

Guía de aprendizaje

El desarrollo de un microservicio

Exp 2 – Semana 4

Desarrollo Full Stack I (DSY2201)

***Facilitador disciplinar:*** Víctor Rosendo.

***Asesor par:*** Alonso Castillo.

# Índice

[Índice 2](#_Toc151546631)

[Introducción a la semana 3](#_Toc151546632)

[Resultado de aprendizaje 4](#_Toc151546633)

[Conceptos relevantes 5](#_Toc151546634)

[Preguntas activadoras 5](#_Toc151546635)

[Actividad 6](#_Toc151546636)

[Comunicación RESTful 7](#_Toc151546637)

[¿Qué es la comunicación RESTful? 7](#_Toc151546638)

[Diseño de endpoints RESTful 7](#_Toc151546639)

[Uso de anotaciones en Spring Boot 7](#_Toc151546640)

[Anexando Base de Datos al microservicio en Spring Boot 8](#_Toc151546641)

[Configuración de la Base de Datos Oracle 9](#_Toc151546642)

[Codificación del microservicio 10](#_Toc151546643)

[Links de Interés 17](#_Toc151546644)

[Lectura complementaria 17](#_Toc151546645)

[Cierre de la semana 17](#_Toc151546646)

[Referencias 19](#_Toc151546647)

[Lecturas de la semana 19](#_Toc151546648)

[Apuntes 20](#_Toc151546649)

# Introducción a la semana

En esta guía, exploraremos en detalle los conceptos y pasos esenciales para construir y desplegar microservicios eficientes y escalables utilizando el Framework Spring Boot. Todo esto lo lograremos explicando conceptos clave como la comunicación RESTful, el acceso a bases de datos Oracle con JPA y la inyección de dependencias en Spring Boot. Luego, te sumergirás en la creación de proyectos Spring Boot, la definición de endpoints RESTful y la implementación de operaciones CRUD.

# Resultado de aprendizaje

### El estudiante será capaz de:

**RA2.** Desarrolla componentes de microservicio creativos, conectándolos entre sí y accediendo a base de datos usando tecnologías del framework de backend, para realizar operaciones CRUD con datos enviados desde frontend.

### Indicador de logro:

**IL4.**  Desarrolla Java sobre Springboot generando componentes que utilizan Comunicación RESTful.

**IL5.**  Genera códigos que se comuniquen con las BBDD que componen el backend con framework ORM/JPA.

**IL8.** Aplica las operaciones en un repositorio de trabajo colaborativo generado durante el desarrollo de la aplicación.

# Conceptos relevantes

|  |  |  |
| --- | --- | --- |
| Microservicios | Spring Boot | JAVA |
| GIT | API | RestFul |
| Control de Versiones | Repositorio | Base de Datos |

# Preguntas activadoras

* ¿Es posible integrar persistencia de datos en nuestros microservicios?
* ¿Cómo podemos generar comunicación externa para manipular información en nuestro microservicio?
* ¿Es posible seguir trabajando de manera colaborativa cuando se incluye una Base de datos a nuestro microservicio?

# Actividad

### Descripción de la actividad

En esta cuarta semana se realizará la actividad formativa 3 de manera individual; a través de un encargo deberán hacer la adaptación del microservicio simple codificado en la actividad formativa 2 (semana 2) y realizar un CRUD simple de comunicación con la Base de Datos Oracle sobre al menos una tabla.

# Comunicación RESTful

Los microservicios suelen exponer una API RESTful para interactuar con otros servicios y clientes. Los componentes clave de una API RESTful incluyen:

* **Recursos**: Representan objetos o datos a los que se accede a través de la API.
* **URLs**: Cada recurso se identifica mediante una URL única.
* **Métodos HTTP**: Los métodos como GET, POST, PUT y DELETE se utilizan para realizar operaciones en los recursos.
* **Respuestas HTTP**: Las respuestas incluyen códigos de estado y datos, generalmente en formato JSON o XML.

## ¿Qué es la comunicación RESTful?

La comunicación RESTful se basa en REST (Representational State Transfer) y utiliza HTTP para intercambiar datos entre sistemas. En una API RESTful, los recursos se representan mediante URLs y se acceden a través de métodos HTTP (GET, POST, PUT, DELETE). Cada recurso tiene un estado y se puede acceder a través de su URI.

## Diseño de endpoints RESTful

* Utiliza sustantivos en plural para los nombres de los recursos (por ejemplo, /usuarios).
* Utiliza métodos HTTP para definir las operaciones (GET para recuperar, POST para crear, PUT para actualizar, DELETE para eliminar).
* Utiliza parámetros de consulta para filtrar datos (por ejemplo, /usuarios?nombre=Juan).

