Ø=ÜØ Proyecto React - Código Fuente (Frontend - Parte 1)

Ruta: src/App.css

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}

.view-navigation {
  display: flex;
  gap: 10px;
  padding: 0 25px 20px;
  border-bottom: 1px solid var(--border-color);
  margin-bottom: 20px;
}

.nav-btn {
  padding: 12px 24px;
  border: 2px solid var(--border-color);
  background: white;
  border-radius: 8px;
  cursor: pointer;
  font-size: 1rem;
  font-weight: 500;
  transition: all 0.2s;
}

.nav-btn.active {
  background: var(--primary-color);
  color: white;
  border-color: var(--primary-color);
}

.nav-btn:hover:not(.active) {
  border-color: var(--primary-color);
  color: var(--primary-color);
}

/* ... (estilos de .logo, .card, etc. que ya tienes) ... */

/* REEMPLAZA tus estilos de .view-navigation y .nav-btn por estos: */

.view-navigation {
  display: flex;
  gap: 5px; /* Pequeño espacio entre botones */
  padding: 5px; /* Padding interno para el contenedor */
```

```
  border-radius: 10px; /* Bordes redondeados para el contenedor */
  background-color: var(--light-color, #f8fafc); /* Color de fondo claro */
  border: 1px solid var(--border-color, #e2e8f0); /* Borde sutil */
  margin: 0 25px 20px; /* Márgenes (reemplaza el padding original) */
}

.nav-btn {
  /* Estilos para alinear icono y texto */
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 8px; /* Espacio entre icono y texto */

  /* Apariencia */
  flex: 1; /* Hace que ambos botones ocupen el mismo espacio */
  padding: 10px 20px;
  border: none; /* Quitamos el borde feo */
  border-radius: 8px; /* Borde redondeado para el botón */
  background: transparent; /* Fondo transparente por defecto */
  color: #64748b; /* Color de texto gris (el mismo de .model-card p) */
  font-size: 0.95rem; /* Tamaño de fuente ligeramente ajustado */
  font-weight: 600; /* Un poco más de peso */
  cursor: pointer;
  transition: all 0.2s ease-in-out;
}

.nav-btn:hover:not(.active) {
  background: #f1f5f9; /* Un fondo muy sutil al pasar el mouse (color de chat-messages-track) */
  color: var(--dark-color, #1e293b);
}

.nav-btn.active {
  background: white; /* El botón activo es blanco */
  color: var(--primary-color, #6366f1); /* Texto con tu color primario */
  /* Sombra sutil (la misma de .model-card) */
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  transform: scale(1.02); /* Efecto ligero de "pop" */
}
```

Ruta: src/App.jsx

```
import React, { useState } from 'react'
import Header from './components/Header'
import ModelsInfo from './components/ModelsInfo'
import Chat from './components/Chat'
import Dashboard from './components/Dashboard'
// Importa los iconos que usaremos
import { MessageSquare, LayoutDashboard } from 'lucide-react'

function App() {
  const [currentView, setCurrentView] = useState('chat')

  return (
    <div className="app">
      <div className="container">
        <Header />
        <ModelsInfo />
        {/* Navegación entre Chat y Dashboard */}
        <div className="view-navigation">
          <button
            className={`nav-btn ${currentView === 'chat' ? 'active' : ''}`}
            onClick={() => setCurrentView('chat')}
          >
            {/* Icono y texto añadidos */}
            <MessageSquare size={18} />
            Chat
          </button>
          <button
            className={`nav-btn ${currentView === 'dashboard' ? 'active' : ''}`}
            onClick={() => setCurrentView('dashboard')}
          >
            {/* Icono y texto añadidos */}
            <LayoutDashboard size={18} />
            Dashboard
          </button>
        </div>

        {currentView === 'chat' ? <Chat /> : <Dashboard />}
      </div>
    </div>
  )
}

export default App
```

Ruta: src/components/Chat.jsx

```
import React, { useState, useRef, useEffect } from 'react'
import { Send, Monitor } from 'lucide-react'
import Message from './Message'
import { sendMessage, startMonitoring, stopMonitoring, getMonitoringStatus } from '../services/api'

const Chat = () => {
    const [messages, setMessages] = useState([{
        id: 1,
        text: 'Hola! Soy tu asistente MAAS. ¿En qué puedo ayudarte hoy?',
        isUser: false,
        timestamp: new Date()
    }])
    const [inputMessage, setInputMessage] = useState('')
    const [isLoading, setIsLoading] = useState(false)
    const [monitoringStatus, setMonitoringStatus] = useState(false)
    const chatContainerRef = useRef(null)

    useEffect(() => {
        checkMonitoringStatus()
        scrollToBottom()
    }, [])

