### **🧩 Actividad 1.1: Explorando la arquitectura de microservicios**

**Parte de la Experiencia de Aprendizaje 1 del RA1 – Total: 2 sesiones**

### **🎯 Objetivo de la Actividad**

Sumergir a los estudiantes en los fundamentos de la arquitectura de microservicios, diferenciándose de otras arquitecturas como la monolítica, y comprender el rol del desarrollo Fullstack en este contexto. El trabajo se realiza de forma **colaborativa**, integrando investigación, análisis, presentación y reflexión.

### **📚 Recursos utilizados**

* **Presentaciones**:  
  + 1.1.1 Introducción a microservicios.pdf
  + 1.1.3 Desarrollo Fullstack.pdf
* **Actividades Guiadas**:  
  + 1.1.2 Exploración de arquitecturas de software.docx
  + 1.1.4 Análisis técnico a empresa tecnológica ML.docx
* **Video entrevista**: "La ENORME Infraestructura de MERCADO LIBRE" – Pelado Nerd

### 

### **🧠 Sesión 1: Introducción a Microservicios**

**Contenidos:**

* ¿Qué es una arquitectura de microservicios?
* Ventajas y desventajas frente a la arquitectura monolítica
* Eventos que impulsaron el cambio tecnológico (ej. pandemia)
* Casos reales de adopción (Mercado Libre)

**Actividad**:

* Investigaciones colaborativas sobre distintos tipos de arquitectura (monolítica, SOA, microservicios, multitenant, event-driven, micro frontends, etc.)
* Creación de tabla comparativa de términos tecnológicos encontrados en el video (API REST, Docker, Kubernetes, etc.)

### **🧠 Sesión 2: Desarrollo Fullstack**

**Contenidos:**

* ¿Qué es un desarrollador Fullstack?
* Diferencias entre frontend y backend
* Responsabilidades del backend en una arquitectura de microservicios (seguridad, escalabilidad, DevOps, pruebas, monitoreo)
* Uso de plataformas como roadmap.sh

**Actividad**:

* Reflexión grupal y preguntas orientadoras
* Presentación de términos clave por equipo (prohibido repetir término entre grupos)
* Apoyo mutuo en ronda de preguntas

### **📝 Evaluación**

* **Formativa 1**: Encargo con presentación y coevaluación entre pares
* Criterios vinculados a los IL1.1, IL1.2 e IL1.3:  
  + Investigación técnica clara y pertinente
  + Uso de herramientas colaborativas (Miro, Trello, Docs)
  + Discusión ética fundamentada

### **🚨 Consideraciones para los docentes**

* Deben **guiar activamente la investigación** y la **discusión ética**.
* Se espera **presentación creativa y colaborativa** con participación activa de todos los integrantes.
* Las sesiones deben cerrar con una **reflexión guiada** que conecte lo aprendido con los desafíos reales en proyectos de software.

### **🧩 Actividad 1.2: La era de los Frameworks**

**Duración total**: 2 sesiones  
 **Enlace al RA**: RA1 – *Analiza en equipo la arquitectura de microservicios y las herramientas involucradas.*

### **🎯 Objetivos de aprendizaje**

* Comprender los fundamentos de una **startup tecnológica**: roles, estructura y propósito.
* Analizar y comparar **lenguajes y frameworks** populares en la industria.
* Fomentar el **trabajo colaborativo**, la **toma de decisiones técnicas** y la documentación en contexto de proyectos reales.

### **🧠 Sesión 1: Emprendiendo en clases – Construyendo una Startup con Miro y Trello**

📌 Basada en:

* [1.2.1] Construyendo una Startup
* [1.2.2] Mi primera startup tecnológica
* [1.2.3] Acta de constitución
* [1.2.5] Minuta de reunión (Scrum meeting ejemplo)
* [1.2.6] Skills del equipo (perfil individual)

**Actividades:**

* Formar equipos de 3 estudiantes.
* Crear una startup ficticia en el ámbito tecnológico.
* Realizar lluvia de ideas con **Miro** para definir nombre y enfoque.
* Crear tablero **Trello** para distribuir tareas y roles.
* Documentar:  
  + **Acta de constitución** (nombre, misión, visión, valores, estructura organizacional, roles).
  + **Minuta de reunión tipo Scrum diario**.
  + **Documento de habilidades (skills)** por integrante del equipo.

**Entrega esperada:**

* Presentación breve del acta de constitución y estructura organizativa del equipo.

### 

### **🧠 Sesión 2: La era de los Frameworks**

📌 Basada en:

* [1.2.7] Presentación "La era de los Frameworks"
* [1.2.8] Actividad de investigación de frameworks

**Actividades:**

* Se asigna un **lenguaje de programación** a cada estudiante (ej: Java, Python, PHP…).
* Se investiga su **framework principal** (ej: Spring, Django, Laravel…).
* Cada estudiante analiza:  
  + Historia, sintaxis, paradigmas, ventajas/desventajas
  + Casos de uso, ejemplos de empresas reales
* Se realiza **discusión técnica grupal** para defender cuál framework sería mejor para determinados proyectos o servicios.
* Se entregan conclusiones en un informe colaborativo.

