UNIVERSIDAD LAICA ELOY ALFARO DE MANABÍ

FACULTAD DE CIENCIAS INFORMÁTICAS

CARRERA DE SOFTWARE

TALLER DE PRÁCTICA

Model Context Protocol (MCP): Integración de IA con Microservicios

Carrera	Software	**Nivel**	Quinto
Asignatura	Aplicación para el Servidor Web	**Paralelos**	A - B
Docente	Ing. John Cevallos	**Período**	2025-2026 (2)
Taller N°	3	**Duración**	3 sesiones (6 horas)
Modalidad	Grupal (3 estudiantes)	**Prerrequisito**	Talleres 1 y 2 completados

1. Descripción General

En este taller, los estudiantes evolucionarán su arquitectura de microservicios desarrollada en los talleres anteriores para integrar **Model Context Protocol (MCP)**, el protocolo estándar de Anthropic que permite a los modelos de IA (como Claude o Gemini) interactuar de manera inteligente con servicios backend.

1.1. ¿Qué es MCP?

MCP es un protocolo que permite a los modelos de Inteligencia Artificial decidir qué herramientas (Tools) ejecutar basándose en la intención del usuario. A diferencia de REST donde el cliente controla el flujo, en MCP la IA toma decisiones inteligentes sobre qué operaciones realizar.

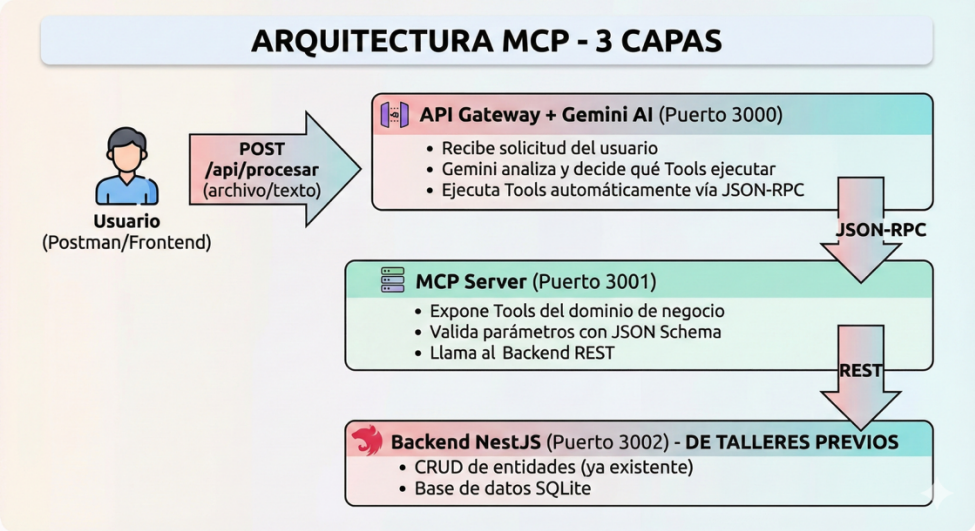
1.2. Evolución desde Talleres Anteriores

La progresión de aprendizaje ha sido:

- **Taller 1:** Microservicios + RabbitMQ (comunicación interna asíncrona)
- **Taller 2:** Webhooks + Serverless (comunicación hacia sistemas externos)
- **Taller 3 (Actual):** MCP + IA (orquestación inteligente de servicios)

1.3. Arquitectura del Proyecto

Los estudiantes implementarán una arquitectura de 3 capas:



2. Objetivos de Aprendizaje

1. **Comprender MCP:** Entender cómo MCP permite a los modelos de IA orquestar servicios de forma inteligente, tomando decisiones basadas en la intención del usuario.
2. **Diseñar Tools:** Crear herramientas (Tools) bien definidas con JSON Schema que expongan la lógica de negocio existente al modelo de IA.
3. **Implementar JSON-RPC 2.0:** Construir un servidor MCP que implemente el protocolo JSON-RPC para comunicación entre el API Gateway y los Tools.
4. **Integrar con Gemini:** Utilizar la API de Gemini con Function Calling para permitir que la IA decida qué Tools ejecutar.
5. **Reutilizar código existente:** Integrar los microservicios desarrollados en talleres anteriores como backend del sistema MCP.

3. Requisitos del Sistema

3.1. Base desde Talleres Anteriores

Cada grupo debe tomar su proyecto de microservicios del Taller 1 o Taller 2 como punto de partida. El sistema debe tener:

- Al menos 2 entidades relacionadas (Maestro-Movimiento)
- Endpoints REST funcionales para CRUD
- Base de datos SQLite operativa

3.2. Nuevos Componentes a Desarrollar

A) MCP Server (TypeScript + Express)

Servidor que expone Tools mediante JSON-RPC 2.0:

1. **Tool de búsqueda:** Permite buscar registros en la entidad principal (ej: buscar_producto, buscar_libro)
2. **Tool de validación:** Valida reglas de negocio (ej: validar_stock, validar_disponibilidad)
3. **Tool de acción:** Ejecuta operaciones transaccionales (ej: crear_egreso, registrar_prestamo)

