Universidad Mariano Gálvez
Programación 2
Ing. Jonathan García

Proyecto individual 2: Desarrollo de un Sistema de Ingreso de Notas utilizando Java 17, Spring Framework, React, y MySQL

Descripción:

El objetivo de este proyecto es desarrollar una aplicación de gestión de notas para estudiantes. El sistema permitirá registrar las notas de diferentes evaluaciones (actividades, primer parcial, segundo parcial, examen final) y calcular el puntaje total final del estudiante. El backend será implementado en Java 17 con el Spring Framework, el frontend será desarrollado utilizando React para aplicaciones web, y los datos se almacenarán en una base de datos MySQL.

Requerimientos Funcionales:

1. Ingreso de Notas:

- El sistema debe permitir registrar los puntajes para las siguientes evaluaciones:
- Actividades: Máximo 35 puntos.
- Primer Parcial: Máximo 15 puntos.
- Segundo Parcial: Máximo 15 puntos.
- Examen Final: Máximo 35 puntos.
- El sistema debe validar que los puntajes ingresados no superen los valores máximos establecidos.

2. Cálculo Automático del Puntaje Total:

- El sistema debe calcular automáticamente el puntaje total de cada estudiante sumando los puntajes de todas las evaluaciones.

3. Guardar y Consultar Notas:

- El sistema debe permitir guardar las notas de los estudiantes en una base de datos MySQL.
- El sistema debe permitir consultar las notas registradas para cada estudiante.

4. Interfaz de Usuario:

- El frontend, desarrollado en React, debe permitir la creación, visualización, y modificación de notas de forma intuitiva y sencilla.

Requerimientos Técnicos:

1. Backend:

- Desarrollado en Java 17 con el Spring Framework.
- API RESTful para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las notas.
- Integración con MySQL para persistencia de datos.
- Validaciones de puntajes en el servidor para asegurar la consistencia de los datos.

2. Frontend:

- Desarrollado en React.
- Interfaz de usuario para ingresar, modificar y consultar las notas de los estudiantes.

3. Base de Datos:

- MySQL será utilizado para almacenar las notas de los estudiantes.
- Las tablas deben estar correctamente normalizadas y deben incluir los campos necesarios para almacenar la información del estudiante y sus puntajes.

4. Despliegue:

- El backend debe ser capaz de ejecutarse localmente.

Rubrica de Evaluación (Máximo 15 Puntos):

1. Funcionalidad del Backend (5 puntos):

- [2 puntos] Implementación correcta de las operaciones CRUD en el backend con Spring Framework.

- [1.5 puntos] Manejo adecuado de validaciones de entradas (puntajes no excedan los límites).
- [1.5 puntos] Correcta integración y persistencia de datos en la base de datos MySQL.

2. Funcionalidad del Frontend (4 puntos):

- [2 puntos] Interfaz de usuario funcional y atractiva, desarrollada en React.
- [1 punto] Correcta integración del frontend con el backend. - [1 punto] Manejo eficiente del estado en React.

3. Cálculo de Puntaje Total y Validaciones (3 puntos):

- [1.5 puntos] Cálculo automático y correcto del puntaje total de cada estudiante.
- [1.5 puntos] Implementación correcta de validaciones para evitar errores (ejemplo: no permitir puntajes fuera de los límites establecidos).

4. Diseño de Base de Datos y Almacenamiento (2 puntos):

- [1 punto] Diseño correcto de la base de datos MySQL, con tablas normalizadas y relaciones adecuadas.
- [1 punto] Funcionamiento adecuado de las operaciones de inserción, consulta y actualización de los datos.

5. Documentación y Despliegue (1 punto):

- [0.5 puntos] Documentación clara y detallada sobre cómo desplegar y utilizar la aplicación.
- [0.5 puntos] Funcionamiento correcto del sistema en un entorno local o en un simulador de dispositivos móviles.

Entrega:

Los estudiantes deberán subir su código fuente en un repositorio de GitHub, junto con un archivo de documentación que explique:

- La arquitectura del proyecto.
- Las tecnologías utilizadas.
- Cómo desplegar y probar la aplicación en un entorno local.

DOCUMENTACION

- INTRODUCCION:

Este proyecto tiene como propósito desarrollar un sistema para la gestión de notas estudiantiles, empleando tecnologías como Java 17, Spring Framework, React y MySQL. El sistema nos permite registrar, consultar y modificar las notas de diferentes evaluaciones, incluyendo actividades, parciales y examen final. A su vez, el sistema calcula automáticamente el puntaje total del estudiante.

