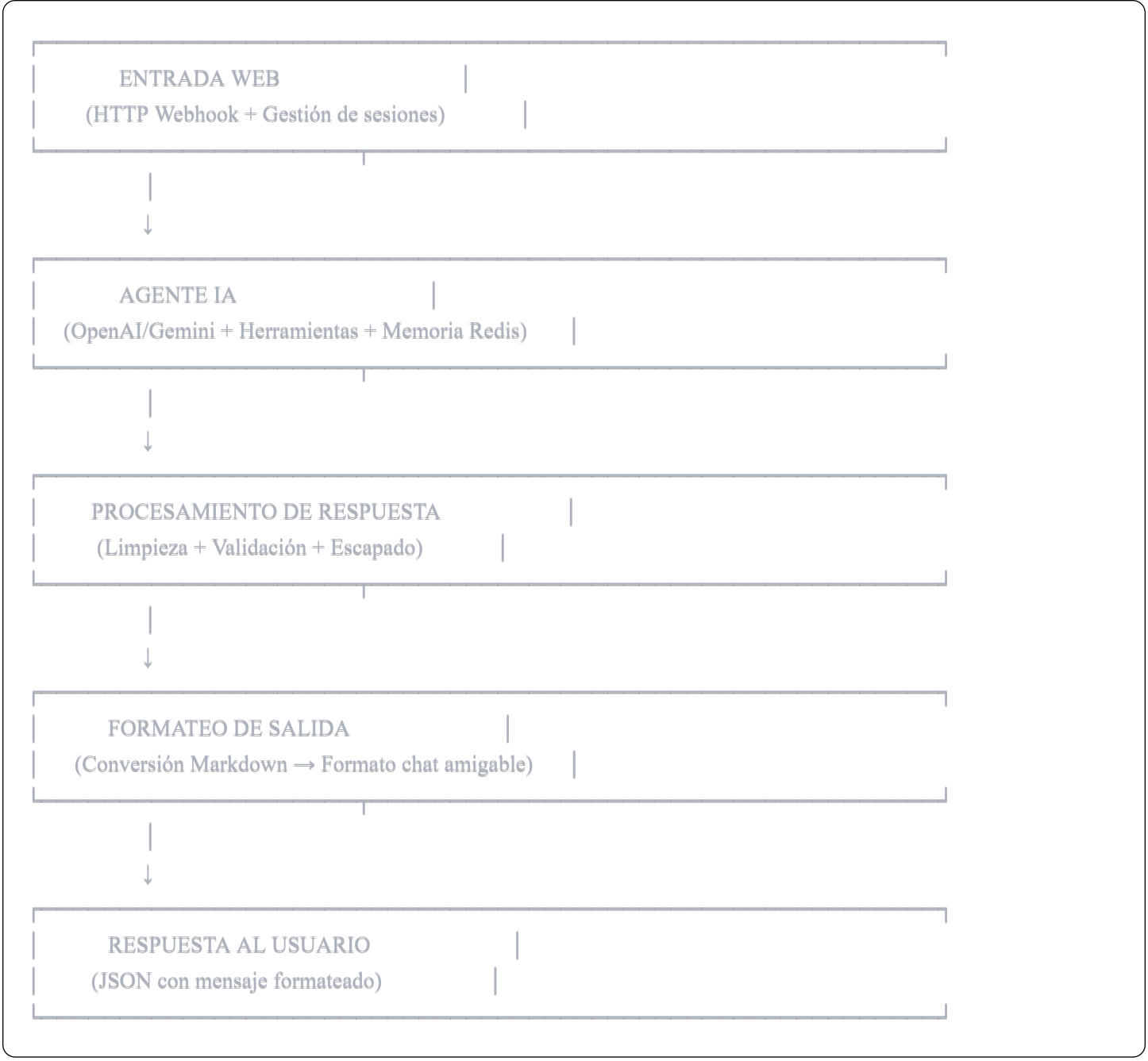
Documentación Técnica - Web Agent N8N

Esta guía explica en detalle cómo funciona el agente web, su arquitectura, componentes y flujo de datos.

