INFORME DEL PROYECTO FINAL

Desarrollo de API REST con Spring Boot

Tecnología Superior en Desarrollo de Software

Integrantes:	Concepción Arequipa y Josue Patiño
Tema:	Estudiante: Gestión de estudiantes de un instituto
Asignatura:	Desarrollo Backend con Spring Boot
Modalidad:	Individual o en parejas de dos estudiantes
Fecha de entrega y defensa	**Entrega:** Domingo 25 de enero hasta las 23:59 en el repositorio de One-Drive. **Defensa:** Lunes 26 de enero a las 10:00 am.

1.Estructura del proyecto

En el proyecto se implementó una arquitectura en capas o paquetes:

- **controller:** Capa encargada de recibir las solicitudes del cliente y responderlas.

  Responsabilidades:
  1. Definir los endpoints REST (GET, POST, PUT, DELETE)
  2. Recibir datos de entrada (JSON)
  3. Llamar a la capa service
  4. Devolver respuestas HTTP adecuadas

- **servicie:** Contiene la lógica de negocio de la aplicación.

  Responsabilidades:

  1. Procesar reglas del negocio
  2. Validar datos
  3. Decidir qué hacer ante errores
  4. Llamar a la capa repository

- **repository:** Se encarga del acceso a la base de datos.
  Responsabilidades:

  1. Guardar datos
  2. Consultar registros
  3. Actualizar información
  4. Eliminar registros

- **exception:** Gestiona los errores personalizados de la aplicación.

  Responsabilidades:

  1. Definir excepciones propias
  2. Manejar errores de forma centralizada
  3. Devolver códigos HTTP correctos (404, 400, 500)

- **entity:** Representa las tablas de la base de datos.

  Responsabilidades:

  1. Definir los atributos del objeto
  2. Mapear la entidad a una tabla
  3. Usar anotaciones JPA (@Entity, @Id)

2. Entidad: Estudiante

La entidad Estudiante fue implementada cumpliendo con los requisitos establecidos para el modelo de datos. Se definieron más de cinco atributos relevantes, incluyendo un identificador único (id) de tipo Long. Se utilizaron correctamente las anotaciones JPA @Entity, @Id y @GeneratedValue para el mapeo de la entidad con la base de datos.

Asimismo, se incorporaron atributos de tipo String y numéricos, y se aplicaron validaciones mediante anotaciones como @NotBlank, @NotNull, @Size, @Min, @Max y @Email, garantizando la integridad y consistencia de los datos antes de su persistencia. De esta manera, se cumple con el uso mínimo de validaciones solicitadas y se asegura un correcto control de la información ingresada en el sistema.

Estructura de la identidad:

Atributo	Tipo	Validación
id	Long	@Id, @GeneratedValue(strategy = GenerationType.IDENTITY)
nombre	String	@NotBlank, @Size(min = 2, max = 50)
apellido	String	@NotBlank, @Size(min = 2, max = 50)
cedula	String	@NotBlank, @Size(min = 10, max = 10), @Column(unique = true)

edad	Integer	@NotNull, @Min(16), @Max(90)
email	String	@NotBlank, @Email, @Column(unique = true)
carrera	String	@NotBlank, @Size(min = 5, max = 100)
semestre	Integer	@NotNull, @Min(1), @Max(10)

3.Implementacion del repositorio de Estudiante

Se implementó la interfaz EstudianteRepository, la cual extiende de JpaRepository<Estudiante, Long>, permitiendo el acceso a la base de datos mediante Spring Data JPA. Gracias a esta extensión, se dispone automáticamente de los métodos CRUD básicos para la entidad Estudiante, tales como listar, buscar por identificador, crear, actualizar y eliminar registros, sin necesidad de definir métodos adicionales.

El repositorio se encuentra correctamente anotado con @Repository, lo que permite su detección y gestión por parte del contenedor de Spring. Esta implementación cumple con el requisito de acceso a datos de la aplicación, manteniendo una correcta separación de responsabilidades dentro de la arquitectura en capas del proyecto.

4. Manejo de errores

Dentro del paquete exception se implementó la clase RecursoNoEncontradoException, la cual extiende de RuntimeException, con el propósito de manejar los casos en los que un estudiante no es encontrado en el sistema. Esta excepción permite generar mensajes de error claros y personalizados, y es utilizada en la capa de servicio para controlar errores de negocio. Su manejo centralizado facilita la devolución de respuestas HTTP adecuadas, como el código 404 (Not Found), mejorando la calidad y robustez de la API REST.