## Uso de anotaciones en Spring Boot

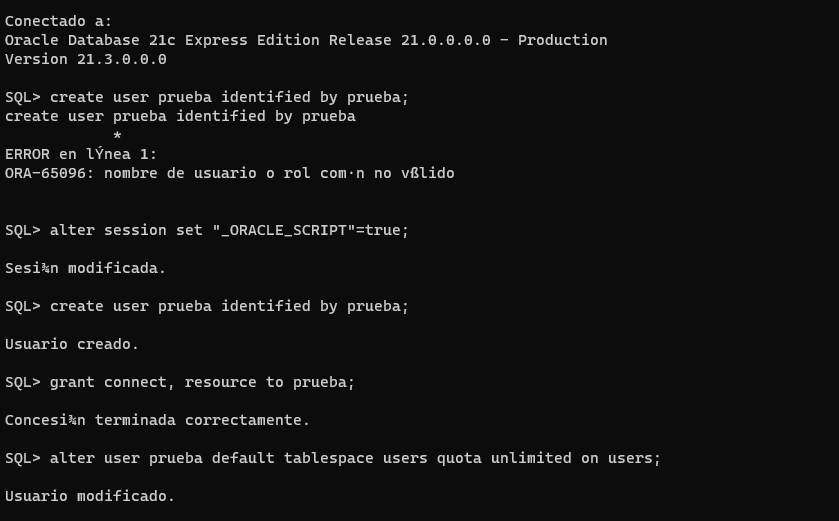
Spring Boot facilita la creación de servicios RESTful mediante anotaciones como @RestController, @GetMapping, @PostMapping, etc. Estas anotaciones se utilizan para definir controladores que gestionan las solicitudes HTTP.

# Anexando Base de Datos al microservicio en Spring Boot

Para comunicarse con una base de datos utilizando JPA, primero debemos configurar la conexión a la base de datos y definir nuestras entidades. Asegúrate de tener una base de datos disponible y los detalles de conexión. Recuerda que usaremos Oracle como manejador de base de datos y se recomienda usar la versión 19 o superior.

**Figura 1**

*Detalles de conexión Oracle.*



*Nota.* Se muestra un print de pantalla de la visualización de la conexión Oracle. SQL Plus (2023).

## Configuración de la Base de Datos Oracle

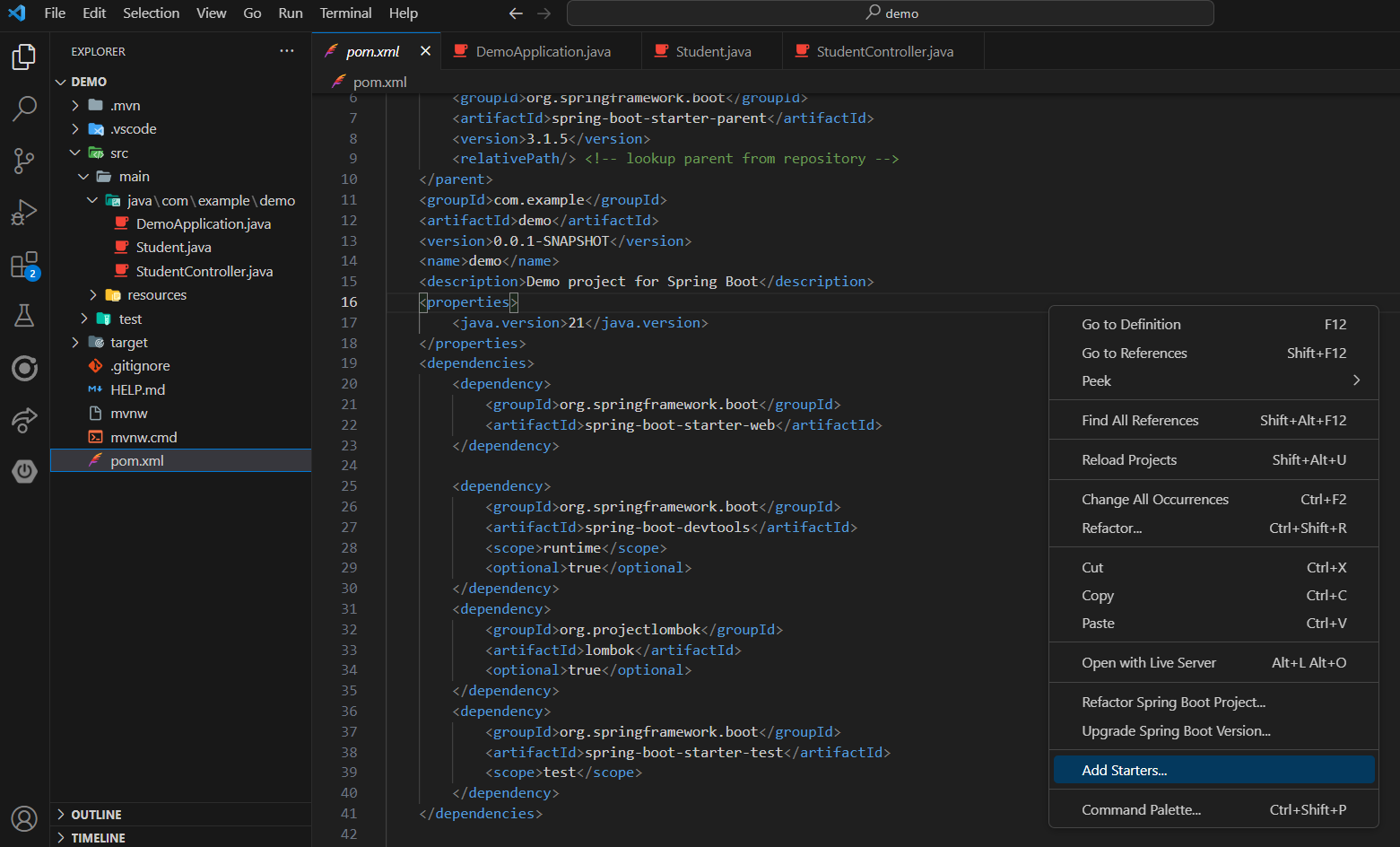
Asegúrate de incluir las dependencias Oracle Driver y Spring Data JPA. Si ya has creado el proyecto puedes incluir nuevas librerías seleccionando el archivo pom.xml, clic derecho add starters y agregar las nuevas.

