Guía paso a paso de cómo integrar una nueva API (ejemplo: CRM) a tu sistema MCP, para que quede **documentada y consultable por la IA**.

Aquí te armo un **informe más detallado** con explicación técnica y documentación del código:

📑 Informe Técnico – Integración de Nuevas APIs en el MCP

🎯 Objetivo

El propósito es diseñar un **sistema centralizado y escalable** que permita integrar múltiples sistemas (CRM, ERP, RH, SABI, etc.) en un **único punto de entrada (MCP)**.
Este MCP:

- Permite **conectar APIs con Swagger, JSON o mixtas**.
- **Genera documentación automática** para que la IA pueda leer y comprender los endpoints.
- Facilita la **consulta de datos** a través de preguntas naturales.
- Soporta **autenticación** (Bearer Token, API Key, OAuth2).

🛠️ 1. Configuración de Nueva API

Archivo: `src/config/apis.config.ts`

Aquí se define la nueva API **CRM** con metadatos y endpoints.

```
export const apiConfigs: APIConfig[] = [
  // ... otras configuraciones de sistemas

  {
    id: 'crm', // Identificador único
    name: 'crm-api',
    baseUrl: 'https://crm.tuempresa.com/api',
    type: 'swagger', // swagger | json | mixed
    description: 'Sistema de Gestión de Relación con Clientes',
    swaggerUrl: 'https://crm.tuempresa.com/api-docs/json',

    // Endpoints estáticos (si Swagger no cubre todos)
    endpoints: [
      {
        path: '/clientes',
        method: 'GET',
        description: 'Obtener lista de clientes',
```

```
        parameters: [
          { name: 'estado', type: 'string', in: 'query', required:
false }
        ]
      },
      {
        path: '/clientes/{id}',
        method: 'GET',
        description: 'Obtener detalles de un cliente específico',
        parameters: [
          { name: 'id', type: 'string', in: 'path', required: true }
        ]
      },
      {
        path: '/ventas',
        method: 'GET',
        description: 'Obtener listado de ventas',
        parameters: [
          { name: 'fecha_inicio', type: 'string', in: 'query',
required: false },
          { name: 'fecha_fin', type: 'string', in: 'query', required:
false }
        ]
      }
    ],

    authentication: {
      type: 'bearer',
      token: process.env.CRM_API_TOKEN || 'tu-token'
    },

    healthCheck: '/clientes'
  }
];
```

🧩 2. Palabras Clave para Detección Automática

Archivo: `src/controllers/chat.controller.ts`

Asociamos cada API a un conjunto de **palabras clave**. Esto permite que el sistema detecte a qué API dirigir la pregunta del usuario.

```
private getAPIKeywords(api: any): string[] {
  const keywordsMap: { [key: string]: string[] } = {
    'sabi': ['usuario', 'activo', 'empresa', 'sabi'],
    'rh': ['empleado', 'recursos humanos', 'nómina'],
    'crm': ['crm', 'cliente', 'ventas', 'marketing', 'lead',
'oportunidad'],
    'erp': ['erp', 'finanzas', 'contabilidad', 'planificación']
  };

  return keywordsMap[api.id] || [api.id, api.name.toLowerCase()];
}
```

🔎 3. Consulta de APIs Según Pregunta

Archivo: `src/controllers/chat.controller.ts`

Aquí se definen las reglas para **interpretar preguntas** y conectarse a los endpoints correctos.

```
private async queryAPIForQuestion(api: any, question: string):
Promise<string | null> {
  const lowerQuestion = question.toLowerCase();
  let endpointData = '';

  try {
    // 🔹 Lógica especial para CRM
    if (api.id === 'crm') {
      if (/cliente|customer/i.test(lowerQuestion)) {
        const data = await apiManager.queryAPI(api.id, '/clientes',
{});
        endpointData += this.formatListResponse(data, 'cliente');
      }
      if (/venta|sales/i.test(lowerQuestion)) {
        const data = await apiManager.queryAPI(api.id, '/ventas', {});
        endpointData += this.formatListResponse(data, 'venta');
      }
    }

    // 🔹 Lógica genérica para otras APIs
    else {
      const apiInfo = await apiManager.getAPI(api.id);
      endpointData = `**${apiInfo?.name}**\n- URL:
${apiInfo?.baseUrl}\n- ${apiInfo?.description}`;
    }

    return endpointData;

  } catch (error) {
    logger.warn(`Error consultando API ${api.id}:`, error);
    return `Error consultando ${api.name}`;
  }
}
```

🧪 4. Pruebas de Integración

```
# Listar APIs disponibles
curl http://localhost:3000/api/apis

# Preguntar clientes del CRM
curl -X POST http://localhost:3000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"question": "listame los clientes del CRM", "model": "claude"}'

# Obtener documentación del CRM
curl http://localhost:3000/api/docs?apiId=crm

# Consultar endpoint específico
curl "http://localhost:3000/api/query?endpoint=/clientes&apiId=crm"
```

🔐 5. Autenticación

Bearer Token

```
authentication: {
  type: 'bearer',
  token: process.env.CRM_API_TOKEN
}
```

API Key

```
authentication: {
  type: 'apiKey',
  key: 'X-API-Key',
  value: process.env.CRM_API_KEY
}
```

📘 6. Tipos de APIs Soportados

- **Swagger** → Descubrimiento automático de endpoints.
- **JSON (sin Swagger)** → Endpoints definidos manualmente.
- **Mixto** → Swagger + endpoints personalizados.

⚠️ 7. Solución de Problemas

- Revisar conectividad (`ping` o Postman).
- Verificar autenticación (Bearer / API Key).
- Chequear logs del servidor.
- Probar manualmente con `curl` o Postman.

✅ Conclusión

Con esta estructura:

- Se pueden **integrar múltiples sistemas** sin modificar la lógica base.
- La IA recibe documentación **unificada y actualizada automáticamente**.
- El código está **escalado para soportar nuevos módulos** (CRM, ERP, RH, Inventario, Compras, etc.).