**Reflexión guiada:**

* ¿Quién decide qué framework usar?
* ¿Qué diferencia hay entre biblioteca y framework?
* ¿Qué frameworks dominan el mercado actual? (se apoya con informe de StackOverflow)

### **📝 Evaluación**

* La actividad es **formativa**, pero se recomienda realizar **coevaluación** entre equipos y presentación al curso para fomentar el aprendizaje activo.
* Se espera entrega formal de:  
  + Acta + minuta + documento de skills
  + Informe de frameworks

### **🧭 Recomendaciones para docentes**

* Acompañar activamente la **formación de equipos funcionales**.
* Promover el uso de herramientas colaborativas (Miro, Trello) en clase.
* Validar que los estudiantes **no repitan frameworks** en sus presentaciones.
* Incentivar el análisis técnico en la comparación entre lenguajes y frameworks.

### **🧩 Actividad 1.3: Elementos éticos en un desarrollo de software**

**Duración**: 1 sesión  
 **Contribuye a**: RA1 (Indicador IL1.3)

### **🎯 Objetivos de Aprendizaje**

* Comprender el **rol de la ética en el desarrollo de software**.
* Identificar y analizar incidentes éticos reales en empresas tecnológicas.
* Desarrollar pensamiento crítico sobre el impacto social, ético y técnico del software.

### **📚 Contenidos abordados**

Basado en la presentación [1.3.1]:

* Principios de ética: privacidad, inclusión, transparencia, equidad e impacto ambiental.
* Modelo CIA: confidencialidad, integridad y disponibilidad.
* Diferencias entre **punto de vista ético** y **técnico** de la seguridad en microservicios.

### 

### **📝 Actividad**

Basado en el documento [1.3.2]:

**Tarea del equipo**:

1. Seleccionar una empresa tecnológica de la lista sugerida (Google, Amazon, Microsoft, Uber, etc.).
2. Investigar un **incidente ético real** asociado a privacidad, seguridad, uso indebido de datos u otro dilema relevante.
3. Analizar el caso considerando:  
   * Descripción del incidente
   * Impacto generado
   * Respuesta de la empresa
   * Consecuencias
4. Elaborar un informe y **presentarlo al curso**.

**Ejemplo incluido**: Caso Facebook y Cambridge Analytica.

### **📌 Evaluación**

* Esta actividad es **formativa**, pero puede ser **coevaluada** por pares y complementada con preguntas de refl al finalizar la presentación.
* Relacionada con el **Indicador IL1.3** del RA1: *Evalúa los diferentes enfoques éticos en la construcción de componentes y los desafíos específicos en el desarrollo backend.*

### 

### **🧭 Recomendaciones para los docentes**

* Fomentar el debate crítico y ético durante las exposiciones.
* Validar que los estudiantes distingan entre la dimensión **técnica** y **moral** de las decisiones de diseñ.
* Enfatizar que el rol del desarrollador incluye **responsabilidad ética** ante el uso de datos y las consecuencias sociales del software.

### **📌 Evaluación Parcial 1 – Caso de análisis semestral**

**RA evaluado:** RA1  
 **Duración:** 7 horas pedagógicas  
 **Modalidad:** Trabajo en equipo (máx. 3 personas)  
 **Evaluación individual:** La presentación es evaluada individualmente  
 **Ponderación en la asignatura:** 30%

### **🧠 Objetivo general**

Analizar el caso semestral desde una mirada crítica y técnica, abordando la **transición de un sistema monolítico hacia una arquitectura de microservicios**, incluyendo los **aspectos éticos involucrados en el desarrollo backend**.

### **📘 Entregables**

1. **Informe (Entr de encargo - 40%)**
2. **Presentación individual (60%)**

### **✍️ Contenido obligatorio del informe**

1. **Definición del problema a resolver**
2. **Estrategia de microservicios propuesta**
3. **Herramientas y frameworks a utilizar (justificación técnica)**
4. **Uso de herramientas colaborativas (ej. Miro, Trello)**
5. **Análisis ético del caso (privacidad, equidad, impacto)**

**Anexos incluidos**:

* Documento de requisitos del sistema
* Análisis del sistema actual
* Diseño detallado de microservicios (casos de uso, clases, despliegue)
* Plan de migración y mitigación de riesgos

### **🎤 Contenido de la presentación individual**

* Exposición clara del problema y contexto
* Explicación de la estrategia de microservicios usada
* Justificación de las herramientas seleccionadas
* Descripción del trabajo colaborativo con herramientas colectivas
* Evaluación de los enfoques éticos adoptados

## **🎓 Para los Estudiantes**

### **🧩 Experiencia de Aprendizaje 1: Fundamentos de Microservicios**

¡Bienvenidos a su primera experiencia del semestre! En esta etapa inicial trabajarán **en equipos** para comprender cómo funciona una **arquitectura de microservicios**, en contraste con los sistemas tradicionales conocidos como **monolíticos**.

### **🔍 ¿Qué haremos?**

Explorarán de forma **colaborativa y práctica**:

* Qué son los microservicios y por qué las empresas los usan.
* Cuáles son sus ventajas y desafíos técnicos.
* Qué **herramientas** se utilizan para construirlos.
* Qué **decisiones éticas** deben considerar los desarrolladores de software.

### **🧠 ¿Cómo lo haremos?**

Tendrán **actividades guiadas** que incluyen:

* Investigar y comparar arquitecturas (monolítica vs. microservicios).
* Analizar casos reales (como Mercado Libre o Facebook).
* Crear su propia **startup tecnológica ficticia**.
* Presentar sus ideas y conclusiones al resto del curso.
* Reflexionar sobre temas como privacidad, equidad y responsabilidad social en el software.

### **📝 ¿Cómo se evaluará?**

Esta experiencia incluye una **evaluación formativa** y una **evaluación parcial**:

* **Encargo (informe grupal)**
* **Presentación individual** Donde demostrarán cómo abordaron el problema del caso y cómo aplicarían una solución basada en microservicios.