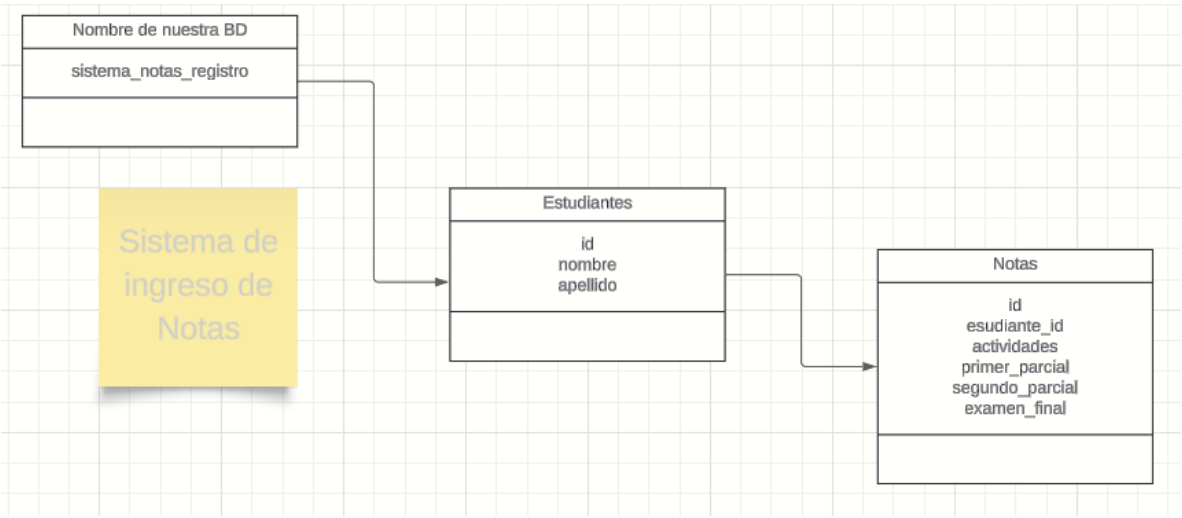
Nuestro backend maneja las operaciones mediante una API RESTful desarrollada en Java con Spring, mientras que el frontend, desarrollado en React, proporciona una interfaz de usuario simple e intuitiva.

Nuestros datos se almacenan en una base de datos MySQL para asegurar su persistencia y organización eficiente.

A través de este proyecto, se busca demostrar la capacidad de implementar una solución completa que integre tanto aspectos de frontend como de backend, cumpliendo con los requerimientos funcionales y técnicos establecidos.