Importante

En el caso que estes iniciando uno nuevo, asegúrate de seguir las instrucciones de creación del proyecto de la guía de aprendizaje de la semana 2.

**Figura 2**

*Agregar nuevas dependencias.*

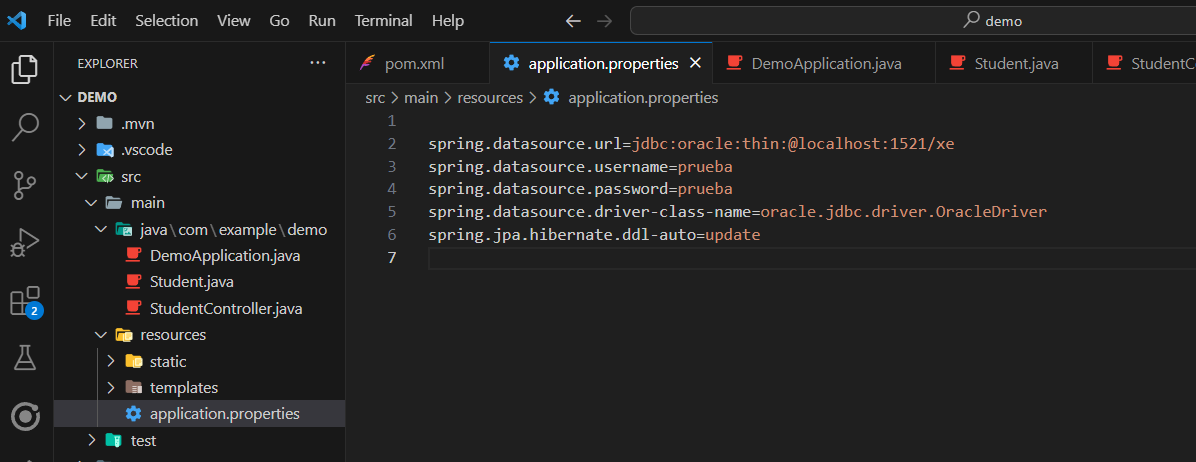


*Nota.* Se puede ver en la figura un print de pantalla de cómo se visualiza cuando se agregan nuevas dependencias. VsCode (2023).

Luego de lo anterior deberás abrir el archivo **src/main/resources/application.properties** o **application.yml** en tu proyecto Spring Boot y configura la conexión a la base de datos Oracle. Recuerda que escribir xe o orcl dependerá de tu versión de Oracle y asegúrate de escribir el usuario y clave que generaste mediante el SQL Plus.

**Figura 3**

*Application.propierties.*



*Nota.* Se presentan en la figura las propiedades de la aplicación. VsCode (2023).

