Proyecto EduTech Innovators Spa.

Equipo: Tiare Acuña

Ricardo Rodríguez

Maximiliano Rubilar

Fecha: 24 de junio de 2025.

Contenido

[1.Resumen ejecutivo 3](#_Toc201664303)

[2.Introducción 3](#_Toc201664304)

[3. Objetivos 4](#_Toc201664305)

[3.1 Objetivo General 4](#_Toc201664306)

[3.2 Objetivos Específicos 4](#_Toc201664307)

[4.Justificación 4](#_Toc201664308)

[5.Marco Teórico 4](#_Toc201664309)

[6.Análisis del problema 4](#_Toc201664310)

[7.Arquitectura del sistema 5](#_Toc201664311)

[8. Diseño e implementación 7](#_Toc201664312)

[8.1 Estructura del proyecto 7](#_Toc201664313)

[Dependencias 7](#_Toc201664314)

[Componentes implementados 22](#_Toc201664315)

[8.2 Base de datos 24](#_Toc201664316)

[9. Git – Git Hub 25](#_Toc201664317)

[10.Pruebas 30](#_Toc201664318)

[11.Resultados obtenidos 30](#_Toc201664319)

[11.1 Calificaciones 30](#_Toc201664320)

[11.2 Usuarios 32](#_Toc201664321)

[11.3 Cursos 33](#_Toc201664322)

[11.4 Inscripciones 34](#_Toc201664323)

[12.Conclusiones y recomendaciones 35](#_Toc201664324)

# 1.Resumen ejecutivo

El presente informe describe el desarrollo de un sistema Fullstack basado en microservicios para la empresa ficticia EduTech Innovators Spa. El sistema permite la gestión académica mediante módulos independientes pero integrados, como usuarios, cursos, inscripciones, evaluaciones y calificaciones. Cada microservicio fue construido en Java con Spring Boot y conectados mediante API REST, mientras que el frontend fue desarrollado en HTML/CSS/JavaScript. El sistema busca escalar fácilmente y facilitar el mantenimiento y evolución del software.

# 2.Introducción

Bienvenidos a esta nueva presentación donde compartiremos los avances más recientes de nuestro proyecto **EduTech Innovators Spa**, una empresa comprometida con la transformación digital en el ámbito educativo. Desde su creación, nuestra misión ha sido proporcionar plataformas educativas accesibles, dinámicas e innovadoras, permitiendo que estudiantes de diversas regiones puedan acceder a herramientas de aprendizaje efectivas sin importar su ubicación geográfica.

Nuestra historia comenzó con una oficina en **Providencia, Santiago**, donde dimos nuestros primeros pasos en la creación de cursos interactivos en línea. Gracias al respaldo de nuestros usuarios y la creciente demanda de soluciones digitales en educación, hemos logrado expandir nuestras operaciones y establecer nuevas oficinas en **Valparaíso y La Serena**, consolidando así nuestra presencia en el mercado y fortaleciendo nuestra infraestructura para ofrecer un servicio más sólido y escalable.

Como se discutió en nuestra presentación anterior, el sistema **EduTech** enfrenta diversos desafíos relacionados con el rendimiento y la disponibilidad. Identificamos y analizamos estos problemas con detalle, presentando propuestas y soluciones que han permitido optimizar la estabilidad del software y mejorar la experiencia del usuario. En esta sesión, nos enfocaremos en la **estructura del proyecto**, sus **componentes clave**, el **modelo de base de datos** y las **implementaciones de servicios** que han sido fundamentales para garantizar un funcionamiento eficiente y escalable.

A lo largo de esta presentación, explicaremos en profundidad cómo hemos refinado la arquitectura del software, detallaremos las mejoras realizadas en la integración de nuestros servicios y mostraremos cómo cada uno de estos cambios contribuye a la misión de **EduTech Innovators Spa**: ofrecer educación accesible y de calidad a través de plataformas tecnológicas avanzadas.

Esperamos que esta actualización sea de gran utilidad para todos los presentes y estamos abiertos a cualquier consulta o sugerencia que ayude a seguir mejorando nuestro sistema.

# 3. Objetivos

## 3.1 Objetivo General

Desarrollar un sistema educativo fullstack modular basado en microservicios que permita gestionar de forma eficiente usuarios, cursos, inscripciones, evaluaciones y calificaciones.

## 3.2 Objetivos Específicos

1. Diseñar una arquitectura desacoplada y escalable basada en microservicios.
2. Implementar APIs REST en Java Spring Boot para cada dominio.
3. Desarrollar vistas HTML funcionales que consuman los servicios REST.
4. Realizar pruebas unitarias e integración para asegurar calidad de cada componente.

# 4.Justificación

El uso de microservicios permite construir sistemas más robustos y mantenibles. Este enfoque responde a las necesidades actuales del mercado, permitiendo una evolución ágil de los sistemas de software. EduTech Innovators Spa representa una empresa en crecimiento que necesita una solución moderna, adaptable y eficiente para su entorno académico.

# 5.Marco Teórico

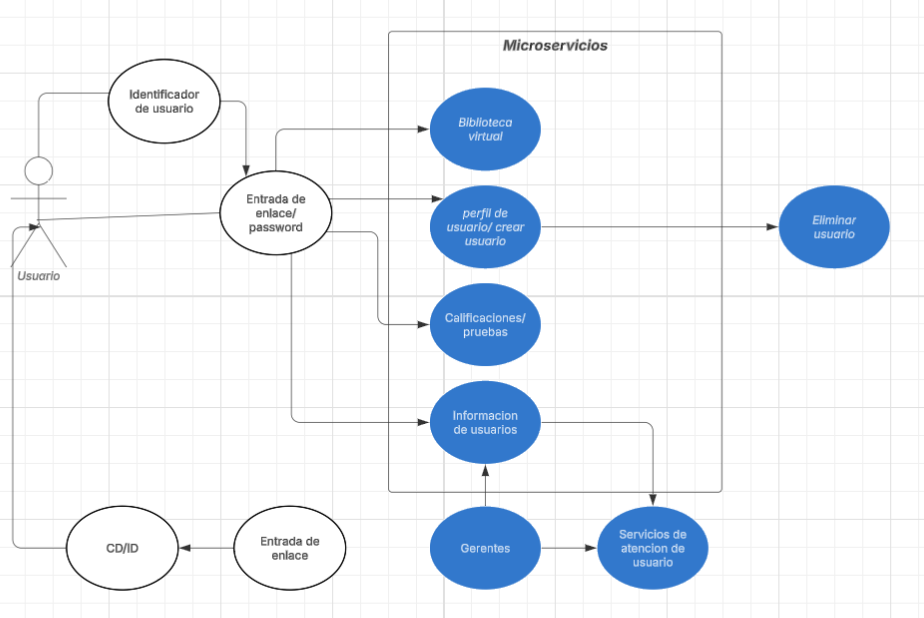
Los microservicios son una arquitectura de software donde las funcionalidades se dividen en servicios pequeños, independientes y desplegables por separado. Spring Boot es un framework de desarrollo ágil para crear aplicaciones Java. En el frontend, se emplea HTML y JavaScript para el consumo de APIs REST, permitiendo una experiencia interactiva al usuario.