Índice

Arquitectura general

El workflow está dividido en **5 módulos principales** que trabajan de forma coordinada:



Flujo de datos

Paso 1: Entrada del usuario (HTTP Webhook)

```json
{
  "chatInput": "Texto del mensaje del usuario",
  "sessionId": "identificador-unico-sesion"
}
```

Nodo: Sin nodo webhook visible (se activa mediante trigger HTTP)

Función:

- Recibe mensajes desde la interfaz web vía HTTP POST
- Identifica la sesión del usuario mediante `sessionId`
- Pasa los datos al siguiente nodo

Paso 2: Preparación de entrada

Nodo: `Entrada agente`

Función: Normaliza los datos de entrada para el agente IA:

```javascript
{
  "content": mensaje_del_usuario,
  "chatId": sessionId,
  "messageId": mensaje_del_usuario,
  "sessionId": sessionId
}
```

Campos clave:

- `content`: El mensaje que procesará el agente
- `chatId`: Identificador único para la memoria Redis
- `sessionId`: Identificador de la sesión web

Paso 3: Procesamiento del agente IA

Nodo: `AI Agent1`

Este es el cerebro del sistema. Integra:

a) Modelo de lenguaje principal

- **Primary:** Google Gemini (temperatura: 0.1, topP: 0.9)
- **Fallback:** OpenAI GPT-4.1 (temperatura: 0.1, topP: 0.9)

b) Memoria conversacional

- **Redis Chat Memory** con ventana de contexto de 10 mensajes
- Permite recordar el historial de cada usuario

c) Herramientas disponibles

1. **Calculator**: Cálculos matemáticos
2. **Think**: Razonamiento interno
3. **MCP Calendario**: Gestión de agenda
4. **MCP Correo**: Envío de emails
5. **AddLEADS**: Registro en Google Sheets

d) Prompt del sistema

Define la personalidad, reglas de negocio y comportamiento del agente.

Salida:

```json
{
  "output": "Respuesta generada por el agente con posible formato markdown"
}
```

Paso 4: Limpieza inicial

Nodo: Code in JavaScript

Función: Primera limpieza de la respuesta del agente:

```javascript
// Elimina etiquetas HTML
.replace(/<[^>]+>/g, '')

// Convierte saltos de línea a espacios
.replace(/\r?\n|\r/g, ' ')

// Convierte tabs a espacios
.replace(/\t+/g, ' ')

// Elimina caracteres especiales excepto letras, números y puntuación básica
.replace(/[^\p{L}\p{N}.:;?!()\"\s]/gu, '')

// Normaliza espacios múltiples
.replace(/\s+/g, ' ')

// Escapa caracteres para JSON
```

Salida:

```json
{
  "output": "Texto limpio y escapado para JSON"
}
```

Paso 5: Validación de salida

Nodo: SALIDA AGENTE

Función: Asegura que la salida tiene el formato correcto antes del formateo final:

```json
{
  "output": "Texto validado"
}
```

Paso 6: Limpieza de markdown

Nodo: LIMPIAR SALIDA

Función: Segunda fase de limpieza específica para markdown:

```javascript
// Elimina etiquetas HTML residuales
.replace(/<[^>]+>/g, '')

// Normaliza saltos de línea
.replace(/\r?\n|\r/g, ' ')

// Normaliza tabs
.replace(/\t+/g, ' ')
```

Salida:

```json
{
  "cleaned": "Texto sin markdown"
}
```

Paso 7: Formateo conversacional

Nodo: Basic LLM Chain

Modelo: OpenAI GPT-4.1-mini

Prompt:

```
Tu objetivo consiste en formatear un mensaje markdown
en un formato más bonito para un usuario. Elimina todos
los caracteres típicos del lenguaje markdown y adapta
la respuesta a un chat.

En tu salida jamás digas que has adaptado el texto a
un formato para chat. Simplemente devuelve la misma
información pero en el formato especificado.
```

Función:

- Convierte markdown a texto plano amigable
- Mantiene la información pero la presenta de forma conversacional
- Elimina símbolos técnicos (#, *, **, -, etc.)