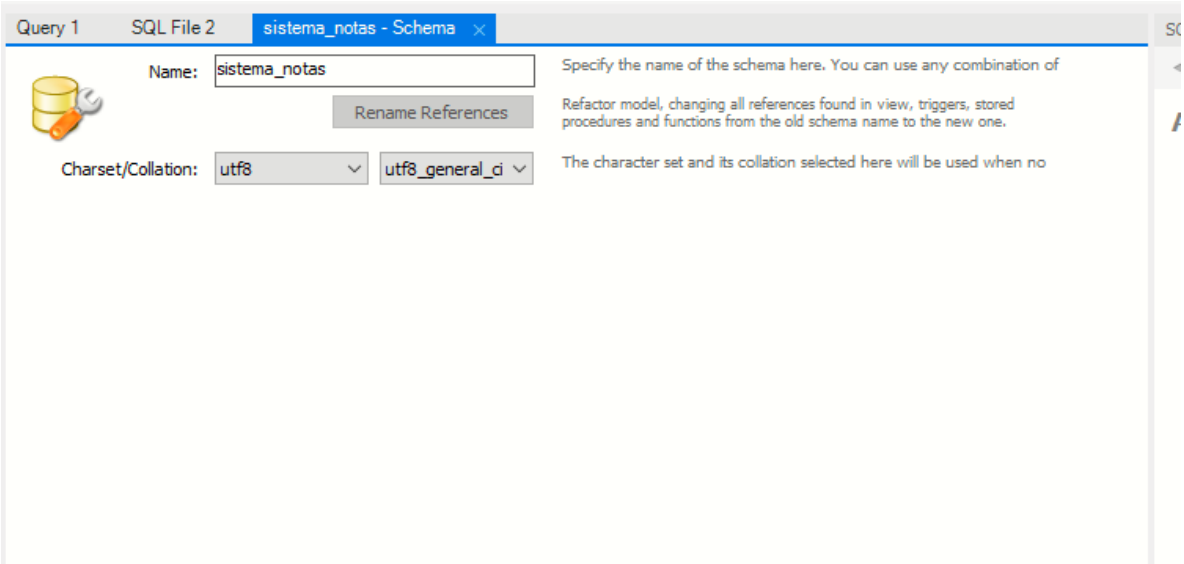
- TECNOLOGIAS UTILIZADAS

- COMO PASO INCIAL TENEMOS EL DISEÑO DE NUESTRA BASE DE DATOS:



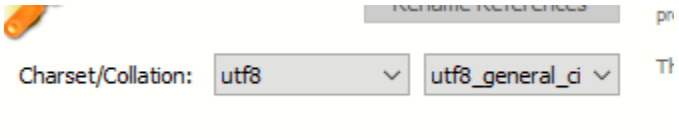
Teniendo esto en cuenta procedemos con la creación de nuestra base de datos:

1. Abrimos nuestro programa MYSQL WORKBENCH. “Colocamos nuestro nombre de la bd”

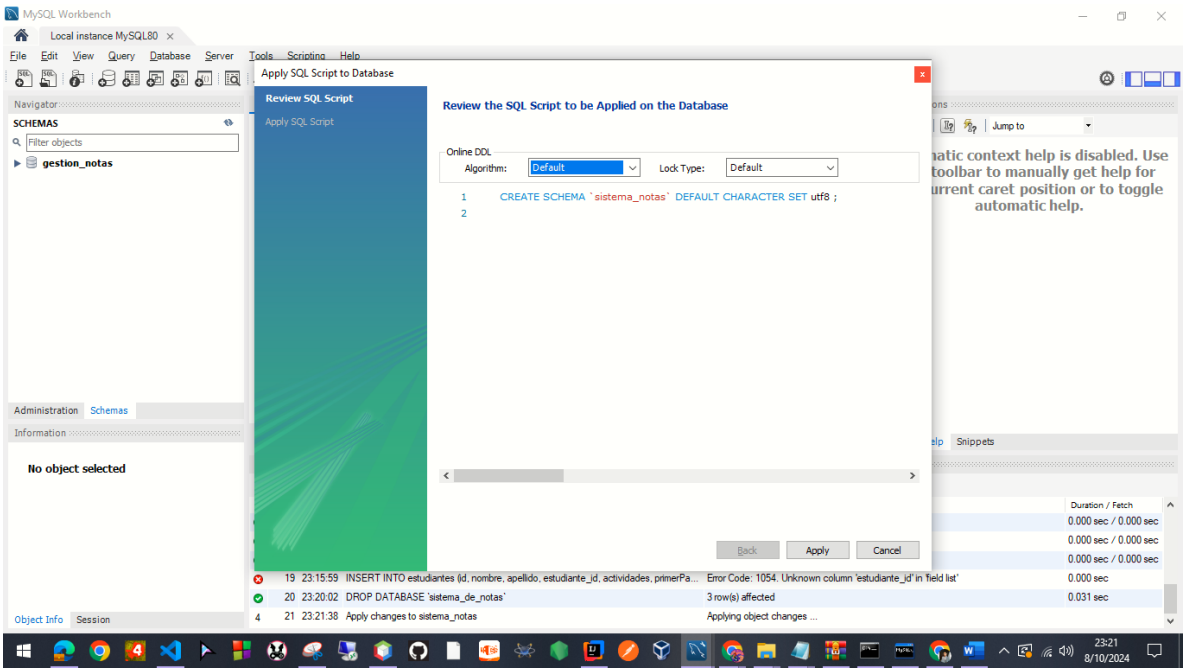


2. Colocamos en las opciones donde indica “Charset/Collation” utf8 – utf_general_ci.

Esto nos permitirá poder ingresar caracteres como la letra ñ.



Damos en “APPLY”.



3. Creamos nuestra base de datos con el siguiente Query:

```
CREATE DATABASE sistema_notas_registro;
```

```
USE sistema_notas_registro;
```

4. Creamos nuestra tabla llamada “estudiantes” con ID, NOMBRE Y APELLIDO.

- ID: Es el nombre de la columna que actúa como identificador, nuestro INT definirá que almacenará datos enteros, INCREMENT indica que se aumentará automáticamente de uno en uno automáticamente y nuestro PRIMARY KEY establece nuestra columna como llave primaria.

