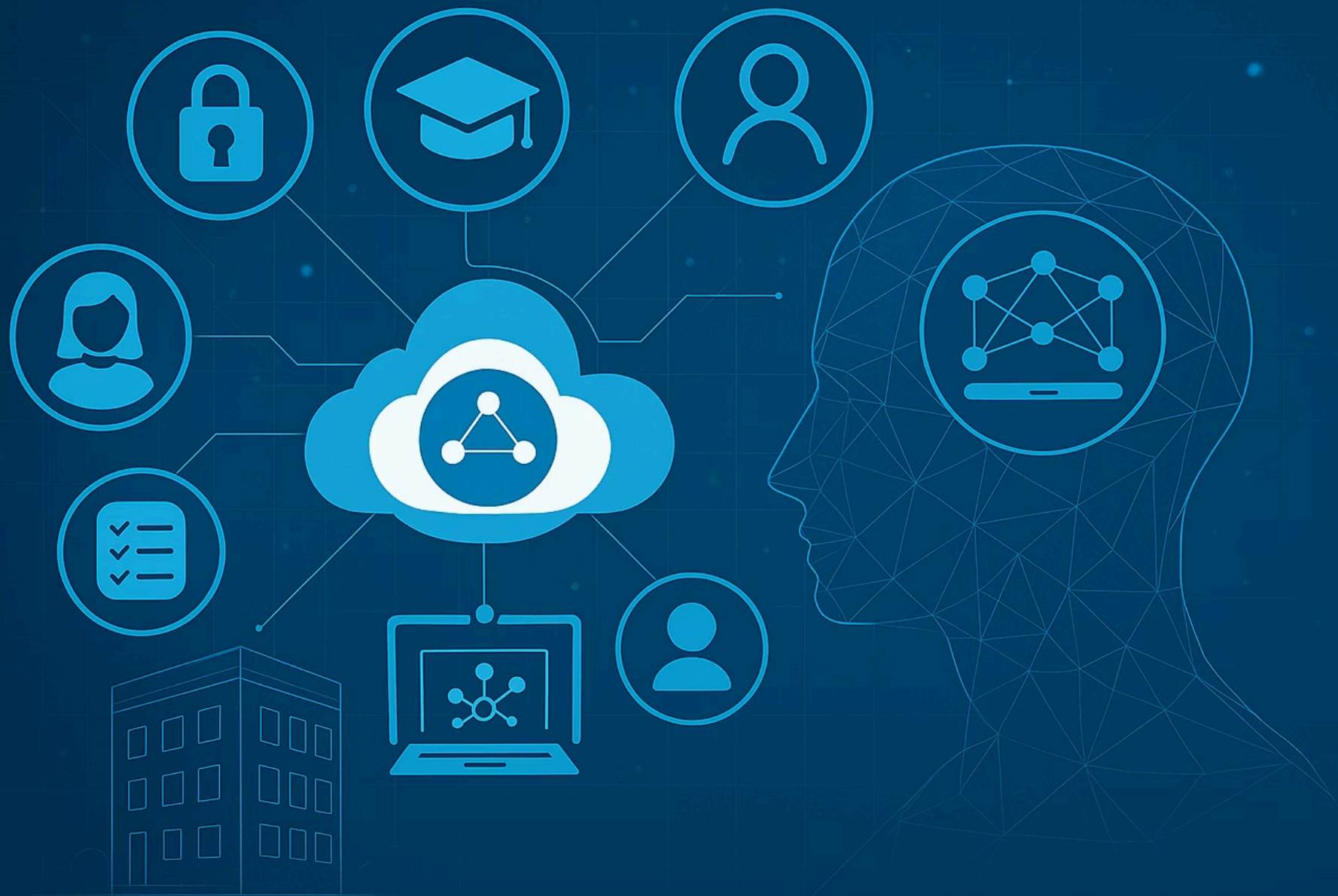


Informe Situación Evaluativa 3: EduTech Innovators SPA



Experiencia 3 – Desarrollo y Operación de Microservicios

Benjamín Torrejón Soto – Alejandra Reyes Duque

Desarrollo Fullstack I 002D

Duoc UC – Ingeniería Informática

Junio 2025

 por Benjamín Torrejón Soto

De Monolito a Microservicios: EduTech Innovators SPA

En esta tercera entrega, el proyecto EduTech Innovators SPA avanza desde una arquitectura monolítica inicial hacia un ecosistema completamente modular basado en microservicios. En las Experiencias 1 y 2 se desarrollaron los servicios de Usuarios, Cursos y Evaluaciones, los cuales sentaron las bases funcionales del sistema. En esta nueva fase se incorporaron cuatro microservicios adicionales: Inscripciones, Contenido, Pagos y Soporte, ampliando significativamente las capacidades de la plataforma. Además, se optimizaron las APIs existentes mediante la unificación de rutas bajo una misma convención REST (/api/[recurso]), la mejora del manejo de errores y la incorporación de Swagger para la documentación automática. Estos avances permitieron implementar pruebas unitarias e integración de manera rigurosa, fortaleciendo la calidad, mantenibilidad y escalabilidad del sistema.

