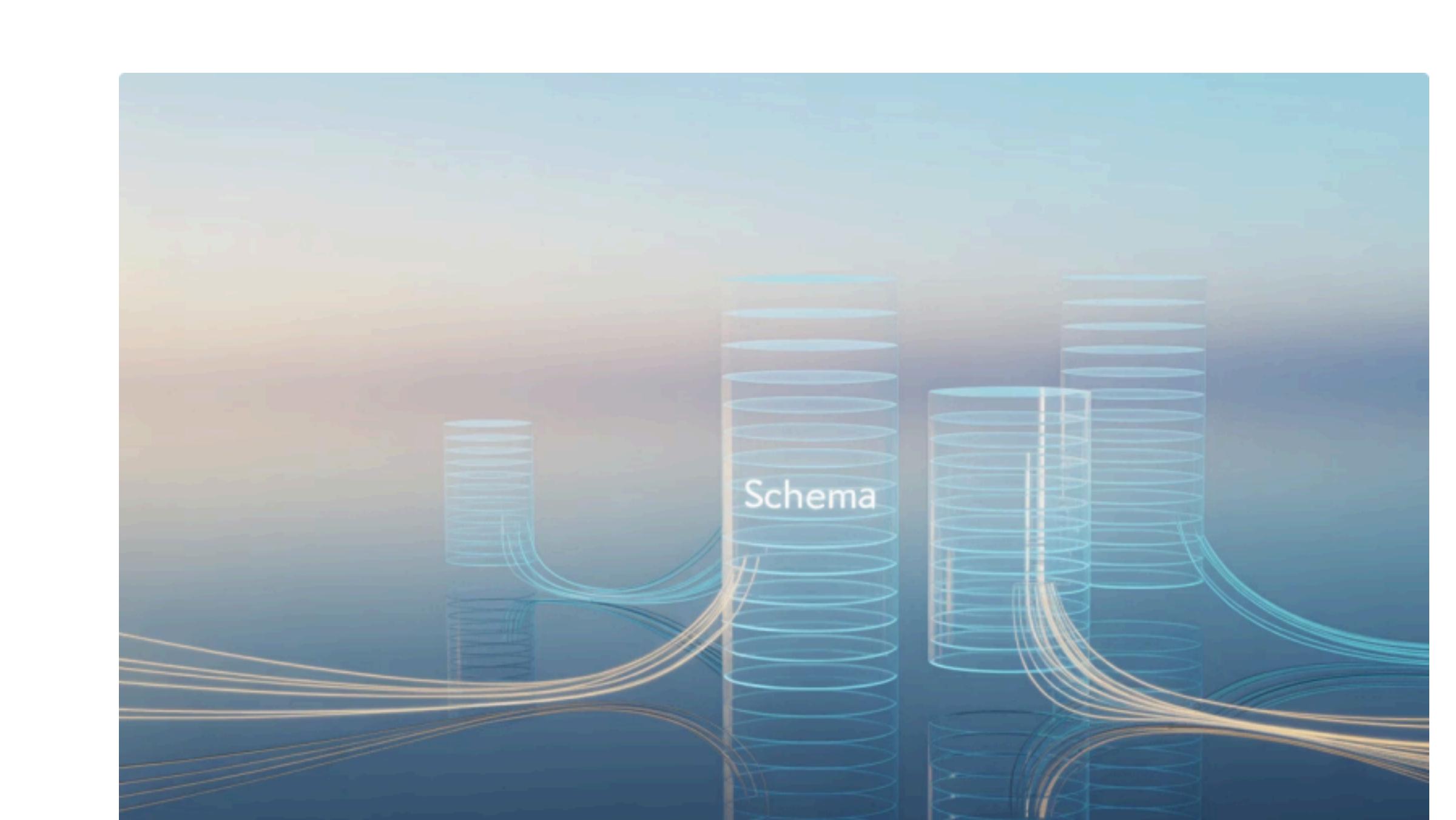
Equipos con sensores que han dado alertas en los últimos 7 días

```
$equiposCriticos = Equipo::whereHas('sensores.lecturas', function($query) {
    $query->where('estado', 'critico')
        ->where('created_at', '>=', now()->subDays(7));
})->with('sensores')->get();
```

Comandos Útiles:

php artisan tinker - Probar consultas en tiempo real

DB::enableQueryLog() - Ver qué consultas SQL se ejecutan



A landscape photograph of a bridge over water at sunset. The sky is a gradient from blue to orange. In the foreground, several glowing, translucent blue and yellow lines represent data flow, originating from the base of three large, vertical cylinders. The cylinders are semi-transparent, showing concentric rings, and are positioned along a curved path. The word "Schema" is written vertically in white text on the middle cylinder. The overall image has a futuristic, digital feel.

Schema

Validaciones Avanzadas con Form Requests

¿Qué son los Form Requests?

Los Form Requests son clases especiales que validan automáticamente los datos que llegan a tu aplicación. Es como tener un "portero" que revisa que toda la información esté correcta antes de procesarla.