# 6.Análisis del problema

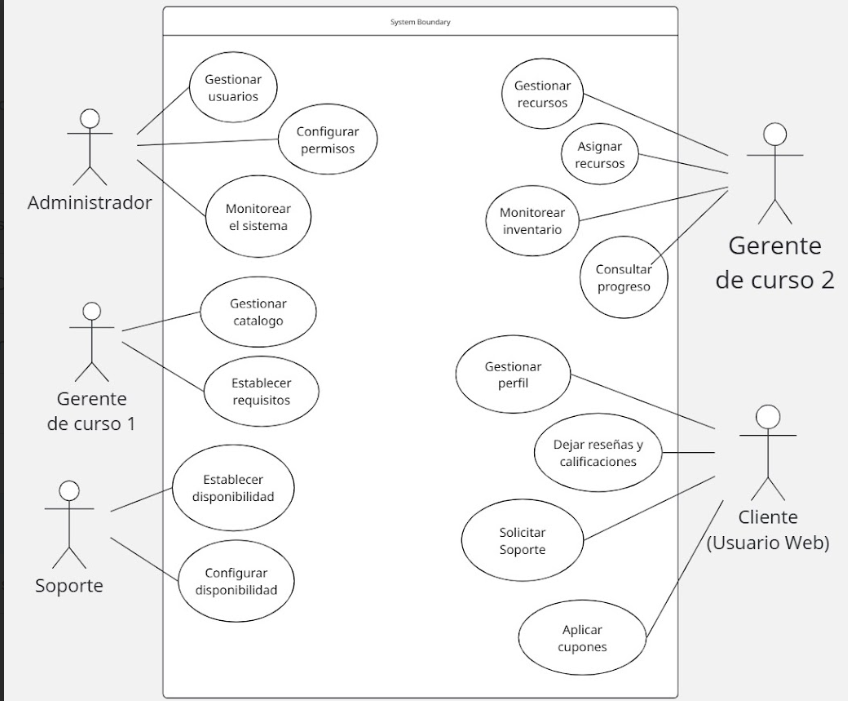
Las plataformas educativas tradicionales tienden a ser monolíticas, dificultando la evolución del sistema. EduTech Innovators Spa requiere una solución modular que le permita escalar sus funcionalidades sin afectar el resto del sistema, asegurando resiliencia, independencia de módulos y eficiencia en las actualizaciones.

# 7.Arquitectura del sistema

El sistema se compone de cinco microservicios principales: usuarios, cursos, inscripciones, evaluaciones y calificaciones. Cada uno maneja su propia lógica de negocio, base de datos y API REST. La comunicación entre servicios es mediante HTTP utilizando JSON. El frontend permite a los usuarios interactuar con cada módulo mediante formularios y vistas dinámicas.



En este diagrama de arquitectura se ve los procesos que sigue cada usuario a la hora de acceder a los servicios los cuales se divide en diferentes secciones basados en los crud (crear, leer actualizar y eliminar) para que así se pueda mantener una base de datos funcional basada en los microservicios que requiere la empresa para su funcionalidad



También se debe agregar el diagrama de los usuarios ya que es un factor determinante a la hora de realizar un programa de desarrollo software ya que es la única forma de saber los requisitos y roles sobre los cuales se basaran para desarrollar una base de datos y programa que se pueda adaptar a las necesidades de los usuarios y personal encargado de la página web.

# 8. Diseño e implementación

Tecnologías utilizadas:

* Backend: Java 17, Spring Boot, Maven, JUnit, Mockito
* Frontend: HTML5, CSS3, JavaScript, Postman (para pruebas)
* Base de datos: MySQL
* Control de versiones: GitHub

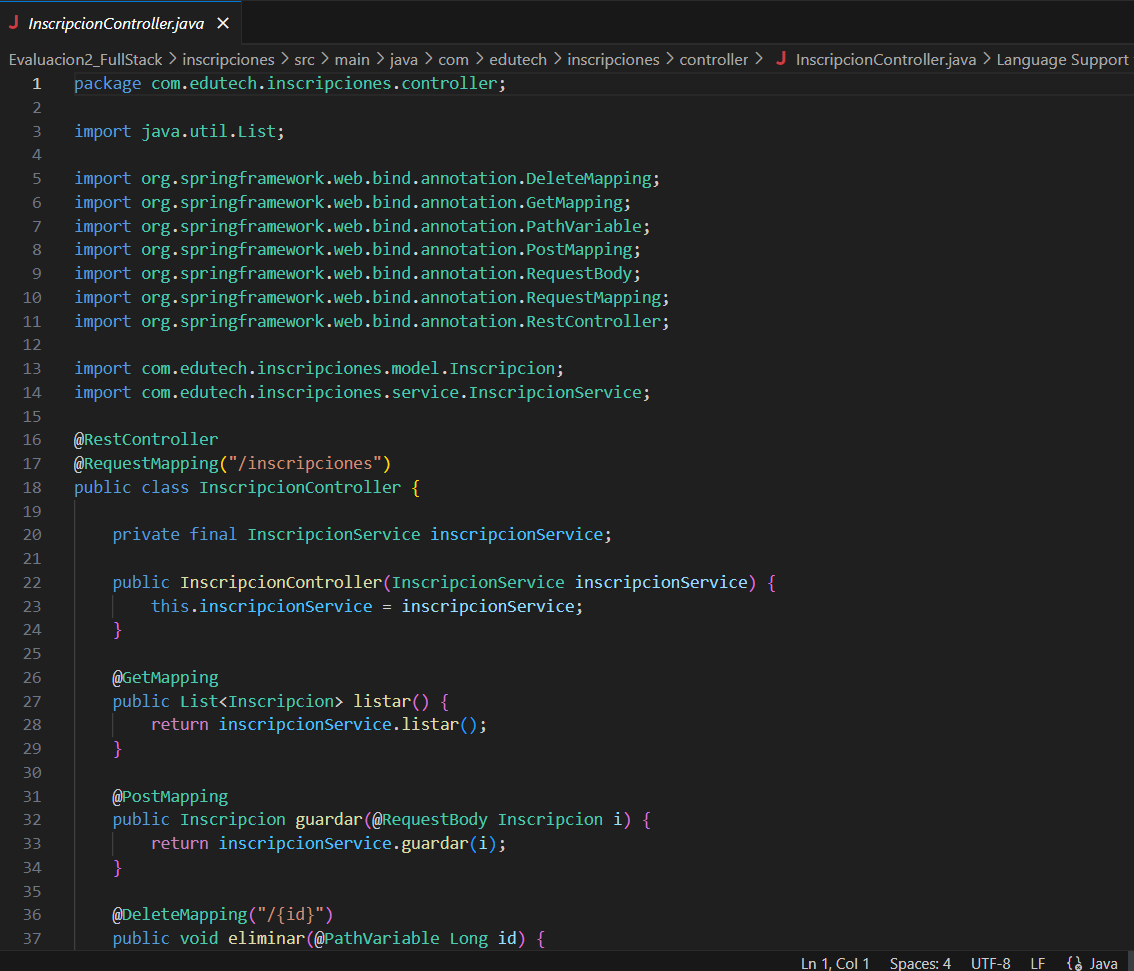
Cada microservicio incluye controladores REST, servicios, entidades y repositorios. Se usaron DTOs para la transferencia de datos y validaciones con Bean Validation.

