



**Nombre del Estudiante:** Alexis Sotomayor

**Fecha:** 10/11/2025

### Examen 001

Este examen busca demostrar la habilidad de documentar una API REST existente (JSONPlaceholder) usando OpenAPI 3.0 y Swagger. La documentación generada describe técnicamente los endpoints de sus seis recursos principales y sus operaciones CRUD, estableciendo un estándar profesional que mejora la comunicación entre desarrolladores y facilita la integración y mantenimiento de sistemas. Esta competencia es esencial en el desarrollo web moderno para garantizar la escalabilidad y una colaboración efectiva.

### Desarrollo de la práctica

El taller es **completamente autocontenido**: un estudiante puede tomar este material y aprender sin necesidad de buscar recursos externos. Todos los conceptos necesarios se explican en detalle, todos los ejemplos son ejecutables, y todas las soluciones están incluidas.

### Enfoque Práctico Inmediato

A diferencia de tutoriales que pasan mucho tiempo en teoría abstracta, este taller **comienza con código funcional en el paso 4**. Los estudiantes ven resultados inmediatos, lo que aumenta la motivación y el engagement.

Antes de crear el contenido, se realizó una investigación exhaustiva sobre:

- Especificación OpenAPI 3.0 oficial
- Tutoriales y cursos existentes sobre Swagger
- Análisis de APIs bien documentadas en la industria
- Feedback de estudiantes sobre materiales de documentación de APIs
- Revisión de literatura pedagógica sobre enseñanza de desarrollo web

Se diseñó la estructura del taller considerando:

- **Tiempo disponible:** Diseñado para completarse en 2.5-3 horas
- **Nivel de los estudiantes:** Ingeniería de Sistemas, VII semestre
- **Conocimientos previos:** HTTP básico, JSON, conceptos de APIs
- **Objetivos del curso:** Integración con el syllabus de Aplicaciones Web

El contenido se desarrolló iterativamente:



1. Creación del índice y estructura de carpetas
2. Redacción del contenido teórico
3. Desarrollo de ejemplos YAML funcionales
4. Diseño de ejercicios progresivos
5. Compilación de recursos adicionales
6. Revisión y refinamiento del material

Se validó el material mediante:

- Prueba de todos los ejemplos YAML en Swagger Editor
- Verificación de que todos los enlaces externos funcionen
- Revisión de la progresión pedagógica
- Comprobación de que los ejercicios sean realizables en el tiempo estimado
- Validación de que las soluciones de ejercicios sean correctas

### Desde la vista de swagger corriendo como servidor de documentación

The screenshot shows the Swagger UI interface running as a local server. The title bar indicates the URL is <http://localhost:3000/api-docs/>. The main content area displays the "JSONPlaceholder API - Documentación Completa" page. It includes a brief description stating it's a REST API for testing and prototyping, providing fake data for CRUD operations. Below this, sections for "Recursos disponibles" (available resources) and "Características" (features) are listed. The "Recursos disponibles" section lists "Posts" (publicaciones), "Comments" (comentarios), "Albums" (álbumes), "Photos" (fotos), "Todos" (todos), and "Users" (usuarios). The "Características" section highlights features like consistent and predictable data, simulated CRUD operations, being ideal for frontend testing, and requiring no authentication. At the bottom, there's a "Servers" dropdown set to "https://jsonplaceholder.typicode.com - Servidor de producción". The "Posts" section shows two operations: "GET /posts" (Obtener todos los posts) and "POST /posts" (Crear un nuevo post).



**Posts** Operaciones relacionadas con publicaciones

GET /posts	Obtener todos los posts
POST /posts	Crear un nuevo post
GET /posts/{id}	Obtener un post por ID
PUT /posts/{id}	Actualizar un post completo
PATCH /posts/{id}	Actualizar parcialmente un post
DELETE /posts/{id}	Eliminar un post
GET /posts/{id}/comments	Obtener comentarios de un post

**Comments** Operaciones relacionadas con comentarios

GET /comments	Obtener todos los comentarios
POST /comments	Crear un nuevo comentario
GET /comments/{id}	Obtener un comentario por ID
PUT /comments/{id}	Actualizar un comentario completo
DELETE /comments/{id}	Eliminar un comentario
GET /posts/{id}/comments	Obtener comentarios de un post

**Users** Operaciones relacionadas con usuarios

GET /users	Obtener todos los usuarios
------------	----------------------------

  