## **👨‍🏫 Para los Docentes**

### **🎯 Objetivo de la Experiencia de Aprendizaje 1**

El foco está en que los estudiantes **entiendan los fundamentos teóricos y éticos de los microservicios**, y que puedan plantear una **propuesta inicial de rediseño** de un sistema monolítico, con un **nivel básico de abstracción**, como parte de una metodología reflexiva previa al aprendizaje práctico de Spring Boot.

### **🧰 Actividades clave**

* **Actividad 1.1**: Introducción y exploración comparativa entre arquitecturas.
* **Actividad 1.2**: Creación de startup ficticia, definición de roles, uso de herramientas colaborativas (Trello, Miro).
* **Actividad 1.3**: Análisis de incidentes éticos en empresas tecnológicas (ej: Cambridge Analytica).

### **📊 Evaluación (Parcial 1 - 30%)**

**Entrega**:

* Informe grupal con análisis del sistema, propuesta de arquitectura básica, herramientas a usar, y plan de migración inicial.
* Reflexión ética integrada.

**Presentación**:

* Evaluación individual centrada en claridad, análisis técnico, uso de herramientas colaborativas y conciencia ética.

Esta primera experiencia no busca que los estudiantes programen microservicios aún, sino que **piensen el problema, planifiquen una solución y reflexionen éticamente**, para más adelante comparar con lo aprendido en Spring Boot y mejorar sus diseños.

## **🎓 Para los Estudiantes**

### **🧩 Experiencia de Aprendizaje 2: Desarrollo y Operación de Microservicios**

**Duración:** 42 horas  
 **Modalidad:** Taller de alto cómputo

### **🔍 ¿Qué aprenderás?**

En esta etapa te introducirás **de forma práctica y técnica** al desarrollo real de microservicios usando **Spring Boot, Maven, Git y bases de datos**. Aprenderás a construir la “base” de una aplicación moderna, desde su configuración inicial hasta la integración entre componentes.

### **⚙️ Tecnologías que usarás:**

* **Spring Boot** (framework para backend)
* **Maven** (gestión de dependencias)
* **JPA/ORM** (acceso a base de datos)
* **Git** (control de versiones)
* **Postman** (pruebas de APIs REST)

### **🧪 ¿Qué harás?**

* Crearás un proyecto desde cero.
* Configurarás tu entorno de trabajo con Spring y Maven.
* Utilizarás Git para versionar tu código.
* Construirás **microservicios funcionales que se comunican entre sí**.
* Implementarás **operaciones CRUD** conectadas a una base de datos real.
* Probarás los servicios con Postman.

### 

### **📝 Evaluaciones**

* **Evaluación formativa 2**: Práctica colaborativa en taller.
* **Evaluación Parcial 2 (35%)**:  
  + Entrega de código funcional (por equipo)
  + Presentación individual donde demostrarás lo que implementaste

## **👨‍🏫 Para los Docentes**

### **🎯 Objetivo de la EA2**

Profundizar en el **desarrollo técnico de microservicios**, configurando proyectos backend reales y conectándolos entre sí mediante **APIs RESTful**. Aquí comienza el trabajo codificado que pone en práctica los conceptos arquitectónicos y éticos planteados en la EA1.

### **📘 Resultados de Aprendizaje y sus Indicadores**

#### **RA2 – Configuración de proyectos backend**

* **IL2.1**: Diseña un proyecto base con Spring/Maven
* **IL2.2**: Configura e integra microservicios en entornos con control de versiones (Git)
* **IL2.3**: Aplica buenas prácticas de desarrollo colaborativo

#### **RA3 – Desarrollo e integración**

* **IL3.1**: Implementa operaciones CRUD con JPA/Postman
* **IL3.2**: Realiza integración RESTful entre microservicios

### **📊 Evaluación**

* **Formativa 2**: Coevaluación práctica (no sumativa)
* **Parcial 2 (35%)**:  
  + 50% ejecución funcional del proyecto (entrega por equipo)
  + 50% presentación individual con foco técnico y reflexivo

## **🧩 Actividad 2.1 – Fundamentos de HTTP y REST**

**Duración:** 2 sesiones  
 **Modalidad:** Taller práctico  
 **RA vinculados:** RA2 y RA3  
 **Evaluación:** Formativa (ejecución práctica y coevaluación)

### **🎓 Para Estudiantes**

### **🎯 ¿Qué aprenderás?**

* Cómo funcionan la Web y el protocolo HTTP.
* Qué es una API y cómo utilizar los métodos HTTP (GET, POST, PUT, DELETE…).
* Qué es REST y por qué es importante para el desarrollo de microservicios.
* Cómo usar **Postman** para probar servicios backend.

### **🧪 Actividades:**

#### **Sesión 1 – Descubriendo y probando APIs públicas**

* Investigar y probar **APIs públicas reales** usando Postman.
* Realizar llamadas con distintos métodos HTTP y documentar las respuestas.
* Analizar qué APIs siguen buenas prácticas REST y cuáles no.

#### **Sesión 2 – Análisis técnico de una empresa de delivery**

* Ver el video: *La infraestructura de Pedidos Ya* (canal Pelado Nerd).
* Identificar y definir tecnologías mencionadas (Kubernetes, Kafka, Load Balancer…).
* Completar tabla técnica comparativa con definiciones y enlaces.