Salida:

```json
{
  "text": "Mensaje formateado para chat"
}
```

Paso 8: Respuesta final

Nodo: RESPUESTA

Función: Prepara la respuesta final que se enviará al usuario:

```json
{
  "message": "Texto final que verá el usuario"
}
```

Esta respuesta se envía de vuelta al cliente web vía HTTP.

Componentes principales

1. AI Agent (Agente IA)

Configuración del agente

Nodo: AI Agent1

Tipo: @n8n/n8n-nodes-langchain.agent

Parámetros clave:

```json
{
  "promptType": "define",
  "text": "={{ $json.content }}",
  "options": {
    "systemMessage": "Prompt completo del sistema..."
  }
}
```

Estructura del prompt del sistema

El prompt define:

1. **Rol y objetivo**
   - Asistente profesional de atención al cliente
   - Experto en IA y automatización
   - Servicio en Sevilla y provincia
   - Comunicación en español de España
2. **Presentación única**

```
"Hola, soy el agente virtual. ¿En qué puedo ayudarte?"
```

- Solo se presenta una vez por conversación nueva

3. **Proceso de agendamiento**
    - Solicitud de datos: nombre, email, teléfono, servicio
    - Verificación de disponibilidad en calendario
    - Confirmación con el usuario
    - Creación del evento
    - Envío de email de confirmación
    - Registro automático del lead
4. **Herramientas y su uso**
    - Conocimiento: Información sobre servicios
    - MCP Calendario: Gestión de agenda (verificar, crear, actualizar, eliminar)
    - MCP Correo: Envío de confirmaciones
    - AddLEADS: Registro en base de datos
    - Calculator: Operaciones matemáticas
    - Think: Razonamiento interno
5. **Restricciones y comportamiento**
    - Horario: Lunes a viernes, 10:00-14:00 y 15:00-19:00
    - Tono: Alegre, simpático y profesional
    - No responde temas fuera de contexto
    - Usa memoria para evitar repetir preguntas
    - Nunca menciona el uso de herramientas
    - Requiere confirmación para acciones destructivas

---

2. Sistema de memoria

Redis Chat Memory

Nodo: `Redis Chat Memory`

Tipo: `@n8n/n8n-nodes-langchain.memoryRedisChat`

Configuración:

```json
{
  "contextWindowLength": 10
}
```

Funcionamiento

Redis almacena el historial conversacional usando el `sessionId` como clave:

```
Key: chat_memory:{sessionId}
Value: [
  {
    "role": "user",
    "content": "Mensaje del usuario"
  },
  {
    "role": "assistant",
    "content": "Respuesta del agente"
  },
  ...
]
```

Ventana de contexto:

- Mantiene los últimos 10 mensajes (5 intercambios)
- Los mensajes más antiguos se eliminan automáticamente
- Permite mantener contexto sin sobrecargar el prompt

Ventajas de Redis

1. **Persistencia:** Las conversaciones sobreviven reinicios del workflow
2. **Velocidad:** Acceso ultra-rápido a la memoria
3. **Escalabilidad:** Soporta múltiples usuarios simultáneos
4. **Aislamiento:** Cada sesión tiene su propia memoria

---

3. Motor de IA

El workflow usa dos modelos de IA en diferentes etapas:

A) Agente principal

Modelos disponibles:

1. **Google Gemini (Principal)**
   - **Nodo:** `Google Gemini Chat Model`
   - **Modelo:** `gemini-pro`
   - **Temperatura:** 0.1 (respuestas consistentes)
   - **Top P:** 0.9 (diversidad controlada)
   - **Uso:** Agente conversacional principal
2. **OpenAI GPT-4.1 (Fallback)**
   - **Nodo:** `OpenAI Chat Model4`
   - **Modelo:** `gpt-4.1`
   - **Temperatura:** 0.1
   - **Top P:** 0.9
   - **Frequency Penalty:** 0
   - **Presence Penalty:** 0
   - **Uso:** Respaldo cuando Gemini no está disponible

¿Por qué dos modelos?