Creando un Form Request:

```
php artisan make:request StoreEquipoRequest  
php artisan make:request UpdateEquipoRequest
```

Ejemplo Completo de Validación:

```
// app/Http/Requests/StoreEquipoRequest.php  
class StoreEquipoRequest extends FormRequest  
{  
    public function authorize()  
    {  
        return true; // Permitir a todos los usuarios autenticados  
    }  
  
    public function rules()  
    {  
        return [ // Reglas de validación  
    }  
}
```

```
{  
    return [  
        'nombre' => 'required|string|max:100|unique:equipos,nombre',  
        'tipo' => 'required|in:Bomba,Compresor,Motor,Turbina,Ventilador',  
        'ubicacion' => 'required|string|max:100',  
        'descripcion' => 'nullable|string|max:500',  
        'fecha_instalacion' => 'required|date|before_or_equal:today',  
        'estado' => 'required|in:activo,mantenimiento,inactivo',  
        'horas_operacion' => 'required|integer|min:0|max:100000'  
    ];  
}
```

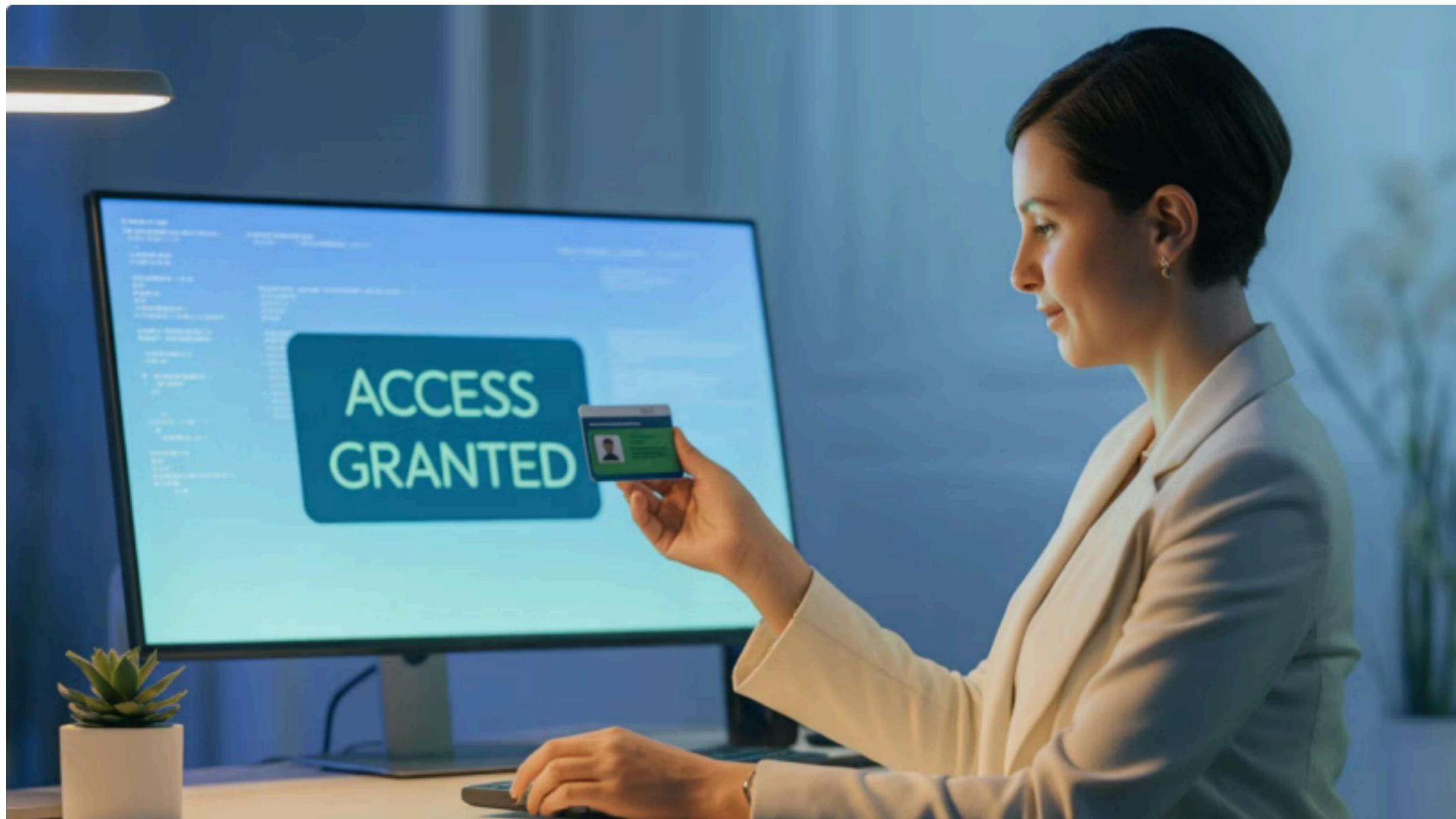
```
public function messages()  
{  
    return [  
        'nombre.required' => 'El nombre del equipo es obligatorio',  
        'nombre.unique' => 'Ya existe un equipo con este nombre',  
        'tipo.in' => 'El tipo debe ser: Bomba, Compresor, Motor, Turbina o Ventilador',  
        'fecha_instalacion.before_or_equal' => 'La fecha no puede ser futura',  
        'horas_operacion.max' => 'Las horas no pueden exceder 100,000'  
    ];  
}
```