## Codificación del microservicio

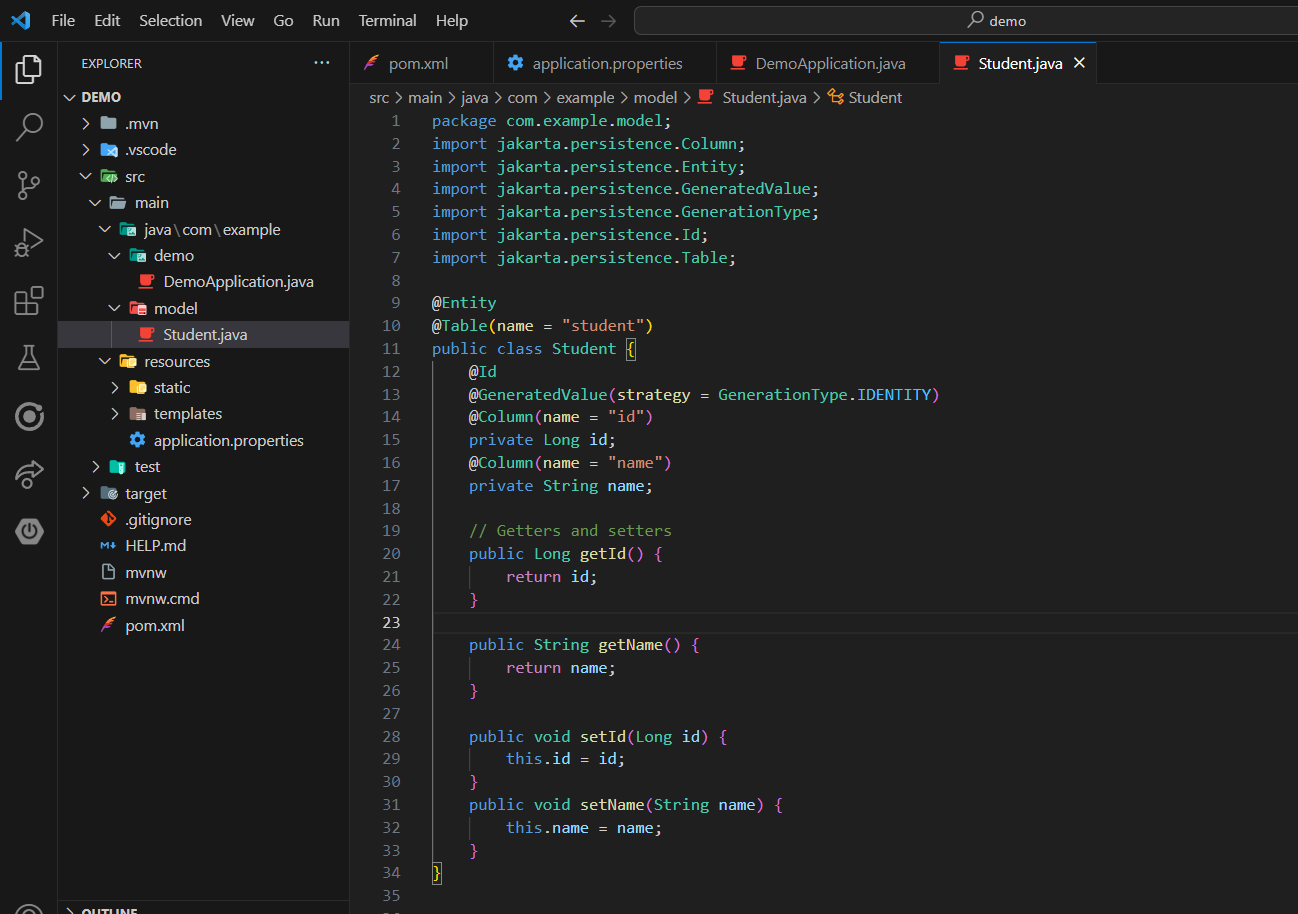
1. Crea una clase Java llamada **Student** en el paquete **com.example.demo.model** con los atributos id y name. Anótala con @Entity para que sea una entidad de JPA y define la tabla en la base de datos.

La tabla en la base de datos Oracle debe crearse previamente, ya sea manualmente o a través de una estrategia de creación de tablas que Spring Boot puede gestionar. Spring Boot proporciona opciones para crear tablas automáticamente (por ejemplo, usando **spring.jpa.hibernate.ddl-auto=update** en **application.properties**), pero esta función puede estar deshabilitada en entornos de producción.

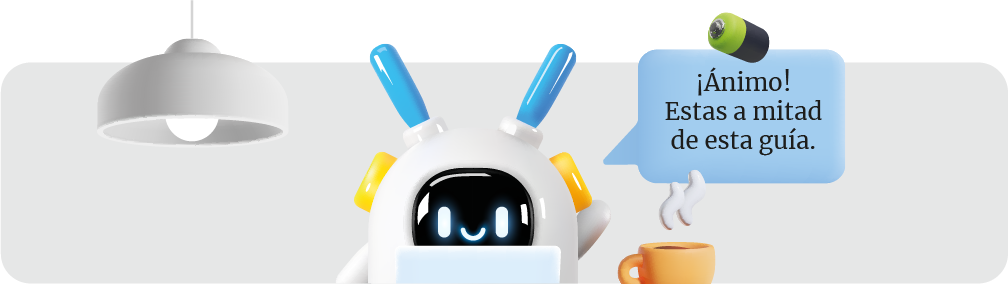
Entonces, para utilizar una tabla de estudiantes en Oracle con Spring Boot, debes crear manualmente la tabla en la base de datos Oracle antes de ejecutar el microservicio. Puedes hacerlo utilizando scripts SQL o mediante herramientas de administración de bases de datos. Luego, Spring Boot mapeará la entidad Student a esa tabla en la base de datos en función de la configuración de JPA y la anotación @Entity.