- **Redundancia:** Si Gemini falla, GPT-4.1 toma el control
- **Flexibilidad:** Puedes cambiar entre modelos según necesidades
- **Coste:** Gemini puede ser más económico para uso continuo

B) Formateo de respuestas

OpenAI GPT-4.1-mini

- **Nodo:** `OpenAI Chat Model1`
- **Modelo:** `gpt-4.1-mini`
- **Uso:** Conversión de markdown a texto amigable
- **Ventaja:** Modelo más ligero y rápido para tareas simples

---

4. Herramientas del agente

El agente tiene acceso a 5 herramientas principales:

A) Calculator

Nodo: `Calculator1`

Tipo: `@n8n/n8n-nodes-langchain.toolCalculator`

Función:

- Realiza operaciones matemáticas básicas
- Ejemplo: "¿Cuánto es 25% de 1500?"

Uso interno:

```
Calculator(1500 * 0.25) → 375
```

B) Think

Nodo: `Think1`

Tipo: `@n8n/n8n-nodes-langchain.toolThink`

Función:

- Permite al agente "pensar en voz alta"
- Razonamiento paso a paso para problemas complejos
- Mejora la calidad de las respuestas

Ejemplo interno:

```
Think: El usuario quiere agendar una cita pero no ha
dado su email. Debo solicitarlo antes de proceder.
```

C) MCP Calendario

Nodo: `MCP Calendario1`

Tipo: `@n8n/n8n-nodes-langchain.mcpClientTool`

Configuración:

```json
{
  "endpointUrl": "https://tu-servidor-mcp/calendario",
  "serverTransport": "httpStreamable",
  "authentication": "headerAuth"
}
```

Funciones disponibles:

1. **Get All Events (Obtener eventos)**

json

```
{
 "action": "get_events",
 "params": {
  "start_date": "2024-12-01T10:00:00Z",
  "end_date": "2024-12-01T11:00:00Z"
 }
}
```

2. **Create Event (Crear evento)**

json

```
{
 "action": "create_event",
 "params": {
  "summary": "Cita con Juan Pérez",
  "start": "2024-12-01T10:00:00Z",
  "end": "2024-12-01T11:00:00Z",
  "description": "Consulta sobre automatización"
 }
}
```

3. **Update Event (Actualizar evento)**

json

```
{
 "action": "update_event",
 "params": {
  "event_id": "evt_123",
  "summary": "Cita modificada"
 }
}
```

4. **Delete Event (Eliminar evento)**

json

```
{
 "action": "delete_event",
 "params": {
  "event_id": "evt_123"
 }
}
```

Flujo de agendamiento:

1. Usuario solicita cita
2. Agente verifica disponibilidad con `get_events`
3. Si está libre, solicita datos del usuario
4. Crea el evento con `create_event`
5. Confirma al usuario

D) MCP Correo

Nodo: `MCP Correo1`

Tipo: `@n8n/n8n-nodes-langchain.mcpClientTool`

Configuración:

```json
{
  "endpointUrl": "https://tu-servidor-mcp/correo",
  "serverTransport": "httpStreamable",
  "authentication": "headerAuth"
}
```

Función principal: Enviar email

```json
{
  "action": "send_email",
  "params": {
    "to": "usuario@email.com",
    "subject": "Confirmación de cita",
    "body": "Tu cita está confirmada para el 1 de diciembre a las 10:00...",
    "html": "<p>Tu cita está confirmada...</p>"
  }
}
```

Uso automático:

- Se ejecuta después de crear una cita
- Envía confirmación con todos los detalles
- Incluye enlace para modificar/cancelar

E) AddLEADS (Google Sheets)

Nodo: AddLEADS

Tipo: n8n-nodes-base.googleSheetsTool

Configuración:

```json
{
  "authentication": "serviceAccount",
  "operation": "append",
  "documentId": "ID_de_tu_Google_Sheet",
  "sheetName": "Leads"
}
```

Función:

- Registra automáticamente nuevos leads
- Se ejecuta después de agendar una cita
- No requiere intervención manual

Datos registrados:

```json
{
  "Fecha": "2024-12-01 10:30:00",
  "Nombre": "Juan Pérez",
  "Email": "juan@email.com",
  "Teléfono": "+34 600 123 456",
  "Servicio": "Automatización con IA",
  "Estado": "Cita agendada"
}
```

