

UNIVERSIDAD DE LAS FUERZAS ARMADAS “ESPE”

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

APLICACIONES DISTRIBUIDAS



SISTEMA DE GESTIÓN DE CURSOS Y DOCENTES (ORM)

Nrc: 2553

Carrera: Ingeniería de Software

Integrantes:

- Erick Andrade
- Ángel Castillo
- Ricardo Lazo
- Vanessa Zurita

Docente: Ing. Darío Morales

## **Tabla de contenido**

|                       |    |
|-----------------------|----|
| Introducción.....     | 3  |
| Desarrollo .....      | 3  |
| Conclusiones.....     | 10 |
| Recomendaciones ..... | 11 |
| Bibliografía.....     | 11 |
| Anexos .....          | 11 |

## Introducción

Los microservicios han emergido como un paradigma arquitectónico fundamental en el desarrollo de software moderno, representando una evolución significativa desde los sistemas monolíticos tradicionales. Esta arquitectura no solo ha transformado la forma en que se construyen las aplicaciones empresariales, sino que también ha redefinido los procesos de desarrollo y despliegue en la industria tecnológica (Newman, 2021).

Según Dragoni et al. (2017), los microservicios surgieron como respuesta a las limitaciones inherentes de las arquitecturas monolíticas, especialmente en un contexto donde la agilidad, la escalabilidad y la resistencia son requisitos fundamentales. Esta arquitectura permite a las organizaciones desarrollar aplicaciones más resilientes y adaptables, facilitando la innovación continua y la evolución tecnológica.

En este informe se desarrollará una aplicación con microservicios utilizando SpringBoot y MySQL como base de datos, además de eso la aplicación se va a dockerizar para evitar errores de ejecución en otros dispositivos, esta aplicación permite observar el gran uso de los microservicios y los beneficios que brindan

## Desarrollo

A continuación, se va a presentar el paso a paso de la creación de una aplicación con microservicios, para comenzar se va a presentar el enunciado del microservicio a desarrollar.

### **Enunciado:**

Desarrollar un sistema de gestión de cursos y docentes.

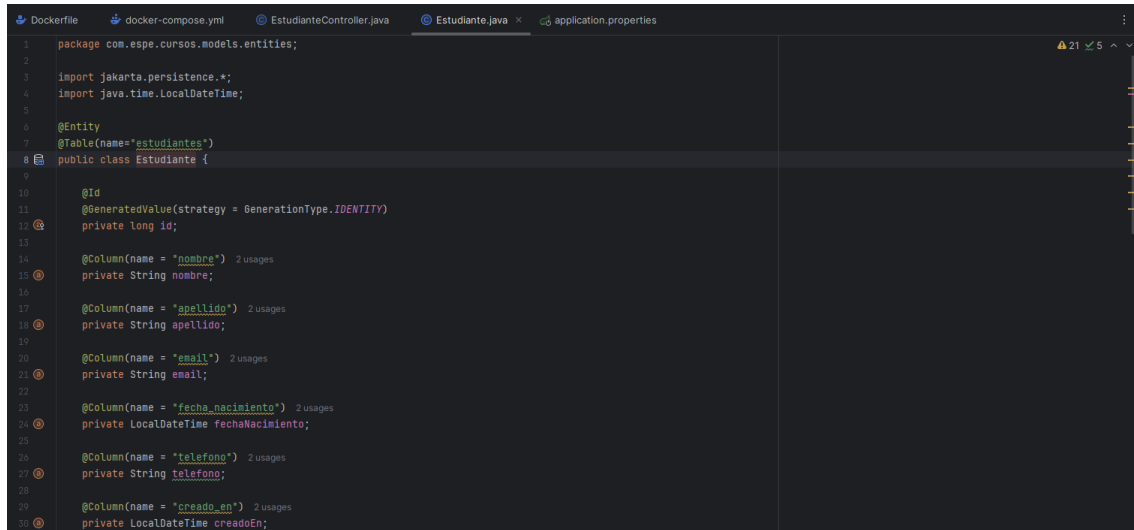
Los estudiantes van a tener estos campos

(id, nombre, apellido, email, fecha\_nacimiento, telefono, creado\_en) y los cursos los siguientes (id, nombre, descripcion, credits, creado\_en).