Arquitectura Modular de EduTech Innovators SPA

Microservicios Autónomos

Siete microservicios: Usuarios, Cursos, Evaluaciones (Experiencias 1 y 2), y Inscripciones, Contenido, Pagos, Soporte (nueva fase).

Configuración Aislada

Puerto exclusivo y DB MySQL aislada para cada microservicio, usando HikariCP.

Estructura Estándar

Cada microservicio con módulo Maven/carpeta raíz y paquetes: entities, repositories, services, controllers, config.

Dependencias Comunes

Maven con spring-boot-starter-web, data-jpa, mysql-connector, springdoc-openapi, spring-boot-starter-test.

Puertos Asignados y Flujos de Comunicación

1 Cursos (8081)

Gestiona alta, baja y listados de cursos. Consumido por Inscripciones y frontend.

2 Evaluaciones (8082)

Registra y consulta notas de estudiantes, basada en Usuarios.

3 Usuarios (8083)

Administra registro y consulta de perfiles. Usado por Inscripciones y Evaluaciones.

4 Inscripciones (8084)

Coordina Usuarios y Cursos, valida existencias, genera solicitudes a Pagos.

5 Pagos (8085)

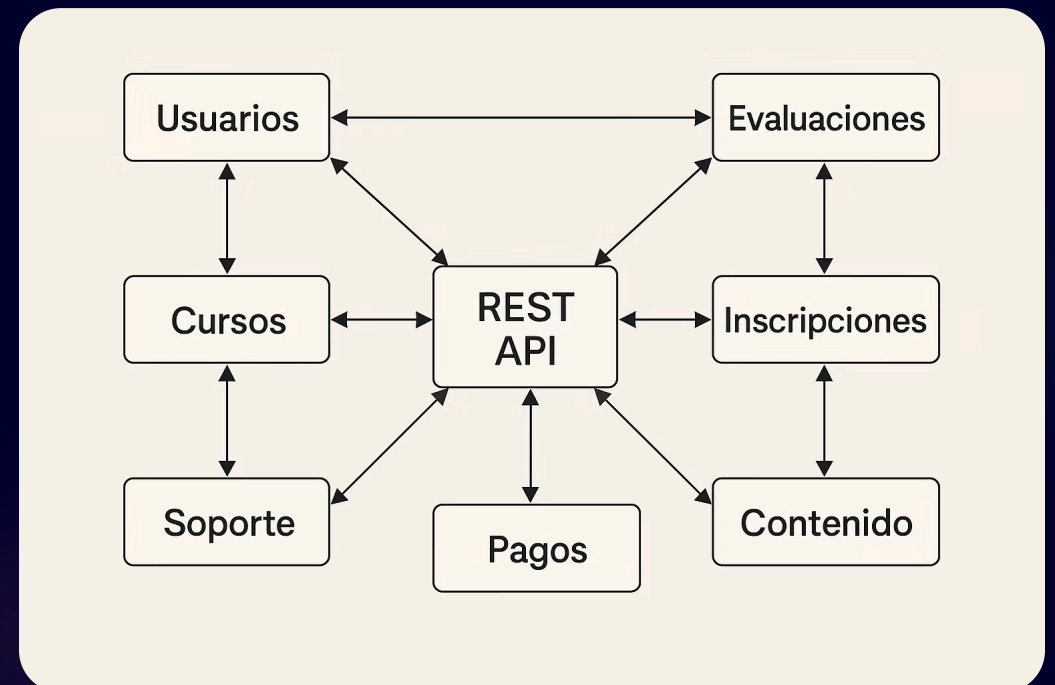
Procesa transacciones de Inscripciones, actualiza estado de inscripción a pagada.

6 Soporte (8086)

Permite creación de tickets de atención, gestiona mensajes y resolución.

7 Contenido (8087)

Administra materiales académicos asociados a cursos.
Consultable desde frontend y relacionado con Inscripciones.



Nota: La arquitectura completa y los flujos de comunicación entre microservicios pueden visualizarse en el diagrama incluido, donde se evidencian las dependencias cruzadas y la interacción con el frontend mediante REST.