- NOMBRE: Es la columna la cual almacenara los nombres de las personas, VARCHAR(255) indica que almacenara datos de cadena con un máximo de 255 caracteres y NOT NULL indica que no permite ingresar valores nulos ósea que siempre debe de contener un valor cuando se inserta una nueva fila.

- APELLIDO: Es el nombre de la columna la cual almacenara los apellidos de las personas, VARCHAR(255) indica que almacenara datos de cadena con un máximo de 255 caracteres y NOT NULL indica que no permite ingresar valores nulos ósea que siempre debe de contener un valor cuando se inserta una nueva fila.

INSERTAMOS EL QUERY:

```
CREATE TABLE Estudiantes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    apellido VARCHAR(255) NOT NULL
);
```

5. Creamos nuestra tabla llamada “notas”

ID: Define la columna la cual se utilizara como identificador único INT es el tipo de dato adecuado para nuestros valores, AUTO_INCREMENT especifica el valor el cual aumentara automáticamente PRIMERY KEY define nuestra columna la cual será llave primaria.

- ESTUDIANTE_ID: Esta columna almacenara nuestro identificador del estudiante relacionado con nuestras notas.

- ACTIVIDADES: Es el nombre de la columna la cual almacenara las notas DECIMAL(5,2) especifica que el tipo de dato es decimal con un máximo de 5 dígitos, de los cuales 2 pudieran ser decimales.

- PRIMER_PARCIAL: Almacenara las notas de nuestro primer parcial con el mismo formato.

- SEGUNDO_PARCIAL: Almacenara las notas del segundo parcial con el mismo formato.

- EXAMEN_FINAL: Almacenara las notas del examen final con el mismo formato.

```
CREATE TABLE Notas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    estudiante_id INT,
    actividades DECIMAL(5,2),
    primer_parcial DECIMAL(5,2),
    segundo_parcial DECIMAL(5,2),
    examen_final DECIMAL(5,2),
    FOREIGN KEY (estudiante_id) REFERENCES Estudiantes(id) ON DELETE CASCADE
);
```

6. Insertamos nuestros alumnos, en esta casilla colocamos el nombre al igual que el apellido.
7.

```
INSERT INTO Estudiantes (nombre, apellido) VALUES
('Juan', 'Pérez'),
('María', 'Gómez'),
('Carlos', 'López'),
('Ana', 'Martínez'),
('Luis', 'Hernández'),
('Sofía', 'Ramírez'),
('Diego', 'Torres');
```

8. Ingresamos las notas de nuestros estudiantes:

- INSERT INTO Notas (estudiante_id, actividades, primer_parcial, segundo_parcial, examen_final)

- (el ID del estudiante, la nota de las actividades no mayor a 35 puntos, la nota del primer parcial no mayo a 15 puntos, la nota de nuestro segundo parcial no mayor a 15 puntos, la nota del examen no mayo a 35 puntos)

```
INSERT INTO Notas (estudiante_id, actividades, primer_parcial, segundo_parcial, examen_final) VALUES
(1, 30.00, 12.00, 14.00, 28.00), -- Juan Pérez
(2, 32.50, 10.00, 13.50, 30.00), -- María Gómez
(3, 25.00, 14.00, 12.00, 20.00), -- Carlos López
(4, 35.00, 15.00, 14.50, 33.00), -- Ana Martínez
(5, 28.00, 11.00, 10.50, 27.00), -- Luis Hernández
(6, 30.50, 13.00, 12.50, 29.00), -- Sofía Ramírez
(7, 34.00, 15.00, 14.00, 34.00); -- Diego Torres
```

9. Validamos nuestras tablas con los registros ingresados:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

nombre	actividades	primer_parcial	segundo_parcial	examen_final	puntaje_total
Juan	30	12	14	28	84
María	32.5	10	13.5	30	86
Carlos	25	14	12	20	71
Ana	35	15	14.5	33	97.5
Luis	28	11	10.5	27	76.5
Sofía	30.5	13	12.5	29	85