Este sistema debe generar API's utilizando Spring boot, desplegar con docker y realizar las pruebas con postman.

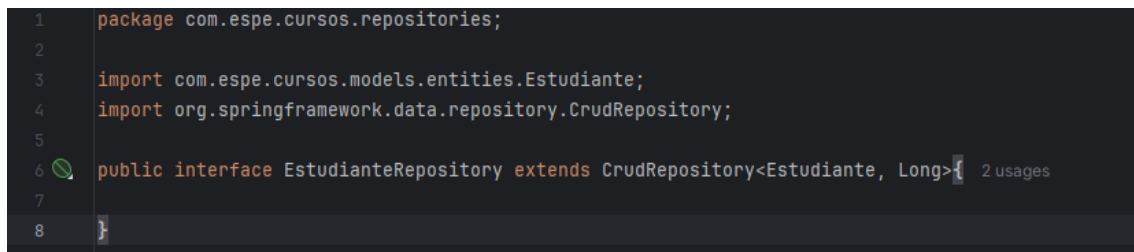
Con toda esta información se procede a realizar la aplicación donde se explican los pasos a desarrollar para el correcto funcionamiento del sistema de gestión de estudiantes y cursos.

1. Primero creamos la clase Estudiantes en la carpeta entities con sus respectivos Getters y Setters.



```
1 package com.espe.cursos.models.entities;
2
3 import jakarta.persistence.*;
4 import java.time.LocalDateTime;
5
6 @Entity
7 @Table(name="estudiantes")
8 public class Estudiante {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private long id;
13
14     @Column(name = "nombre") 2 usages
15     private String nombre;
16
17     @Column(name = "apellido") 2 usages
18     private String apellido;
19
20     @Column(name = "email") 2 usages
21     private String email;
22
23     @Column(name = "fecha_nacimiento") 2 usages
24     private LocalDateTime fechaNacimiento;
25
26     @Column(name = "telefono") 2 usages
27     private String telefono;
28
29     @Column(name = "creado_en") 2 usages
30     private LocalDateTime creadoEn;
```

2. En la carpeta de repositories crearemos un archivo llamado EstudianteRepository



```
1 package com.espe.cursos.repositories;
2
3 import com.espe.cursos.models.entities.Estudiante;
4 import org.springframework.data.repository.CrudRepository;
5
6 public interface EstudianteRepository extends CrudRepository<Estudiante, Long> {
7
8 }
```

3. Creamos el archivo EstudiantesService en la carpeta services en donde se pondrán las funcionalidades.

```

1 package com.espe.cursos.services;
2
3 import com.espe.cursos.models.entities.Estudiante;
4
5 import java.util.List;
6 import java.util.Optional;
7
8 public interface EstudianteService { 3 usages 1 implementation
9
10     List<Estudiante> findAll(); 1 usage 1 implementation
11     Optional<Estudiante> findById(Long id); 2 usages 1 implementation
12     Estudiante save(Estudiante estudiante); 1 usage 1 implementation
13     void deleteById(Long id); 1 usage 1 implementation
14
15 }
16

```

4. Creamos otro archivo llamado EstudianteServiceImpl en donde implementaremos las funciones antes previstas en el archivo EstudiantesService.

```

1 package com.espe.cursos.services;
2
3 import com.espe.cursos.models.entities.Estudiante;
4 import com.espe.cursos.repositories.EstudianteRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @Service
12 public class EstudianteServiceImpl implements EstudianteService {
13
14     @Autowired
15     private EstudianteRepository repository;
16
17     @Override 1 usage
18     public List<Estudiante> findAll() {
19         return (List<Estudiante>) repository.findAll();
20     }
21
22     @Override 2 usages
23     public Optional<Estudiante> findById(Long id) {
24         return repository.findById(id);
25     }
26
27     @Override 1 usage
28     public Estudiante save(Estudiante estudiante) {
29         return repository.save(estudiante);
30     }
31
32     @Override 1 usage
33     public void deleteById(Long id) {