Usando en el Controlador:

```
public function store(StoreEquipoRequest $request)
{
    // Los datos ya están validados automáticamente
    $equipo = Equipo::create($request->validated());
    return response()->json($equipo, 201);
}
```

Validaciones Personalizadas:

```
// Validar que el equipo no tenga más de 10 sensores
'sensores' => [
    'array',
    'max:10',
    function ($attribute, $value, $fail) {
        if (count($value) > 10) {
            $fail('Un equipo no puede tener más de 10 sensores.');
        }
    }
]
```



Jobs y Queues - Procesamiento en Segundo Plano

Los Jobs son tareas que se ejecutan en segundo plano sin que el usuario tenga que esperar. Es como enviar una carta por correo: la entregas y sigues con tu vida mientras el cartero la lleva.

Casos de Uso Prácticos:

Enviar emails de alerta

Procesar datos de sensores

Generar reportes PDF

Hacer predicciones con IA

Limpiar archivos temporales

Configuración Básica:

```
# Configurar la cola en .env  
QUEUE_CONNECTION=database
```

```
# Crear tabla de jobs  
php artisan queue:table  
php artisan migrate
```

Creando Jobs Específicos:

```
php artisan make:job ProcesarDatosSensor  
php artisan make:job EnviarAlertaEmail  
php artisan make:job GenerarReporteMantenimiento
```

Ejemplo: Job para Procesar Datos de Sensores

```
// app/Jobs/ProcesarDatosSensor.php  
class ProcesarDatosSensor implements ShouldQueue  
{  
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;  
  
    protected $sensorId;  
    protected $datos;  
  
    public function __construct($sensorId, $datos)
```

```
{  
    $this->sensorId = $sensorId;  
    $this->datos = $datos;  
}  
  
public function handle()  
{  
    $sensor = Sensor::find($this->sensorId);  
  
    // Procesar cada lectura  
    foreach ($this->datos as $lectura) {  
        // Crear registro en base de datos  
        Lectura::create([  
            'sensor_id' => $this->sensorId,  
            'valor' => $lectura['valor'],  
            'timestamp' => $lectura['timestamp'],  
            'estado' => $this->evaluarEstado($lectura['valor'], $sensor)  
        ]);  
  
        // Si el valor es crítico, crear alerta  
        if ($this->esCritico($lectura['valor'], $sensor)) {  
            EnviarAlertaEmail::dispatch($sensor->equipo, $lectura);  
        }  
    }  
}
```

```
}

private function evaluarEstado($valor, $sensor)
{
    if ($valor > $sensor->limite_critico) return 'critico';
    if ($valor > $sensor->limite_advertencia) return 'advertencia';
    return 'normal';
}
```

Despachando Jobs:

```
// En tu controlador o donde necesites
ProcesarDatosSensor::dispatch($sensorId, $datosRecibidos);

// Con delay (retraso)
EnviarAlertaEmail::dispatch($equipo, $alerta)->delay(now()->addMinutes(5));

// Con prioridad
GenerarReporteMantenimiento::dispatch($equipold)->onQueue('high-priority');
```

Comandos para Manejar Colas:

```
# Ejecutar worker (procesar jobs)  
php artisan queue:work
```

```
# Ver jobs fallidos  
php artisan queue:failed
```

```
# Reintentar job fallido  
php artisan queue:retry 1
```

```
# Limpiar jobs fallidos  
php artisan queue:flush
```