**Figura 4**

*Entidad Estudiantes.*



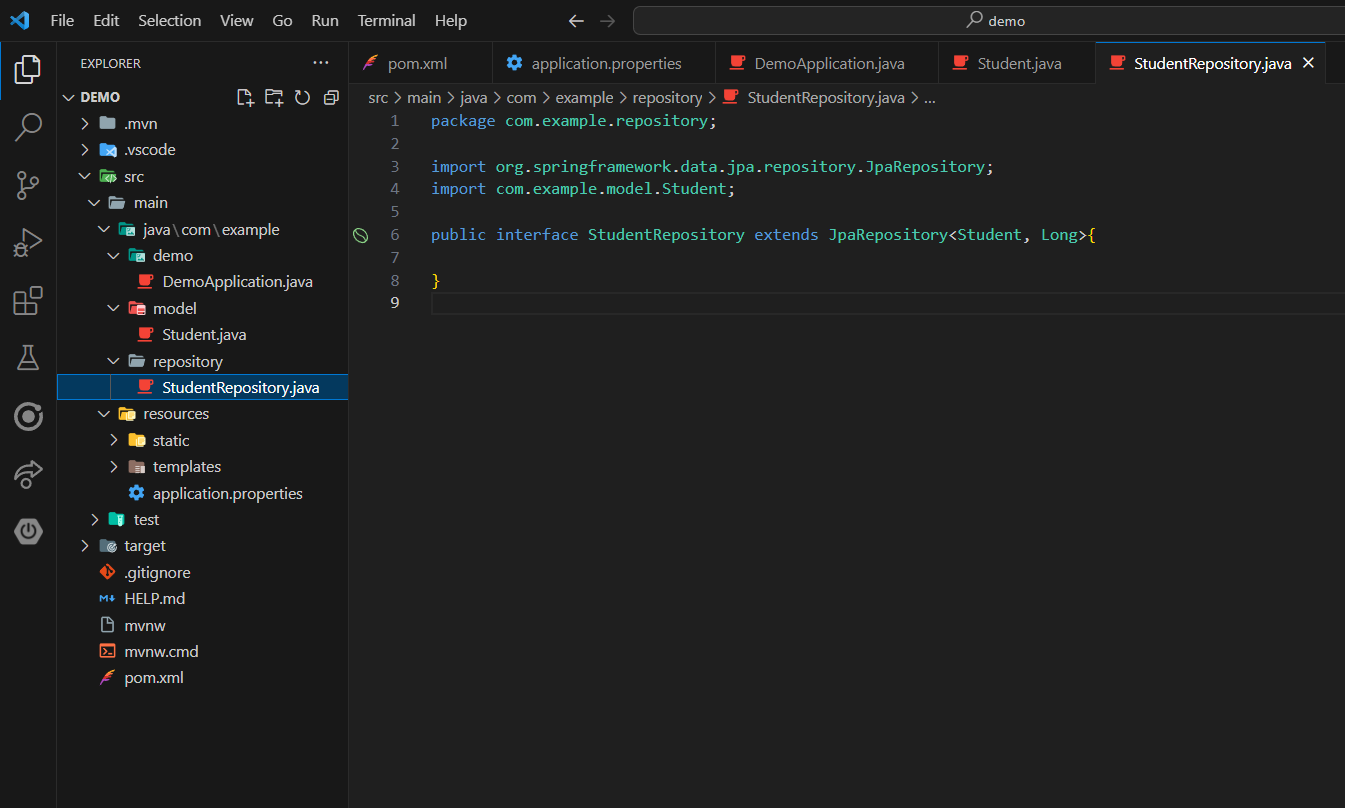
*Nota.* La figura muestra cómo se mapea la entidad estudiantes en la herramienta. VsCode (2023).



1. Crea una interfaz llamada **StudentRepository** en el paquete **com.example.demo.repository** que extienda **JpaRepository** para realizar operaciones CRUD en la entidad **Student**.