```

5. En la carpeta controller crearemos un archivo Estudiante Controller implementaremos los métodos a ejecutar en la base de datos ya antes mencionados.

```

1 package com.espe.cursos.controllers;
2
3 import com.espe.cursos.models.entities.Estudiante;
4 import com.espe.cursos.services.EstudianteService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.util.List;
11 import java.util.Optional;
12
13 @RestController
14 @RequestMapping("/api/estudiantes")
15 public class EstudianteController {
16
17     @Autowired
18     private EstudianteService service;
19
20     // Crear un nuevo estudiante
21     @PostMapping
22     public ResponseEntity<?> crear(@RequestBody Estudiante estudiante) {
23         return ResponseEntity.status(HttpStatus.CREATED).body(service.save(estudiante));
24     }
25
26     // Obtener todos los estudiantes
27     @GetMapping
28     public ResponseEntity<List<Estudiante>> obtenerTodos() {
29         List<Estudiante> estudiantes = service.findAll();
30         return ResponseEntity.ok(estudiantes);
31     }
32
33     // Obtener un estudiante por su ID

```

6. Para subir nuestro proyecto a Docker crearemos el siguiente archivo llamado Dockerfile con la siguiente configuración en la raíz del proyecto.

- a. Lo nuevo que podemos encontrar es que conseguir el .jar necesitaremos ir a la carpeta target y si no está ahí ejecutaremos el siguiente comando mvn clean package

```

1 # Usar una imagen base de OpenJDK
2 FROM openjdk:17-jdk-slim
3
4 # Definir el directorio de trabajo dentro del contenedor
5 WORKDIR /app
6
7 # Copiar el archivo JAR de la aplicación al contenedor
8 COPY target/cursos-0.0.1-SNAPSHOT.jar app.jar
9
10 # Exponer el puerto donde la aplicación escucha
11 EXPOSE 8002
12
13 # Comando para ejecutar la aplicación Spring Boot
14 ENTRYPOINT ["java", "-jar", "app.jar"]
15

```

7. Cerca de terminar tenemos que crear otro archivo en la raíz del proyecto con el siguiente nombre docker-compose.yml

```

1  version: '3.8'
2
3  services:
4    spring-boot-app:
5      build: .
6      container_name: spring-boot-app
7      ports:
8        - "8002:8002"
9      environment:
10       - SPRING_DATASOURCE_URL=jdbc:mysql://mysql-db:3306/cursos_db
11       - SPRING_DATASOURCE_USERNAME=root
12       - SPRING_DATASOURCE_PASSWORD=admin
13      depends_on:
14        - mysql-db
15
16    mysql-db:
17      image: mysql:8
18      container_name: mysql-db
19      environment:
20       - MYSQL_ROOT_PASSWORD=admin
21       - MYSQL_DATABASE=cursos_db
22      ports:
23       - "3307:3306"
24      volumes:
25       - mysql-data:/var/lib/mysql
26
27  volumes:
28    mysql-data:
29

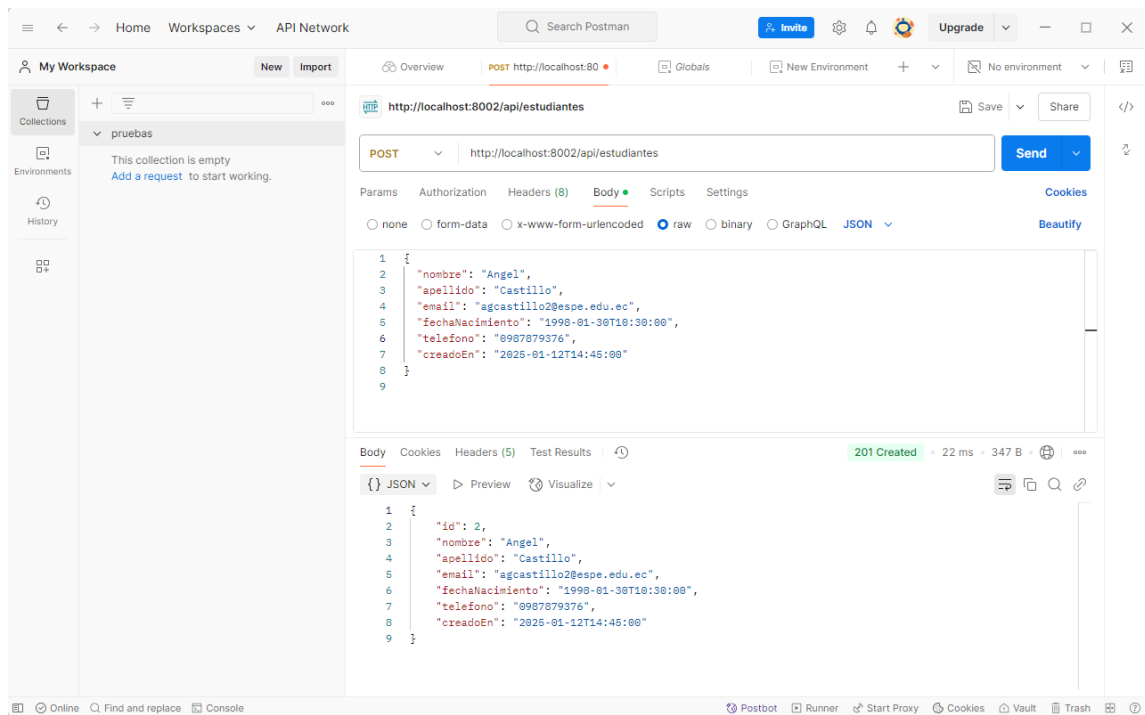
```