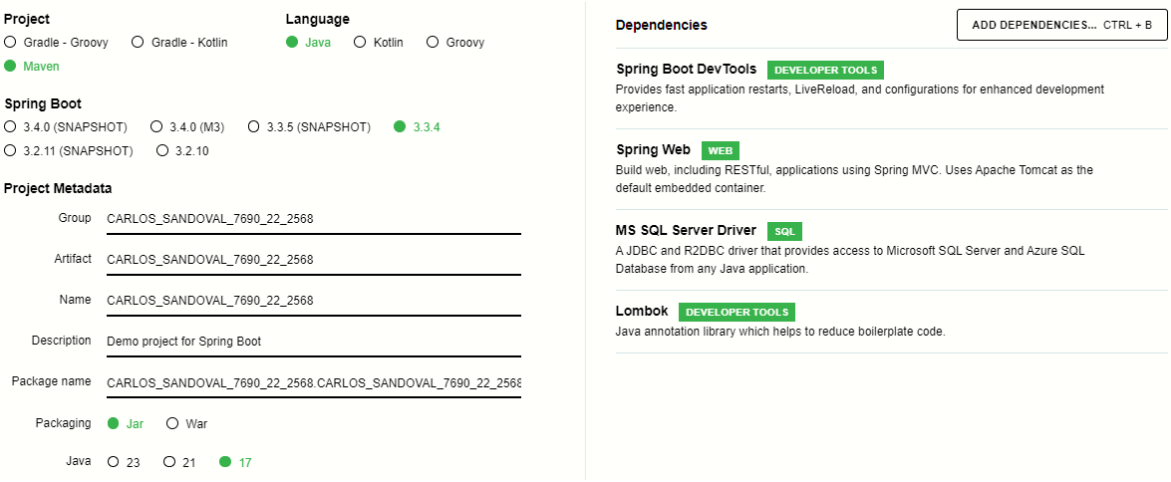
- Este es nuestro Query utilizado:

```
SELECT
    e.nombre,
    n.actividades,
    n.primer_parcial,
    n.segundo_parcial,
    n.examen_final,
    (n.actividades + n.primer_parcial + n.segundo_parcial + n.examen_final) AS puntaje_total
FROM
    Notas n
JOIN
    Estudiantes e
ON
    n.estudiante_id = e.id;
```

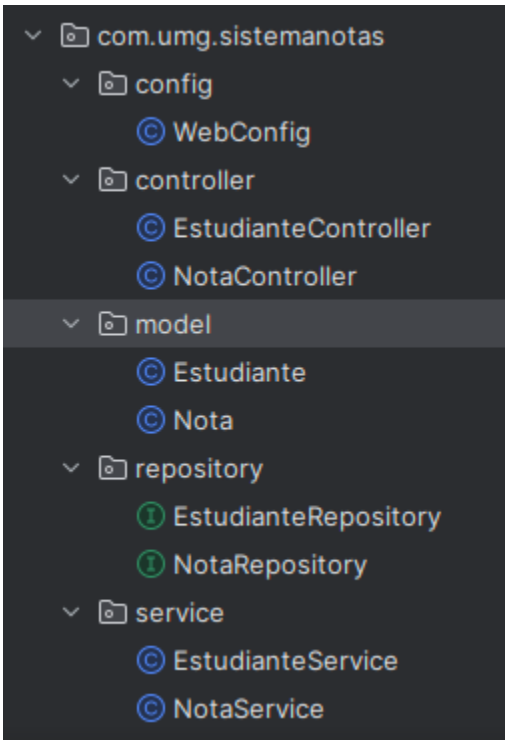
Link de nuestro diagrama en UML

https://lucid.app/lucidchart/5d5a8eeb-6aa4-4c50-b0d7-2798bf756384/edit?viewport_loc=-897%2C-180%2C1997%2C905%2C0_0&invitationId=inv_ede6ae05-b486-486d-989a-b80e3ffd58e4