**Figura 5**

*Repositorio Estudiantes.*

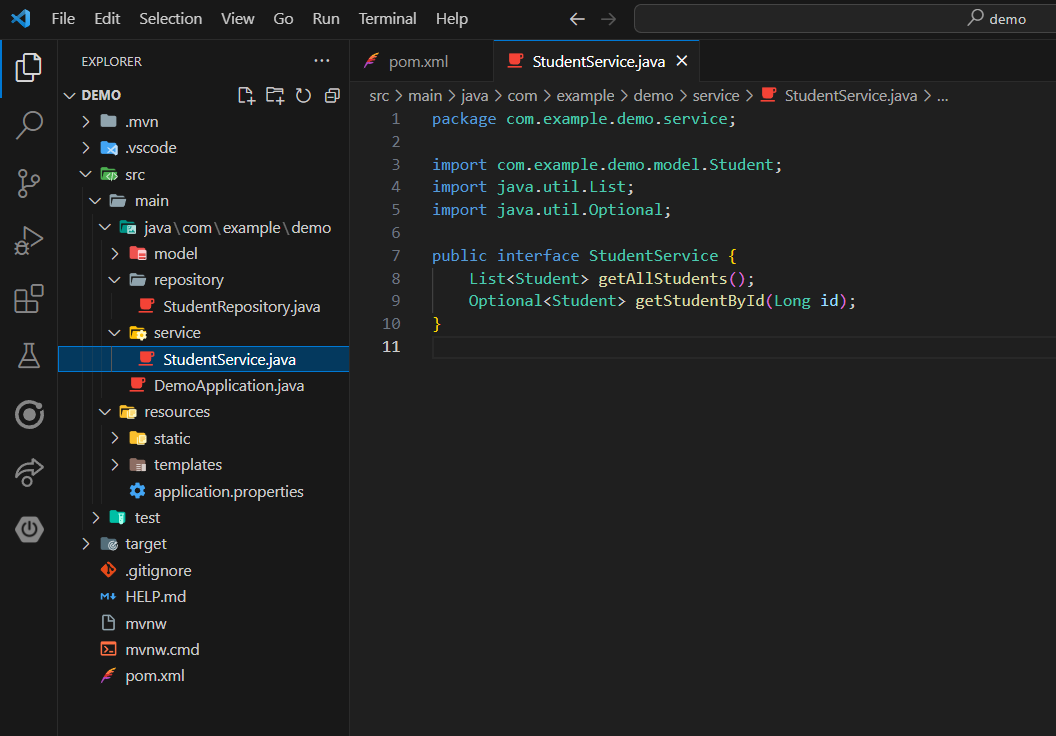


*Nota.* Esta figura muestra la visualización de pantalla cuando se crea el interfaz para crear repositorio estudiantes. VsCode (2023).

1. Crea una interfaz llamada **StudentService** en el paquete **com.example.demo.service** que defina métodos para realizar operaciones CRUD en estudiantes.