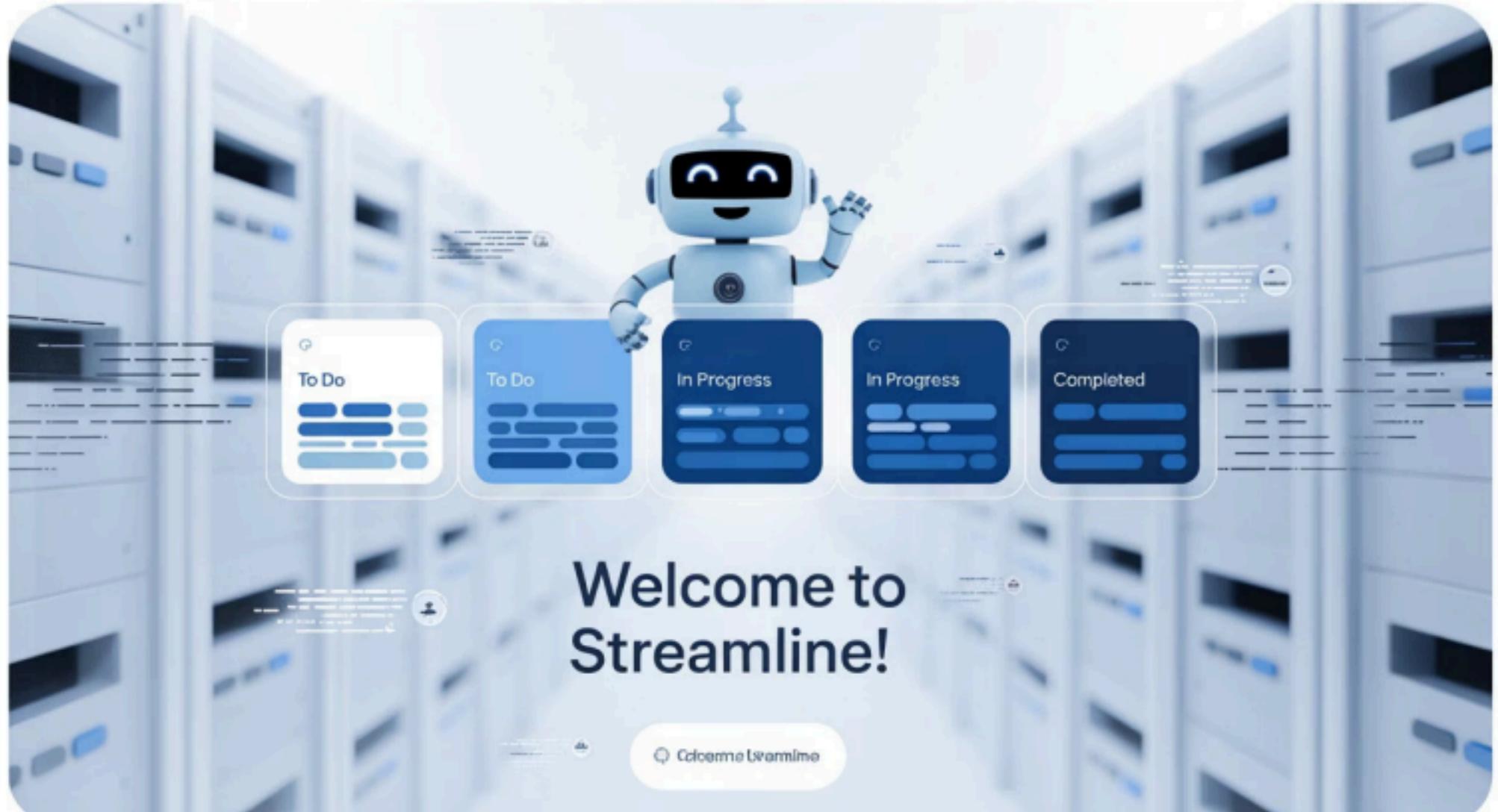
Dashboard

Projects

Tasks

Reports

Bryan Ronald



Middleware Personalizado - Control de Acceso Avanzado

¿Qué es un Middleware?

Un middleware es como un "filtro" o "checkpoint" que revisa cada petición antes de que llegue a tu controlador. Es como el guardia de seguridad de un edificio que verifica tu identificación.

Casos de Uso Prácticos:

Verificar si el usuario es administrador

Registrar todas las acciones en un log

Limitar el número de peticiones por minuto

Verificar si el equipo pertenece al usuario

Convertir datos automáticamente

Creando Middleware Personalizado:

```
php artisan make:middleware VerificarPropietarioEquipo  
php artisan make:middleware LogearAccionesUsuario  
php artisan make:middleware SoloAdministradores
```

Ejemplo: Middleware para Verificar Propietario

```
// app/Http/Middleware/VerificarPropietarioEquipo.php  
class VerificarPropietarioEquipo  
{  
    public function handle(Request $request, Closure $next)  
    {  
        $equipold = $request->route('equipo'); // Obtener ID de la URL  
        $usuario = auth()->user();  
  
        // Verificar si el equipo existe  
        $equipo = Equipo::find($equipold);
```

```
if (!$equipo) {
    return response()->json(['error' => 'Equipo no encontrado'], 404);
}

// Verificar si el usuario es propietario o administrador
if ($equipo->user_id !== $usuario->id && !$usuario->es_admin) {
    return response()->json([
        'error' => 'No tienes permiso para acceder a este equipo'
    ], 403);
}

// Si todo está bien, continuar
return $next($request);
}
```

Ejemplo: Middleware para Logging

```
// app/Http/Middleware/LogearAccionesUsuario.php
class LogearAccionesUsuario
{
    public function handle(Request $request, Closure $next)
    {
        $usuario = auth()->user();
```

```
$accion = $request->method() . ' ' . $request->path();

// Registrar la acción en la base de datos
LogAccion::create([
    'user_id' => $usuario->id,
    'accion' => $accion,
    'ip' => $request->ip(),
    'user_agent' => $request->userAgent(),
    'timestamp' => now()
]);

return $next($request);
}
}
```

Registrando Middleware:

```
// app/Http/Kernel.php
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'propietario.equipo' => \App\Http\Middleware\VerificarPropietarioEquipo::class,
    'log.acciones' => \App\Http\Middleware\LogearAccionesUsuario::class,
    'solo.admin' => \App\Http\Middleware\SoloAdministradores::class,
];
```