📄 Herramientas y material de apoyo:

* Presentación: [Fundamentos de HTTP y REST – Buenas prácticas]
* Guía: [Cómo usar Postman]
* Actividades: [2.1.2 APIs públicas], [2.1.3 Empresa de delivery]

### **👨‍🏫 Para Docentes**

### **📘 Objetivo de la Actividad**

Introducir a los estudiantes en los **conceptos técnicos base que sustentan las APIs RESTful**, desde la teoría del protocolo HTTP hasta la práctica con herramientas profesionales (Postman). Esta actividad **prepara el terreno para desarrollar microservicios** que serán evaluados en la Parcial 2.

### **💡 Enfoque metodológico:**

* **Constructivismo + Práctica guiada**: Se parte de APIs reales para aprender desde la exploración.
* **Aprendizaje colaborativo**: Se fomenta compartir hallazgos y resultados entre pares.
* **Reflexión crítica**: Los estudiantes analizan qué prácticas son correctas o incorrectas en APIs reales.

### **📊 Criterios clave a observar:**

* Correcta identificación de métodos HTTP y uso de Postman.
* Capacidad para interpretar respuestas HTTP (códigos, estructuras).
* Comprensión de conceptos REST y arquitectura sin estado.
* Nivel de profundidad en la tabla técnica del análisis de Pedidos Ya.

## **🧩 Actividad 2.2 – Mi Primer Proyecto en Spring Boot**

**Duración**: 2 sesiones  
 **RA evaluados**: RA2 y RA3  
 **Modalidad**: Taller práctico guiado

### **🎓 Para Estudiantes**

### **🎯 ¿Qué vas a aprender?**

* Cómo crear tu primer proyecto Java usando **Spring Boot** y **Maven**.
* Qué es un **controlador** y cómo se usa para crear endpoints en una API REST.
* Cómo implementar rutas básicas (GET) para entregar resultados desde un backend.
* Cómo usar **Git y GitHub** para guardar tu avance y trabajar en equipo.

### **🧪 Actividades**

#### **📘 Sesión 1: Guía práctica – Hola Mundo con Spring Boot**

Basada en:

* [2.2.1] Presentación teórica
* [2.2.2] Guía paso a paso: *Mi primer proyecto en Spring Boot*

**Pasos clave:**

1. Crear un proyecto en [Spring Initializr](https://start.spring.io/)
2. Importarlo en el IDE (VSCode o IntelliJ)
3. Crear el paquete controller
4. Programar un controlador con un endpoint /hola
5. Ejecutar y probar tu backend en http://localhost:8080/hola

#### **💻 Sesión 2: Explorando el paquete controller y anotaciones Spring**

Basada en:

* [2.2.4] Actividad sobre controladores
* [2.2.3] Guía Git y GitHub

**Ejercicios clave:**

* Investigar anotaciones: @RestController, @GetMapping, @RequestMapping.
* Crear rutas básicas con parámetros.
* Implementar operaciones aritméticas con rutas como /api/suma/10/5.
* Crear 2 nuevos controladores: ClienteController y ProductoController.
* Usar listas en Java y lógica condicional para buscar productos (simulación CRUD).

**Entrega esperada:**

* Repositorio en GitHub con todo el avance.
* Presentación de las rutas y resultados de las pruebas.

### **👨‍🏫 Para Docentes**

### **🎯 Propósito de la Actividad**

Esta actividad introduce por primera vez el desarrollo real de microservicios en Spring Boot. El enfoque está en:

* Configurar correctamente el entorno.
* Comprender la estructura del proyecto Spring.
* Implementar rutas funcionales en Java.
* Fomentar buenas prácticas con control de versiones (Git/GitHub).

### **📊 Evaluación Formativa Recomendada**

* Observación directa del avance del código en clase.
* Verificación del uso de Git (commits, ramas).
* Discusión técnica de cada ruta implementada.
* Pruebas en vivo con Postman o navegador (localhost).

## **🧩 Actividad 2.3 – Interfaz RESTful sin base de datos**

**Duración estimada**: 2 sesiones  
 **Modalidad**: Taller guiado  
 **RA involucrados**: RA2 y RA3  
 **Evaluación**: Formativa mediante entrega funcional y pruebas con Postman

### **🎓 Para Estudiantes**

### **🎯 ¿Qué aprenderás?**

* Cómo implementar una **interfaz RESTful completa (CRUD)** en Spring Boot.
* Cómo usar la arquitectura **Controller – Service – Repository – Model (CSR/MVC)**.
* Cómo organizar tu proyecto en carpetas bien definidas.
* Cómo probar tus endpoints REST con **Postman**.
* Todo esto sin necesidad de una base de datos: los datos se almacenan en **arreglos de Java**.

### **🛠️ Actividades**

#### **📘 1. Estructura del proyecto**

📂 [2.3.1] Presentación sobre la arquitectura MVC y CSR

* Aprenderás qué hace cada capa: Model, Repository, Service, Controller.
* Definirás la estructura de carpetas en tu proyecto Spring Boot.
* Reflexionarás sobre conceptos como @RestController, @Service, @Repository.

#### **💻 2. Proyecto Biblioteca – Inventario de Libros**

📄 [2.3.2] Guía práctica paso a paso

Implementarás una aplicación CRUD para gestionar libros:

* Crear clase Libro con Lombok (@Data, @AllArgsConstructor, etc.)
* Implementar LibroRepository usando listas (sin DB).
* Crear LibroService y LibroController.
* Crear, leer, actualizar y eliminar libros usando Postman.

#### **📈 3. Reportes y lógica de negocio básica**

📄 [2.3.3] Proyecto Biblioteca – Reportes

Extenderás la aplicación con funcionalidades extra:

* Obtener el total de libros
* Buscar por ISBN, autor o año
* Ordenar libros por fecha
* Encontrar el más nuevo o más antiguo

Esto permite ejercitar el trabajo entre capas (Service → Repository).