Estructura de Microservicios y Configuración

La solución EduTech Innovators SPA se estructura en siete microservicios independientes, cada uno respetando una convención de paquetes que favorece la claridad y mantenibilidad del código: `entities` para las clases de dominio con anotaciones JPA, `repositories` para interfaces que extienden `JpaRepository`, `services` para la lógica de negocio anotada con `@Service`, `controllers` para exponer endpoints REST bajo la convención `/api/[recurso]`, `config` para clases como `CorsConfig` que habilitan la comunicación entre frontend y backend, y `dto` cuando es necesario estandarizar solicitudes y respuestas JSON.

Cada microservicio cuenta con su propio archivo `application.properties`, donde se define un puerto exclusivo (por ejemplo, 8081 para Cursos, 8082 para Evaluaciones, hasta 8087 para Contenido), los parámetros de conexión a una base de datos MySQL aislada (`spring.datasource.url`, `username`, `password`) y la configuración de JPA e HikariCP (`spring.jpa.hibernate.ddl-auto=update`, `hibernate.dialect`, `pool size`). En el archivo `pom.xml` de cada servicio se declaran dependencias comunes como: `spring-boot-starter-web` para la capa web, `spring-boot-starter-data-jpa` y `mysql-connector-java` para persistencia, `spring-boot-devtools` para recarga en caliente, `springdoc-openapi-starter-webmvc-ui` para documentación Swagger disponible en `/swagger-ui/index.html`, y `spring-boot-starter-test` para pruebas unitarias e integración con JUnit 5, Mockito y MockMvc. Esta estructura modular, homogénea y bien documentada facilita la escalabilidad, promueve la separación de responsabilidades y cumple con los criterios exigidos por la rúbrica.

Desarrollo de Controladores REST



InscripcionController

Expone **/api/inscripciones** para listar (GET) y registrar (POST) inscripciones, validando usuario y curso. Delega lógica al servicio.



ContenidoController

Accesible en **/api/contenidos**. Gestiona ciclo completo de recursos académicos (GET, POST, PUT, DELETE), usando DTOs para datos.



PagoController

Disponible en **/api/pagos**. Procesa transacciones (POST) y consulta estado (GET /api/pagos/{inscripcionId}). Valida inscripción y calcula descuentos.

REST API



Ejecución de Pruebas Rigurosas

1 Pruebas de Carga de Contexto

Uso de **ApplicationTests** específicas (**CursosApplicationTests**, **EvaluacionesApplicationTests**, **UsuariosApplicationTests**) con **@SpringBootTest**.

2 Verificación de Entorno

Aseguran la correcta inicialización de Spring Boot, detección de beans y ausencia de conflictos de configuración.

3 Confirmación de Integridad

Ejecución de **mvn test** resultó en **BUILD SUCCESS** para cada test, confirmando la integridad estructural y base sólida para pruebas futuras.










Gestión de Versiones con Git y GitHub

En el repositorio de GitHub (<https://github.com/moonnnluv/Desarrollo-Fullstack-I-002D>) inicializamos el control de versiones con **git init**, añadimos el origen remoto mediante **git remote add origin https://github.com/moonnnluv/Desarrollo-Fullstack-I-002D.git**, y para cada entrega seguimos el flujo habitual: **git add .**, **git commit -m "Experiencia 3: pruebas e integración"** y **git push -u origin main**.

