**TRABAJO PRACTICO ARQUITECTURA WEB**

**Aplicacion Web**



**PROFESORES**

DIEGO MARAFETTI

ADRIAN EZEQUIEL MARTINEZ

**ALUMNO**

EDUARDO NICOLAS ZURBRIGGEN

**MATERIA**

ARQUITECTURA WEB

Contenido

[Descripción de la aplicación desarrollada (no olvidar el alcance) 3](#_Toc201066250)

[Descripción 3](#_Toc201066251)

[Alcance de la Aplicación 3](#_Toc201066252)

[Justificación de la arquitectura, lenguajes de programación y toda decisión en torno al proyecto 4](#_Toc201066253)

[Arquitectura de App 4](#_Toc201066254)

[Capas del sistema: 4](#_Toc201066255)

[Lenguajes de Programación y Frameworks 5](#_Toc201066256)

[Por cada endpoint del backend: Endpoint, verbo HTTP, descripción 6](#_Toc201066257)

[Endpoints del Backend 6](#_Toc201066258)

[Entidad: Alumno 6](#_Toc201066259)

[Entidad: Curso 7](#_Toc201066260)

[Entidad: Inscripción 7](#_Toc201066261)

[Testeo de endpoints con PostMan 9](#_Toc201066262)

# Descripción de la aplicación desarrollada (no olvidar el alcance)

## Descripción

Esta aplicación web se desarrollo para permitir la gestión de Alumnos, Cursos e Inscripciones de Alumnos a Cursos. Cada Alumno una vez creado puede inscribirse a un Curso existente.

En la sección de Inscripciones podemos ver la lista de alumnos inscriptos a los respectivos cursos y tenemos la posibilidad de realizar la inscripción de Alumnos a Cursos.

## Alcance de la Aplicación

La aplicación permite realizar las siguientes operaciones:

* **Gestión de Alumnos:**
  + **Listar:** Visualizar todos los alumnos registrados en el sistema.
  + **Crear:** Registrar nuevos alumnos con sus datos pertinentes.
  + **Actualizar:** Modificar la información existente de un alumno.
  + **Eliminar:** Dar de baja a un alumno del sistema.
* **Gestión de Cursos:**
  + **Listar:** Visualizar todos los cursos disponibles.
  + **Crear:** Añadir nuevos cursos con sus detalles (nombre, descripción, etc.).
  + **Eliminar:** Eliminar un curso del sistema.
* **Gestión de Inscripciones:**
  + **Listar:** Visualizar todas las inscripciones realizadas.
  + **Crear:** Inscribir a un alumno en un curso específico.
  + **Eliminar:** Cancelar la inscripción de un alumno a un curso.

La interfaz de usuario (frontend) facilita la interacción con estas funcionalidades, presentando la información de forma clara y permitiendo las operaciones de creación, lectura, actualización y eliminación (CRUD).

# Justificación de la arquitectura, lenguajes de programación y toda decisión en torno al proyecto

## Arquitectura de App

La aplicación sigue una arquitectura **en capas**, que permite separar responsabilidades y facilitar el mantenimiento y escalabilidad del sistema.

### Capas del sistema:

* **Capa de Presentación (Front-end):**
  + Desarrollada en React.js.
  + Se comunica con la API REST del back-end mediante peticiones HTTP.
  + Se encarga de mostrar los datos y manejar las interacciones del usuario.
* **Capa de Controlador (Controller):**
  + Define los endpoints REST usando anotaciones de Spring (@RestController).
  + Recibe las peticiones HTTP del front-end y delega la lógica a la capa de servicio.
* **Capa de Servicio (Service):**
  + Contiene la lógica de negocio, realiza validaciones necesarias.
  + Separa las reglas de negocio del acceso a datos.
  + Permite testear la lógica sin depender de la base de datos.
* **Capa de Persistencia (Repository):**
  + Usa Spring Data JPA para interactuar con la base de datos H2.
  + Define interfaces que Spring implementa automáticamente.

Springboot gestiona los componentes como Singletons por defecto. Por ejemplo, cuando definimos un service, repository o controller, se crea una única instancia de esta clase y es utilizada por toda la app.

Se utilizan DTOs (data transfer object) para evitar utilizar las entidades y evitar exponer datos sensibles. En el caso de esta app no contamos con password, pero si las hubiera, mediante el uso de DTOs, se estaría evitando exponer ese dato.