Ventajas:

- Centraliza todos los leads
- Permite análisis posterior
- Integración con CRM si es necesario
- Nunca se menciona al usuario

Procesamiento de respuestas

Fase 1: Limpieza inicial (Code in JavaScript)

Objetivo: Eliminar elementos no deseados y preparar para JSON

```javascript
return items.map(item => {
  const originalText = item.json.output;

  // 1. Limpiezas básicas
  let cleaned = originalText
    .replace(/<[^>]+>/g, '')          // Quita HTML
    .replace(/\r?\n|\r/g, ' ')        // Saltos → espacios
    .replace(/\t+/g, ' ')             // Tabs → espacios
    .replace(/[^\p{L}\p{N}.:;?!()\"\s]/gu, '') // Solo chars válidos
    .replace(/\s+/g, ' ')             // Normaliza espacios
    .trim();                          // Quita espacios extremos

  // 2. Escapado para JSON
  cleaned = cleaned
    .replace(/\\/g, '\\\\')           // Escapa backslashes
    .replace(/"/g, '\\"')             // Escapa comillas
    .replace(/\r/g, '\\r')            // Escapa CR
    .replace(/\n/g, '\\n')            // Escapa LF
    .replace(/\t/g, '\\t');           // Escapa tabs

  return { output: cleaned };
});
```

Fase 2: Limpieza de markdown (LIMPIAR SALIDA)

Objetivo: Eliminar caracteres markdown residuales

```javascript
return items.map(item => {
  const originalText = item.json.output;

  let cleaned = originalText
    .replace(/<[^>]+>/g, '')    // HTML residual
    .replace(/\r?\n|\r/g, ' ')  // Saltos de línea
    .replace(/\t+/g, ' ');      // Tabs

  return { json: { cleaned } };
});
```

Fase 3: Formateo conversacional (Basic LLM Chain)

Objetivo: Convertir a texto natural amigable

Entrada:

```
**Nombre:** Juan Pérez
**Email:** juan@email.com
**Teléfono:** +34 600 123 456

¿Son correctos estos datos?
```

Salida:

```
Perfecto, he registrado tus datos:
- Nombre: Juan Pérez
- Email: juan@email.com
- Teléfono: +34 600 123 456

¿Confirmas que todo es correcto?
```

Transformaciones:

- `**texto**` → texto sin formato
- `#` → eliminado
- `*` → `-` o eliminado
- Listas markdown → listas naturales
- Estructura técnica → conversacional

Integración web

Endpoint del workflow

Una vez activado el workflow, n8n expone un endpoint:

```
POST http://localhost:5678/webhook/AGENTE_WEB_GITHUB
```

Request format

json

```
{
  "chatInput": "Mensaje del usuario",
  "sessionId": "unique-session-id-12345"
}
```

Response format

json

```
{
  "message": "Respuesta formateada del agente"
}
```

Ejemplo de integración con JavaScript

javascript

```javascript
class ChatWebAgent {
  constructor(webhookUrl) {
    this.webhookUrl = webhookUrl;
    this.sessionId = this.generateSessionId();
  }

  generateSessionId() {
    return 'session_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
  }

  async sendMessage(message) {
    try {
      const response = await fetch(this.webhookUrl, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          chatInput: message,
          sessionId: this.sessionId
        })
      });

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const data = await response.json();
      return data.message;
    } catch (error) {
      console.error('Error sending message:', error);
      return 'Lo siento, ha ocurrido un error. Por favor, intenta de nuevo.';
    }
  }
}

// Uso
const agent = new ChatWebAgent('http://localhost:5678/webhook/AGENTE_WEB_GITHUB');

agent.sendMessage('Hola').then(response => {
  console.log('Agente:', response);
});
```