#### **📚 4. Solicitud de libros (nueva entidad)**

📄 [2.3.4] Actividad – Solicitud de libros

Se crea una nueva clase Prestamo, que contiene:

* id\_prestamo, id\_libro, run\_solicitante, fecha\_solicitud, fecha\_entrega, cantidad\_dias, multas.

Se debe:

* Crear PrestamoModel, PrestamoRepository, PrestamoService y PrestamoController.
* Probar las rutas en Postman bajo /api/v1/solicitudes.

### **👨‍🏫 Para Docentes**

### **🎯 Propósito de esta actividad**

Esta actividad busca que los estudiantes:

* Comprendan la arquitectura en capas CSR/MVC.
* Implementen una API REST usando solo memoria (arrays), sin distracciones por configuración de DB.
* Apliquen **buenas prácticas de desarrollo RESTful** (nombres de rutas, métodos HTTP, respuestas).
* Refuercen el uso de Git y trabajo colaborativo.

### **🔍 Enfoques didácticos recomendados**

* Acompañamiento activo en Postman y estructuras de clases.
* Uso de Pair Programming para los métodos más complejos (como buscar por ID o parámetros).
* Evaluación informal mediante revisión en vivo del código y pruebas.
* Reflexión crítica al final: ¿qué pasará cuando usemos base de datos real?

## **🧩 Actividad 2.4 – Servicios RESTful conectados a Base de Datos**

**Modalidad**: Trabajo individual  
 **Evaluación**: Avance funcional + documentación técnica + presentación  
 **RA involucrados**: RA3 (Desarrollar microservicios CRUD conectados a datos reales)

### **🎓 Para Estudiantes**

### **🎯 ¿Qué aprenderás?**

Con esta actividad llevarás tu proyecto al siguiente nivel, conectando tu backend con una **base de datos real (MySQL)** y aplicando todo lo que has aprendido sobre APIs RESTful y microservicios.

### **🛠️ ¿Qué harás paso a paso?**

1. **Configurar la conexión MySQL**
   * Editar application.properties
   * Agregar credenciales, URL de conexión, y dialéctico MySQL
   * Añadir dependencias en pom.xml (spring-boot-starter-data-jpa, mysql-connector-java)
2. **Diseñar la base de datos relacional**
   * Identificar entidades y relaciones (1:N, N:M)
   * Definir estructura en diagrama y traducir a modelos Java
3. **Crear los modelos Java (entidades)**
   * Usar anotaciones @Entity, @Id, @GeneratedValue
   * Relacionar entidades con @ManyToOne, @OneToMany, @ManyToMany según corresponda
4. **Implementar los repositorios**
   * Interfaces que extienden JpaRepository
   * Métodos personalizados opcionales para búsquedas específicas
5. **Construir los servicios y controladores REST**
   * Lógica en @Service para CRUD y relaciones
   * Rutas en @RestController con @GetMapping, @PostMapping, etc.
6. **Probar con Postman y documentar la API**
   * Validar operaciones y relaciones
   * Preparar una **presentación técnica final** del funcionamiento del sistema con BD

### 

### **👨‍🏫 Para Docentes**

### **🎯 Propósito didáctico**

Esta actividad busca consolidar la transición desde microservicios simulados (con listas) hacia una **API funcional respaldada por MySQL**, con el uso completo de las capas de arquitectura y el ecosistema de Spring Boot.

### **🔍 Enfoque recomendado**

* Supervisar **estructura correcta del proyecto** (model, repository, service, controller)
* Evaluar el uso adecuado de **anotaciones JPA** y diseño de relaciones
* Observar el uso **funcional de Postman** y el registro correcto en base de datos
* Validar que la presentación incluya:  
  + Diagrama relacional inicial
  + Explicación del modelo en Java
  + Pruebas reales con datos persistidos
  + Reflexión del estudiante sobre lo aprendido

## **📘 Formato del Informe – Evaluación Parcial 2**

**Asignatura:** DSY1103 – Desarrollo Full Stack I  
 **Evaluación Parcial 2** – *35% del curso* **Tipo de trabajo:** Grupal (máx. 3 integrantes)  
 **Evaluación del encargo: grupal (40%)** **Evaluación de la presentación: individual (60%)**

### **🧩 Estructura del Informe (Entrega de Encargo – 40%)**

#### **1. Portada**

* Nombre del equipo y sus integrantes
* Nombre del caso (Perfulandia, EduTech, EcoMarket)
* Fecha de entrega
* Enlace al repositorio GitHub del proyecto

#### **2. Desarrollo de Microservicios**

* Breve descripción del sistema general
* ¿Qué microservicios han desarrollado hasta ahora?
* Estructura del proyecto (CSR): modelos, controladores, servicios y repositorios
* Imagen del diagrama o arquitectura

#### **3. Base de Datos**

* Conexión a MySQL documentada (application.properties)
* Modelo entidad-relación implementado
* Relaciones entre tablas (1:N, N:M) con capturas de código (@OneToMany, etc.)