5.Base de datos

Para el desarrollo del proyecto se utilizará una base de datos MySQL ejecutada localmente, la cual se emplea para el almacenamiento y gestión de la información de los estudiantes. La conexión a la base de datos se realiza mediante las configuraciones proporcionadas por Spring Boot, permitiendo la persistencia de los datos durante la ejecución de la aplicación.

5.Dependencias

Para el desarrollo del sistema se configuró un proyecto Spring Boot incorporando las dependencias necesarias para la construcción de una API REST y la persistencia de datos:

1. Spring Web: Usada para la creación de controladores y la exposición de servicios REST, permitiendo la comunicación entre el cliente y el servidor mediante solicitudes HTTP.

2. Spring Data JPA: Usada para la gestión de la persistencia, facilitando la interacción con la base de datos mediante repositorios y eliminando la necesidad de escribir consultas SQL manuales para las operaciones básicas.

3. Validation: Se empleó para validar los datos de entrada, garantizando la integridad y consistencia de la información enviada por el usuario.

4. MySQL Driver: Permite la conexión entre la aplicación y la base de datos.

5. Spring Boot DevTools: Se uso para agilizar el proceso de desarrollo, proporcionando recarga automática de la aplicación ante cambios en el código.

6. Cabe destacar que no se utilizó H2 Database, ya que el proyecto trabaja directamente con una base de datos MySQL local.

6. Implementación de la capa Service

La capa **service** fue implementada para centralizar la lógica de negocio del sistema de gestión de estudiantes. En esta capa se definieron los métodos necesarios para realizar las operaciones CRUD sobre la entidad **Estudiante**, actuando como intermediaria entre el controlador y el repositorio.

Responsabilidades cumplidas:

- Aplicar reglas de negocio.
- Validar la existencia del estudiante antes de realizar operaciones.
- Manejar excepciones personalizadas.
- Delegar el acceso a datos al repositorio.

En caso de que un estudiante no exista al momento de buscarlo, actualizarlo o eliminarlo, se lanza la excepción personalizada **RecursoNoEncontradoException**, garantizando un control adecuado de errores.

7. Implementación del controlador (Controller)

La capa **controller** fue desarrollada para exponer los servicios REST del sistema, permitiendo la interacción con los clientes mediante solicitudes HTTP. Se definieron correctamente los endpoints REST para la entidad **Estudiante**, siguiendo las buenas prácticas de una API RESTful.

Endpoints implementados:

- **GET /estudiantes**
  Permite obtener la lista de todos los estudiantes registrados.

- **GET /estudiantes/{id}**
  Permite obtener un estudiante específico mediante su identificador.

- **POST /estudiantes**
  Permite registrar un nuevo estudiante en el sistema.

- **PUT /estudiantes/{id}**
  Permite actualizar la información de un estudiante existente.

- **DELETE /estudiantes/{id}**
  Permite eliminar un estudiante del sistema.

El controlador recibe los datos en formato JSON, los valida mediante anotaciones y delega el procesamiento a la capa de servicio, devolviendo respuestas HTTP apropiadas como **200 OK**, **201 Created**, **404 Not Found** y **400 Bad Request**.

8. Validación de datos

Se aplicaron validaciones en la entidad **Estudiante** utilizando anotaciones del paquete EstudianteService asegurando que los datos enviados por el cliente cumplan con los requisitos establecidos antes de ser persistidos en la base de datos.

Estas validaciones permiten:

- Evitar el ingreso de datos incompletos o incorrectos.

- Garantizar la integridad de la información.

- Reducir errores a nivel de base de datos.

  La validación se ejecuta automáticamente al recibir solicitudes **POST** y **PUT**, devolviendo mensajes de error claros cuando los datos no cumplen las restricciones definidas.

9. Pruebas del sistema

El funcionamiento de la API REST fue verificado mediante herramientas de prueba como **Postman**, realizando solicitudes HTTP para cada una de las operaciones CRUD implementadas.