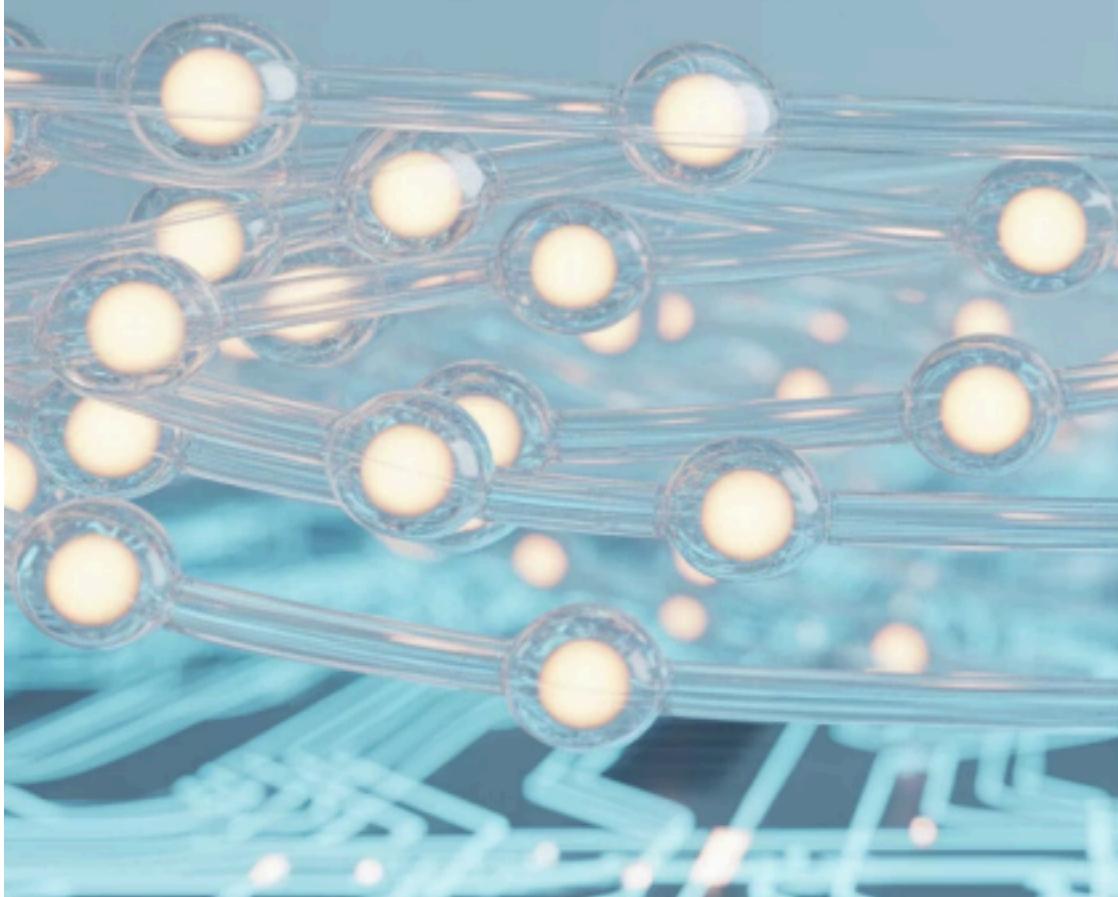
Usando Middleware en Rutas:

```
// routes/api.php
Route::middleware(['auth', 'propietario.equipo'])->group(function () {
    Route::get('/equipos/{equipo}', [EquipoController::class, 'show']);
    Route::put('/equipos/{equipo}', [EquipoController::class, 'update']);
    Route::delete('/equipos/{equipo}', [EquipoController::class, 'destroy']);
});

// Aplicar a un controlador completo
Route::middleware(['auth', 'log.acciones'])->resource('equipos', EquipoController::class);
```

Middleware con Parámetros:

```
// Middleware que acepta roles específicos
Route::middleware(['auth', 'role:admin, supervisor'])->group(function () {
    Route::get('/reportes/confidenciales', [ReporteController::class, 'confidenciales']);
});
```



Eventos y Listeners - Sistema de Notificaciones Inteligente

¿Qué son los Eventos?

Los eventos son "señales" que se disparan cuando algo importante sucede en tu aplicación. Los listeners son "oyentes" que reaccionan a esas señales. Es como un sistema de alarma: cuando se detecta humo (evento), se activa la alarma (listener).

Casos de Uso en Mantenimiento Predictivo:

Cuando se crea un equipo ³ Enviar email de bienvenida

Cuando un sensor da lectura crítica ³ Crear alerta + Enviar SMS

Cuando se programa mantenimiento ³ Notificar técnicos

Cuando se completa mantenimiento ³ Actualizar historial

Creando Eventos y Listeners:

```
php artisan make:event EquipoCreado
```

```
php artisan make:event LecturaCriticaDetectada
```

```
php artisan make:event MantenimientoCompletado
```

```
php artisan make:listener EnviarEmailBienvenida --event=EquipoCreado
php artisan make:listener CrearAlertaCritica --event=LecturaCriticaDetectada
php artisan make:listener NotificarTecnicos --event=LecturaCriticaDetectada
```