#### **4. Configuración Técnica**

* pom.xml con dependencias utilizadas
* Configuración de entorno (IDE, Git, GitHub, Spring Boot)
* Estructura de carpetas del proyecto

#### **5. Integración de Servicios y API**

* Enlaces funcionales de endpoints
* Documentación de rutas implementadas (GET, POST, PUT, DELETE)
* Evidencia de pruebas con Postman (capturas de solicitud/respuesta)
* Descripción breve del flujo de integración entre microservicios

#### **6. Buenas prácticas y colaboración**

* Uso de Git: ramas, commits, pull requests, integraciones
* Análisis de buenas prácticas aplicadas (modularidad, legibilidad, RESTful)

#### **7. Pruebas Unitarias**

* Descripción de test realizados (JUnit u otros)
* Resultados de las pruebas

#### **8. Conclusión**

* Qué se ha logrado y qué falta por desarrollar
* Dificultades enfrentadas y cómo las resolvieron

#### **9. Anexos**

* Enlace al repositorio GitHub
* Enlace a la base de datos (si aplica)
* Evidencias de ejecución (Postman, consola, entorno)

## **🎤 Guía para la Presentación (Evaluación Individual – 60%)**

Cada estudiante debe responder y presentar:

1. **Diseño general del proyecto Spring**
2. **Uso de Maven (gestión de dependencias y ciclo de vida del proyecto)**
3. **Línea base de trabajo (estructura del proyecto, componentes backend)**
4. **Configuración del proyecto y uso de Git (flujo de trabajo)**
5. **Explicación de la arquitectura interna de los microservicios**
6. **Buenas prácticas aplicadas (modularidad, escalabilidad, mantenibilidad)**
7. **Desarrollo con Spring y Maven (ejemplos concretos)**
8. **Control de versiones colaborativo (evidencia de Git)**
9. **Implementación de operaciones CRUD (con JPA/Hibernate y Postman)**
10. **Comunicación RESTful entre microservicios (flujo técnico)**
11. **Pruebas con Postman (demostración real)**
12. **Reflexión personal sobre el avance del caso**

## **🧩 Experiencia de Aprendizaje 3 – Testing, Documentación y Calidad**

**Duración**: 42 horas  
 **Ambiente**: Taller de alto cómputo  
 **RA involucrados**: RA4 (Pruebas unitarias) y RA5 (Documentación con OAS y HATEOAS)

### **🎓 Para Estudiantes**

### **🎯 ¿Qué aprenderás?**

En esta experiencia aprenderás a **asegurar la calidad técnica de tus microservicios** mediante:

* **Pruebas unitarias** usando **JUnit**
* **Documentación de APIs RESTful** con **OpenAPI Specification (OAS)**, como Swagger o Springdoc
* Implementación de **HATEOAS** para facilitar la navegabilidad de tu API
* Despliegue (teórico o básico) de tu API documentada para que otros desarrolladores puedan consumirla

### **🛠️ ¿Qué herramientas usarás?**

* **JUnit 5** para pruebas unitarias
* **Springdoc OpenAPI** o Swagger UI
* **Spring HATEOAS**
* (Opcional) **Postman** para validar y consumir la API
* (Teórico/práctico) Despliegue en servicios cloud (por ejemplo, Render, Railway o localhost documentado)

### **📝 ¿Cómo se evaluará?**

* **Formativa 3**: Encargo parcial de documentación o testeo
* **Parcial 3 (35%)**: Entrega de encargo con presentación (evaluación individual)

### **👨‍🏫 Para Docentes**

### **🎯 Objetivo didáctico**

Guiar a los estudiantes en la **incorporación de buenas prácticas de calidad** en el desarrollo backend. Esta experiencia permite:

* Verificar la cobertura de pruebas básicas en componentes críticos
* Validar que la API esté **documentada y navegable**
* Introducir buenas prácticas de despliegue y exposición de servicios

### **📘 Resultados de Aprendizaje e Indicadores**

#### **RA4 – Pruebas Unitarias**

* **IL4.1**: Selección justificada de herramientas para testing
* **IL4.2**: Implementación de pruebas unitarias en JUnit para validar métodos de servicio

#### **RA5 – Documentación y Navegabilidad**

* **IL5.1**: Uso de OpenAPI para describir endpoints
* **IL5.2**: Implementación de HATEOAS para navegación entre recursos
* **IL5.3**: Despliegue de la API o documentación en un entorno comprensible para terceros

### **📊 Evaluación**

**Parcial 3 – 35% del curso**

* **Entrega de encargo grupal**
* **Presentación individual (obligatoria)**

## **🧪 Actividad 3.1 – Pruebas Unitarias en Microservicios**

### **🎯 Objetivo**

Desarrollar y aplicar pruebas unitarias en componentes backend utilizando **JUnit 5**, complementado con **Mockito** y la base de datos en memoria **H2**, para asegurar la calidad del código en los microservicios desarrollados.

## **🎓 Para Estudiantes**

### **🛠️ ¿Qué aprenderás?**

* Qué son y para qué sirven las pruebas unitarias.
* Cómo utilizar **JUnit 5** para validar métodos en servicios y controladores.
* Cómo utilizar **Mockito** para simular dependencias.
* Cómo configurar un entorno de prueba utilizando una base de datos **H2**.
* Cómo ejecutar los test con Maven y analizar los resultados.
* Cómo documentar tus pruebas.