    useEffect(() => {
        scrollToBottom()
    }, [messages])

    const scrollToBottom = () => {
        if (chatContainerRef.current) {
            chatContainerRef.current.scrollTop = chatContainerRef.current.scrollHeight
        }
    }

    const checkMonitoringStatus = async () => {
        try {
            const status = await getMonitoringStatus()
            setMonitoringStatus(status.monitoreo_activo)
        } catch (error) {
            console.error('Error checking monitoring status:', error)
        }
    }

    const handleSend = async () => {
        if (!inputMessage.trim() || isLoading) return

        const userMessage = {
            id: Date.now(),
            text: inputMessage.trim(),
            isUser: true,
            timestamp: new Date()
        }

        setInputMessage('')
        setIsLoading(true)

        // Add user message immediately
        setMessages(prev => [...prev, userMessage])

        try {
            console.log('Ø=Ý  DEBUG Frontend: Enviando mensaje al backend...')

            // Send to backend
            const response = await sendMessage(userMessage.text)
            console.log('  DEBUG Frontend: Respuesta recibida:', response)

            // Verificar que la respuesta tenga el formato correcto
            if (!response || response.respuesta === undefined) {
                throw new Error('La respuesta del servidor no tiene el formato esperado')
            }

            // Add bot response
            const botMessage = {
                id: Date.now() + 1,
                text: response.respuesta,
                isUser: false,
                timestamp: new Date()
            }

            setMessages(prev => [...prev, botMessage])

        } catch (error) {
            console.error('L DEBUG Frontend: Error en handleSend:', error)
```

```
        // Mostrar el error real en lugar del mensaje genérico
        let errorText = error.message
        if (error.message.includes('Failed to fetch') || error.message.includes('NetworkError')) {
            errorText = 'No se pudo conectar con el servidor. Verifica que el backend esté ejecutándose.'
        }

        const errorMessage = {
            id: Date.now() + 1,
            text: `'L ${errorText}`,
            isUser: false,
            timestamp: new Date()
        }
        setMessages(prev => [...prev, errorMessage])
    } finally {
        setIsLoading(false)
    }
}

const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault()
        handleSend()
    }
}

const handleMonitoringToggle = async () => {
    try {
        if (monitoringStatus) {
            await stopMonitoring()
            setMonitoringStatus(false)
            const stopMessage = {
                id: Date.now(),
                text: 'Ø=Ý  Monitoreo de máquinas detenido',
                isUser: false,
                timestamp: new Date()
            }
            setMessages(prev => [...prev, stopMessage])
        } else {
            await startMonitoring()
            setMonitoringStatus(true)
            const startMessage = {
                id: Date.now(),
                text: 'Ø=Ý  Monitoreo de máquinas iniciado. Recibirás notificaciones de cambios.',
                isUser: false,
                timestamp: new Date()
            }
            setMessages(prev => [...prev, startMessage])
        }
    } catch (error) {
        console.error('Error toggling monitoring:', error)
        const errorMessage = {
            id: Date.now(),
            text: `'L Error al cambiar el estado del monitoreo: ${error.message}`,
            isUser: false,
            timestamp: new Date()
        }
        setMessages(prev => [...prev, errorMessage])
    }
}

return (
    <div className="chat-container">
        <div className="chat-controls">
            <button
                className={`monitor-btn ${monitoringStatus ? 'active' : ''}`}
                onClick={handleMonitoringToggle}
                type="button"
            >
                <Monitor size={18} />
                {monitoringStatus ? 'Detener Monitoreo' : 'Iniciar Monitoreo'}
            </button>
        </div>
        <div className="chat-messages" ref={chatContainerRef}>
            {messages.map((message) => (
                <Message
                    key={message.id}
                    message={message.text}
                    isUser={message.isUser}
                    timestamp={message.timestamp}
                />
            ))}
            {isLoading && (
                <div className="message bot-message">
```

```
                <div className="message-avatar">
                    <Monitor size={16} />
                </div>
                <div className="message-content">
                    <div className="typing-indicator">
                        <span></span>
                        <span></span>
                        <span></span>
                    </div>
                </div>
            </div>
        )}
      </div>
      <div className="chat-input">
        <input
          type="text"
          value={inputMessage}
          onChange={(e) => setInputMessage(e.target.value)}
          onKeyPress={handleKeyPress}
          placeholder="Escribe tu mensaje..."
          disabled={isLoading}
        />
        <button
          onClick={handleSend}
          disabled={isLoading || !inputMessage.trim()}
          type="button"
        >
          <Send size={18} />
        </button>
      </div>
    </div>
  )
}

export default Chat
```

Ruta: src/components/Dashboard.jsx

```
import React, { useState, useEffect } from 'react'
import { Server, Cpu, HardDrive, Network, AlertCircle, Play, PowerOff, RefreshCw, AlertTriangle } from 'lucide-react'

const Dashboard = () => {
  const [metricas, setMetricas] = useState(null)
  const [maquinas, setMaquinas] = useState([])
  const [alertas, setAlertas] = useState([])
  const [loading, setLoading] = useState(true)
  const [autoRefresh, setAutoRefresh] = useState(false)
  const [error, setError] = useState(null)

  const cargarDatos = async () => {
    try {
      setError(null)