**Comments** Operaciones relacionadas con comentarios

GET /comments	Obtener todos los comentarios
POST /comments	Crear un nuevo comentario
GET /comments/{id}	Obtener un comentario por ID
PUT /comments/{id}	Actualizar un comentario completo
DELETE /comments/{id}	Eliminar un comentario
GET /posts/{id}/comments	Obtener comentarios de un post

**Users** operaciones relacionadas con usuarios

GET /users	Obtener todos los usuarios
POST /users	Crear un nuevo usuario
GET /users/{id}	Obtener un usuario por ID

**Albums** Operaciones relacionadas con álbumes

GET /albums	Obtener todos los álbumes
GET /albums/{id}	Obtener un álbum por ID

**Photos** Operaciones relacionadas con fotos

GET /photos	Obtener todas las fotos
GET /photos/{id}	Obtener una foto por ID

**Photos** Operaciones relacionadas con fotos

GET /photos	Obtener todas las fotos
GET /photos/{id}	Obtener una foto por ID

**Todos** Operaciones relacionadas con tareas

GET /todos	Obtener todas las tareas
GET /todos/{id}	Obtener una tarea por ID

**Schemas**

Post >
NewPost >
Comment >
NewComment >
User >

## Análisis de resultados

El proyecto desarrollado constituye un recurso educativo integral y autocontenido que cumple exitosamente con el objetivo de enseñar documentación de APIs REST usando Swagger a estudiantes de Ingeniería de



Sistemas. La estructura del taller demuestra una progresión pedagógica bien fundamentada, iniciando con 1,500+ palabras de teoría que establece bases sólidas sobre APIs REST, OpenAPI 3.0 y la importancia de la documentación técnica, seguido por objetivos de aprendizaje claramente definidos con competencias técnicas y blandas medibles. La guía práctica de 12 pasos proporciona un andamiaje cognitivo efectivo, llevando a los estudiantes desde la configuración básica hasta la implementación de componentes reutilizables complejos, cada paso validado con checkpoints y soluciones a problemas comunes. Los tres archivos YAML (básico con 50 líneas, completo con 20+ endpoints, y avanzado con autenticación OAuth2) ofrecen ejemplos ejecutables y escalables que permiten aprendizaje progresivo. Los 10 ejercicios propuestos, organizados en niveles de dificultad creciente desde básico hasta un proyecto final integrador de biblioteca completa, garantizan práctica activa y aplicación de conocimientos. La inclusión de múltiples métodos de ejecución (Swagger Editor online sin instalación, servidor Node.js local, y Docker) asegura accesibilidad universal independientemente de las capacidades técnicas o recursos del estudiante.

## Conclusiones

El taller logra su propósito fundamental de capacitar estudiantes en documentación profesional de APIs siguiendo estándares de la industria. La estructura de contenidos, que va desde fundamentos teóricos hasta aplicaciones prácticas avanzadas, garantiza que estudiantes de diferentes niveles puedan beneficiarse del material. La inclusión de objetivos específicos medibles, competencias técnicas y blandas, y niveles de dominio claros (básico, intermedio, avanzado, experto) proporciona un marco evaluativo completo para estudiantes e instructores.

La decisión de priorizar Swagger Editor Online como método principal elimina completamente las barreras técnicas de instalación, permitiendo que cualquier estudiante con un navegador web comience inmediatamente. Las alternativas de servidor Node.js local y Docker atienden a usuarios más avanzados que deseen profundizar. Esta arquitectura multi-nivel respeta diferentes contextos de aprendizaje (presencial, autónomo, flipped classroom) y niveles de experiencia técnica.



## Prompt utilizado

### Contexto

Eres un experto en desarrollo web y documentación de APIs REST. Necesito que me ayudes a crear un taller educativo completo para estudiantes de Ingeniería de Sistemas (VII semestre) sobre documentación de APIs REST usando Swagger y OpenAPI 3.0.

### Objetivo

Crear un recurso educativo autocontenido, profesional y práctico que enseñe a documentar APIs REST usando JSONPlaceholder como API de ejemplo.