Ejemplo: Evento de Lectura Crítica

```
// app/Events/LecturaCriticaDetectada.php
class LecturaCriticaDetectada
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $lectura;
    public $sensor;
    public $equipo;

    public function __construct(Lectura $lectura)
    {
        $this->lectura = $lectura;
        $this->sensor = $lectura->sensor;
        $this->equipo = $lectura->sensor->equipo;
    }

    public function broadcastOn()
    {
```

```
{  
    return new PrivateChannel('equipo.' . $this->equipo->id);  
}  
}
```

Ejemplo: Listener para Crear Alerta

```
// app/Listeners/CrearAlertaCritica.php  
class CrearAlertaCritica  
{  
    public function handle(LecturaCriticaDetectada $event)  
    {  
        // Crear alerta en base de datos  
        $alerta = Alerta::create([  
            'equipo_id' => $event->equipo->id,  
            'sensor_id' => $event->sensor->id,  
            'tipo' => 'critica',  
            'mensaje' => "Valor crítico detectado: {$event->lectura->valor} en {$event->sensor->tipo}",  
            'nivel' => 'alto',  
            'estado' => 'pendiente'  
        ]);  
  
        // Disparar evento de alerta creada para más acciones  
        AlertaCreada::dispatch($alerta);  
    }  
}
```

```
 }  
 }
```

Registrando Eventos y Listeners:

```
// app/Providers/EventServiceProvider.php  
protected $listen = [  
    EquipoCreado::class => [  
        EnviarEmailBienvenida::class,  
        CrearCarpetaDocumentos::class,  
    ],  
  
    LecturaCriticaDetectada::class => [  
        CrearAlertaCritica::class,  
        NotificarTecnicos::class,  
        EnviarSMSEmergencia::class,  
    ],  
  
    MantenimientoCompletado::class => [  
        ActualizarHistorial::class,  
        EnviarReporteCompletion::class,  
        ProgramarProximoMantenimiento::class,  
    ],  
];
```

Disparando Eventos:

```
// En tu controlador o modelo
public function store(Request $request)
{
    $equipo = Equipo::create($request->all());

    // Disparar evento
    EquipoCreado::dispatch($equipo);

    return response()->json($equipo, 201);
}
```

```
// En el simulador de sensores
if ($lectura->estado === 'critico') {
    LecturaCriticaDetectada::dispatch($lectura);
}
```

Listeners con Colas (Asíncronos):

```
class EnviarEmailBienvenida implements ShouldQueue
{
```

```
use InteractsWithQueue;

public function handle(EquipoCreado $event)
{
    Mail::to($event->equipo->responsable_email)
        ->send(new EquipoCreadoMail($event->equipo));
}
```



SYSTEM ALERT



SYSTEM ALERT



SLIM MESSAGE



WHINCTING DUAIN

Comandos Personalizados - Automatización Avanzada

¿Qué son los Comandos Artisan Personalizados?

Son herramientas que creas para automatizar tareas específicas de tu proyecto. Es como crear tus propios "botones mágicos" que ejecutan procesos complejos con un solo comando.

Casos de Uso Prácticos:

Generar reportes automáticos

Limpiar datos antiguos

Sincronizar con sistemas externos

Ejecutar diagnósticos del sistema

Importar/exportar datos masivos

Creando Comandos Personalizados:

- php artisan make:command GenerarReporteMantenimiento
- php artisan make:command LimpiarDatosAntiguos
- php artisan make:command DiagnosticoSistema

```
- php artisan make:command ImportarEquiposCSV
```

Estructura Básica de un Comando:

Los comandos tienen una signature (firma) que define cómo se llaman y qué parámetros aceptan, y un método handle() que contiene la lógica. **Ejemplo de Comando para Diagnóstico:**

Un comando que verifica el estado del sistema:

Conexión a base de datos

Número de sensores activos

Espacio disponible en disco

Estado de las colas de trabajo

Programando Comandos Automáticos:

Puedes programar comandos para que se ejecuten automáticamente:

Reportes diarios a las 6 AM

Limpieza de datos antiguos los domingos

Diagnósticos cada hora

Ejecutando Comandos:

- php artisan reporte:mantenimiento - Comando básico
- php artisan reporte:mantenimiento --equipo=1 --formato=excel - Con opciones
- php artisan help reporte:mantenimiento - Ver ayuda del comando

Comandos con Progreso Visual:

Los comandos pueden mostrar barras de progreso y mensajes coloridos para una mejor experiencia de usuario.

Interactividad:

Los comandos pueden hacer preguntas al usuario, confirmar acciones y solicitar información adicional durante la ejecución.



Vue.js Avanzado - Componentes Reactivos y Composables

¿Qué son los Composables en Vue 3?

Los composable son funciones reutilizables que encapsulan lógica reactiva. Es como tener "superpoderes" que puedes usar en cualquier componente para manejar datos, hacer peticiones HTTP, o gestionar el estado.