**Figura 6**

*Servicio Estudiantes.*

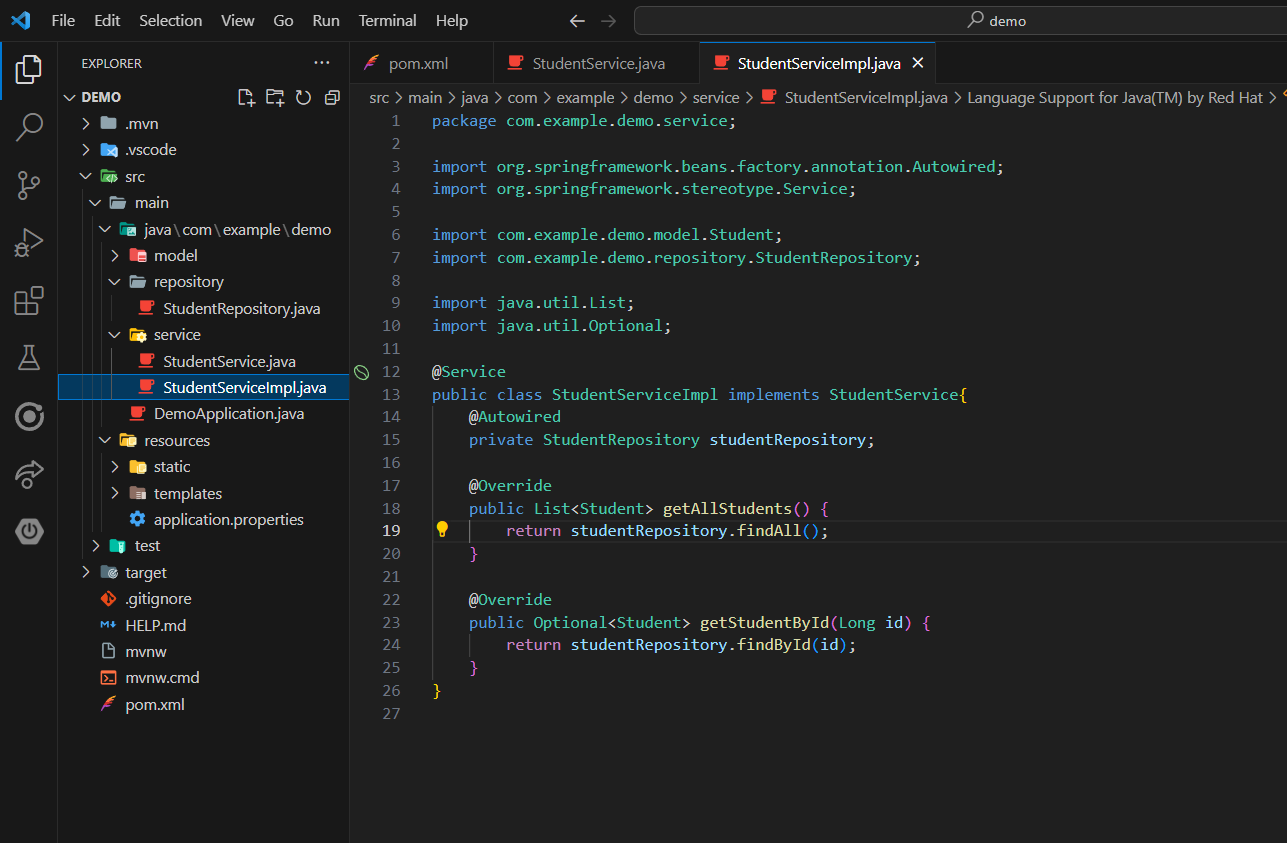


*Nota.* Se visualiza en la figura la pantalla una vez definido los métodos para realizar operaciones CRUD en estudiantes. VsCode (2023).

1. Implementa la interfaz **StudentService** en una clase llamada **StudentServiceImpl** en el mismo paquete.

**Figura 7**

*Clase StudentServiceImpl.*

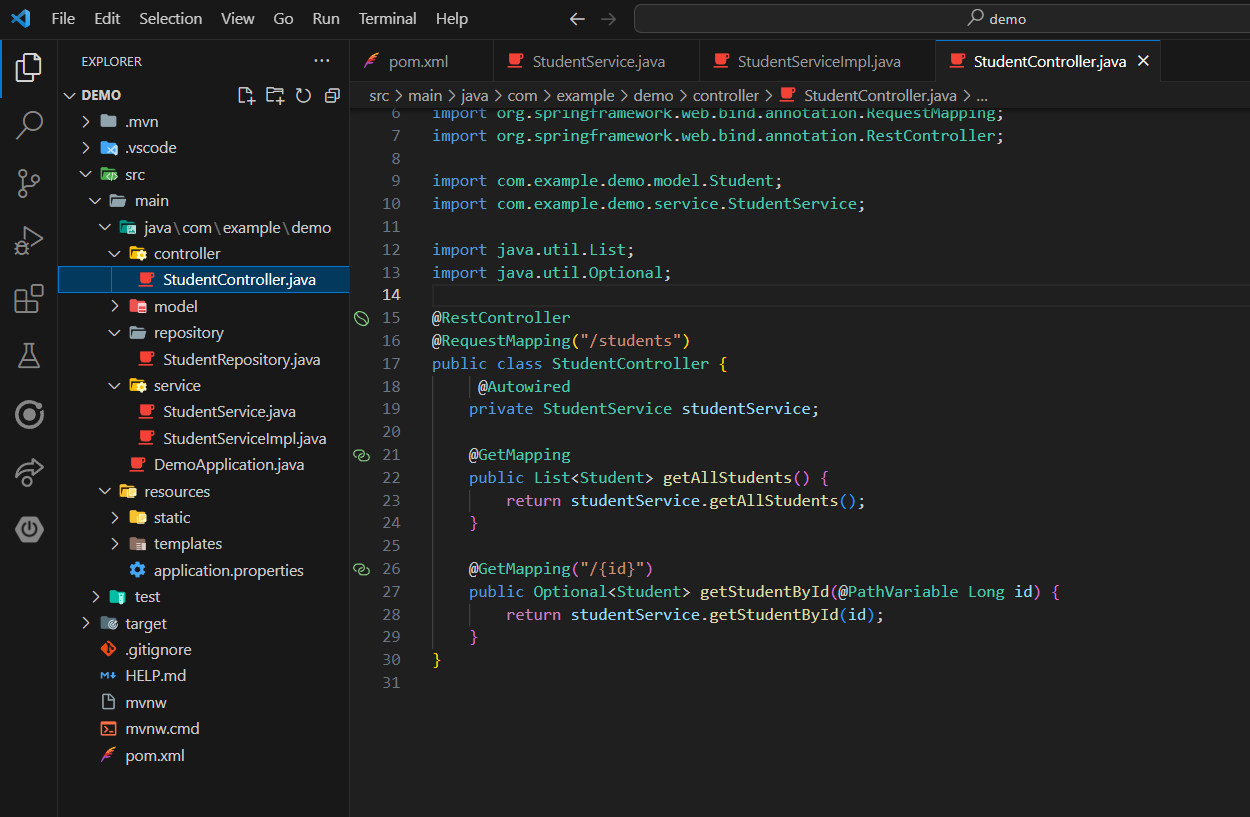


*Nota.* Se presenta la clase llamada StudentServicelmpl. VsCode (2023).

1. Crea un controlador REST llamado **StudentController** en el paquete **com.example.demo.controller** para manejar las solicitudes CRUD.