      // Cargar solo las métricas principales, no los endpoints individuales
      const metricasResponse = await fetch('/api/dashboard/metricas')
      if (!metricasResponse.ok) {
        throw new Error(`Error ${metricasResponse.status}: ${metricasResponse.statusText}`)
      }

      const metricasData = await metricasResponse.json()

      setMetricas(metricasData)
      setMaquinas(Array.isArray(metricasData.maquinas) ? metricasData.maquinas : [])
      setAlertas(Array.isArray(metricasData.alertas) ? metricasData.alertas : [])

    } catch (error) {
      console.error('Error general cargando datos:', error)
      setError(`Error al cargar los datos: ${error.message}`)
      setMaquinas([])
      setAlertas([])
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    cargarDatos()

    if (autoRefresh) {
      const interval = setInterval(cargarDatos, 10000)
      return () => clearInterval(interval)
    }
  }, [autoRefresh])

  const getSaludColor = (salud) => {
    switch (salud) {
      case 'healthy': return 'text-green-500'
      case 'warning': return 'text-yellow-500'
      case 'critical': return 'text-red-500'
      default: return 'text-gray-500'
    }
  }

  const getSaludBgColor = (salud) => {
    switch (salud) {
      case 'healthy': return 'bg-green-50 border-green-200'
      case 'warning': return 'bg-yellow-50 border-yellow-200'
      case 'critical': return 'bg-red-50 border-red-200'
      default: return 'bg-gray-50 border-gray-200'
    }
  }

  const getPowerStateIcon = (powerState) => {
    return powerState === 'on' ?
      <Play size={16} className="text-green-500" /> :
      <PowerOff size={16} className="text-red-500" />
  }

  if (loading) {
    return (
      <div className="dashboard-loading">
        <RefreshCw size={32} className="animate-spin text-primary-color" />
        <p>Cargando dashboard...</p>
      </div>
    )
  }

  if (error) {
    return (
```

```
    <div className="dashboard-error">
      <AlertTriangle size={48} className="text-red-500 mb-4" />
      <h2>Error al cargar el dashboard</h2>
      <p>{error}</p>
      <button onClick={cargarDatos} className="btn-refresh mt-4">
        <RefreshCw size={16} />
        Reintentar
      </button>
    </div>
  )
}

return (
  <div className="dashboard">
    {/* Header del Dashboard */}
    <div className="dashboard-header">
      <h1>Dashboard MAAS</h1>
      <div className="dashboard-controls">
        <button
          onClick={cargarDatos}
          className="btn-refresh"
        >
          <RefreshCw size={16} />
          Actualizar
        </button>
        <label className="auto-refresh-toggle">
          <input
            type="checkbox"
            checked={autoRefresh}
            onChange={(e) => setAutoRefresh(e.target.checked)}
          />
          Auto-refresh cada 10s
        </label>
      </div>
    </div>

    {/* Estado de conexión */}
    {metricas?.error && (
      <div className="alerta-card warning">
        <AlertTriangle size={20} />
        <div className="alerta-content">
          <strong>Advertencia</strong>
          <p>Algunos datos pueden estar incompletos: {metricas.error}</p>
        </div>
      </div>
    )}

    {/* Métricas Principales */}
    {metricas?.resumen && Object.keys(metricas.resumen).length > 0 && (
      <div className="metricas-grid">
        <div className="metrica-card">
          <div className="metrica-icon">
            <Server size={24} />
          </div>
          <div className="metrica-content">
            <h3>Total Máquinas</h3>
            <p className="metrica-valor">{metricas.resumen.total_maquinas || 0}</p>
            <div className="metrica-sub">
              <span className="text-green-500">Ø=ßâ {metricas.resumen.maquinas_encendidas || 0}</span>
              <span className="text-red-500">Ø=Ý4 {metricas.resumen.maquinas_apagadas || 0}</span>
            </div>
          </div>
        </div>

        <div className="metrica-card">
          <div className="metrica-icon">
            <Cpu size={24} />
          </div>
          <div className="metrica-content">
            <h3>CPU Total</h3>
            <p className="metrica-valor">{metricas.resumen.total_cpu_cores || 0} núcleos</p>
          </div>
        </div>