Ejemplo: Composable para Datos de Sensores

```
// composable/useSensores.js
import { ref, reactive, computed, onMounted } from 'vue'
import axios from 'axios'

export function useSensores() {
  // Estado reactivo
  const sensores = ref([])
  const loading = ref(false)
  const error = ref(null)
```

```
// Estado para filtros
const filtros = reactive({
  tipo: '',
  estado: '',
  equipo_id: null
})

// Computed para sensores filtrados
const sensoresFiltrados = computed(() => {
  return sensores.value.filter(sensor => {
    if (filtros.tipo && sensor.tipo !== filtros.tipo) return false
    if (filtros.estado && sensor.estado !== filtros.estado) return false
    if (filtros.equipo_id && sensor.equipo_id !== filtros.equipo_id) return false
    return true
  })
})

// Función para obtener sensores
const obtenerSensores = async () => {
  loading.value = true
  error.value = null

  try {
```

```
const response = await axios.get('/api/sensores')
sensores.value = response.data
} catch (err) {
error.value = 'Error al cargar sensores: ' + err.message
} finally {
loading.value = false
}
}

// Función para crear sensor
const crearSensor = async (datosSensor) => {
try {
const response = await axios.post('/api/sensores', datosSensor)
sensores.value.push(response.data)
return response.data
} catch (err) {
error.value = 'Error al crear sensor: ' + err.message
throw err
}
}

// Cargar datos al montar
onMounted(() => {
obtenerSensores()
```

```
})
// Retornar todo lo que queremos exponer
return {
sensores,
sensoresFiltrados,
loading,
error,
filtros,
obtenerSensores,
crearSensor
}
}
```

Usando el Composable en un Componente:

```
// components/ListaSensores.vue
<template>
<div class="lista-sensores">
<div class="filtros">
<select v-model="filtros.tipo">
<option value="">Todos los tipos</option>
<option value="temperatura">Temperatura</option>
<option value="vibracion">Vibración</option>
```

```
<option value="presion">Presión</option>
</select>
```

```
<select v-model="filtros.estado">
<option value="">Todos los estados</option>
<option value="activo">Activo</option>
<option value="inactivo">Inactivo</option>
</select>
</div>
```

```
<div v-if="loading" class="loading">
Cargando sensores...
</div>
```

```
<div v-else-if="error" class="error">
{{ error }}
</div>
```

```
<div v-else class="sensores-grid">
<div
v-for="sensor in sensoresFiltrados"
:key="sensor.id"
class="sensor-card"
:class="{ 'critico': sensor.estado === 'critico' }"
```

```
>
<h3>{{ sensor.nombre }}</h3>
<p>Tipo: {{ sensor.tipo }}</p>
<p>Estado: {{ sensor.estado }}</p>
<p>Última lectura: {{ sensor.ultima_lectura }}</p>
</div>
</div>
</div>
</template>

<script setup>
import { useSensores } from '@/composables/useSensores'

// Usar el composable
const {
  sensoresFiltrados,
  loading,
  error,
  filtros,
  obtenerSensores
} = useSensores()

// Función para refrescar datos
const refrescar = () => {
```

```
obtenerSensores()
}
</script>
```