10. Creamos nuestro paquete en: https://start.spring.io/



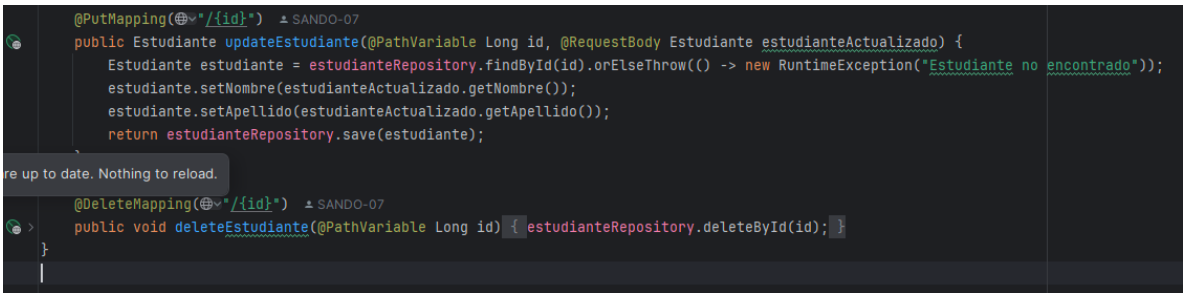
11. Creamos nuestras carpetas y clases las cual vamos a utilizar para nuestro BACKEND.



- En el apartado de WebConfig agregamos la configuración de CORS ya que esto permite que el frontend (React) en localhost:3000 interactúe con el backend en otro servidor (Spring Boot) mediante solicitudes HTTP, habilitando ciertos métodos y encabezados.

- En la clase EstudianteController **agregamos la** aplicación Spring Boot la cual maneja operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los estudiantes para realizar las operaciones en la base de datos.

```java
package com.umg.sistemanotas.controller;

import ...

@RestController  SANDO-07
@RequestMapping("/api/estudiantes")
public class EstudianteController {
    @Autowired
    private EstudianteRepository estudianteRepository;

    @GetMapping  SANDO-07
    public List<Estudiante> getAllEstudiantes() { return estudianteRepository.findAll(); }

    @GetMapping("/{id}")  SANDO-07
    public Estudiante getEstudianteById(@PathVariable Long id) {
        return estudianteRepository.findById(id).orElseThrow(() -> new RuntimeException("Estudiante no encontrado"));
    }

o to date. Nothing to reload.
    SANDO-07
    public Estudiante createEstudiante(@RequestBody Estudiante estudiante) {
        return estudianteRepository.save(estudiante);
    }
```



```java
    @PutMapping("/{id}")  SANDO-07
    public Estudiante updateEstudiante(@PathVariable Long id, @RequestBody Estudiante estudianteActualizado) {
        Estudiante estudiante = estudianteRepository.findById(id).orElseThrow(() -> new RuntimeException("Estudiante no encontrado"));
        estudiante.setNombre(estudianteActualizado.getNombre());
        estudiante.setApellido(estudianteActualizado.getApellido());
        return estudianteRepository.save(estudiante);
    }

re up to date. Nothing to reload.
    @DeleteMapping("/{id}")  SANDO-07
    public void deleteEstudiante(@PathVariable Long id) { estudianteRepository.deleteById(id); }
}
```

- En la clase NotaController, como su nombre lo indica es un **controlador** en una aplicación Spring Boot que maneja las operaciones CRUD (Crear, Leer, Actualizar) para las notas de los estudiantes. Se conecta con dos repositorios: NotaRepository y EstudianteRepository.

```java
package com.umg.sistemanotas.controller;
import ...
@RestController  SANDO-07 *
@RequestMapping("/api/notas")
public class NotaController {
    @Autowired
    private NotaRepository notaRepository;
    @Autowired
    private EstudianteRepository estudianteRepository;
    @GetMapping  SANDO-07
    public List<Nota> getAllNotas() { return notaRepository.findAll(); }
    @GetMapping("/{id}")  SANDO-07
    public Nota getNotaById(@PathVariable Long id) {
        return notaRepository.findById(id).orElseThrow(() -> new RuntimeException("Nota no encontrada"));
    }
    @PostMapping  SANDO-07
    public Nota createNota(@RequestBody Nota nota) {
        Estudiante estudiante = estudianteRepository.findById(nota.getEstudiante().getId())
                .orElseThrow(() -> new RuntimeException("Estudiante no encontrado"));
        nota.setEstudiante(estudiante);
        nota.calcularPuntajeTotal();
        return notaRepository.save(nota);
    }
```

```java
    @PutMapping("/{id}")  SANDO-07
    public Nota updateNota(@PathVariable Long id, @RequestBody Nota notaActualizada) {
        Nota nota = notaRepository.findById(id).orElseThrow(() -> new RuntimeException("Nota no encontrada"));
        Estudiante estudiante = estudianteRepository.findById(notaActualizada.getEstudiante().getId())
                .orElseThrow(() -> new RuntimeException("Estudiante no encontrado"));
        nota.setEstudiante(estudiante);
        nota.setActividades(notaActualizada.getActividades());
        nota.setPrimerParcial(notaActualizada.getPrimerParcial());
        nota.setSegundoParcial(notaActualizada.getSegundoParcial());
        nota.setExamenFinal(notaActualizada.getExamenFinal());
        nota.calcularPuntajeTotal();
        return notaRepository.save(nota);
```