Ejemplo con interfaz HTML

html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chat con Agente IA</title>
  <style>
    #chat-container {
      max-width: 600px;
      margin: 50px auto;
      border: 1px solid #ccc;
      border-radius: 10px;
      padding: 20px;
    }
    #messages {
      height: 400px;
      overflow-y: auto;
      border: 1px solid #eee;
      padding: 10px;
      margin-bottom: 20px;
    }
    .message {
      margin-bottom: 10px;
      padding: 8px 12px;
      border-radius: 8px;
    }
    .user-message {
      background-color: #007bff;
      color: white;
      text-align: right;
    }
    .agent-message {
      background-color: #f1f1f1;
      color: black;
    }
    #input-container {
      display: flex;
      gap: 10px;
    }
    #message-input {
      flex: 1;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 5px;
```

```
    }
    #send-button {
      padding: 10px 20px;
      background-color: #007bff;
      color: white;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }
    #send-button:hover {
      background-color: #0056b3;
    }
  </style>
</head>
<body>
  <div id="chat-container">
    <h2>Chat con Agente IA</h2>
    <div id="messages"></div>
    <div id="input-container">
      <input type="text" id="message-input" placeholder="Escribe tu mensaje...">
      <button id="send-button">Enviar</button>
    </div>
  </div>

  <script>
    class ChatWebAgent {
      constructor(webhookUrl, messagesContainer) {
        this.webhookUrl = webhookUrl;
        this.messagesContainer = messagesContainer;
        this.sessionId = this.generateSessionId();
      }

      generateSessionId() {
        return 'session_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
      }

      addMessage(text, isUser) {
        const messageDiv = document.createElement('div');
        messageDiv.className = 'message ' + (isUser ? 'user-message' : 'agent-message');
        messageDiv.textContent = text;
        this.messagesContainer.appendChild(messageDiv);
        this.messagesContainer.scrollTop = this.messagesContainer.scrollHeight;
      }

      async sendMessage(message) {
        try {
          this.addMessage(message, true);
```

```
      const response = await fetch(this.webhookUrl, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          chatInput: message,
          sessionId: this.sessionId
        })
      });

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const data = await response.json();
      this.addMessage(data.message, false);
    } catch (error) {
      console.error('Error:', error);
      this.addMessage('Lo siento, ha ocurrido un error. Por favor, intenta de nuevo.', false);
    }
  }
}

// Inicialización
const messagesContainer = document.getElementById('messages');
const agent = new ChatWebAgent(
  'http://localhost:5678/webhook/AGENTE_WEB_GITHUB',
  messagesContainer
);

const messageInput = document.getElementById('message-input');
const sendButton = document.getElementById('send-button');

sendButton.addEventListener('click', () => {
  const message = messageInput.value.trim();
  if (message) {
    agent.sendMessage(message);
    messageInput.value = '';
  }
});

messageInput.addEventListener('keypress', (e) => {
  if (e.key === 'Enter') {
    sendButton.click();
```

```
    }
  });

  // Mensaje de bienvenida
  agent.sendMessage('Hola');
</script>
</body>
</html>
```

Troubleshooting avanzado

Problema 1: El agente no responde

Diagnóstico:

```bash
# 1. Verifica que el workflow está activo
curl http://localhost:5678/webhook/AGENTE_WEB_GITHUB \
  -X POST \
  -H "Content-Type: application/json" \
  -d '{"chatInput":"test","sessionId":"test123"}'

# 2. Verifica Redis
docker exec -it redis redis-cli -a tu_password KEYS "chat_memory:*"

# 3. Verifica logs de n8n
docker logs n8n --tail 50
```

Soluciones:

1. **Workflow inactivo:** Actívalo en n8n
2. **Redis desconectado:** Reinicia el contenedor
3. **Credenciales incorrectas:** Revisa todas las credenciales
4. **Error en el prompt:** Revisa sintaxis del system message

---

Problema 2: Respuestas incoherentes

Causas posibles:

1. **Temperatura muy alta:**
   - Solución: Reduce a 0.1 en los modelos
2. **Memoria corrupta:**

```bash
# Limpia la memoria de una sesión
docker exec -it redis redis-cli -a tu_password DEL "chat_memory:session_id"
```

3. **Prompt ambiguo:**
   - Solución: Refina las instrucciones del sistema

Problema 3: Herramientas no funcionan

MCP Calendario no responde:

```bash
# Test del endpoint MCP
curl https://tu-servidor-mcp/calendario/health \
  -H "Authorization: Bearer tu_token"