**Figura 8**

*Controlador Student.*

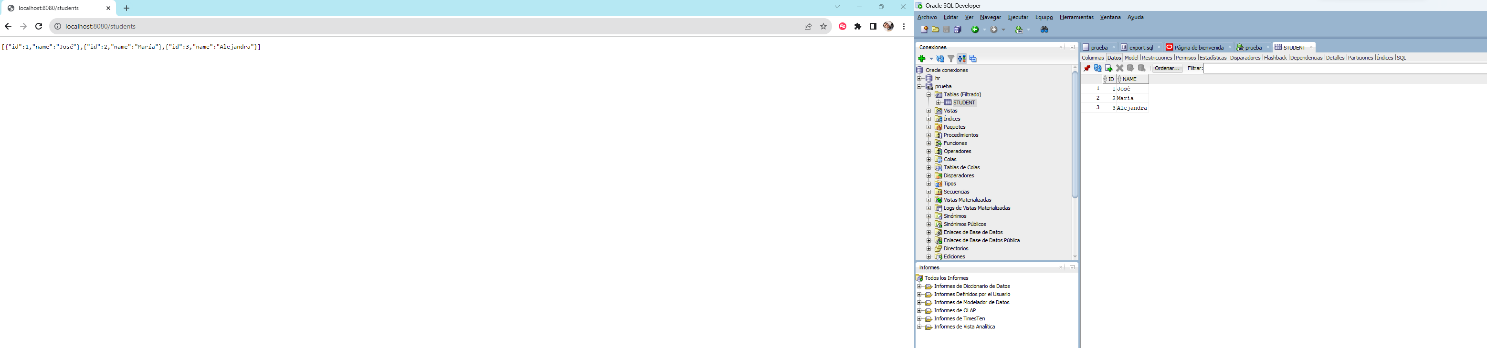


*Nota.* La figura muestra la pantalla de visualización de la creación de un controlador Student. VsCode (2023).

1. Ejecuta la aplicación Spring Boot haciendo clic en el botón "Run" en Visual Studio Code o utilizando el comando mvn spring-boot:run desde la línea de comandos en la carpeta del proyecto.

**Figura 9**

*Ejecución Microservicio.*



*Nota.* La figura muestra la visualización de los comandos en la carpeta del proyecto. Google Chrome y SQLDeveloper (2023).

# Links de Interés

Mantente al día con las actualizaciones del Framework y profundiza sobre los elementos aprendidos durante la experiencia de aprendizaje en el siguiente enlace: <https://spring.io/projects/spring-boot>

# Lectura complementaria

Aprende más sobre Spring en el siguiente enlace: <https://spring.io/>

# Cierre de la semana

Hemos creado un microservicio en Spring Boot que se conecta a una base de datos Oracle 21c y proporciona operaciones CRUD (específicamente, la operación de consulta o GET) para una entidad de "Estudiante" con atributos de "id" y "name". A lo largo de este ejercicio, hemos aprendido lo siguiente:

Configuración de Spring Boot: Hemos configurado un proyecto de Spring Boot utilizando Spring Initializer y hemos agregado las dependencias necesarias para Spring Web, Spring Data JPA y el controlador de Oracle.

Definición de la entidad: Hemos creado una clase Student anotada con @Entity, que representa la entidad "Estudiante" en la base de datos. Esta clase mapea los atributos de id y name a columnas en la tabla de la base de datos.

Repositorio de Estudiantes: Hemos definido un repositorio StudentRepository que extiende JpaRepository para realizar operaciones de lectura (en este caso, la operación de consulta o GET) en la base de datos de manera sencilla sin necesidad de escribir consultas SQL.

Controlador REST: Hemos creado un controlador StudentController que maneja solicitudes HTTP GET para obtener la lista de estudiantes y también para obtener un estudiante por su ID. Utilizamos las anotaciones @RestController y @GetMapping para definir la estructura de las rutas y gestionar las respuestas de las solicitudes HTTP.

Servicio de Estudiantes: Hemos definido una interfaz StudentService y su implementación StudentServiceImpl para separar la lógica de negocio del controlador y mantener una estructura limpia y modular.

En resumen, hemos desarrollado un microservicio que ofrece operaciones de consulta (GET) para obtener la información de los estudiantes. Esto es fundamental en la construcción de aplicaciones RESTful y servicios web, ya que permite a los clientes acceder a los datos de manera eficiente. El ejemplo proporciona una base sólida para extenderlo con operaciones adicionales de creación, actualización y eliminación (CRUD completo) en un entorno real.



# Referencias

* Turnquist, G. Syer, D. Heckler, M. Long, J. (2022). *Learning Spring Boot 3.0: Simplify the development of production-grade applications using Java and Spring*. Packt Publishing.

# Lecturas de la semana

* Chapter 3: Querying for Data with Spring Boot.  
  Fuente: Turnquist, G. Syer, D. Heckler, M. Long, J. (2022). *Learning Spring Boot 3.0: Simplify the development of production-grade applications using Java and Spring*. Packt Publishing. Páginas: 55 - 76

# Apuntes

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Reservados todos los derechos Fundación Instituto Profesional Duoc UC. No se permite copiar, reproducir, reeditar, descargar, publicar, emitir, difundir, de forma total o parcial la presente obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de Fundación Instituto Profesional Duoc UC La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.