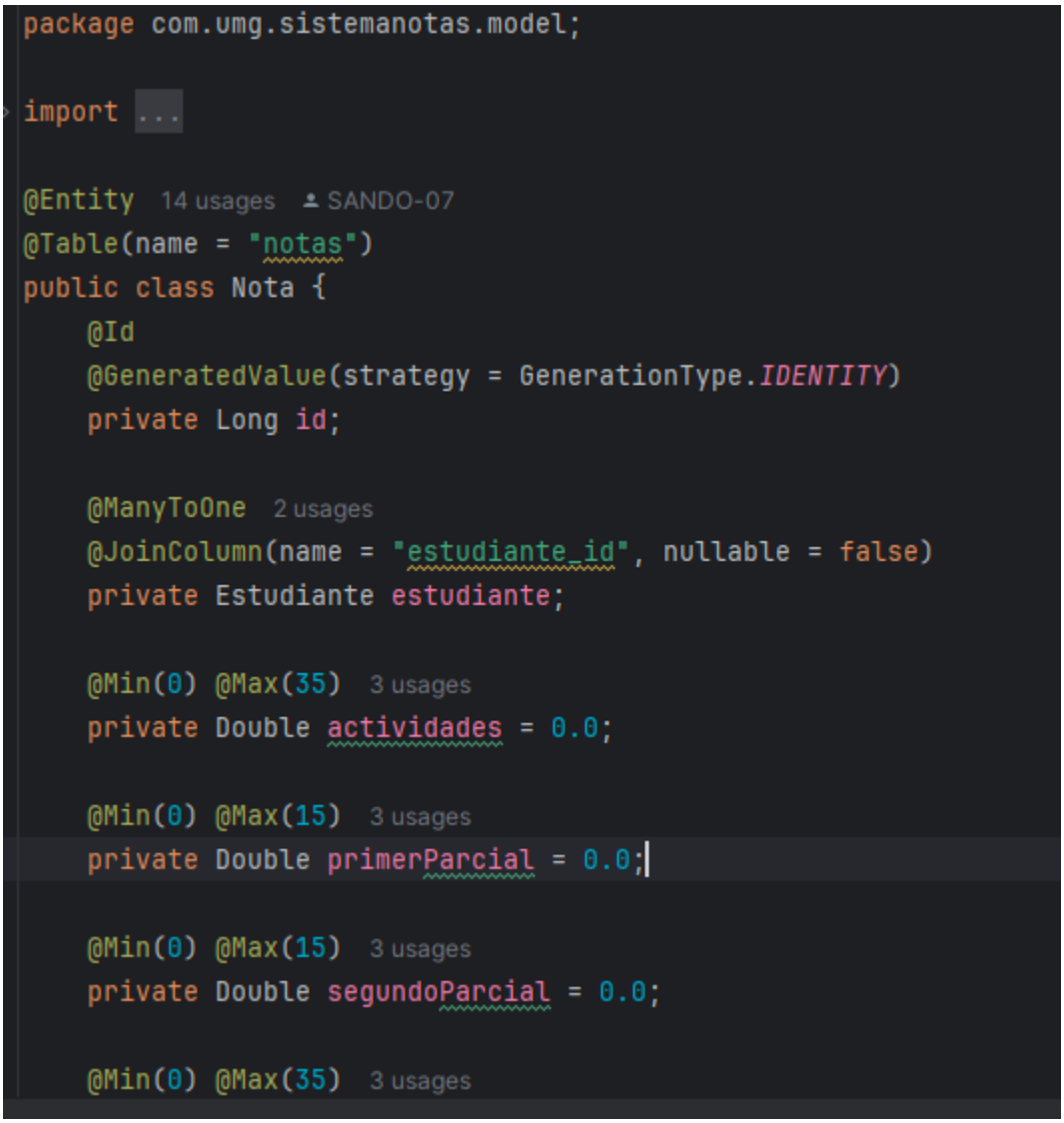
ses are up to date. Nothing to reload.