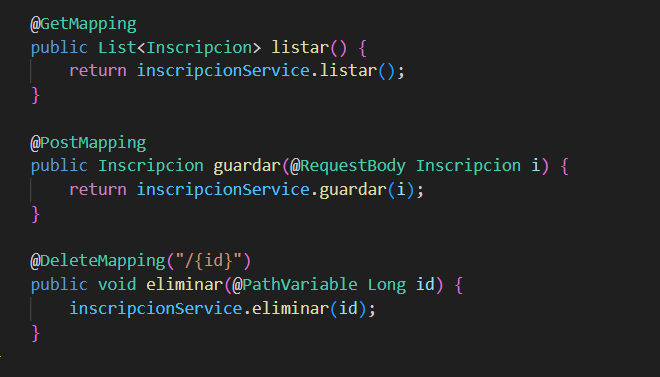
## 8.1 Estructura del proyecto

### Dependencias

**Dependencias Controller**

**Primera dependencia: InscripcionController.java**

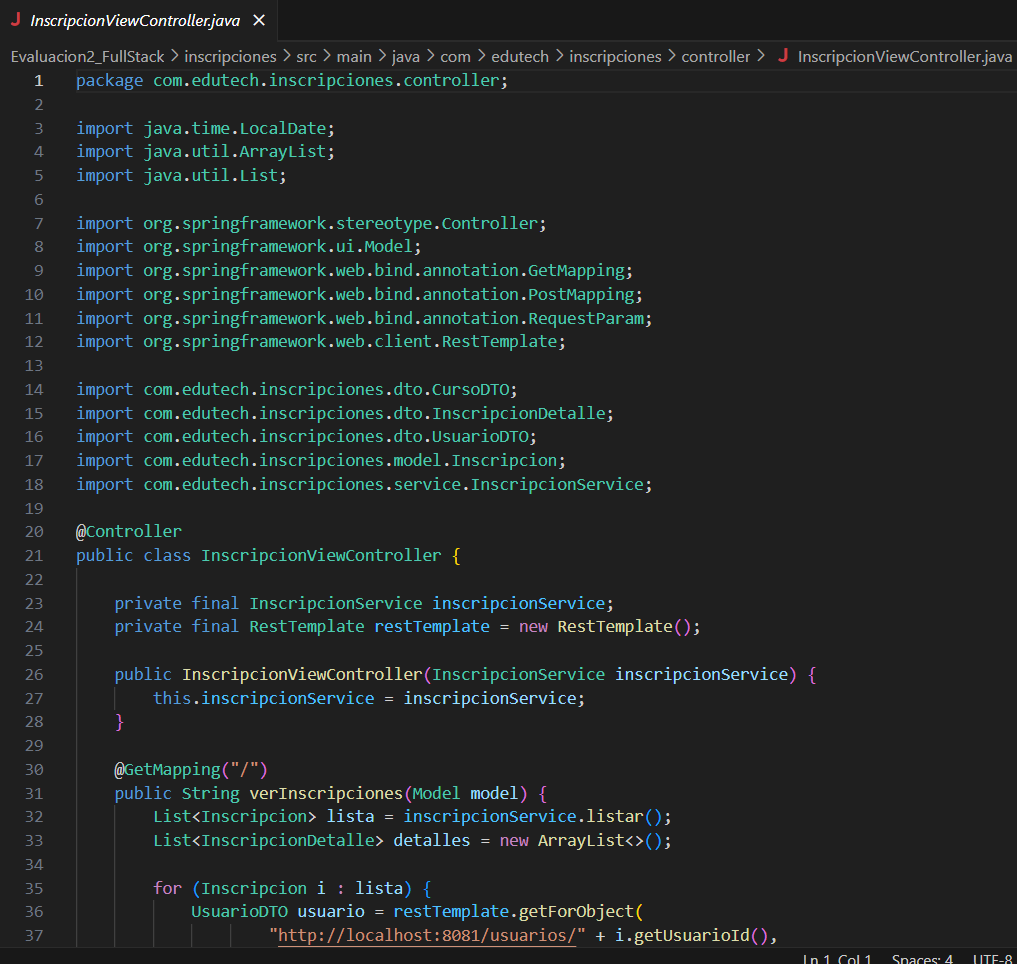


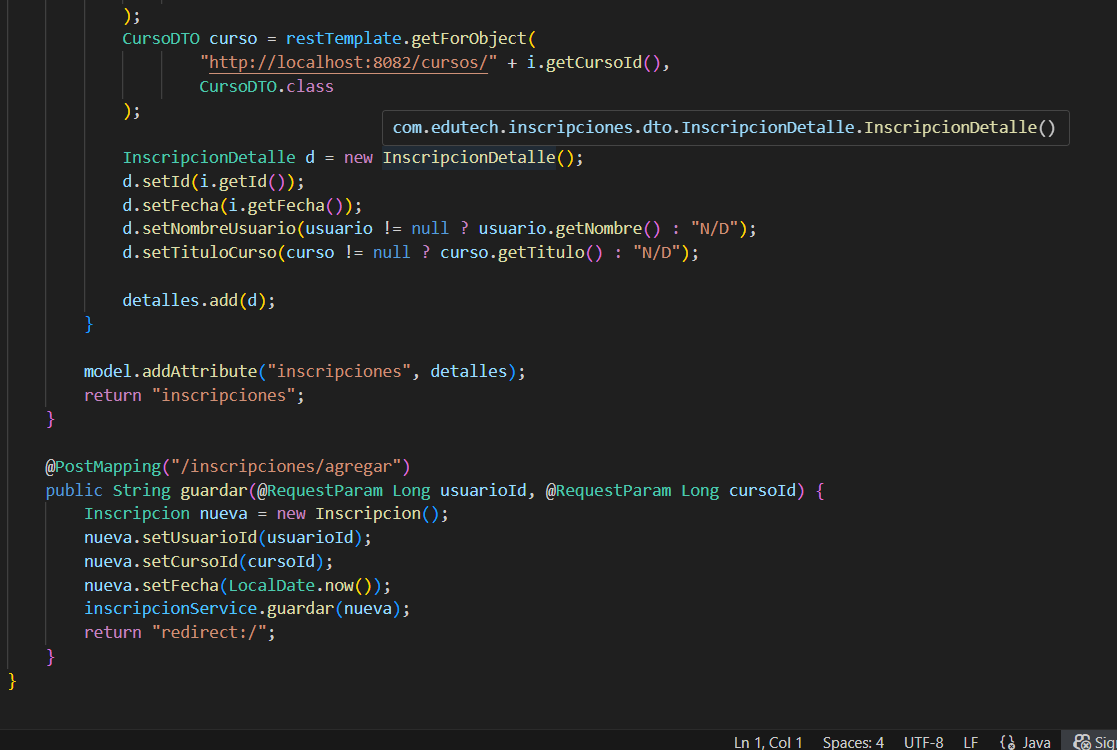


Este controlador rest lo usamos para poder gestionar las inscripciones. ya que su propósito es manejar las solicitudes de HTTP relacionadas con las inscripciones para a su vez delegarlas al servicio correspondiente. Este posee un

* “@Rest controller y @RequestMapping” (controlador Rest que responde a la ruta inscripciones)
* listar() (@GetMapping): Devuelve una lista de todas las inscripciones disponibles en el sistema
* guardar(inscripción i) (@PostMapping): Permite registrar una nueva inscripción, recibiendo los datos en el cuerpo de la solicitud (@RequestBody)
* eliminar(Long id) (@DeleteMapping(“/{id}”)): sirve para borrar una inscripción específica identificada por si ID

**Segunda dependencia: InscripcionViewController.java**

****



Este “InscripicionViewController” sirve para definir un controlador para la vista de esta aplicación (SpringBoot con Thymeleaf) esto con el fin de ayudar a renderizar las páginas de HTML con los datos de inscripciones de la instancia anterior. Sus funciones principales son:

* @Controller: Indica que esta clase manejara vistas en la aplicación, a diferencia del @RestController que solo devuelve el Json
* RestTemplate: Permite hacer llamadas a otros servicios externos, como el obtener la información de usuarios y cursos
* Model: Se usa para poder pasar los datos dinámicos a la vista de HTML

Otros metodos clave de esta instancia son:

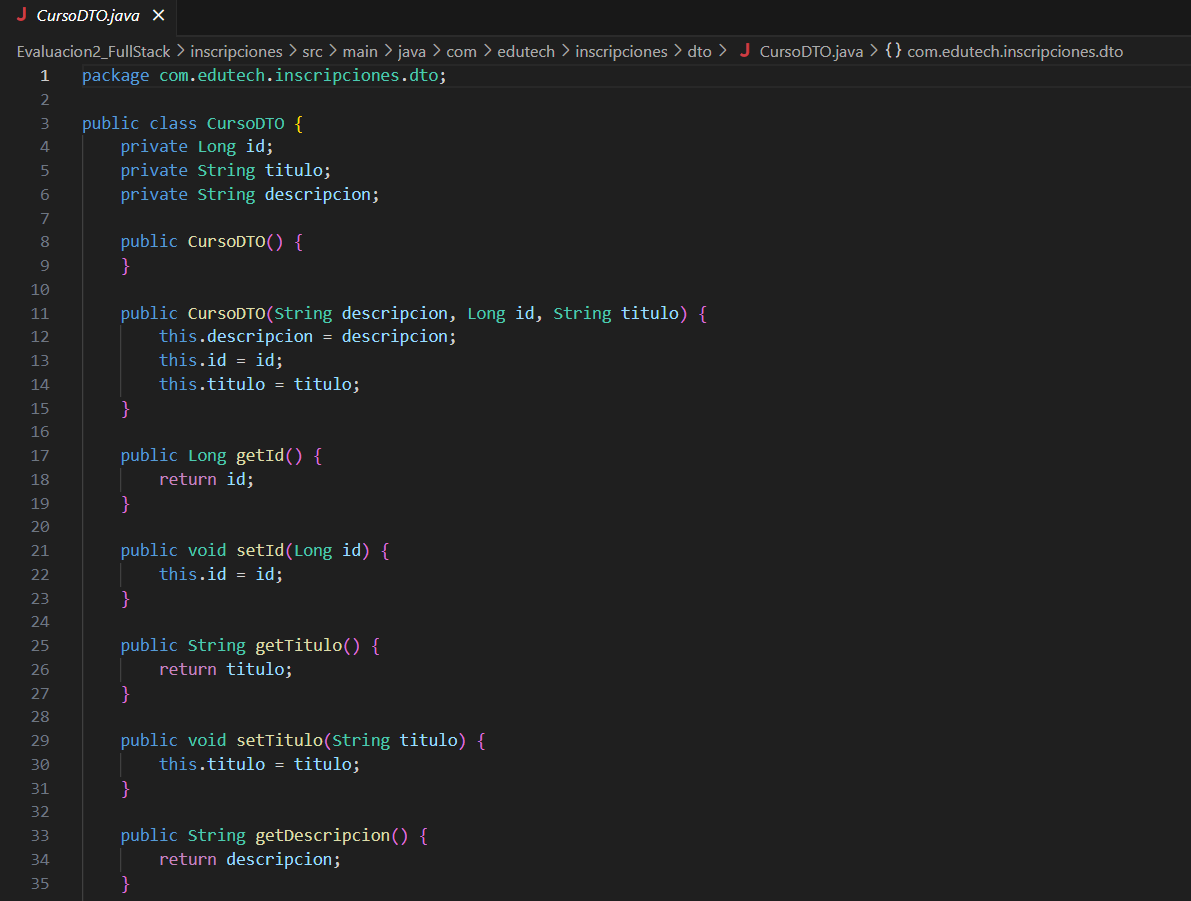
* Obtener todas las inscripciones registradas (**inscripcionService.listar()**). Usa **RestTemplate** para llamar a servicios externos y obtener datos de usuarios y cursos.
* Crea una lista de **InscripcionDetalle** con información combinada (nombre de usuario, curso, fecha).
* Agrega los datos al modelo **(model.addAttribute("inscripciones", detalles)**) y muestra la vista **"inscripciones"**.

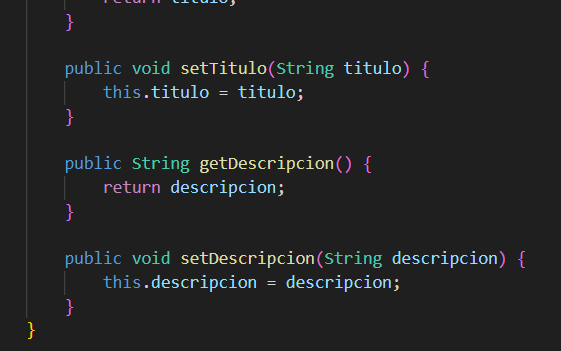
**guardar(Long usuarioId, Long cursoId) (@PostMapping("/inscripciones/agregar"))**

* Recibe parámetros desde un formulario (**@RequestParam)**.
* Crea una nueva inscripción con la fecha actual (**LocalDate.now()**).
* Guarda la inscripción usando el servicio (**inscripcionService.guardar(nueva))**.
* Redirige al usuario a la página principal (**redirect:/**).