Composable para WebSockets en Tiempo Real:

```
// composables/useWebSocket.js
import { ref, onMounted, onUnmounted } from 'vue'

export function useWebSocket(url) {
  const socket = ref(null)
  const isConnected = ref(false)
  const messages = ref([])
  const error = ref(null)

  const connect = () => {
    try {
      socket.value = new WebSocket(url)

      socket.value.onopen = () => {
        isConnected.value = true
        error.value = null
        console.log('WebSocket conectado')
      }
    
```

```
socket.value.onmessage = (event) => {
    const data = JSON.parse(event.data)
    messages.value.push(data)

    // Limitar mensajes en memoria
    if (messages.value.length > 100) {
        messages.value.shift()
    }
}

socket.value.onclose = () => {
    isConnected.value = false
    console.log('WebSocket desconectado')
}

socket.value.onerror = (err) => {
    error.value = 'Error de WebSocket: ' + err.message
}

} catch (err) {
    error.value = 'Error al conectar WebSocket: ' + err.message
}
}
```

```
const disconnect = () => {
if (socket.value) {
socket.value.close()
}
}

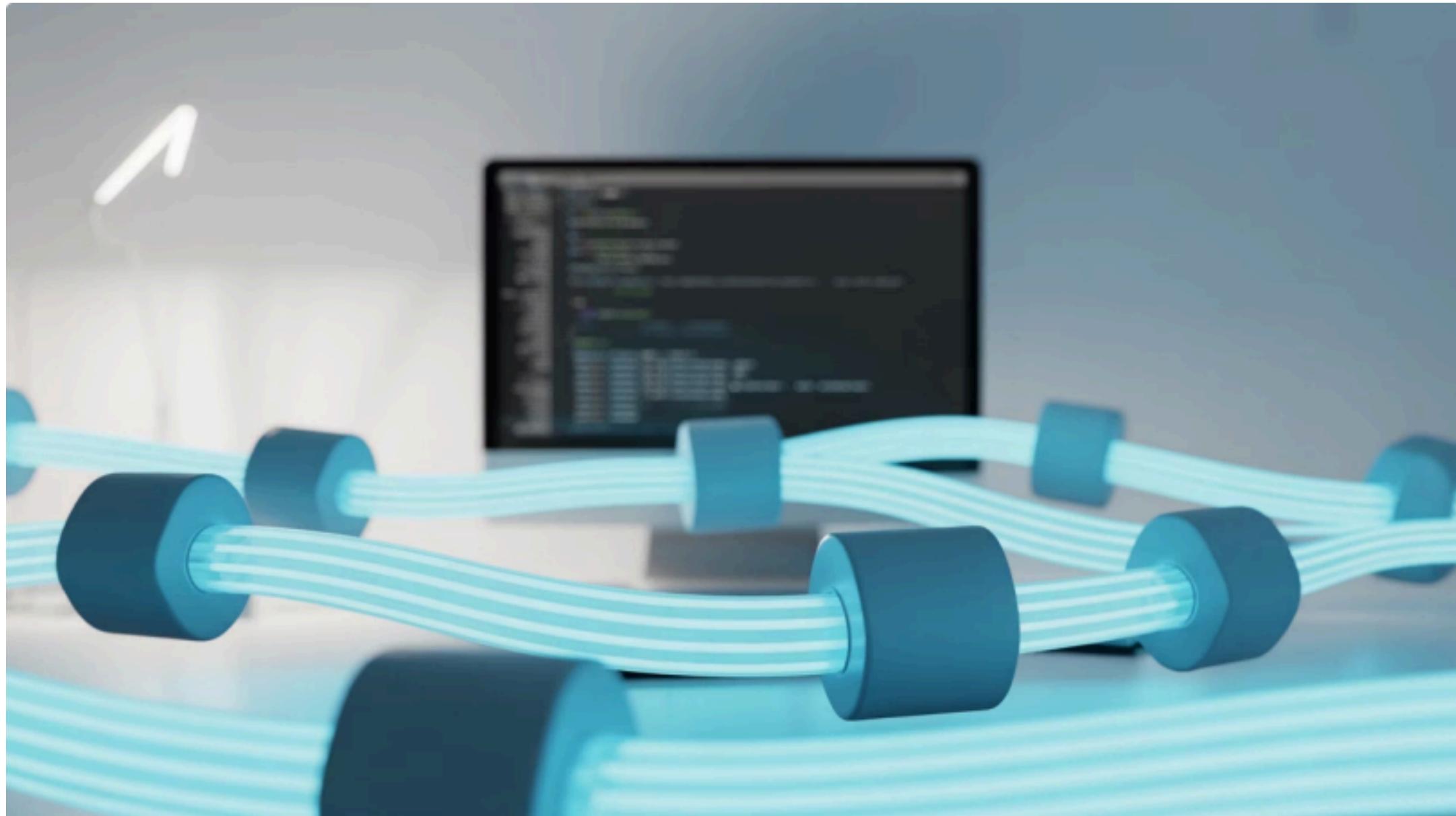
const sendMessage = (message) => {
if (socket.value && isConnected.value) {
socket.value.send(JSON.stringify(message))
}
}

onMounted(() => {
connect()
})

onUnmounted(() => {
disconnect()
})

return {
isConnected,
messages,
```

```
error,  
sendMessage,  
connect,  
disconnect  
}  
}
```



Algoritmos de Machine Learning Explicados Paso a Paso

¿Por qué Random Forest para Mantenimiento Predictivo?

Random Forest es como tener un "comité de expertos" donde cada experto (árbol) da su opinión sobre si una máquina va a fallar, y luego se toma la decisión por mayoría. Es perfecto para nuestro caso porque:

Maneja datos mixtos:

Temperatura (números), tipo de equipo (categorías)

Resistente a ruido:

Si un sensor da datos erróneos, no afecta mucho

Explica decisiones:

Podemos saber qué factores son más importantes

Preparación de Datos - Conceptos Clave:

Extracción de Características:

Promedio de valores de sensores

Desviación estándar (qué tan variables son los datos) Valores máximos y mínimos

Rango de operación

Coeficiente de variación

Entrenamiento del Modelo:

Usar 100 árboles de decisión

Cada árbol aprende de una muestra diferente de datos Combinar predicciones de todos los árboles

Codificación de Variables:

Convertir texto ("Bomba", "Compresor") a números Escalar valores para que estén en el mismo rango Dividir datos en entrenamiento (80%) y prueba (20%)

Evaluación del Modelo:

Precisión: ¿Cuántas predicciones fueron correctas? Recall: ¿Detectamos todos los fallos reales? F1-Score: Balance entre precisión y recall

Haciendo Predicciones en Tiempo Real:

Entrada del Modelo:

- Tipo de equipo
- Horas de operación
- Datos actuales de sensores
(temperatura, vibración,

presión)

Salida del Modelo:

- Probabilidad de fallo (0-100%) Nivel de riesgo (BAJO, MEDIO, ALTO)
- Recomendación específica
- Factores más importantes en la decisión

Interpretación de Resultados:

Probabilidad > 70%: Mantenimiento inmediato

Probabilidad 40-70%: Monitoreo cercano

Probabilidad < 40%: Operación normal

Forest: No necesita muchos datos para empezar

Ventajas de Random

Funciona bien con datos faltantes

Proporciona importancia de características

Difícil de sobreajustar (overfitting)

Mejoras Continuas: Reentrenar el

modelo mensualmente

Agregar nuevos tipos de sensores Incluir datos de

mantenimiento histórico Ajustar umbrales según la

experiencia