## Lenguajes de Programación y Frameworks

* **Backend: Java con Spring Boot**
  + **Java:** Lenguaje robusto, maduro, de alto rendimiento y con una vasta comunidad y ecosistema. Ideal para aplicaciones empresariales que requieren estabilidad y escalabilidad.
  + **Spring Boot:** Framework para el desarrollo de aplicaciones Java.
    - **Desarrollo Rápido:** Minimiza la configuración, permitiendo un desarrollo ágil de APIs REST.
    - **Ecosistema Extenso:** Integración sencilla con bases de datos (JPA/Hibernate), seguridad (Spring Security), validación, y mucho más.
    - **Mantenibilidad:** El código es limpio, organizado y fácil de mantener gracias a las convenciones de Spring.
* **Frontend: React con Vite**
  + **React:** Biblioteca de JavaScript declarativa y basada en componentes para construir interfaces de usuario.
    - **Componentes:** Facilita la creación de UI complejas y reutilizables, mejorando la modularidad y el mantenimiento.
    - **Rendimiento:** Utiliza un Virtual DOM para optimizar las actualizaciones de la interfaz, resultando en una experiencia de usuario fluida.
  + **Vite:** Herramienta de construcción de frontend de próxima generación.
    - **Velocidad de Desarrollo:** Servidor de desarrollo extremadamente rápido..
    - **Build Óptimo:** Utiliza Rollup para generar builds de producción altamente optimizados y pequeños.
    - **Moderno:** Soporte nativo para ES Modules, TypeScript, JSX.

Base de Datos: H2 (en memoria)

* **H2 Database:** Base de datos relacional ligera y de código abierto escrita en Java.
  + **Simplicidad para Desarrollo y Pruebas:** Su modo "en memoria" es ideal para entornos de desarrollo y pruebas unitarias. La base de datos se inicializa y destruye con cada ejecución de la aplicación, garantizando un estado limpio.
  + **Facilidad de Integración:** Se integra sin esfuerzo con Spring Boot a través de Spring Data JPA (solo se debe configurar el archivo **application.properties**.
  + **Se designa una clase (Java) que se encarga de cargar la base de datos h2, para su prueba.**

Inicialmente configure el archivo **“application.properties”** que se encuentra en la carpeta **“resourses”** del backend para la conexión con MySQL local, pero ante la necesidad de que la aplicación cuente con datos pre cargados decidi optar por el uso de H2.

# Por cada endpoint del backend: Endpoint, verbo HTTP, descripción

## Endpoints del Backend

El backend expone una API RESTfull para gestionar las tres entidades principales: Alumnos, Cursos e Inscripciones.

### Entidad: Alumno

Para la gestión de alumnos:

* **GET /api/alumnos**: Permite obtener una lista completa de todos los alumnos registrados en el sistema.
* **GET /api/alumnos/{legajo}**: Se utiliza para obtener los detalles específicos de un alumno, identificándolo por su ID único.
* **POST /api/alumnos**: Permite crear un nuevo alumno. Los datos del alumno (en formato JSON) deben ser enviados en el cuerpo de la solicitud.
* **PUT /api/alumnos/{legajo}**: Se usa para actualizar la información de un alumno existente, identificado por su ID. Los datos actualizados (en formato JSON) se envían en el cuerpo de la solicitud.
* **DELETE /api/alumnos/{legajo}**: Permite eliminar un alumno del sistema, especificando su ID.

### Entidad: Curso

Para la gestión de cursos:

* **GET /api/cursos**: Proporciona una lista de todos los cursos disponibles en el sistema.
* **GET /api/cursos/{codCurso}**: Permite obtener los detalles de un curso específico, identificado por su ID.
* **POST /api/cursos**: Se utiliza para crear un nuevo curso. Los detalles del curso (en formato JSON) deben ser enviados en el cuerpo de la solicitud.
* **PUT /api/cursos/{codCurso}**: Permite actualizar la información de un curso existente, identificado por su ID. Los datos actualizados (en formato JSON) se envían en el cuerpo de la solicitud.
* **DELETE /api/cursos/{codCurso}**: Se usa para eliminar un curso del sistema, especificando su ID.

### Entidad: Inscripción

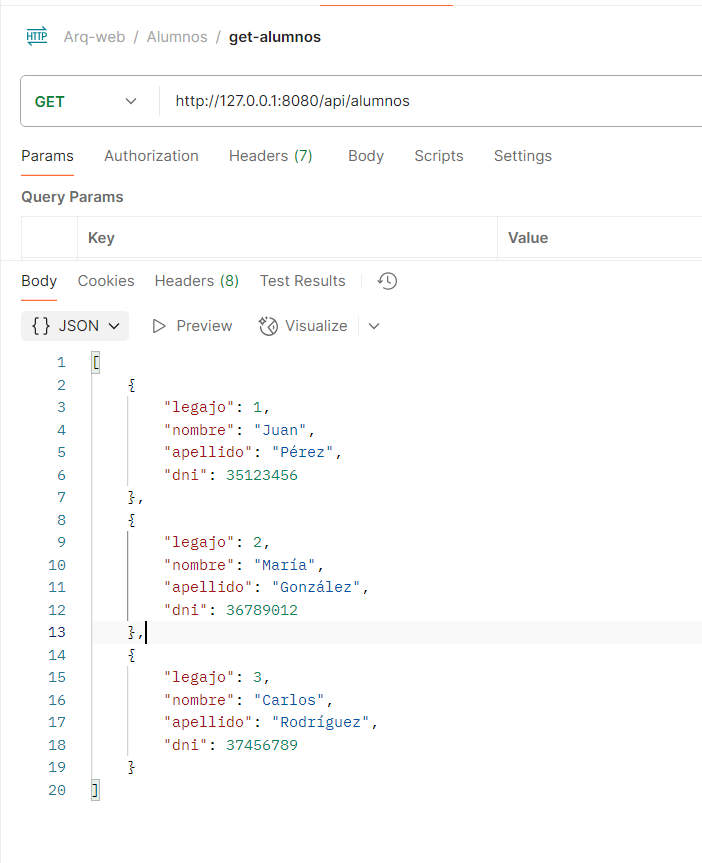
Para la gestión de inscripciones:

* **GET /api/inscripciones**: Permite obtener una lista de todas las inscripciones realizadas en el sistema.
* **GET /api/inscripciones/{codInscripcion}**: Se utiliza para obtener los detalles de una inscripción específica, identificándola por su ID.
* **POST /api/inscripciones**: Permite crear una nueva inscripción. Los IDs del alumno y del curso involucrados (en formato JSON) deben ser enviados en el cuerpo de la solicitud.
* **PUT /api/inscripciones/{codInscripcion}**: Se usa para actualizar los detalles de una inscripción existente, identificada por su ID. Los datos actualizados (en formato JSON) se envían en el cuerpo de la solicitud.
* **DELETE /api/inscripciones/{codInscripcion}**: Permite eliminar una inscripción del sistema, especificando su ID.

### Testeo de endpoints con PostMan

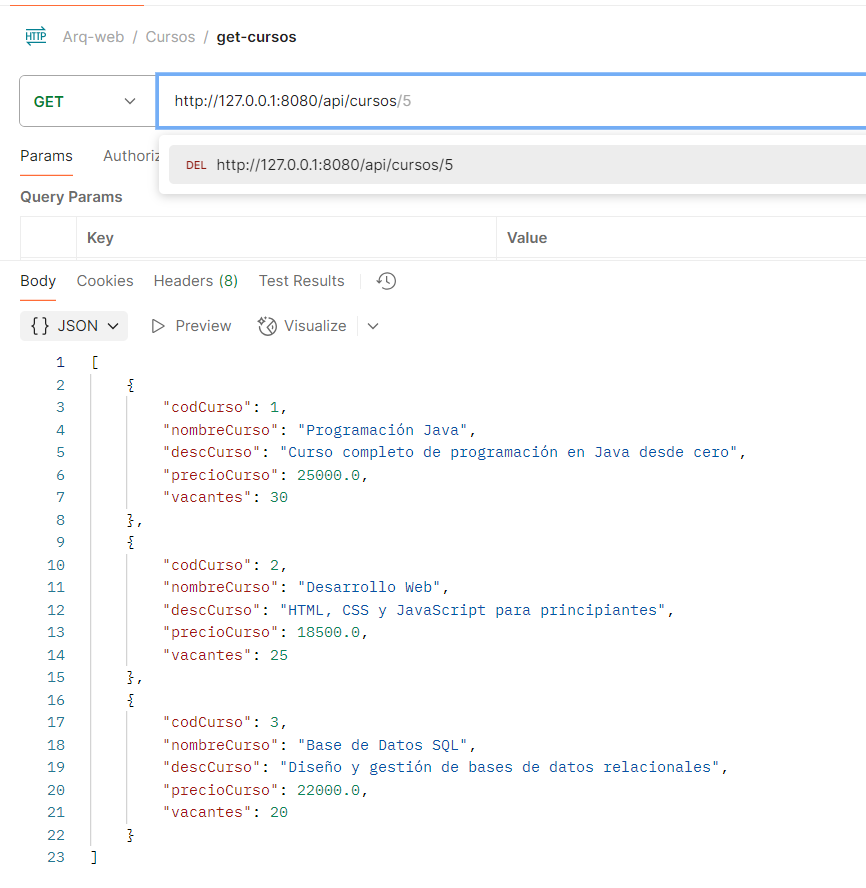
**GET /api/alumnos**

<http://127.0.0.1:8080/api/alumnos>



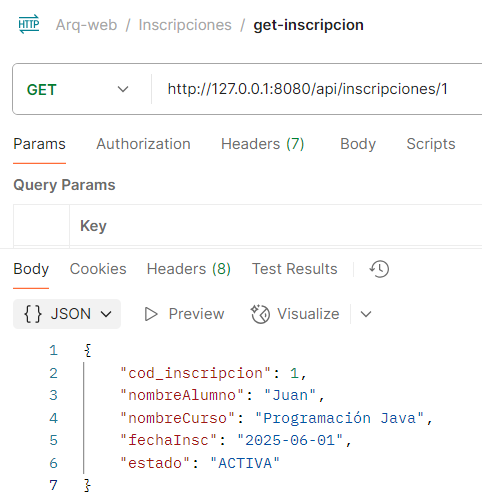
**GET /api/cursos**

<http://127.0.0.1:8080/api/cursos>



**GET /api/inscripciones/{codInscripcion}**

<http://127.0.0.1:8080/api/inscripciones/1>



**DELETE /api/inscripciones/{codInscripcion}**:

<http://127.0.0.1:8080/api/inscripciones/1>