**Dependencias DTO:**

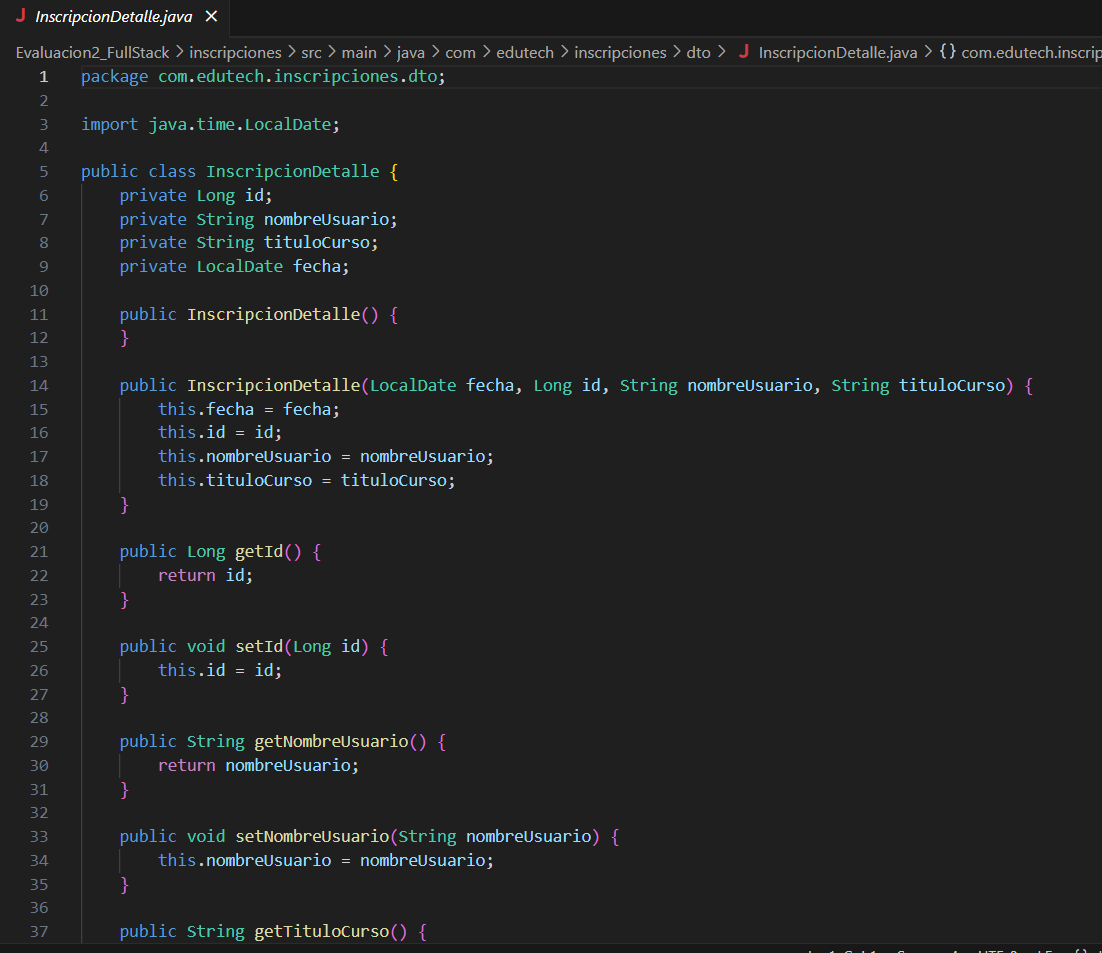
**Primera dependencia: CursoDTO.java**

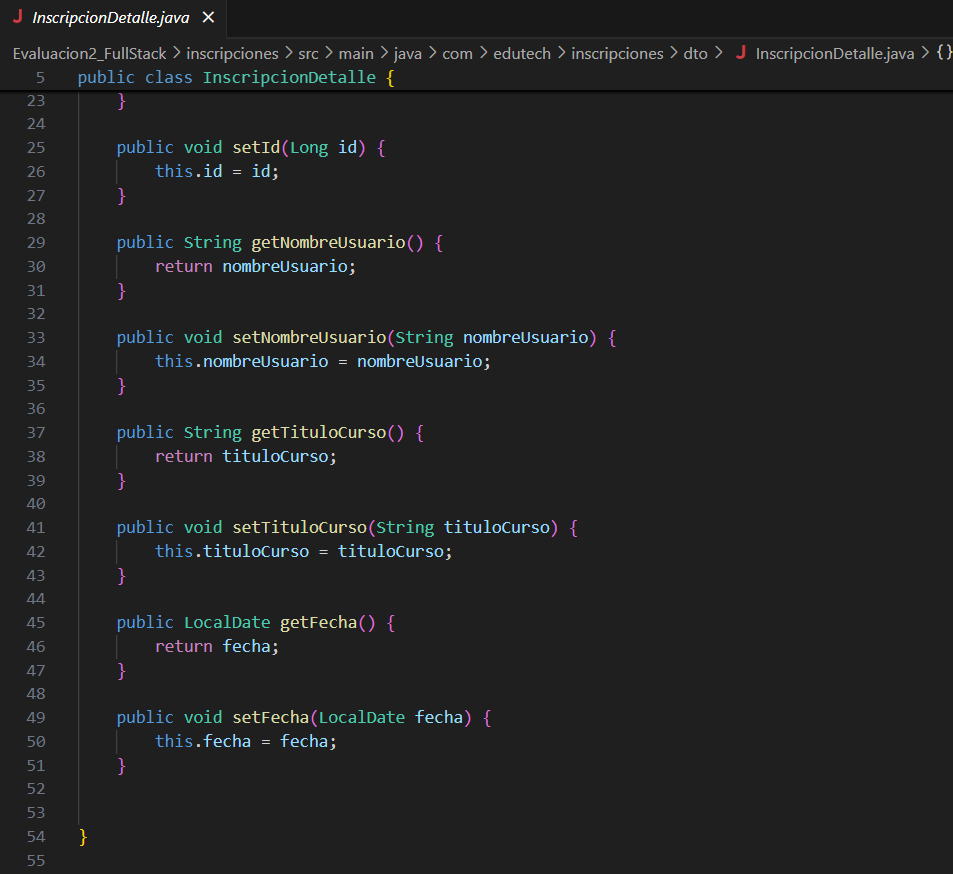




Esta instancia se usa para definir y transferir datos a otras instancias y otras partes diferentes de la aplicación, sin tener que exponer por completo la entidad del modelo. Por ende su propósito es:

* Encapsular datos de un curso (Id, Título y descripción )
* Poder facilitar la comunicación entre los demás servicios de la aplicación ejemplo: El consumo de una API externa
* Evitar exponer detalles internos del modelo para de esta forma poder mejorar la seguridad de la aplicación software
* Mejorar la optimización de las APIs Rest enviando solo la información completa en vez de la entidad completa

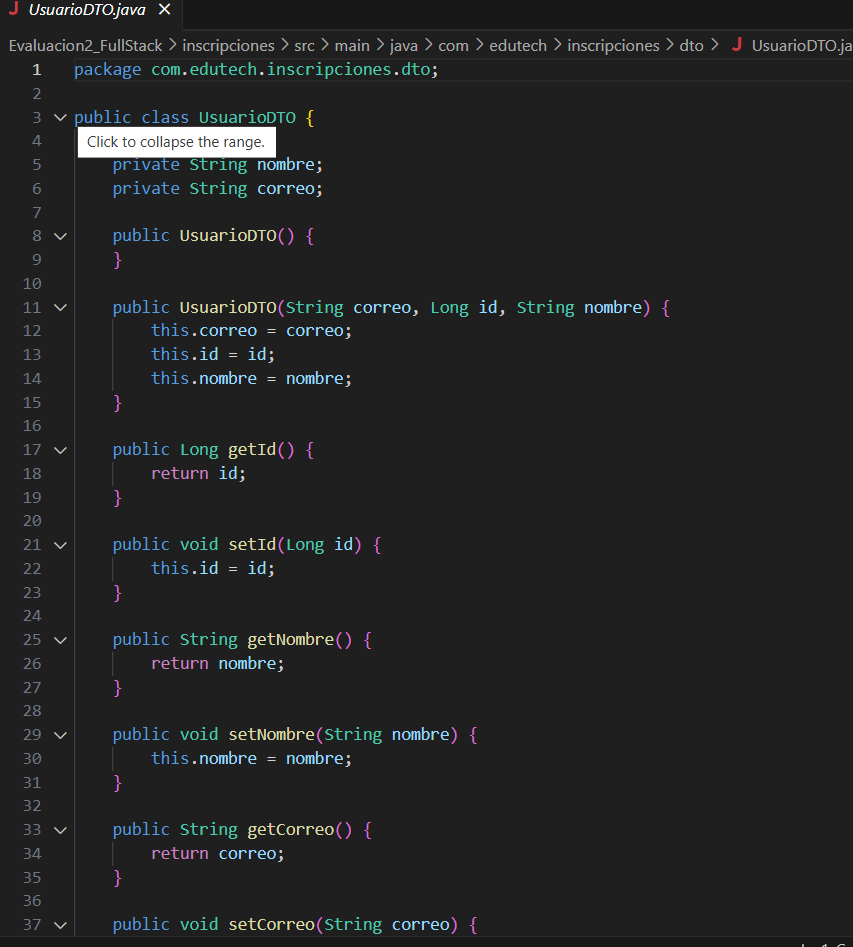
**Segunda instancia: InscripcionDetalle.java**

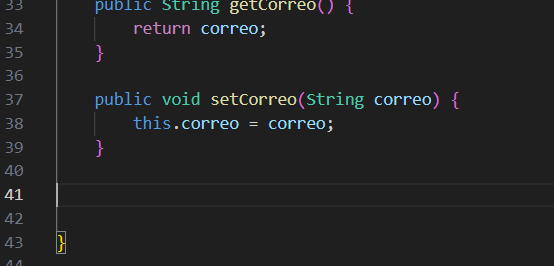


El propósito de esta clase es representar la información detallada de las inscripciones de los usuarios y a la vez no depender de otras entidades del modelo. Por lo que su funciones son:

* **Encapsular información** relevante de una inscripción, incluyendo: ID de la inscripción **(id)**  
   - Nombre del usuario **(nombreUsuario)**  
   - Título del curso **(tituloCurso)**  
   - Fecha de inscripción **(fecha)** **Facilitar la comunicación entre componentes** de la aplicación sin exponer la entidad de **Inscripcion** directamente.
* **Optimizar la transferencia de datos** en controladores y vistas, permitiendo mostrar información combinada sin hacer múltiples consultas.