# Verifica que devuelve 200 OK
```

Google Sheets falla:

1. Verifica permisos del service account
2. Comprueba que el Sheet está compartido con el email del service account
3. Verifica el ID del documento y nombre de la hoja

Problema 4: Errores de formato en respuestas

Síntoma: El usuario ve caracteres extraños o JSON

Causa: Fallo en el formateo o limpieza

Solución:

1. Verifica que todos los nodos de limpieza están conectados
2. Revisa que el modelo de formateo (GPT-4.1-mini) tiene créditos
3. Comprueba el nodo `Basic LLM Chain` no tiene errores

Problema 5: Sesiones se mezclan

Síntoma: El agente recuerda conversaciones de otros usuarios

Causa: `sessionId` no único o corrupto

Solución:

```javascript
// Genera sessionId único por navegador
function generateSessionId() {
  let sessionId = localStorage.getItem('chat_session_id');
  if (!sessionId) {
    sessionId = 'session_' + Date.now() + '_' +
                Math.random().toString(36).substr(2, 9);
    localStorage.setItem('chat_session_id', sessionId);
  }
  return sessionId;
}
```

Problema 6: El agente no agenda citas

Diagnóstico paso a paso:

1. **Verifica que solicita los datos:**
   - El agente debe pedir: nombre, email, teléfono, servicio

2. **Verifica la herramienta MCP Calendario:**

```bash
# Test manual
curl https://tu-servidor-mcp/calendario/get_events \
  -H "Authorization: Bearer tu_token" \
  -H "Content-Type: application/json" \
  -d '{
    "start_date": "2024-12-01T10:00:00Z",
    "end_date": "2024-12-01T11:00:00Z"
  }'
```

3. **Verifica el envío de email:**
   - Comprueba que MCP Correo está configurado
   - Revisa logs del servidor MCP

4. **Verifica el registro de lead:**
   - Abre Google Sheets y verifica si se creó la fila

Causas comunes:

- **Calendario retorna error:** Verifica credenciales MCP
- **Email no se envía:** Revisa configuración SMTP del servidor MCP
- **Lead no se guarda:** Verifica permisos en Google Sheets

Monitoreo y logs

Ver ejecuciones en n8n

1. En n8n, ve a **Executions**
2. Filtra por:
   - **Workflow:** AGENTE WEB GITHUB
   - **Status:** Success / Error
   - **Date range:** Últimas 24 horas
3. Haz clic en una ejecución para ver:
   - Datos de entrada
   - Salida de cada nodo
   - Errores si los hay
   - Tiempo de ejecución

Logs de Redis

```bash
# Monitorear en tiempo real
docker exec -it redis redis-cli -a tu_password MONITOR

# Ver memoria usada
docker exec -it redis redis-cli -a tu_password INFO memory

# Ver todas las sesiones activas
docker exec -it redis redis-cli -a tu_password KEYS "chat_memory:*"

# Ver contenido de una sesión específica
docker exec -it redis redis-cli -a tu_password GET "chat_memory:session_id"
```

Métricas importantes

1. **Tiempo de respuesta:**
   - Óptimo: < 3 segundos
   - Aceptable: < 5 segundos
   - Mejorable: > 5 segundos
2. **Tasa de error:**
   - Objetivo: < 1%
   - Monitorizar: Ejecuciones fallidas en n8n
3. **Uso de memoria Redis:**
   - Monitorizar: Crecimiento continuo indica fuga
   - Solución: Limpieza periódica de sesiones antiguas

Optimizaciones de rendimiento

1. Caché de respuestas comunes

Implementa un nodo de caché antes del agente para preguntas frecuentes:

```javascript
// Pseudo-código
const commonQuestions = {
  "hola": "¡Hola! ¿En qué puedo ayudarte?",
  "horario": "Nuestro horario es de lunes a viernes, 10:00-14:00 y 15:00-19:00",
  "ubicación": "Estamos en Sevilla, España"
};

if (commonQuestions[input.toLowerCase()]) {
  return commonQuestions[input.toLowerCase()];
} else {
  // Continuar con el agente
}
```

---

2. Limpieza automática de sesiones antiguas

Script de mantenimiento Redis:

```bash
#!/bin/bash
# cleanup_old_sessions.sh

# Conectar a Redis
redis-cli -a tu_password <<EOF
# Obtener todas las keys de sesiones
KEYS chat_memory:*
# Por cada key, verificar TTL
# Si > 24 horas, eliminar
EOF
```

Configura como cron job:

```bash
# Ejecutar diariamente a las 3 AM
0 3 * * * /path/to/cleanup_old_sessions.sh
```

---

3. Usar modelos más rápidos para tareas simples

- **Agente principal:** GPT-4.1 o Gemini Pro (precisión)
- **Formateo:** GPT-4.1-mini (velocidad)
- **Clasificación inicial:** Modelo local pequeño (opcional)

Seguridad

1. Rate limiting

Implementa límite de mensajes por sesión:

javascript

```javascript
// En n8n, antes del agente
const sessionId = $json.sessionId;
const key = `rate_limit:${sessionId}`;

// Incrementar contador
const count = await redis.incr(key);
await redis.expire(key, 60); // 1 minuto

if (count > 10) {
  return {
    json: {
      message: "Has excedido el límite de mensajes. Espera un momento."
    }
  };
}
```

---

2. Validación de entrada

javascript

```javascript
// Validar sessionId
if (!/^session_[0-9]+_[a-z0-9]+$/.test($json.sessionId)) {
  throw new Error('Invalid session ID');
}

// Sanitizar entrada
const input = $json.chatInput
  .trim()
  .slice(0, 1000) // Máximo 1000 caracteres
  .replace(/<script>/gi, ''); // Prevenir XSS
```

---

3. Autenticación del webhook (Opcional)

Añade un API key al webhook:

```javascript
// En el código del cliente
const response = await fetch(webhookUrl, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-API-Key': 'tu_api_key_secreta'
  },
  body: JSON.stringify({
    chatInput: message,
    sessionId: sessionId
  })
});
```

En n8n, verifica el header al inicio del workflow.

Extensiones futuras

1. Soporte multiidioma

Detectar idioma y responder acorde:

```javascript
// Pseudo-código
const detectedLanguage = detectLanguage($json.chatInput);

if (detectedLanguage !== 'es') {
  // Traducir entrada al español
  const translatedInput = await translate($json.chatInput, 'es');
  // Procesar con el agente
  const response = await agent.process(translatedInput);
  // Traducir respuesta al idioma original
  return await translate(response, detectedLanguage);
}
```

---

2. Análisis de sentimiento

Detectar si el usuario está frustrado y escalar a humano:

```javascript
const sentiment = await analyzeSentiment($json.chatInput);

if (sentiment.score < -0.5) {
  return {
    message: "Noto que podrías estar frustrado. ¿Te gustaría hablar directamente con un asesor humano?"
  };
}
```

---

3. Integración con CRM

Sincronizar leads con Salesforce, HubSpot, etc.:

```javascript
// Después de crear lead en Google Sheets
await syncToCRM({
  name: leadData.name,
  email: leadData.email,
  phone: leadData.phone,
  source: 'Web Agent',
  status: 'New'
});
```

---

4. Dashboard de métricas

Visualizar en tiempo real:

- Conversaciones activas
- Tiempo promedio de respuesta
- Tasa de conversión (leads generados)
- Preguntas más frecuentes
- Horarios de mayor actividad

Conclusión

Este Web Agent N8N es un sistema completo de atención al cliente que:

✅ Mantiene conversaciones contextuales gracias a Redis
✅ Agenda citas inteligentemente verificando disponibilidad
✅ Envía confirmaciones automáticas por email
✅ Registra leads sin intervención humana
✅ Se integra fácilmente con cualquier interfaz web
✅ Es escalable y mantenible

Arquitectura modular: Cada componente puede mejorarse independientemente
Extensible: Fácil añadir nuevas herramientas y capacidades
Profesional: Listo para producción con las configuraciones adecuadas

---

Para soporte técnico o personalizaciones, consulta la documentación de cada servicio integrado o contacta con el equipo de desarrollo.