        <div className="metrica-card">

          <div className="metrica-content">
            <h3>RAM Total</h3>
            <p className="metrica-valor">{metricas.resumen.total_ram_gb || 0} GB</p>
          </div>
        </div>

        <div className="metrica-card">
          <div className="metrica-icon">
```

```
        <HardDrive size={24} />
      </div>
      <div className="metrica-content">
        <h3>Almacenamiento</h3>
        <p className="metrica-valor">{metricas.resumen.total_almacenamiento_gb || 0} GB</p>
      </div>
    </div>

    <div className="metrica-card">
      <div className="metrica-icon">
        <Network size={24} />
      </div>
      <div className="metrica-content">
        <h3>Subredes</h3>
        <p className="metrica-valor">{metricas.red?.total_subredes || 0}</p>
      </div>
    </div>

    <div className="metrica-card">
      <div className="metrica-icon">
        <AlertCircle size={24} />
      </div>
      <div className="metrica-content">
        <h3>Alertas Activas</h3>
        <p className="metrica-valor">{alertas.length}</p>
        <div className="metrica-sub">
          {alertas.filter(a => a.tipo === 'critical').length} críticas
        </div>
      </div>
    </div>
  </div>
)}

{/* Alertas */}
{alertas.length > 0 && (
  <div className="alertas-section">
    <h2>Alertas Activas</h2>
    <div className="alertas-grid">
      {alertas.map((alerta, index) => (
        <div key={index} className={`alerta-card ${alerta.tipo}`}>
          <AlertCircle size={20} />
          <div className="alerta-content">
            <strong>{alerta.maquina}</strong>
            <p>{alerta.mensaje}</p>
            <small>{new Date(alerta.timestamp).toLocaleString()}</small>
          </div>
        </div>
      ))}
    </div>
  </div>
)}

{/* Lista de Máquinas */}
<div className="maquinas-section">
  <h2>Máquinas ({maquinas.length})</h2>

  {maquinas.length === 0 ? (
    <div className="no-data">
      <Server size={48} className="text-gray-400" />
      <p>No se encontraron máquinas</p>
    </div>
  ) : (
    <div className="maquinas-grid">
      {maquinas.map((maquina, index) => (
        <div key={maquina.system_id || index} className={`maquina-card ${getSaludBgColor(maquina.salud)}`}>
          <div className="maquina-header">
            <div className="maquina-info">
              <h3>{maquina.hostname}</h3>
              <span className={`salud-badge ${getSaludColor(maquina.salud)}`}>
                {maquina.salud}
              </span>
            </div>
            <div className="maquina-estado">
              {getPowerStateIcon(maquina.power_state)}
              <span>{maquina.power_state === 'on' ? 'Encendida' : 'Apagada'}</span>
            </div>
          </div>

          <div className="maquina-details">
            <div className="detail-row">
              <span>IP:</span>
              <span>{maquina.ip}</span>
            </div>
            <div className="detail-row">
```

```
                <span>SO:</span>
                <span>{maquina.so}</span>
              </div>
              <div className="detail-row">
                <span>CPU:</span>
                <span>{maquina.cpu_cores} núcleos</span>
              </div>
              <div className="detail-row">
                <span>RAM:</span>
                <span>{maquina.ram_gb} GB</span>
              </div>
              <div className="detail-row">
                <span>Almacenamiento:</span>
                <span>{maquina.almacenamiento_gb} GB</span>
              </div>
              <div className="detail-row">
                <span>Zona:</span>
                <span>{maquina.zona}</span>
              </div>
            </div>
          </div>
        ))}
      </div>
    ))}
  </div>
 </div>
 )
}

export default Dashboard
```

Ruta: src/components/Header.jsx

```
import React from 'react'
import { Cpu, MessageCircle } from 'lucide-react'

const Header = () => {
  return (
    <header className="header">
      <div className="header-content">
        <div className="logo">
          <Cpu className="logo-icon" />
          <h1>Chat MAAS</h1>
        </div>
        <p>Model as a Service - Chatea con nuestros modelos de IA</p>
      </div>
    </header>
  )
}

export default Header
```

Ruta: src/components/Message.jsx

```
import React from 'react'
import { User, Bot } from 'lucide-react'

const Message = ({ message, isUser, timestamp }) => {
  const formatTime = (date) => {
    return date.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' })
  }

  return (
    <div className={`message ${isUser ? 'user-message' : 'bot-message'}`}>
      <div className="message-avatar">
        {isUser ? <User size={16} /> : <Bot size={16} />}
      </div>
      <div className="message-content">
        <div className="message-text">{message}</div>
        <div className="message-time">
          {timestamp ? formatTime(timestamp) : formatTime(new Date())}
        </div>
      </div>
    </div>
  )
}

export default Message
```