### **📚 Actividades y Recursos**

#### **1. Fundamentos de pruebas unitarias**

📄 [3.1.1] Presentación “Pruebas Unitarias en Proyectos de Microservicios”  
 Incluye:

* Beneficios del testing
* Introducción a JUnit y Mockito
* Rol de las pruebas en CI/CD

#### 

#### **2. Pruebas prácticas en Spring Boot**

📘 [3.1.2] Guía DataFaker

* Crear un DataLoader que genera datos de prueba automáticamente al iniciar la app en modo desarrollo (perfil dev)
* Cargar entidades como Estudiante, Carrera, Reserva, Sala

📘 [3.1.3] Análisis a los test

* Crear test unitarios de servicios como CarreraService, EstudianteService
* Analizar pruebas existentes y completar preguntas guiadas
* Crear archivos application-test.properties para separar ambientes
* Ejecutar los test con Maven (mvn test)

#### **3. Actividad de análisis técnico**

📘 [3.1.4] Análisis técnico a una empresa tecnológica

* Video: Entrevista a equipo DevOps de RAPPI (Pelado Nerd)
* Investigar tecnologías mencionadas (Kubernetes, Kafka, etc.)
* Completar tabla con definiciones y compartir hallazgos

### **📦 Entregables**

* Pruebas unitarias de al menos 2 servicios
* Capturas de ejecución exitosa con mvn test
* Documentación con:  
  + Objetivo del test
  + Anotaciones utilizadas
  + Resultados y mejoras sugeridas
* Tabla técnica con términos del video RAPPI

## **👨‍🏫 Para Docentes**

### **🎯 Enfoque didáctico**

Esta actividad permite que los estudiantes pasen de un desarrollo funcional a un desarrollo confiable. Se recomienda:

* **Supervisión del entorno de test** con H2 y perfiles separados (dev / test)
* **Validación de pruebas reales**, no solo teóricas
* Revisión de:  
  + Uso correcto de @Test, @Mock, @Autowired
  + Separación clara entre pruebas unitarias y de integración
* Promover reflexión: ¿Qué pasa si no hay tests? ¿Qué código se podría romper?

## **🧩 Actividad 3.2 – Documentación de Microservicios con OpenAPI Specification (OAS)**

### **🎯 Objetivo**

Que los estudiantes aprendan a **documentar correctamente sus microservicios** usando herramientas basadas en **OpenAPI**, como **Swagger UI** o **Redoc**, generando una documentación estándar, automática y navegable para cualquier consumidor de la API.

## **🎓 Para Estudiantes**

### **🧠 ¿Qué aprenderás?**

* Qué es **OpenAPI Specification (OAS)** y por qué es importante en el desarrollo de APIs.
* Cómo integrar **Swagger** en un proyecto Spring Boot usando **Springdoc**.
* Cómo describir de forma técnica y visual los **endpoints, parámetros, respuestas, cuerpos de solicitud y modelos de datos**.
* Cómo personalizar y compartir tu documentación desde tu red local.

### **📚 Materiales y recursos**

#### **✅ Fundamento teórico**

📘 [3.2.1] Presentación "Documentación con OpenAPI Specification"

* ¿Qué es OAS?
* Beneficios: documentación estándar, interoperabilidad, automatización
* Introducción a Swagger y cómo usarlo

#### **✅ Implementación práctica**

📘 [3.2.2] Guía: Proyecto Biblioteca – Swagger  
 Pasos clave:

1. Agregar dependencia de springdoc-openapi en pom.xml.

Configurar application.properties con rutas como:

springdoc.api-docs.enabled=true

springdoc.swagger-ui.enabled=true

springdoc.swagger-ui.path=/doc/swagger-ui.html

1. Crear clase de configuración SwaggerConfig.
2. Documentar los endpoints con anotaciones como:  
   * @Tag, @Operation, @ApiResponse, @RequestBody, @Schema, @ExampleObject.

#### **✅ Publicación en red local (LAN)**

📘 [3.2.3] Guía: Compartir de forma LAN

* Cambiar server.address=0.0.0.0 y mantener server.port=8080.
* Acceder a la documentación desde otra máquina de la red usando IP:  
   http://<IP\_LOCAL>:8080/doc/swagger-ui/index.html

### 

### **📝 Entregables**

* Proyecto Spring Boot con Swagger integrado.
* Documentación de todos los endpoints implementados en la Parcial 2.
* Capturas de Swagger UI navegando los endpoints.
* Documentación escrita (PDF/Word) que incluya:  
  + Explicación de qué es OAS.
  + Cómo se implementó y configuró en el proyecto.
  + Ejemplos concretos de anotaciones usadas.
  + Imagen o acceso a la URL local del Swagger UI.

## **👨‍🏫 Para Docentes**

### **🎯 Propósito de la actividad**

Formar en los estudiantes una cultura de documentación técnica clara, replicable y profesional, promoviendo la comprensión de los estándares de la industria actual.

### **📌 Enfoques sugeridos**

* Validar implementación funcional de Swagger/OpenAPI.
* Verificar que cada endpoint esté documentado (método, parámetros, respuestas).
* Fomentar el uso de anotaciones descriptivas y ejemplos en modelos (@Schema, @ExampleObject).
* Si es posible, invitar a los estudiantes a compartir su documentación con compañeros en LAN.

## **🧩 Actividad 3.3 – Implementación de HATEOAS**

### **🎯 Objetivo**

Aprender a implementar **HATEOAS (Hypermedia as the Engine of Application State)** en una API RESTful para enriquecer las respuestas con enlaces navegables y hacer la API **autodescriptiva, escalable y fácil de usar**.

## **🎓 Para Estudiantes**

### **🧠 ¿Qué aprenderás?**

* Qué es **HATEOAS** y su rol dentro de una arquitectura RESTful.
* Cómo usar **Spring HATEOAS** para incluir enlaces (\_links) en las respuestas de tu API.
* Cómo estructurar controladores versión 2 (ControllerV2) para diferenciar endpoints con y sin HATEOAS.
* Cómo desarrollar y documentar enlaces dinámicos que permitan navegar tu API sin conocer previamente todas las rutas disponibles.