La colección de Postman con las pruebas de todos los endpoints se encuentra en:

coleccion-postman/Proyecto_gestionEstudiantes-postman_collection.json

Indicaciones de uso:

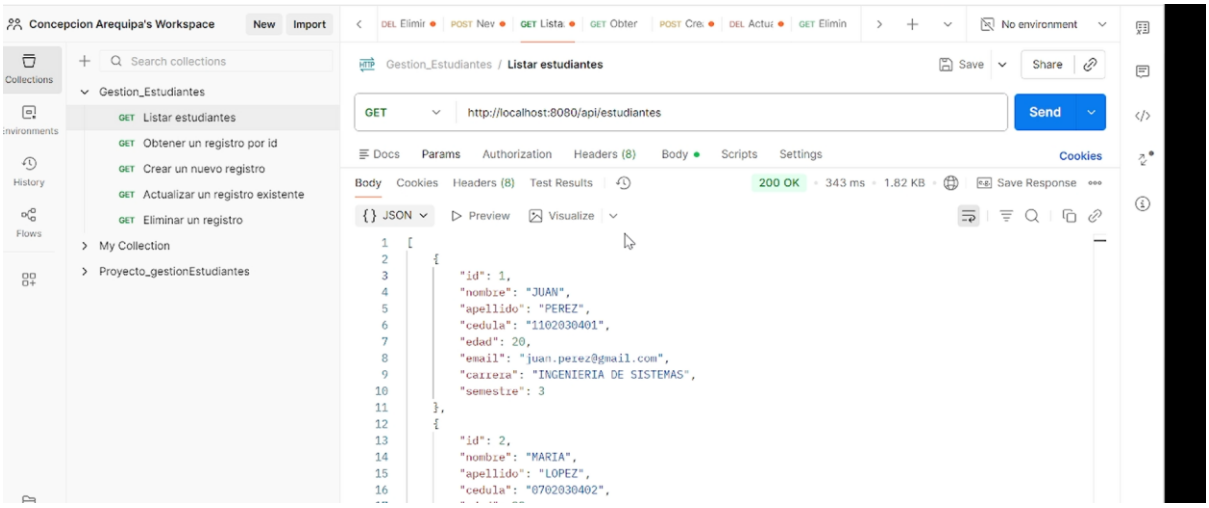
1. Abrir Postman

2. Import → seleccionar el archivo JSON

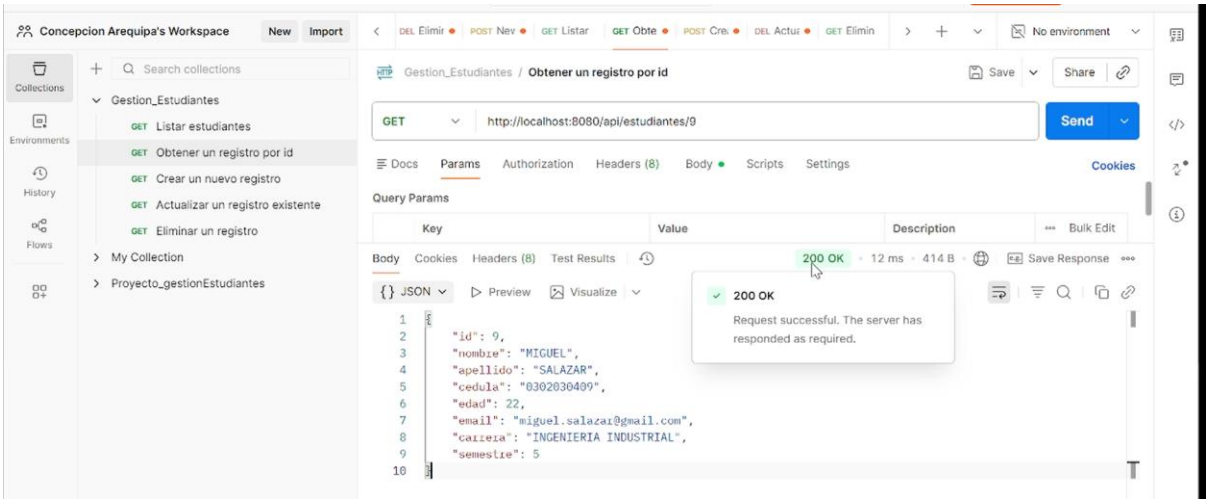
3. Ejecutar los endpoints con el backend en ejecución

Pruebas realizadas:

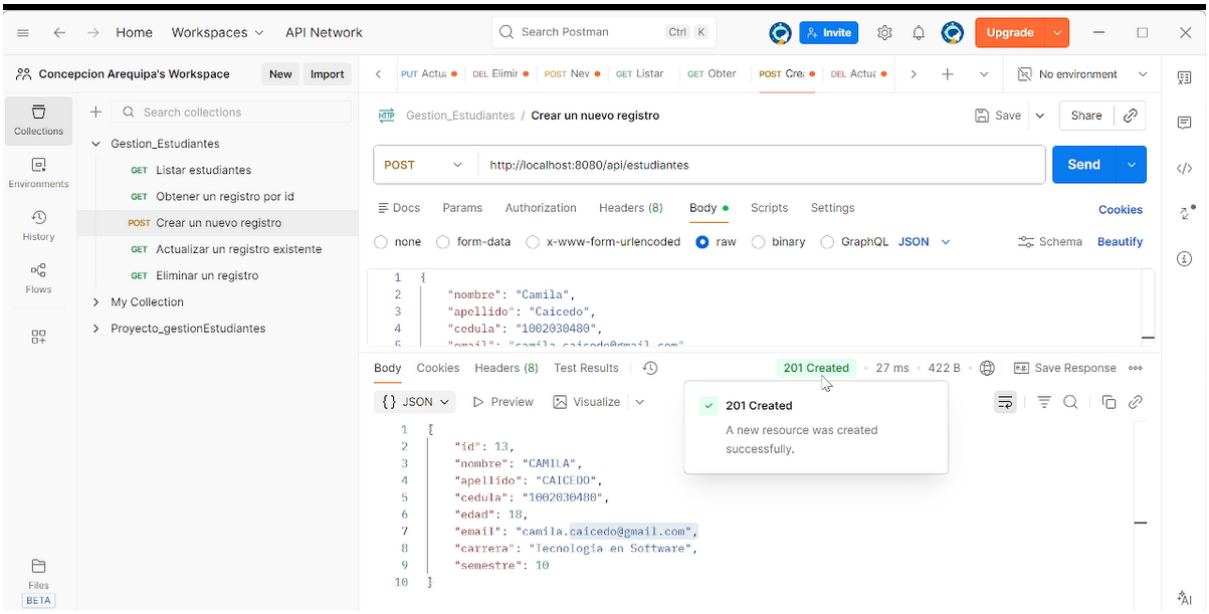
1. Listar todos los estudiantes (GET /api/entidad)



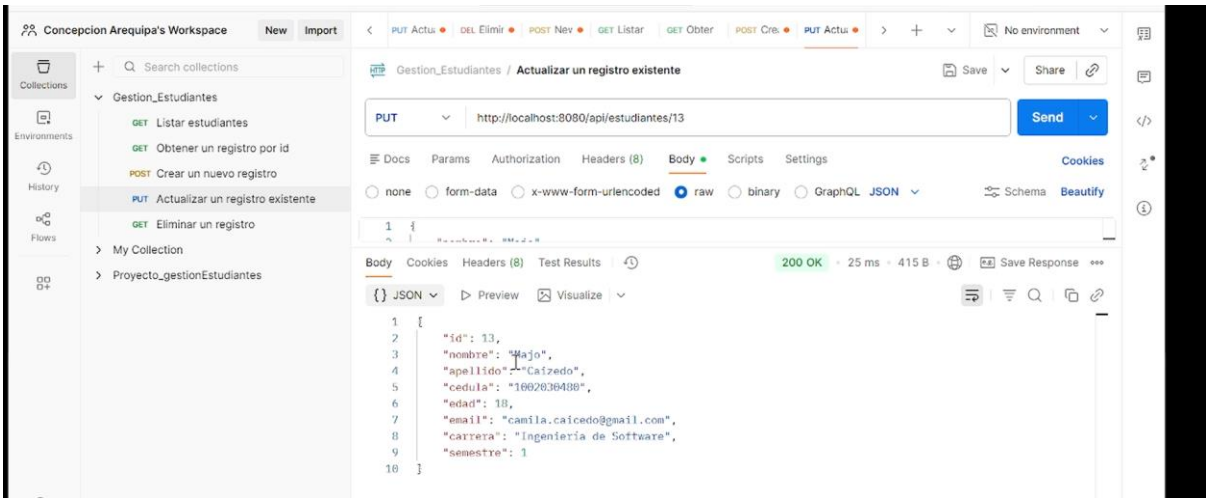
2. Obtener un estudiante por ID (GET /api/entidad/{id})