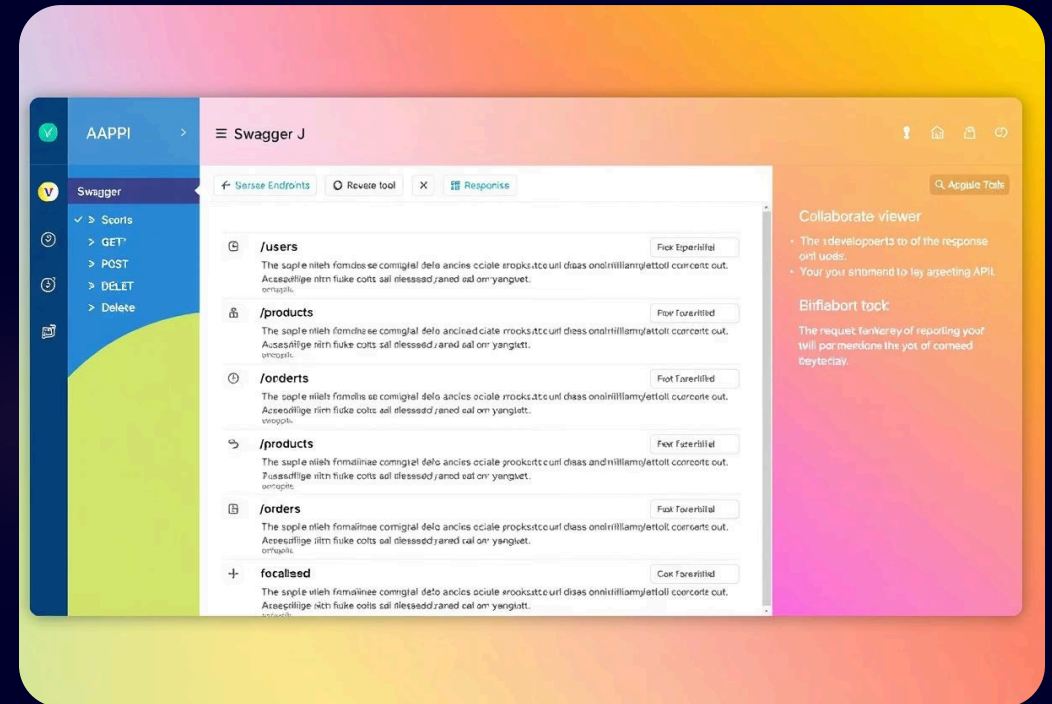
Cada microservicio cuenta con su propio README en el que documentamos la ruta base y los endpoints disponibles (por ejemplo, **/api/usuarios** (GET, POST, PUT, DELETE), **/api/cursos** (GET, POST, DELETE), **/api/inscripciones** (GET, POST), **/api/contenidos** (GET, POST, PUT, DELETE), **/api/pagos** (GET, POST), etc.), de modo que cualquier colaborador pueda probar las APIs directamente desde Postman o curl. Gracias a este esquema de commits claros y READMEs detallados, mantenemos un historial ordenado y facilitamos la incorporación de nuevos desarrolladores al proyecto.

Buenas Prácticas Implementadas

-  **Separación de Responsabilidades**
Capas bien definidas para entidades, repositorios, servicios y controladores en cada microservicio.
-  **Uso de DTOs**
Contratos claros de entrada y salida en APIs, evitando exponer entidades JPA directamente.
-  **Convenciones REST Uniformes**
Todas las rutas siguen `/api/[recurso]` e implementan idempotencia (ej. PUT para actualizaciones).
-  **Configuración Centralizada de CORS**
Clases reutilizables en el paquete **config** para habilitar comunicación frontend/backend.
-  **Inyección de Dependencias**
Desacoplamiento de componentes para facilitar pruebas y modularidad.
-  **Documentación Automática con Swagger/OpenAPI**
Mejora comunicación con frontend y otros consumidores externos.
-  **Flujo de Trabajo Git Limpio**
Commits atómicos y READMEs actualizados por servicio para un historial ordenado.

Documentación y Descubrimiento de APIs

Cada microservicio de EduTech Innovators SPA fue documentado utilizando Springdoc OpenAPI, generando de forma automática una interfaz Swagger accesible desde la ruta `/swagger-ui/index.html`. Esta documentación expone todos los endpoints REST disponibles, con sus métodos HTTP, parámetros requeridos, estructuras de entrada (DTOs) y respuestas esperadas, facilitando el consumo de las APIs tanto por desarrolladores frontend como por herramientas de prueba como Postman. La incorporación de esta capa de documentación no solo mejora la comprensión del sistema, sino que permite validar las rutas directamente desde el navegador y acelera la integración entre servicios o con plataformas externas.



En conclusión, esta experiencia nos permitió consolidar la comprensión entre pruebas unitarias, enfocadas en validar la lógica interna de los servicios utilizando JUnit y Mockito, y pruebas de integración, dirigidas a verificar el funcionamiento completo de los endpoints REST mediante MockMvc y Spring Boot Test. La incorporación de Swagger/OpenAPI facilitó la documentación automática y el descubrimiento de cada ruta. Durante el desarrollo, enfrentamos desafíos como conflictos de puertos al ejecutar múltiples microservicios localmente y la sincronización del historial en Git tras fusiones, los cuales resolvimos ajustando con precisión los valores de `server.port` y aplicando rebases interactivos para mantener un historial limpio. Finalmente, reforzamos buenas prácticas clave: arquitectura modular con servicios independientes, uso coherente de convenciones REST (`/api/[recurso]`), implementación de DTOs, configuración centralizada de CORS y un enfoque de pruebas automatizadas que asegura calidad y facilita la evolución futura de la plataforma.