**Tercera instancia: UsuarioDTO.java**

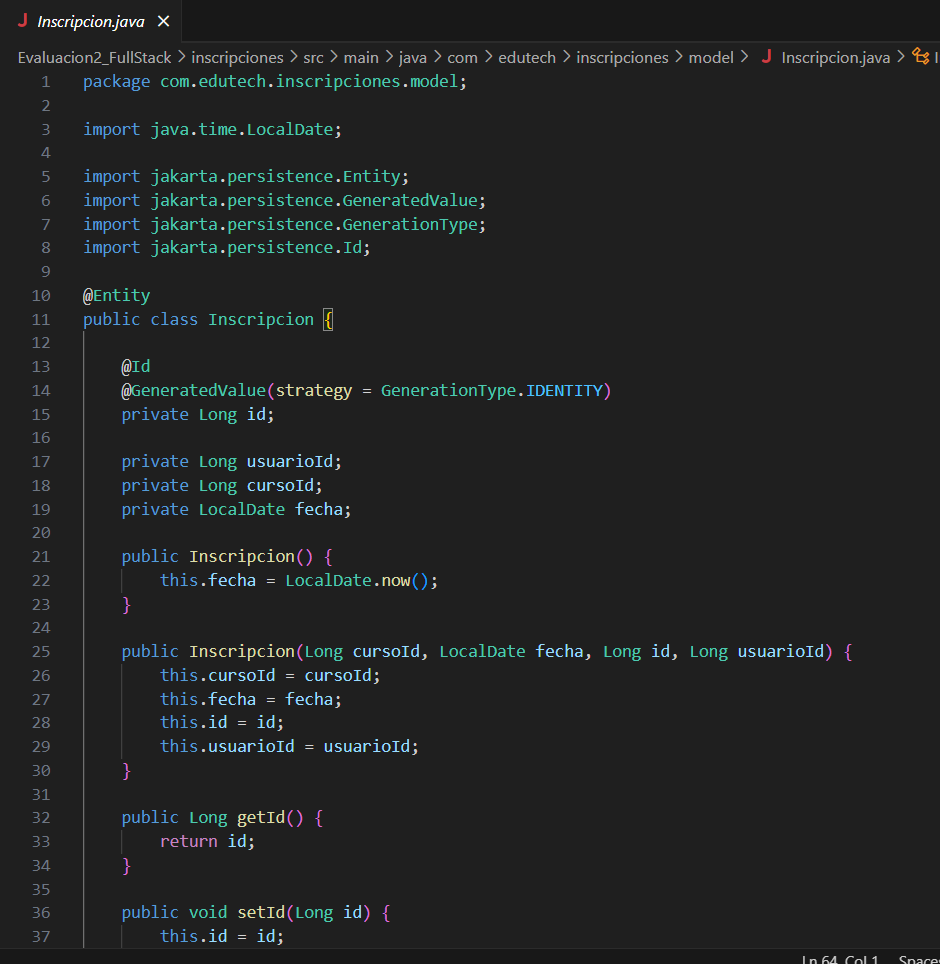




Este código presentado sirve para transferir la información necesaria de un usuario en la aplicación sin depender directamente de otras entidades del modelo software al igual que las 2 instancias anteriores de este modelo. Por lo cual este modelo tiene estas funciones:

* **Encapsular datos** de un usuario: ID, nombre y correo electrónico. **Facilitar la comunicación entre servicios** (por ejemplo, al consumir APIs externas).
* **Evitar exponer detalles internos** de la entidad **Usuario**.
* **Optimizar la transferencia de datos** en controladores y vistas, enviando solo la información relevante.

**Dependencia: Model**

**Instancia: Inscripcion.java**



Esta clase es un modelo de datos el cual se almacena en una base de datos para poder registrar las inscripciones de los usuarios en cursos. Este funciona como una tabla de en la base de datos permitiendo manejar información como:

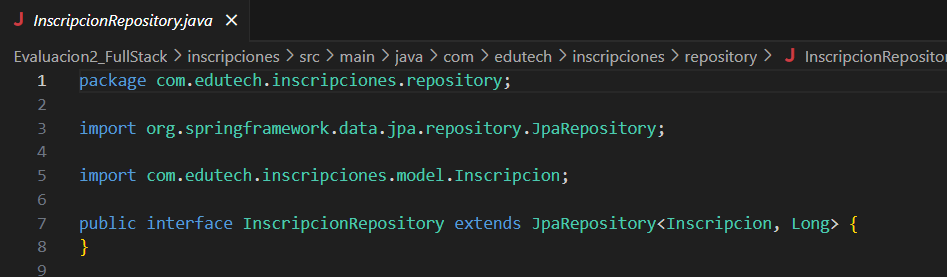
**Usuario ID** (**usuarioId**) → Relación con un usuario inscrito.  
**Curso ID** (**cursoId**) → Indica el curso al que se inscribe el usuario.  
**Fecha** (**fecha**) → Registro de cuándo se realizó la inscripción.

🔹 **Características clave**

* **@Entity** → Marca esta clase como una entidad de base de datos.
* **@Id** → Define el atributo **id** como clave primaria.
* **@GeneratedValue(strategy** **= GenerationType.IDENTITY)** → Permite que la base de datos pueda generar automáticamente el ID.
* Constructor vacío → Inicializa la fecha con **LocalDate.now().**

**Dependencia: Repository**

**Instancia: InscripcionRepository.java**

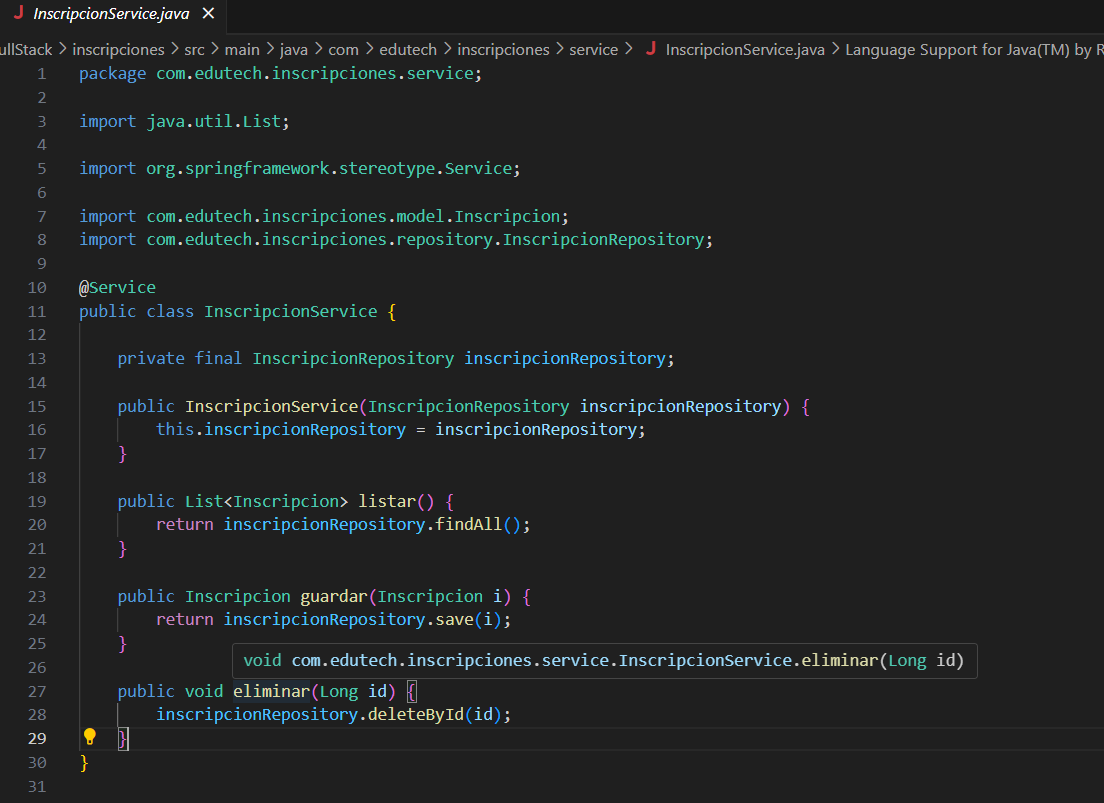
****

Esta dependencia es para poder realizar operaciones Crud (crear, leer, actualizar y eliminar inscripciones). Por ende sus funciones son:

* Consultar datos sin la necesidad de escribir en le SQL de manera manual
* Integrarse a la base de datos mediante el Spring JPA, simplificando del acceso a la base de datos

**Dependencia: Service**

**Instancia: InscripcionService.java**

****

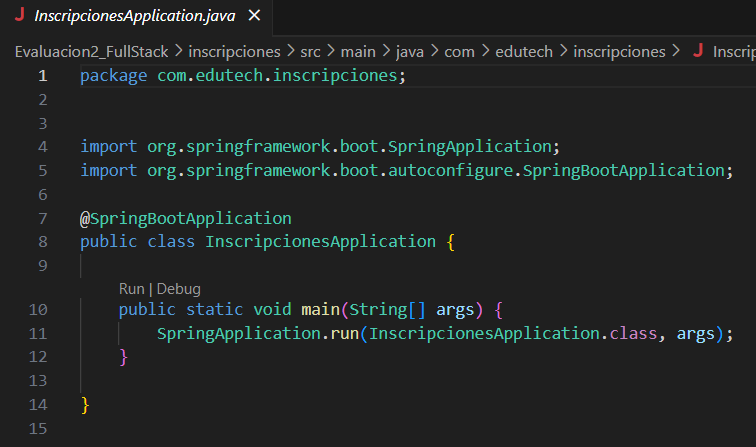
Esta instancia sirve para definir este servicio Spring Boot, el cual actúa como una capa intermedia entre el controlador y el repositorio. Por ende su propósito es:

**Abstrae la interacción con la base de datos**, evitando que los controladores accedan directamente a **InscripcionRepository.**  
 ✔ **Facilita la reutilización del código**, permitiendo que distintos controladores usen la misma lógica.  
 ✔ **Mejora la organización del proyecto**, separando responsabilidades en capas (controlador → servicio → repositorio).