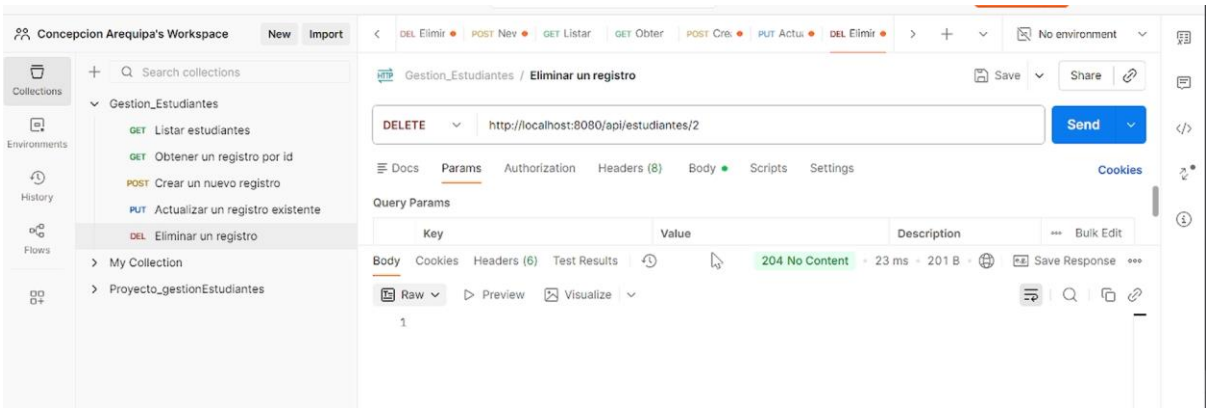
3. Crear un nuevo estudiante (POST /api/entidad)



4. Actualizar un estudiante existente (PUT /api/entidad/{id})



5. Eliminar un estudiante (DELETE /api/entidad/{id})



Las pruebas permitieron comprobar que los endpoints funcionan correctamente, que las validaciones se aplican de forma adecuada y que los errores son manejados correctamente mediante excepciones personalizadas.

10. Bonificaciones (opcional)

- Documentación de la API con Swagger (OpenAPI)

Para facilitar la documentación y prueba de los endpoints de la API REST, se integró Swagger mediante la especificación OpenAPI, lo que permite visualizar y probar los servicios de manera interactiva desde el navegador, sin necesidad de herramientas externas como Postman.

Swagger genera automáticamente la documentación de la API a partir de los controladores definidos en el proyecto, mostrando información clara sobre los endpoints disponibles, los métodos HTTP utilizados, los parámetros requeridos y los posibles códigos de respuesta.

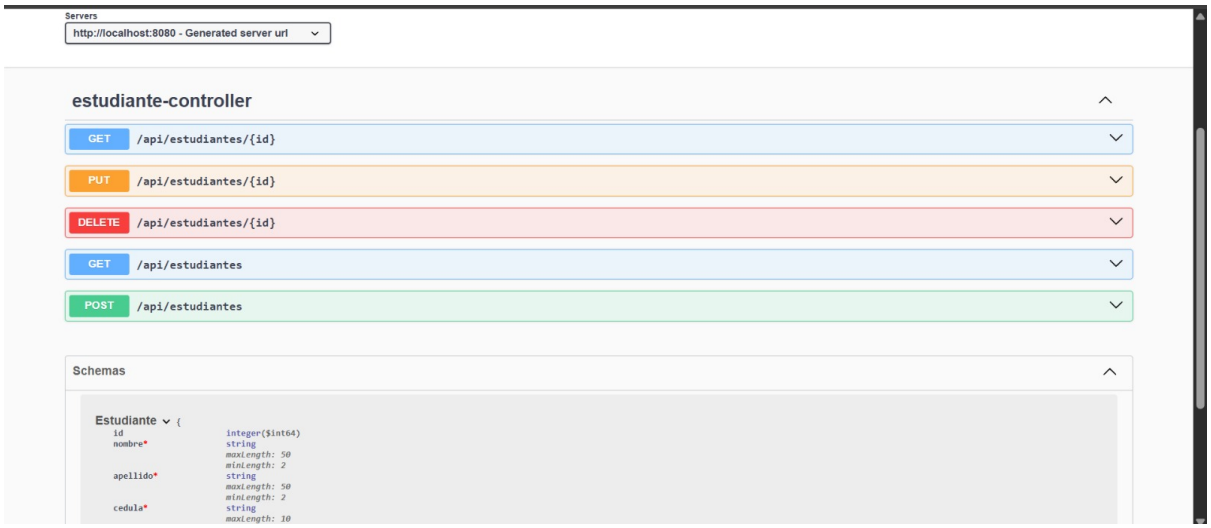
Características principales:

- Visualización automática de todos los endpoints REST.
- Pruebas directas de los servicios GET, POST, PUT y DELETE.
- Descripción clara de los modelos de datos utilizados.
- Mejora la comprensión y mantenibilidad de la API.

La documentación de Swagger se encuentra disponible en la siguiente ruta una vez que la aplicación está en ejecución:

http://localhost:8080/swagger-ui/index.html

La implementación de Swagger permite una mejor experiencia para desarrolladores y evaluadores, facilitando la validación del funcionamiento correcto de la API y sirviendo como documentación técnica oficial del proyecto.



Swagger api/estudiantes

- **Implementación básica de Frontend (Listado de estudiantes)**

Como complemento a la API REST, se desarrolló una interfaz frontend básica, cuyo objetivo principal es listar todos los estudiantes registrados en el sistema. Este frontend funciona como una vista de solo lectura y no implementa operaciones avanzadas como búsqueda por ID, creación, edición o eliminación de registros.

El frontend se conecta a la API REST mediante una solicitud HTTP GET al endpoint:

GET /estudiantes

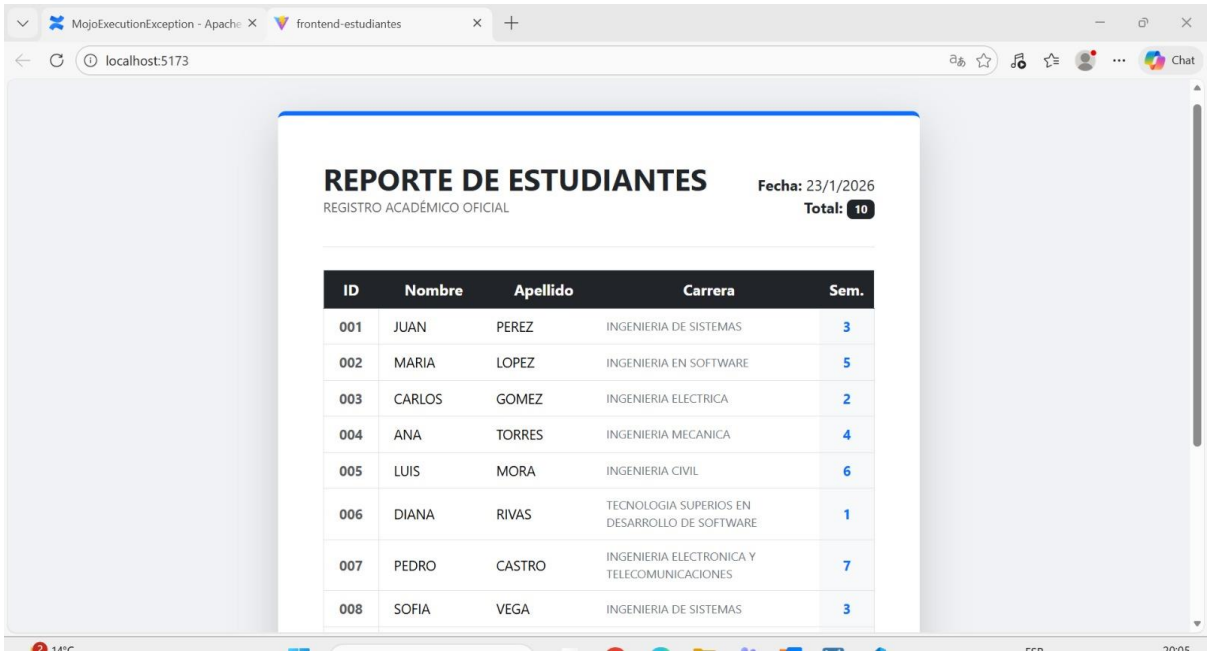
Funcionalidad implementada:

- Consumo de la API REST desde el frontend.
- Visualización de la lista completa de estudiantes.
- Presentación de los datos en formato de tabla .
- Uso exclusivo del método GET (sin operaciones CRUD adicionales).
- Boton para imprimir la lista de estudiantes.

