# **Informe de Proyecto - Desarrollo Backend con Microservicios en Spring Boot**

## **1. Herramientas utilizadas y su justificación**

Para el desarrollo del backend del proyecto **EcoMarket SPA**, se utilizaron diversas herramientas que cumplieron un rol fundamental en la implementación de una arquitectura basada en microservicios. A continuación, se detalla cada una de ellas, junto con su funcionalidad específica y su aporte al desarrollo:

* **Java 17:** Lenguaje base para toda la lógica del proyecto, proporcionando robustez y compatibilidad con las librerías modernas.
* **Spring Boot 3:** Framework principal para el desarrollo del backend. Permite una configuración mínima y una estructura estandarizada para servicios RESTful. Facilitó la creación rápida de controladores, servicios y entidades, manteniendo una arquitectura desacoplada y escalable.
* **Spring Web:** Utilizado para la creación de endpoints REST y manejo de peticiones HTTP.
* **Spring Data JPA:** Facilita la conexión entre la capa de persistencia y la base de datos mediante la abstracción de operaciones CRUD, evitando la necesidad de escribir consultas SQL manualmente.
* **Spring Security:** Implementado para la gestión de autenticación y autorización (según alcance del proyecto).
* **Maven:** Herramienta para la gestión de dependencias, compilación, pruebas y empaquetado del proyecto. Su configuración mediante pom.xml permitió agregar librerías clave y gestionar el ciclo de vida completo.
* **JUnit 5 + Mockito:** Herramientas para la creación de pruebas unitarias, simulando el comportamiento de repositorios y validando el correcto funcionamiento de los servicios.
* **Postman:** Utilizada para probar manualmente los endpoints REST, validando las respuestas del backend y los códigos HTTP correspondientes.
* **Git y GitHub:** Para control de versiones, permitiendo mantener un historial organizado de cambios y facilitando la colaboración y respaldo remoto.
* **Base de datos (MySQL / H2):** MySQL para el entorno de producción y H2 como base de datos en memoria para pruebas y desarrollo local.

## **2. Enfoques éticos y desafíos en desarrollo backend**

Durante la construcción del backend, se consideraron aspectos éticos fundamentales:

* **Privacidad de datos:** Aunque el proyecto es una simulación, se aplicaron buenas prácticas como validación de entradas y manejo seguro de identificadores para evitar fugas de información.
* **Responsabilidad en el diseño:** Se implementaron mensajes de error claros y controlados para mejorar la experiencia del usuario y evitar respuestas inesperadas o inseguras.
* **Desafíos comunes:** Mantener la consistencia de los datos, respetar los principios SOLID para una arquitectura limpia y desacoplada, y asegurar la robustez del API frente a errores o condiciones atípicas.

## **3. Configuración y diseño del proyecto con Spring**

El proyecto está organizado en capas bien definidas:

* **Modelo (model):** Define las entidades persistentes como Producto y Proveedor.
* **Repositorio (repository):** Interfaces que extienden JpaRepository, encargadas de la interacción con la base de datos.
* **Servicio (service):** Contiene la lógica de negocio, validaciones y procesamiento de datos.
* **Controlador (controller):** Expone los endpoints REST que permiten consumir los servicios desde clientes externos.

La configuración de Spring Boot se realiza mediante el archivo application.properties, donde se definen parámetros del servidor y la conexión a la base de datos.

## **4. Uso de Maven para la gestión del ciclo de vida**

Maven se utilizó para:

* Declarar las dependencias necesarias en pom.xml.
* Ejecutar comandos como mvn clean install y mvn spring-boot:run para compilar, testear y levantar el servidor.
* Gestionar perfiles y propiedades para diferentes entornos.

## **5. Pruebas unitarias con JUnit y Mockito**

Se crearon pruebas unitarias para validar funcionalidades clave:

* Guardar entidades y verificar que los datos sean correctos.
* Obtener por ID, tanto en casos existentes como no existentes.
* Actualizar y eliminar entidades, validando comportamientos esperados.
* Simular el comportamiento de los repositorios con Mockito, evitando dependencia de la base de datos en las pruebas.

## **6. Validación y consumo de servicios con Postman**

Postman permitió realizar pruebas manuales de los endpoints con distintos métodos HTTP:

* GET: Listados y consultas específicas.
* POST: Creación de nuevas entidades.
* PUT: Actualización de entidades existentes.
* DELETE: Eliminación por ID.

Se comprobó que el backend responde con los códigos HTTP apropiados y con la información esperada.

## **7. Arquitectura interna de los microservicios**

El diseño sigue una arquitectura modular y escalable, separando responsabilidades claramente y permitiendo en un futuro la división en servicios independientes. La comunicación entre capas se realiza con bajo acoplamiento, favoreciendo la mantenibilidad.

## **8. Listado de tecnologías usadas y dónde se usan (clases o paquetes principales)**

| **Tecnología** | **Funcionalidad principal** | **Ubicación o ejemplo de clases** |
| --- | --- | --- |
| Java 17 | Lenguaje base para toda la lógica | Todo el proyecto (paquetes model, service, controller) |
| Spring Boot 3 | Framework para microservicios REST | Clase principal (@SpringBootApplication), controladores, servicios |
| Spring Web | Creación de endpoints REST y manejo HTTP | ProductoController, ProveedorController, etc. |
| Spring Data JPA | Persistencia y manejo ORM con base de datos | ProductoRepository, ProveedorRepository, modelos |
| Spring Security | Seguridad y autorización | Configuración en clases de seguridad (si implementado) |
| Maven | Gestión de dependencias y ciclo de vida | Archivo pom.xml y scripts de compilación |
| JUnit 5 + Mockito | Pruebas unitarias y simulación de dependencias | Clases de test en src/test/java |
| Postman | Pruebas manuales de endpoints REST | No está en el código, pruebas externas |
| Git / GitHub | Control de versiones y respaldo remoto | Repositorio completo en GitHub |
| MySQL / H2 | Base de datos para persistencia | Configuración en application.properties, entidades modelo |