🔹 **Funciones principales**

* **listar()** → Obtiene todas las inscripciones usando **findAll().**
* **guardar(Inscripcion i**) → Guarda una nueva inscripción en la base de datos con **save(i).**
* **eliminar(Long id)** → Borra una inscripción por su ID usando **deleteById(id).**

**Instancia: InscripcionApplication.java**



Esta instancia ayuda a definir la clase principal ya que este sería el punto de entrada para iniciar la aplicación software. Por lo que su propósito es inicializar y configurar de manera automáticamente los componentes de spring. Entonces sus funciones son:

**@SpringBootApplication** → Marca esta clase como la configuración principal de la aplicación, activando automáticamente el escaneo de componentes y la configuración de Spring Boot.  
- **SpringApplication.run(InscripcionesApplication.class, args);** → Inicia la aplicación, creando el contexto de Spring y levantando los servicios necesarios.  
- **Main(String[] args)** → Es el método principal que se ejecuta cuando la aplicación arranca.

### Componentes implementados

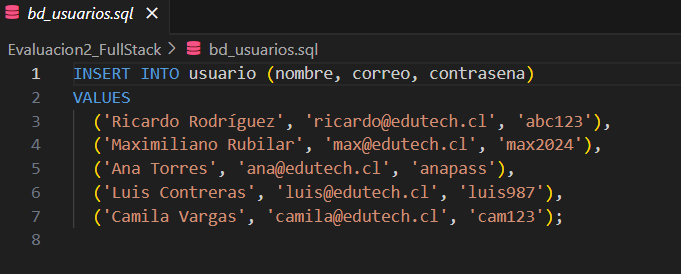
Se aplicaron muchos componentes en este programa software los cuales cada uno cumplió su propósito. Estos componentes son lo Siguientes:

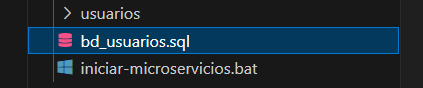
* **@Request controller y @RequestMapping**: El @Request controller significa que una clase es un **controlador REST**, lo que significa que manejará solicitudes HTTP y devolverá respuestas en formato JSON o XML. Básicamente, convierte la clase en un punto de acceso para la API. Mientras que el @RequestMapping sirve para definir la base URL que el controlador maneja al especificar los métodos HTTP
* **@PostMapping:** Ayuda a registrar una nueva inscripción, recibiendo los datos en el cuerpo de la solicitud
* **@RequestBody:** Sirve para extraer los datos del cuerpo de una solicitud HTTP y convertirlos en un objeto de java. Bastante útil al recibir datos en formato de JSON o XML y necesitarás procesarlos en la aplicación
* **@DeleteMapping (“/{id}))** : Este componente sirve para borrar inscripciones específica identificada como id
* **@RestTemplate:** Es para llamar a los servicios externos y así poder obtener datos de usuarios y cursos
* **(model.addAttribute(“inscripciones”, detalles)):** Este componente se usa para poder pasar datos desde un controlador a la vista de la aplicación web
* **@RequestParam :** Sirve para poder obtener parámetros de solicitud de HTTP, generalmente está relacionado con consultar URL o formularios. Útil a la hora capturar valores específicos como los de un cuerpo JSON
* **(LocalDate.now()):** Este componente se usa para crear una nueva inscripción con su fecha actual
* **(inscripcionService.guardar(nueva))** y **(redirect:/):** El primero sirve para guardar las inscripciones usando el servicio y la segunda para redirigir a la usuario a la página principal
* **Encapsulación de información:** Es útil a la hora de restringir los datos específicos de un objeto con métodos específicos para de esta forma poder mejorar la seguridad en los códigos
* **@Entity:** Este componente se usa para poder marcar una clase como una entidad de base datos dentro de un JPA(Java Persistence Api) por ende este funciona solo dentro de una API en un programa software
* **@Id:** Sirve para definir un atributo en este caso el de un **id** como clave primaria
* **@GeneratedValue**(**strategy** **= GenerationType.IDENTITY):** Este componente logra que cada base de datos pueda generar de manera automática un id
* **listar():** Este componente sirve para poder obtener todas las inscripciones de los usuarios
* **FindAll():** Sirve para obtener todos los registros de una entidad en la base de datos. Este componente es prácticamente el la operación más común de un Crud(Crear, leer, actualizar, eliminar) y también permite recuperar una lista completa de objetos sin la necesidad de tener que escribir consultas desde la base de datos agilizando el programa software
* **Guardar(inscripcion i):** Este componente sirve para guardar la nueva inscripción de los usuarios en la base de datos mediante un save**(i)**
* **Eliminar(Long id):** Como se ve en el nombre de este componente. Este sirve para eliminar las inscripciones de los usuarios designados usando su ID como registro
* **deleteById(id):** Es el componente o mejor dicho código, el cual se encarga de eliminar al usuario utilizando el Id de este mismo al igual que el código anterior
* **@SpringBootAplication:** Es el componente más usado y más importante ya que este está encargado de configurar, habilitar modificaciones y configuraciones, también permite que el Spring escanee los paquetes y registre los componentes del objeto
* **SpringApplication.run(InscripcionesApplication.class, args);** Este componente es para iniciar la aplicación, creando el contexto de spring y levantando los servicios necesarios para el sistema software
* **Main(String[ ] args):** Es el método principal que se ejecuta cuando la aplicación arranca ya que sin este el programa no compilaria ni sería funcional

## 8.2 Base de datos

En la implementación de la base de datos logramos implementar mediante visualcode estudios y la base de datos se creó en MySql Workbench. esto es posible gracias a las apis y implementos de instancias en base al modelo de tipo crud (Crear, Leer, Actualizar, Eliminar)

Aquí los ejemplos:





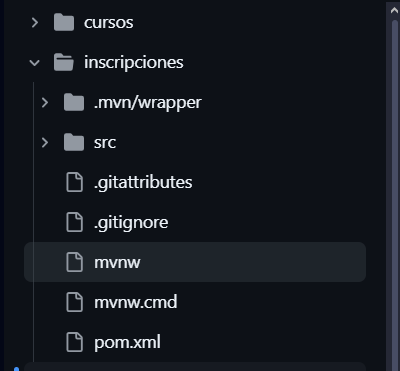
Para finalizar la explicación se puede decir que en db\_usuarios.sql se almacenó la información de los usuarios que están usando la aplicación hasta el momento para que así puedan quedar integrados en la base de datos logrando de esta manera que el programa software quede funcional

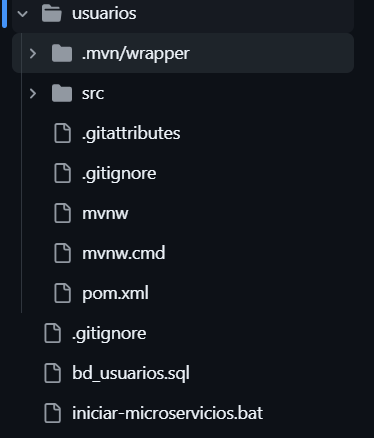
# 9. Git – Git Hub

En github dividimos los codigos segun su proposito y funcionalidades en el sistema como se puede apreciar en esta imagen



En esta imagen se ve como se ordeno y clasifico el Programa software. Este está ordenado de la misma manera que en visual Studio code ya que de esta forma es más rápido que este corra y evitar confusiones o perder alguno de los documentos del programa