8. Por último, ejecutaremos dos comandos uno para construir y el otro para subir y ejecutar son los comandos docker-compose build y docker-compose up.

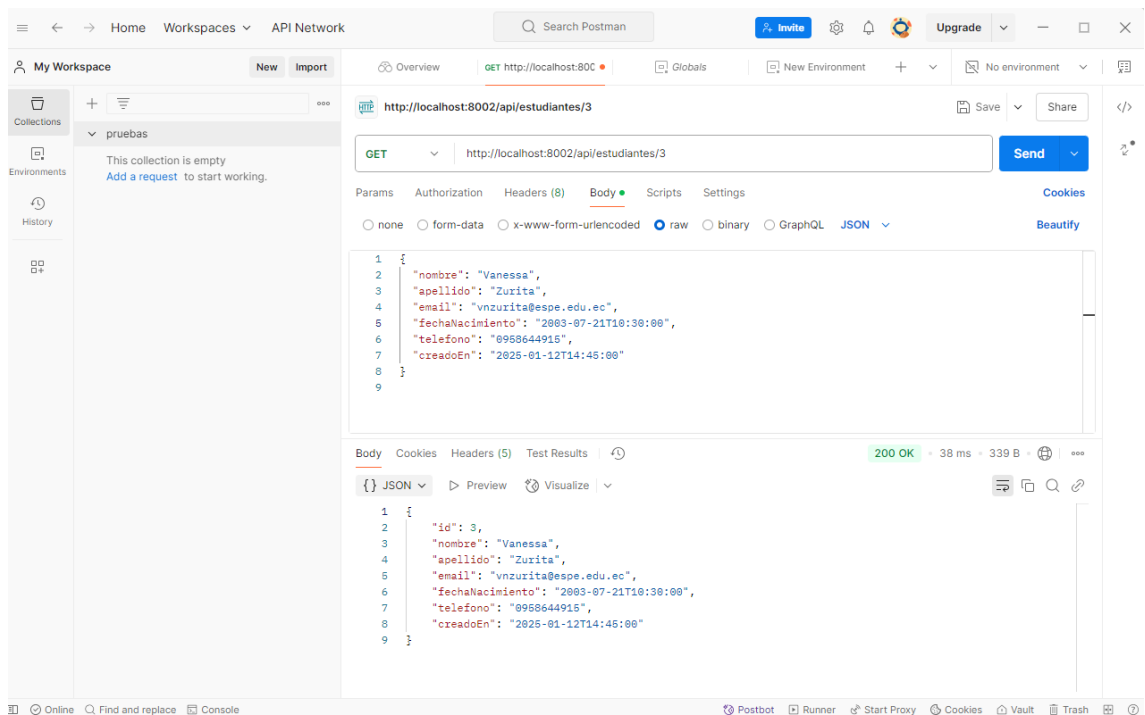
The screenshot shows the Docker Desktop interface. On the left, the 'Containers' tab is selected, showing two containers: 'mysql-db' and 'spring-boot-app'. The 'spring-boot-app' container is highlighted, showing its configuration: image 'cursos-spring-boot-app', ports '8002:8002', and a link to 'View configurations'. On the right, the logs for the 'spring-boot-app' container are displayed, showing the application starting and initializing the Spring DispatcherServlet. The logs include timestamps, log levels (INFO, DEBUG), and the application's output.