## ESTRUCTURA REQUERIDA DEL PROYECTO

### Carpeta Raíz: "Examen 001/"

### 2. 01-teoria.md

Debe incluir:

- ¿Qué es una API REST?
- Importancia de documentar APIs (4-5 razones con ejemplos)
- ¿Qué es Swagger y OpenAPI? (diferencias y componentes)
- Conceptos fundamentales de OpenAPI 3.0:
  - \* Estructura básica (openapi, info, servers, paths, components)
  - \* Métodos HTTP (GET, POST, PUT, DELETE, PATCH)
  - \* Schemas y tipos de datos
  - \* Parámetros (path, query, header, cookie)
  - \* Request body y responses
- Tabla completa de códigos de estado HTTP (2xx, 4xx, 5xx)
- Mejores prácticas de documentación
- Mínimo 1500 palabras, con ejemplos de código

### ### 3. 02-objetivos.md

Debe definir:

- Objetivo general (específico y medible)
- 5 objetivos específicos (usando verbos de acción: comprender, implementar, aplicar, etc.)
- Competencias a desarrollar:
  - \* Técnicas (3-4 competencias)
  - \* Blandas (2-3 competencias)
- Resultados medibles del aprendizaje (indicadores concretos)
- 4 niveles de dominio:
  - \* Básico (principiante)
  - \* Intermedio
  - \* Avanzado
  - \* Experto

Cada nivel con criterios claros de evaluación

- Tiempo estimado: 2.5-3 horas

### ### 4. 03-guia-practica.md

Paso a paso detallado con 12 pasos:

#### \*\*Pasos 1-3: Configuración\*\*

- Acceder a Swagger Editor
- Crear estructura básica OpenAPI 3.0
- Configurar servidor apuntando a JSONPlaceholder

#### \*\*Pasos 4-6: Documentación Básica\*\*

- Documentar primer endpoint GET /posts
- Crear schema Post

- Agregar descripciones detalladas

**\*\*Pasos 7-9: Operaciones Avanzadas\*\***

- POST /posts (con request body)
- PUT /posts/{id} (con parámetros de ruta)
- DELETE /posts/{id}

**\*\*Pasos 10-11: Componentes Reutilizables\*\***

- Schemas complejos con referencias
- Parámetros reutilizables
- Responses reutilizables

**\*\*Paso 12: Validación y Pruebas\*\***

- Validar sintaxis
- Probar endpoints con "Try it out"
- Verificar respuestas

Cada paso debe incluir:

- Descripción clara
- Código YAML completo
- Resultado esperado
- Checkpoint de validación
- Problemas comunes y soluciones

**### 5. RECURSOS.md**

Lista curada de recursos organizados en:

- Documentación oficial (OpenAPI, Swagger)
- Tutoriales y guías (básicos a avanzados)

- Herramientas adicionales:

\* Convertidores (Postman to OpenAPI)

- Frameworks con soporte OpenAPI:

\* Node.js (Express + swagger-ui-express)

## Carpeta: "swagger/"

### ### 6. jsonplaceholder-basic.yaml

Archivo OpenAPI 3.0 básico con:

- Información básica (title, description, version)
- Servidor: <https://jsonplaceholder.typicode.com>
- Solo 1 endpoint: GET /posts
- Schema simple: Post (userId, id, title, body)
- Comentarios explicativos en línea
- Máximo 50 líneas

### ### 7. jsonplaceholder-complete.yaml

Documentación completa con:

- 6 recursos: posts, comments, users, albums, photos, todos
- Operaciones CRUD completas donde aplique
- Schemas complejos con referencias (\$ref)
- Parámetros reutilizables (components/parameters)
- Responses reutilizables (components/responses)
- Ejemplos de request/response
- Mínimo 20 endpoints documentados
- 300-500 líneas de código

### ### 8. swagger-config.yaml

Configuración avanzada mostrando:



- Múltiples servidores (producción, desarrollo, staging)
- Esquemas de autenticación:
  - \* Bearer Token
  - \* API Key (header y query)
  - \* OAuth 2.0 (authorization code flow)
- Links entre operaciones
- Headers de respuesta personalizados
- Callbacks/Webhooks
- Tags para organizar endpoints
- ExternalDocs

## Carpeta: "ejercicios/"

### 9. ejercicios-propuestos.md

10 ejercicios progresivos:

**\*\*Básico (1-3):\*\***

- Agregar endpoint GET /users
- Crear schema User
- Documentar parámetros de query

**\*\*Intermedio (4-6):\*\***

- POST /comments con request body
- PUT /albums/{id}
- DELETE con múltiples respuestas (200, 404, 500)

**\*\*Avanzado (7-9):\*\***

- Schema anidado (Comment con User embebido)



- Paginación (limit, offset, \_page, \_limit)
- Búsqueda y filtrado

**\*\*Proyecto Final (10):\*\***

Sistema completo de biblioteca con:

