

# Taller 02 – Diseño e Implementación de una API REST con OpenAPI/Swagger

## 1. Objetivo del taller

Aplicar los principios del diseño RESTful para construir y documentar una API que sirva como punto de integración entre sistemas, utilizando **Apache Camel** como línea base, con opción a implementar en otro stack cumpliendo condiciones técnicas equivalentes.

## 2. Competencia a desarrollar

- Diseñar una API REST siguiendo estándares de interoperabilidad.
- Documentar endpoints con **OpenAPI/Swagger**.
- Implementar y probar la API en un entorno controlado.
- Comprender la equivalencia de integración en distintos lenguajes y frameworks.

## 3. Requisitos previos

- **Java 17 o superior** instalado.
- **Apache Maven 3.9+** instalado y configurado.
- **Visual Studio Code** con extensiones Java.
- **Postman** o herramienta equivalente para pruebas de APIs.
- Conocimientos básicos de HTTP, JSON y estructuras de datos.

## 4. Flexibilidad Tecnológica

Este taller está diseñado con una **línea base oficial en Java + Maven + Apache Camel**, que sirve como guía de referencia.

Sin embargo, los equipos pueden optar por implementar la solución en otro stack (Node.js, Python, .NET u otro) **siempre que cumplan las condiciones técnicas mínimas:**

### Condiciones del stack libre:

1. Implementar la misma funcionalidad y endpoints definidos en el enunciado.

2. Incluir un contrato **OpenAPI 3.0** válido (`openapi.yaml` o `swagger.json`).
3. Proveer ejecución reproducible:
  - a. mediante `Dockerfile` o `docker-compose.yml`, o
  - b. comandos claros (`npm start`, `uvicorn main:app`, etc.).
4. Entregar **colección de pruebas Postman/Newman** con evidencias de cada endpoint.
5. Exponer un endpoint de salud `/health`.
6. Incluir un **README técnico** con pasos de instalación, ejecución y pruebas.

La **evaluación será idéntica** para todos los stacks, enfocándose en:

- Cumplimiento funcional.
- Aplicación del patrón de integración.
- Calidad del contrato API.
- Evidencias y documentación.

## 5. Escenario del taller

Una empresa de logística necesita exponer una **API REST** para gestionar envíos.

El servicio debe permitir:

1. Consultar todos los envíos (`GET /envios`)
2. Consultar un envío por ID (`GET /envios/{id}`)
3. Registrar un nuevo envío (`POST /envios`)

## 6. Línea Base: Implementación Java + Maven + Apache Camel

### Paso 1. Crear proyecto

```
mvn -q archetype:generate \
  -DgroupId=edu.udla.isw \
  -DartifactId=api-rest-lab \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

Abre la carpeta `api-rest-lab` en VS Code.

## Paso 2. Editar pom.xml

Incluye las dependencias:

```
<dependencies>
  <dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-core</artifactId>
    <version>4.7.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-main</artifactId>
    <version>4.7.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-rest</artifactId>
    <version>4.7.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-openapi-java</artifactId>
    <version>4.7.0</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>2.0.13</version>
  </dependency>
</dependencies>
```

## Paso 3. Crear definición OpenAPI

Archivo src/main/resources/openapi.yaml:

```
openapi: 3.0.1
info:
  title: API de Envíos
  version: 1.0.0
  description: API REST para registrar y consultar envíos.
paths:
  /envios:
    get:
      summary: Listar todos los envíos
      responses:
        '200':
```

```

        description: Lista de envíos
    post:
        summary: Crear un nuevo envío
        requestBody:
            required: true
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/Envio'
        responses:
            '201':
                description: Envío creado
    /envios/{id}:
        get:
            summary: Obtener un envío por ID
            parameters:
                - name: id
                  in: path
                  required: true
                  schema:
                      type: string
            responses:
                '200':
                    description: Detalle del envío
components:
    schemas:
        Envio:
            type: object
            properties:
                id:
                    type: string
                destinatario:
                    type: string
                direccion:
                    type: string
                estado:
                    type: string

```

#### Paso 4. Implementar clase principal App.java

```

package edu.udla.isw;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.main.Main;

public class App extends RouteBuilder {

    public static void main(String[] args) throws Exception {

```

```

Main main = new Main();
main.configure().addRoutesBuilder(new App());
main.run(args);
}

@Override
public void configure() {
    restConfiguration()
        .component("netty-http")
        .port(8080)
        .contextPath("/api")
        .apiContextPath("/api-doc")
        .apiProperty("api.title", "API de Envíos")
        .apiProperty("api.version", "1.0.0");

    rest("/envios").description("Gestión de Envíos")
        .get().to("direct:listarEnvios")
        .post().to("direct:crearEnvio");

    rest("/envios/{id}")
        .get().to("direct:obtenerEnvio");

    from("direct:listarEnvios")
        .setBody(constant("[{\"id\":\"001\",\"destinatario\":\"Juan Pérez\",\"estado\":\"En tránsito\"}"]));

    from("direct:crearEnvio")
        .log("Nuevo envío recibido: ${body}")
        .setBody(constant("{\"mensaje\":\"Envío registrado correctamente\"}"));

    from("direct:obtenerEnvio")
        .log("Consultando envío con ID: ${header.id}")

    .setBody(simple("{\"id\":\"${header.id}\",\"destinatario\":\"Cliente X\",\"estado\":\"Entregado\"}"));
}
}

```

## Paso 5. Ejecutar y probar

Compilar y ejecutar:

```

mvn clean package
java -jar target/api-rest-lab-1.0-SNAPSHOT.jar

```

Endpoints disponibles: - GET <http://localhost:8080/api/envios> - GET  
<http://localhost:8080/api/envios/{id}> - POST  
<http://localhost:8080/api/envios>

Documentación Swagger:

👉 <http://localhost:8080/api/api-doc>

## 7. Pruebas con Postman

1. **GET** /envios → retorna lista de envíos.
2. **GET** /envios/001 → retorna detalle del envío.
3. **POST** /envios con cuerpo JSON:

```
{  
  "id": "002",  
  "destinatario": "María López",  
  "direccion": "Av. Central 123",  
  "estado": "Registrado"  
}
```

## 9. Entregables

Cada equipo debe subir un Informe que incluya:

- Link a repositorio Git con
- Código fuente (App. java u otro framework equivalente).
- Archivo openapi.yaml o URL de Swagger.
- Colección Postman (JSON).
- Evidencias de ejecución (capturas o logs).
- **README.md** con pasos de instalación y ejecución.
- Reflexión individual:
  - ¿Qué ventajas ofrece una API REST bien diseñada frente a los enfoques tradicionales de integración?"

## 10. Bibliografía

- Hohpe, G. & Woolf, B. (2004). *Enterprise Integration Patterns*. Addison-Wesley.
- Apache Camel Documentation – <https://camel.apache.org>
- OpenAPI Specification – <https://swagger.io/specification/>