9. Comprobar que funciona con postman.

- a. Insertar:

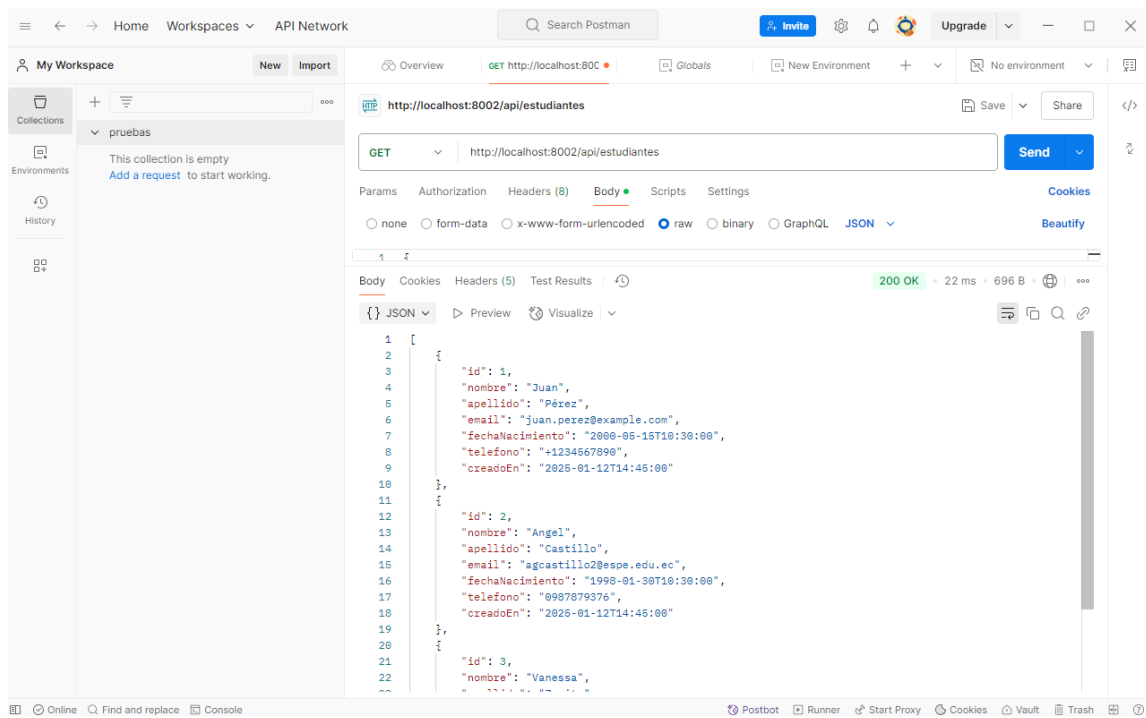


## b. Visualizar uno:

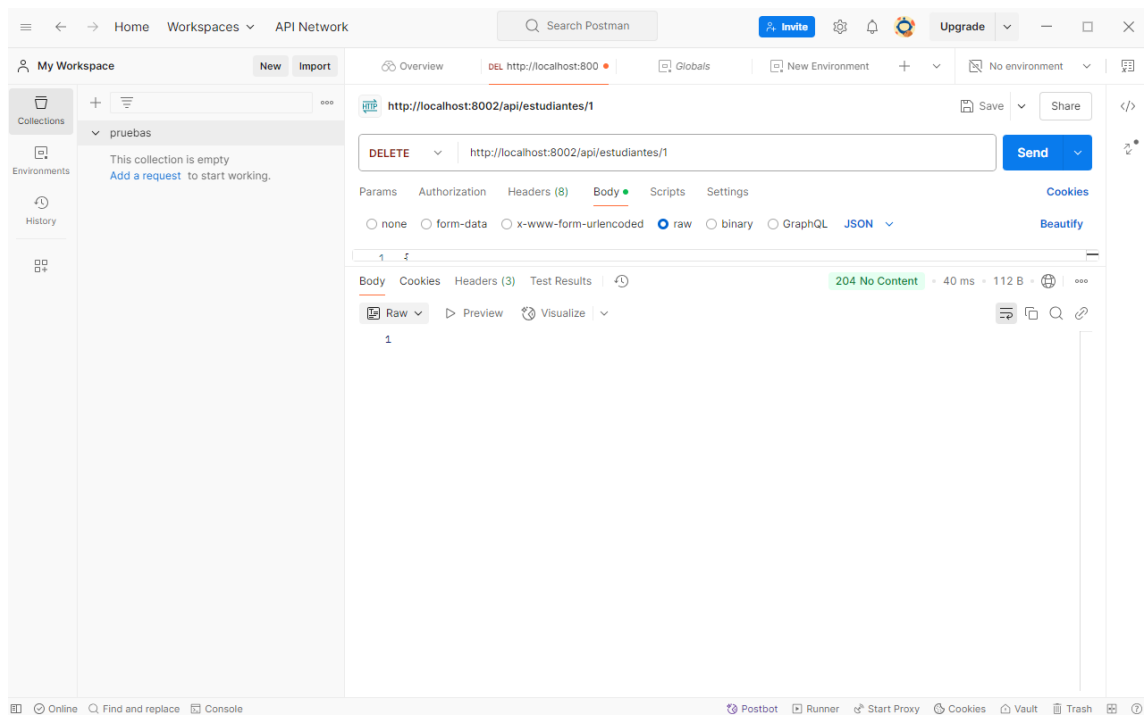


## c. Visualizar todo:

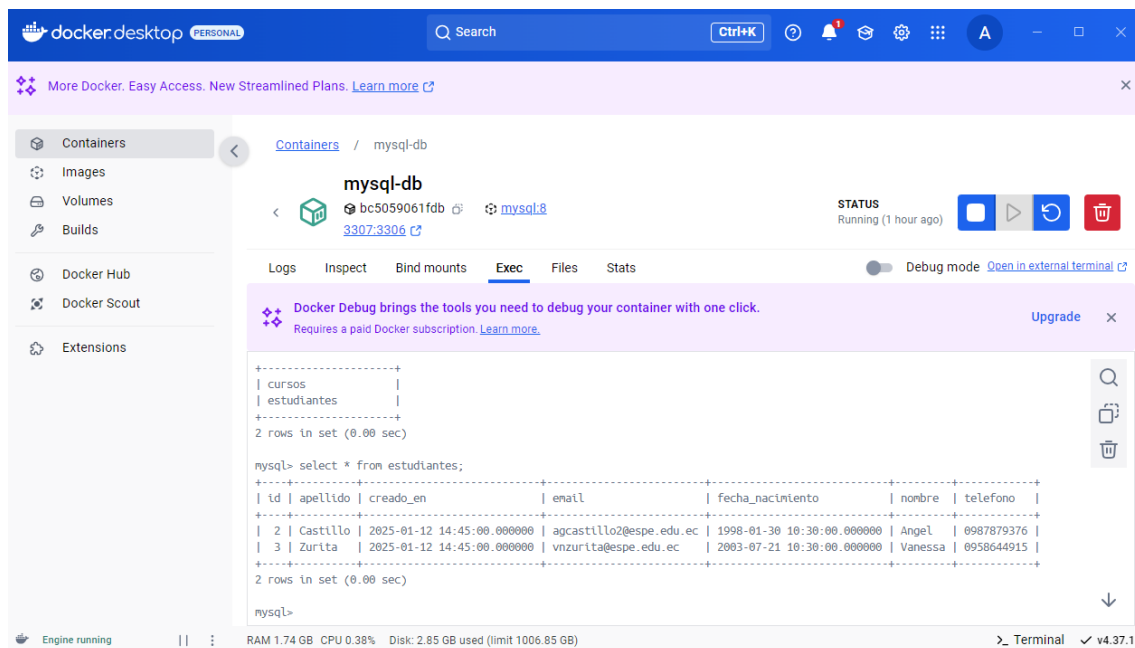
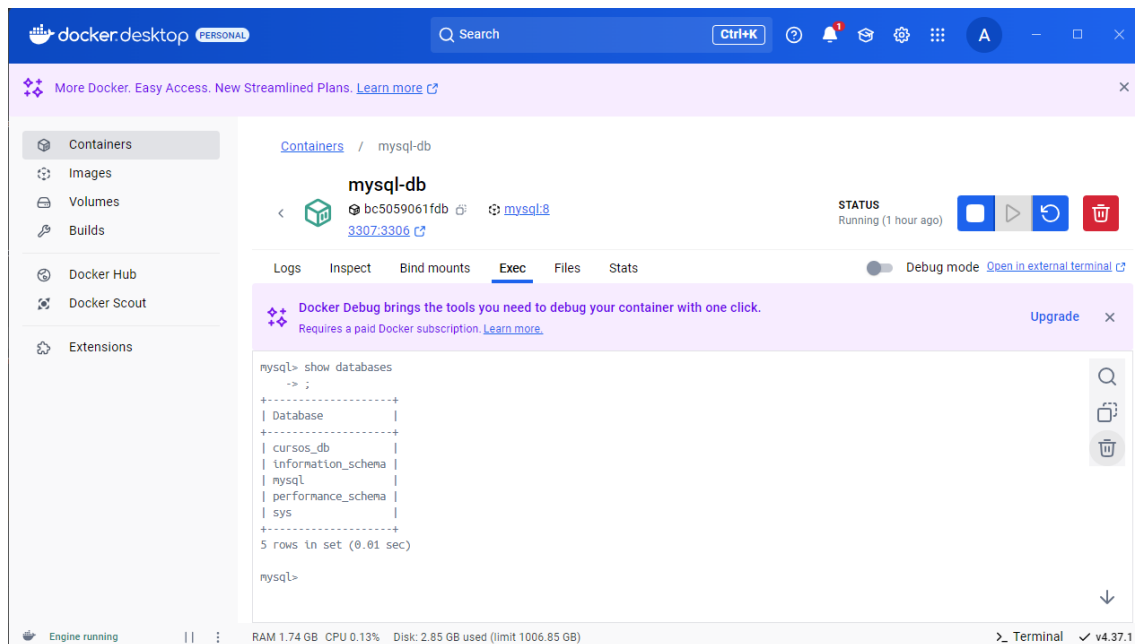




#### d. Eliminar:



#### 10. Comprobar en la base de datos de Docker.



## Conclusiones

- El uso de microservicios permite crear aplicaciones modulares y flexibles, mientras que Docker asegura su ejecución consistente, reduciendo errores y mejorando la productividad.
- La implementación de esta aplicación basada en microservicios con Spring Boot y MySQL, junto con su containerización en Docker, ha demostrado ser una

solución eficiente que maximiza la escalabilidad y reduce la complejidad en el despliegue del sistema.

### Recomendaciones

Una recomendación general para "dockerizar" un proyecto es mantener la modularidad y escalabilidad. Diseña los servicios en tu archivo docker-compose.yml de forma que cada componente como aplicación, base de datos, servicios adicionales sea independiente pero fácilmente configurable.

### Bibliografía

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). "Microservices: Yesterday, Today, and Tomorrow". Present and Ulterior Software Engineering, Springer.

Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems (2nd Edition). O'Reilly Media, Inc. ISBN: 9781492034025

### Anexos

[https://github.com/vanessazurita/AplicacionesDistribuidas/tree/Programar/Segundo%20Parcial/Tarea\\_ORM/cursos](https://github.com/vanessazurita/AplicacionesDistribuidas/tree/Programar/Segundo%20Parcial/Tarea_ORM/cursos)