Alcance y limitaciones:

- No se implementó búsqueda por ID.
- No se permite registrar, editar ni eliminar estudiantes desde el frontend.
- El objetivo es únicamente demostrar la correcta integración entre frontend y backend.

Este frontend cumple una función demostrativa, evidenciando que la API puede ser consumida correctamente por una aplicación cliente y validando el correcto funcionamiento del endpoint de listado general.



Froontend React

- **Valor agregado del proyecto**

  La incorporación de Swagger y un frontend básico aporta un valor adicional al proyecto, ya que:

  1. Facilita la documentación técnica y pruebas de la API.
  2. Demuestra la interoperabilidad entre backend y frontend.
  3. Mejora la comprensión del sistema por parte de terceros.
  4. Refuerza el enfoque práctico del desarrollo de la API REST.

11. Resultados obtenidos

Como resultado del desarrollo del proyecto, se obtuvo una API REST funcional para la gestión de estudiantes de un instituto, cumpliendo con los requisitos establecidos en la asignatura **Desarrollo Backend con Spring Boot**.

El sistema permite:

- Gestionar estudiantes de forma eficiente.
- Mantener una correcta separación de responsabilidades.
- Garantizar la integridad de los datos.
- Manejar errores de forma clara y controlada.

12. Conclusiones

- Se implementó correctamente una arquitectura en capas, facilitando el mantenimiento y escalabilidad del sistema.
- El uso de Spring Boot y Spring Data JPA permitió simplificar el desarrollo de la API REST.
- La aplicación de validaciones y manejo de excepciones mejoró la seguridad y confiabilidad del sistema.
- El proyecto cumple con los objetivos planteados y demuestra el correcto uso de buenas prácticas en el desarrollo backend.
- La integración de Swagger (OpenAPI) permitió documentar la API REST de forma clara e interactiva, facilitando la visualización y prueba de los endpoints disponibles.
- Frontend básico utilizando React, el cual permite únicamente listar todos los estudiantes registrados mediante el consumo del endpoint GET /estudiantes.

13. Recomendaciones y mejoras futuras

Como trabajo futuro, el sistema puede ser mejorado mediante:

- Implementación de autenticación y autorización con Spring Security.
- Registro de auditoría (logs).
- Paginación y filtrado de resultados.
- Implementación de pruebas unitarias.

14. Entregables

1. **Código fuente y documentación asociada** con enlace a repositorio GitHub.

   https://github.com/ConcepcionArequipa/Proyecto_Final_POO.git

2. **Colección de Postman/ApiDog** con las pruebas de cada endpoint

   Nota: Se encuentra en el proyecto dentro de la carpeta **coleccion-postman**

3. **Video en youtube o tik tok con visibilidad pública, donde muestren el funcionamiento de la aplicación.**

https://youtu.be/ggYgfj8rMV8

4. **Sustentación oral** (5 a 10 minutos) explicando el código y 5 minutos adicional respondiendo preguntas del docente, por cada minuto que se exceda es un punto menos al promedio.

https://www.canva.com/design/DAG_d7c6Czk/XnLbn4s82mf-M2Ki15aRpw/edit?utm_content=DAG_d7c6Czk&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

15. Bibliografía:

Microservicios. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/microservices/

Pattern: Database per service. (s. f.). microservices.io. https://microservices.io/patterns/data/database-per-service.html

GeeksforGeeks. (2025, 23 julio). *MVC Architecture vs. Microservices Architecture*. GeeksforGeeks. https://www.geeksforgeeks.org/system-design/mvc-architecture-vs-microservices-architecture/

Lau Sanabria. (2025, July 11). *Así de fácil es crear una API Rest en Java con Spring Boot.. Te lo explico paso a paso!* [Video]. YouTube. https://www.youtube.com/watch?v=YqlxKOY2QmY

DATACLOUDER. (2020, June 3). *Desarrollar Api Rest con Java Spring Boot, explicación completa en 20 min.* [Video]. YouTube. https://www.youtube.com/watch?v=vTu2HQrXtyw