B) API Gateway (NestJS + Gemini)

Gateway inteligente que integra el modelo de IA:

- Recibe solicitudes del usuario (texto o archivos)
- Consulta los Tools disponibles al MCP Server
- Envía la solicitud a Gemini con los Tools definidos
- Ejecuta automáticamente los Tools que Gemini decide usar
- Retorna respuesta consolidada al usuario

4. Ejemplos de Dominio por Proyecto

Según el proyecto autónomo de cada grupo, estos son ejemplos de Tools que podrían implementar:

Proyecto	Tool Búsqueda	Tool Validación	Tool Acción
Sistema de Biblioteca	buscar_libro	validar_disponibilidad	registrar_prestamo
Sistema de Inventario	buscar_producto	validar_stock	crear_egreso
Sistema de Citas Médicas	buscar_medico	validar_horario	agendar_cita
Sistema de Reservas	buscar_habitacion	validar_fechas	crear_reserva
Sistema de Pedidos	buscar_menu	validar_ingredientes	crear_pedido

5. Estructura de Archivos Requerida

proyecto-mcp/

```
├── apps/
│   ├── backend/                    # Microservicio existente
│   │   ├── src/
│   │   │   ├── [entidad-1]/        # Módulo entidad principal
│   │   │   └── [entidad-2]/        # Módulo entidad secundaria
│   │   └── data/inventario.db      # SQLite
│   │
│   ├── mcp-server/                 # NUEVO: Servidor MCP
│   │   ├── src/
│   │   │   ├── tools/              # Definición de Tools
│   │   │   │   ├── registry.ts     # Registro de Tools
│   │   │   │   ├── buscar-[x].tool.ts
│   │   │   │   ├── validar-[x].tool.ts
│   │   │   │   └── crear-[x].tool.ts
│   │   │   ├── services/
│   │   │   │   └── backend-client.ts
│   │   │   └── server.ts           # Servidor Express
│   │   └── package.json
│   │
│   └── api-gateway/                # NUEVO: Gateway con Gemini
│       ├── src/
│       │   ├── gemini/             # Integración Gemini
│       │   ├── mcp-client/         # Cliente MCP
│       │   └── [dominio]/          # Controlador principal
│       └── package.json
│
└── README.md                       # Documentación completa
```

6. Flujo de Ejecución Esperado

Ejemplo con Sistema de Biblioteca:

1. **Usuario envía:** *"Quiero prestar el libro 'Clean Code' para el estudiante Juan Pérez"*
2. **API Gateway:** Envía el texto a Gemini junto con los Tools disponibles
3. **Gemini decide:** Primero buscar_libro('Clean Code'), luego validar_disponibilidad(libro_id), finalmente registrar_prestamo(libro_id, 'Juan Pérez')
4. **MCP Server:** Ejecuta cada Tool secuencialmente llamando al Backend REST
5. **Respuesta:** *"El préstamo del libro 'Clean Code' para Juan Pérez ha sido registrado exitosamente. Fecha de devolución: 15/01/2026"*

7. Tecnologías y Recursos

7.1. Stack Tecnológico

Componente	Tecnología	Puerto
Backend	NestJS + TypeORM + SQLite	3002
MCP Server	TypeScript + Express + JSON-RPC	3001
API Gateway	NestJS + @google/generative-ai	3000
Modelo IA	Gemini 2.0 Flash (gratuito)	API Cloud

7.2. Recursos de Referencia

- **Proyecto de ejemplo del docente:** mcp-inventario-facturas (disponible en repositorio del curso)
- **Documentación MCP:** https://modelcontextprotocol.io
- **Gemini AI Studio:** https://aistudio.google.com
- **Especificación JSON-RPC 2.0:** https://www.jsonrpc.org/specification

8. Entregables

1. **Repositorio Git** con la estructura de carpetas especificada y código funcional
2. **README.md** con instrucciones de instalación, configuración y ejecución
3. **Video demostrativo** (3-5 minutos) mostrando el flujo completo funcionando
4. **Documentación de Tools** con descripción, parámetros de entrada y ejemplos de uso
5. **Pruebas documentadas** con capturas de Postman o Thunder Client

9. Rúbrica de Evaluación

Criterio	Puntos	Descripción
MCP Server funcional	25	3 Tools implementados con JSON Schema correcto
API Gateway con Gemini	25	Integración correcta con Function Calling
Integración Backend	15	Conexión correcta con microservicio existente
Flujo End-to-End	15	Demostración de flujo completo funcionando
Documentación y README	10	Instrucciones claras y completas
Calidad del código	10	Código limpio, comentarios, TypeScript tipado
TOTAL	**100**	

10. Cronograma Sugerido

Sesión	Actividad	Entregable Parcial
Sesión 1 (2h)	Conceptos MCP + Setup proyecto	MCP Server con 1 Tool básico funcionando
Sesión 2 (2h)	Completar Tools + Integrar Gemini	3 Tools + API Gateway con Function Calling
Sesión 3 (2h)	Integración Backend + Pruebas	Sistema completo funcionando + video demo

"La IA no reemplaza al desarrollador, lo potencia. MCP es el puente."