- Recursos: books, authors, categories, loans
- Relaciones entre recursos
- Autenticación
- Paginación y filtros
- Operaciones CRUD completas

Cada ejercicio debe incluir:

- Descripción del problema
- Objetivos específicos
- Pistas (sin dar la solución directa)
- Tiempo estimado
- Solución completa en YAML
- Bonus challenge (opcional)

## # REQUISITOS TÉCNICOS

## Formato:

- Todo en Markdown (.md) excepto archivos YAML
- Usar emojis contextuales para mejor navegación
- Bloques de código con sintaxis highlighting (```yaml, ```bash, ```javascript)
- Tablas donde sea apropiado
- Listas ordenadas y desordenadas según contexto
- Enlaces internos entre documentos

## Estilo de Código YAML:

- OpenAPI 3.0.0
- Indentación: 2 espacios
- Comentarios explicativos en inglés o español
- Nombres descriptivos (no abreviaturas crípticas)
- Seguir convenciones de OpenAPI

## Tono y Estilo:

- Profesional pero accesible
- Explicaciones claras sin jerga innecesaria
- Ejemplos concretos antes de conceptos abstractos
- Enfoque práctico: "ver-hacer-modificar-crear"
- Anticipar dudas comunes

## Validación:

- Todos los archivos YAML deben ser válidos según OpenAPI 3.0
- Todos los endpoints deben apuntar a JSONPlaceholder real
- Los ejemplos deben ser ejecutables en Swagger Editor
- Las rutas y referencias deben ser correctas

## # ENTREGABLES ADICIONALES

## 10. DOCUMENTACION.md

Documento reflexivo sobre el proyecto que incluya:

- Resumen ejecutivo (200 palabras)
- Objetivos del proyecto (por qué se creó)
- Metodología de desarrollo (cómo se estructuró)

- Metodología pedagógica aplicada:

- \* Aprendizaje progresivo (scaffolding)
- \* Aprendizaje basado en ejemplos
- \* Feedback inmediato
- \* Aprendizaje activo
- Resultados esperados del aprendizaje
- Características destacadas del proyecto
- Impacto educativo esperado
- Casos de uso del material (presencial, autónomo, flipped classroom)
- Posibles extensiones futuras
- Métricas de éxito del taller
- Mínimo 2000 palabras

## 11. server.js (en raíz del proyecto)

Código Node.js completo para:

- Servidor Express
- Integración con swagger-ui-express
- Cargar YAML desde ruta: 'Examen 001/swagger/jsonplaceholder-complete.yaml'
- Ruta / que redirige a /api-docs
- Ruta /basic con documentación básica
- Manejo de errores robusto
- Logs informativos con emojis
- Puerto configurable (3000 por defecto)
- Código completo, funcional, comentado

## 12. package.json

Con:

- Nombre: "swagger-taller-apis"



- Versión: "1.0.0"
- Descripción del proyecto
- Scripts: start, dev
- Dependencias: express, swagger-ui-express, js-yaml
- Autor y licencia

## # CRITERIOS DE CALIDAD

- Completitud: Todos los archivos solicitados
- Funcionalidad: Todo el código debe ejecutarse sin errores
- Claridad: Documentación comprensible para estudiantes
- Progresión: Dificultad incremental lógica
- Profesionalismo: Siguiendo estándares de la industria
- Validez: YAML válido según OpenAPI 3.0
- Consistencia: Estilo uniforme en todos los documentos
- Exhaustividad: Cubrir desde básico hasta avanzado

## # REFERENCIAS

- API de ejemplo: <https://jsonplaceholder.typicode.com/>
- Especificación: OpenAPI 3.0 (<https://swagger.io/specification/>)
- Herramienta: Swagger Editor (<https://editor.swagger.io/>)

## # RESULTADO ESPERADO

Un proyecto educativo completo, autocontenido y profesional que un estudiante pueda usar para aprender documentación de APIs desde cero hasta nivel avanzado en 2.5-3 horas, con material de referencia para consulta futura.

## # IMPORTANTE

- Genera TODOS los archivos completos (no resúmenes)
- El código YAML debe ser copiable y ejecutable directamente
- Incluye ejemplos reales y funcionales
- Anticipa problemas comunes y proporciona soluciones
- Usa un tono educativo pero profesional



ESCUELA  
POLITÉCNICA  
NACIONAL

**Escuela Politécnica Nacional**  
**Aplicaciones Web**



- Prioriza la claridad sobre la brevedad