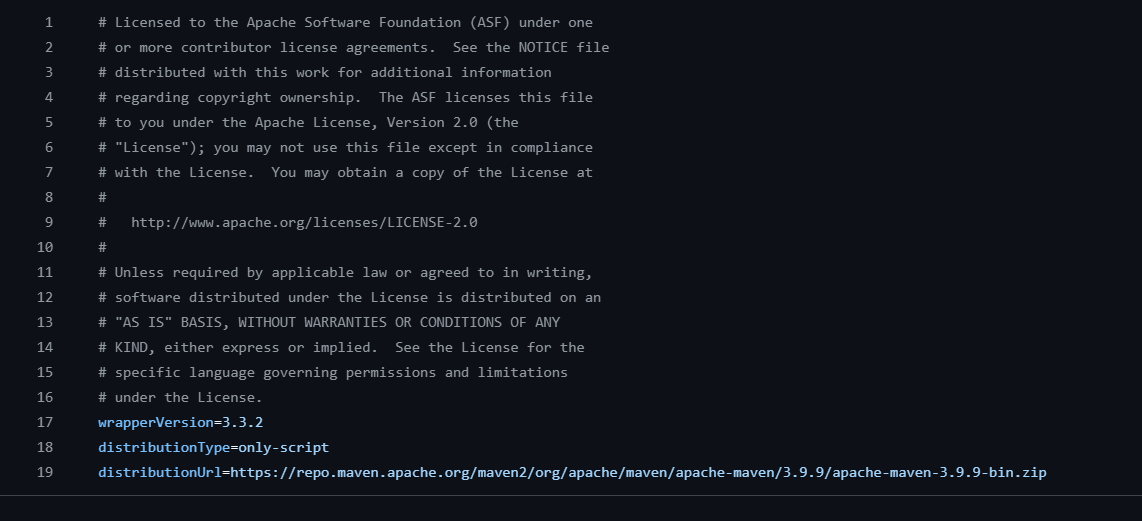






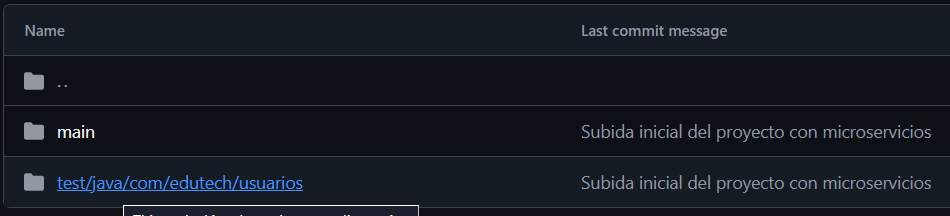
A continuación se muestran el orden en el que se organizó los documentos del programa por github en el cual van primero los documentos .mvn/wrapper , src, .gitattributes, .gitignore, mvnw, mvnw.cmd y por último pom.xml

En .mvn/wrapper se encuentra el documento [maven-wrapper.properties](https://github.com/ac-tiare/Implementacion-de-servicios-REST/blob/main/usuarios/.mvn/wrapper/maven-wrapper.properties)



Este guarda un código que sirve para gestión de dependencias y construcción de proyectos en Java.

En el documento src se encuentra el [main](https://github.com/ac-tiare/Implementacion-de-servicios-REST/tree/main/usuarios/src/main) y el [test/java/com/edutech/usuarios](https://github.com/ac-tiare/Implementacion-de-servicios-REST/tree/main/usuarios/src/test/java/com/edutech/usuarios)



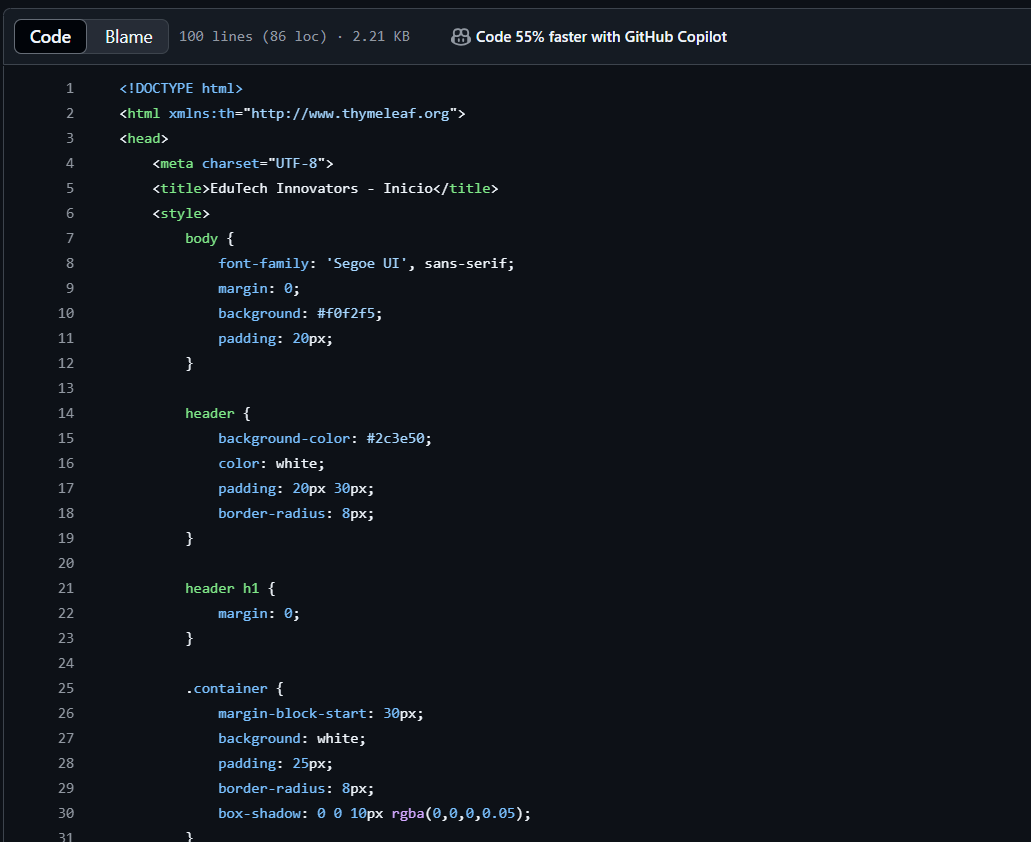
En en main se encuentra el [java/com/edutech/usuarios](https://github.com/ac-tiare/Implementacion-de-servicios-REST/tree/main/usuarios/src/main/java/com/edutech/usuarios) este se encuentra los repositorios de las instancias del proyecto de software ordenados en controller, model, repository, service y UsuariosApplication.java como se muestra en la siguiente imagen. Este representa los códigos del programa software en el que se trabajó en este grupo



En resource se encuentra templates y application.properties donde el templates representa las lineas de codigo de html la cual es la pantalla donde el usuario interactúa el acceder a la aplicación de estudio y realizar sus actividades



Vista del HTML el cual está nombrado como index.html



y en ApplicactionProperties se presenta la conexión de la base de datos al programa software mediante el uso del localhost como se muestra en la siguiente imagen





Por último, estos microservicios donde el gitignore sirve para poder evitar que ciertos archivos y carpetas puedan ser rastreados para de esta forma reforzar la seguridad en la aplicación software. Mientras que el bd\_usuarios.sql es donde se almacena la información de los usuarios que están integrados en la base de datos y para finalizar el iniciar-microservicos.bat es para inicializar múltiples microservicios en un proyecto SpringBoot específicamente diseñado para el entorno digital de edutech Innovators Spa

# 10.Pruebas

Las pruebas unitarias se realizaron con JUnit y Mockito, validando la lógica de negocio y los controladores. Las pruebas de integración permitieron verificar la correcta comunicación entre microservicios y el comportamiento completo del sistema.

# 11.Resultados obtenidos

El sistema logró cumplir con los objetivos propuestos. Se obtuvo una aplicación modular, funcional y testeada. El uso de microservicios permitió aislar errores y facilitar el desarrollo paralelo. Las vistas permiten gestionar cursos, usuarios, inscripciones, evaluaciones y notas de forma efectiva.

## 11.1 Calificaciones

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## 11.2 Usuarios

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## 11.3 Cursos

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## 11.4 Inscripciones

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

# 12.Conclusiones y recomendaciones

El enfoque basado en microservicios facilita el mantenimiento del sistema y su escalabilidad. Se recomienda utilizar herramientas de orquestación como Docker y Kubernetes en futuras versiones, así como implementar autenticación y autorización para robustecer la seguridad del sistema.