### **📚 Materiales y Recursos**

#### **✅ Fundamento teórico**

📘 [3.3.1] Presentación “Implementación de HATEOAS en la Documentación de APIs”  
 Incluye:

* Qué es HATEOAS
* Cómo funciona en una API RESTful
* Beneficios: hipertexto navegable, autodescripción, descubrimiento de acciones

#### 

#### **✅ Guía práctica**

📘 [3.3.2] Guía y ejercicios – Proyecto Bib【169†soos clave:\*\*

1. **Agregar dependencia** en pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-hateoas</artifactId>

</dependency>

1. **Crear ensambladores** (por ejemplo, CarreraModelAssembler) en el paquete assemblers  
   * Encapsulan la lógica para agregar enlaces a las entidades.
2. **Controladores versión 2 (V2)**
   * Implementan HATEOAS en rutas como /api/v2/carreras
   * Devuelven respuestas con \_links y posibles acciones
3. **Probar con Postman o Swagger**
   * Confirmar que las respuestas REST ahora contienen enlaces a otras acciones

### 

### **🧪 Ejercicios prácticos**

* Implementar al menos 5 endpoints personalizados que devuelvan enlaces navegables, como:  
  + Reservas por fecha
  + Reservas por estado
  + Total de reservas por estudiante
  + Reservas entre fechas, etc.

**Desafío adicional:** Desarrollar 5 reportes más con criterios compuestos (ver guía) y validar su respuesta HATEOAS.

### **📝 Entregables**

* Proyecto Spring Boot con **HATEOAS implementado**
* Código fuente con:  
  + Eambladores (ModelAssembler)
  + Controladores V2 con respuestas enriquecidas
* Evidencia en Postman o Swagger con \_links
* Informe que incluya:  
  + ¿Qué es HATEOAS?
  + Cómo se implementó (capturas + código)
  + Ejemplos de enlaces generados y su utilidad
  + Reflexión sobre cómo esto mejora la experiencia del consumidor de API

## **👨🏫 Para Docentes**

### **🎯 Propósito de esta actividad**

Introducir a los estudiantes en una de las prácticas más avanzadas y profesionales del desarrollo backend: **respuestas enriquecidas con hipermedios**, que favorecen la **descubribilidad de recursos y acciones**.

### **📌 Recomendaciones para evaluación**

* Validar que los ensambladores (ModelAssembler) se utilizan correctamente
* Verificar los endpoints V2 devuelvan respuestas con campos \_links
* Confirmar que los enlaces generados sean funcionales y útiles
* Incentivar el uso de respuestas JSON documentadas visualmente en Swagger o en Postman

## **📘 Evaluación Parcial 3 – Calidad, Pruebas y Documentación**

**Ponderación**: 35% del curso  
 **Duración**: 7 horas pedagógicas  
 **Tipo de trabajo**: Grupal (máximo 3 estudiantes por equipo)  
 **Evaluación de presentación**: Individual  
 **Ambiente**: Taller de alto cómputo  
 **RA evaluados**:

* **RA4**: Pruebas unitarias para asegurar calidad
* **RA5**: Documentación técnica y navegabilidad de APIs

### **🎯 Objetivo General**

Consolidar el proyecto iniciado en las evaluaciones anteriores asegurando su **calidad técnica, trazabilidad y comprensión** a través de:

* Pruebas unitarias con JUnit
* Documentación de endpoints con OpenAPI (Swagger)
* Enriquecimiento de la API con enlaces navegables (HATEOAS)
* Despliegue teórico en un entorno cloud (como Render o Railway)

Esta evaluación representa la **última etapa del desarrollo progresivo del caso semestral**, y permite evaluar la madurez técnica y documental alcanzada por los estudiantes.

### 

### 

### **📝 ¿Qué deben entregar?**

#### **✅ 1. Código del proyecto**

* Microservicios implementados desde la Parcial 2
* Pruebas unitarias con JUnit
* Anotaciones de documentación con Swagger / Springdoc
* Implementación de respuestas HATEOAS en controladores versión 2

#### **✅ 2. Informe técnico (entrega grupal – 40%)**

Estructura:

* Documentación del sistema (diagramas, Swagger)
* Plan de pruebas y resultados (JUnit, cobertura básica)
* Documentación HATEOAS
* Caso hipotético de despliegue en nube (teórico)
* Conclusiones y anexos

#### **✅ 3. Presentación individual (60%)**

Cada estudiante debe demostrar dominio de:

* Las herramientas de testeo usadas y su utilidad
* Las pruebas implementadas y sus resultados
* Cómo se documentaron los endpoints con OAS
* Cómo se implementaron los enlaces HATEOAS
* Qué incluiría un despliegue en nube
* Qué aprendió sobre calidad y mantenimiento

### 

### **📌 Puntos importantes para estudiantes**

* La calidad del código ahora **se mide con pruebas y documentación**, no solo con funcionalidad.
* La documentación debe ser **navegable y clara**, como si otro desarrollador usara tu API.
* El despliegue **no se ejecuta realmente**, pero debes saber cómo lo harías.
* El foco está en dejar tu backend **listo para producción** (aunque en versión formativa).

### **👨‍🏫 Puntos clave para docentes**

* Verificar uso correcto de JUnit y organización del paquete test
* Validar Swagger documentado con anotaciones y rutas completas
* Confirmar la presencia de ModelAssembler y controladores V2 con \_links
* Valorar el nivel de comprensión en la presentación oral individual
* No exigir despliegue real, pero sí comprensión clara de cómo se haría