- La clase Estudiante representa el modelo de datos para los estudiantes en una aplicación Spring Boot. Utiliza anotaciones de JPA (Jakarta Persistence API) para mapear la clase a una tabla en la base de datos.

```java
package com.umg.sistemanotas.model;

import jakarta.persistence.*;

@Entity  21 usages  SANDO-07
@Table(name = "estudiantes")
public class Estudiante {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)  2 usages
    private String nombre;

    @Column(nullable = false)  2 usages
    private String apellido;

    public Long getId() {  SANDO-07
        return id;
    }

    public void setId(Long id) {  SANDO-07
        this.id = id;
    }
```

```java
    public String getNombre() {  1 usage   SANDO-07
        return nombre;
    }

    public void setNombre(String nombre) {  1 usage   SANDO-07
        this.nombre = nombre;
    }

    public String getApellido() {  1 usage   SANDO-07
        return apellido;
    }

    public void setApellido(String apellido) {  1 usage   SANDO-07
        this.apellido = apellido;
    }
}
```

- En la clase Nota representa el modelo de datos para las notas de los estudiantes de nuestra aplicación Spring Boot.



```java
package com.umg.sistemanotas.model;

import ...

@Entity  14 usages  SANDO-07
@Table(name = "notas")
public class Nota {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne  2 usages
    @JoinColumn(name = "estudiante_id", nullable = false)
    private Estudiante estudiante;

    @Min(0) @Max(35)  3 usages
    private Double actividades = 0.0;

    @Min(0) @Max(15)  3 usages
    private Double primerParcial = 0.0;

    @Min(0) @Max(15)  3 usages
    private Double segundoParcial = 0.0;

    @Min(0) @Max(35)  3 usages
```

```java
@Min(0) @Max(35)  3 usages
private Double examenFinal = 0.0;

private Double puntajeTotal = 0.0;  3 usages

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public Estudiante getEstudiante() { return estudiante; }

public void setEstudiante(Estudiante estudiante) { this.estudiante = estudiante; }

public Double getActividades() { return actividades; }

public void setActividades(Double actividades) { this.actividades = actividades; }

public Double getPrimerParcial() { return primerParcial; }

public void setPrimerParcial(Double primerParcial) { this.primerParcial = primerParcial; }

public Double getSegundoParcial() { return segundoParcial; }
```

```java
    public Double getSegundoParcial() { return segundoParcial; }

    public void setSegundoParcial(Double segundoParcial) { this.segundoParcial = segundoParcial; }

    public Double getExamenFinal() { return examenFinal; }

    public void setExamenFinal(Double examenFinal) { this.examenFinal = examenFinal; }

    public Double getPuntajeTotal() { return puntajeTotal; }

    public void setPuntajeTotal(Double puntajeTotal) { this.puntajeTotal = puntajeTotal; }

    public void calcularPuntajeTotal() {  3 usages  SANDO-07
        this.puntajeTotal = this.actividades + this.primerParcial + this.segundoParcial + this.examenFinal;
    }
}
```

- En la clase EstudianteRepository al igual que la NotaRepository son las que actúan como nuestros repositorios para las entidades **Estudiante** en nuestra aplicación Spring Boot.

```java
package com.umg.sistemanotas.repository;

import ...

@Repository 6 usages SANDO-07
public interface EstudianteRepository extends JpaRepository<Estudiante, Long> {
}
```

```java
package com.umg.sistemanotas.repository;

import ...

@Repository 4 usages SANDO-07
public interface NotaRepository extends JpaRepository<Nota, Long> {
}
```

- En la clase **EstudianteService**, es la que actúa como un servicio de nuestra aplicación Spring Boot para gestionar las operaciones relacionadas con la entidad **Estudiante**

```java
package com.umg.sistemanotas.service;

import ...

@Service no usages SANDO-07
public class EstudianteService {

    @Autowired
    private EstudianteRepository estudianteRepository;

    public List<Estudiante> getAllEstudiantes() { return estudianteRepository.findAll(); }

    public Optional<Estudiante> getEstudianteById(Long id) { return estudianteRepository.findById(id); }

    public Estudiante saveEstudiante(Estudiante estudiante) { return estudianteRepository.save(estudiante); }

    public void deleteEstudiante(Long id) { estudianteRepository.deleteById(id); }
}
```

- la clase **NotaService**, es la que actúa como nuestro servicio de nuestra aplicación Spring Boot la cual nos sirve para gestionar las operaciones relacionadas con la entidad **Nota.**

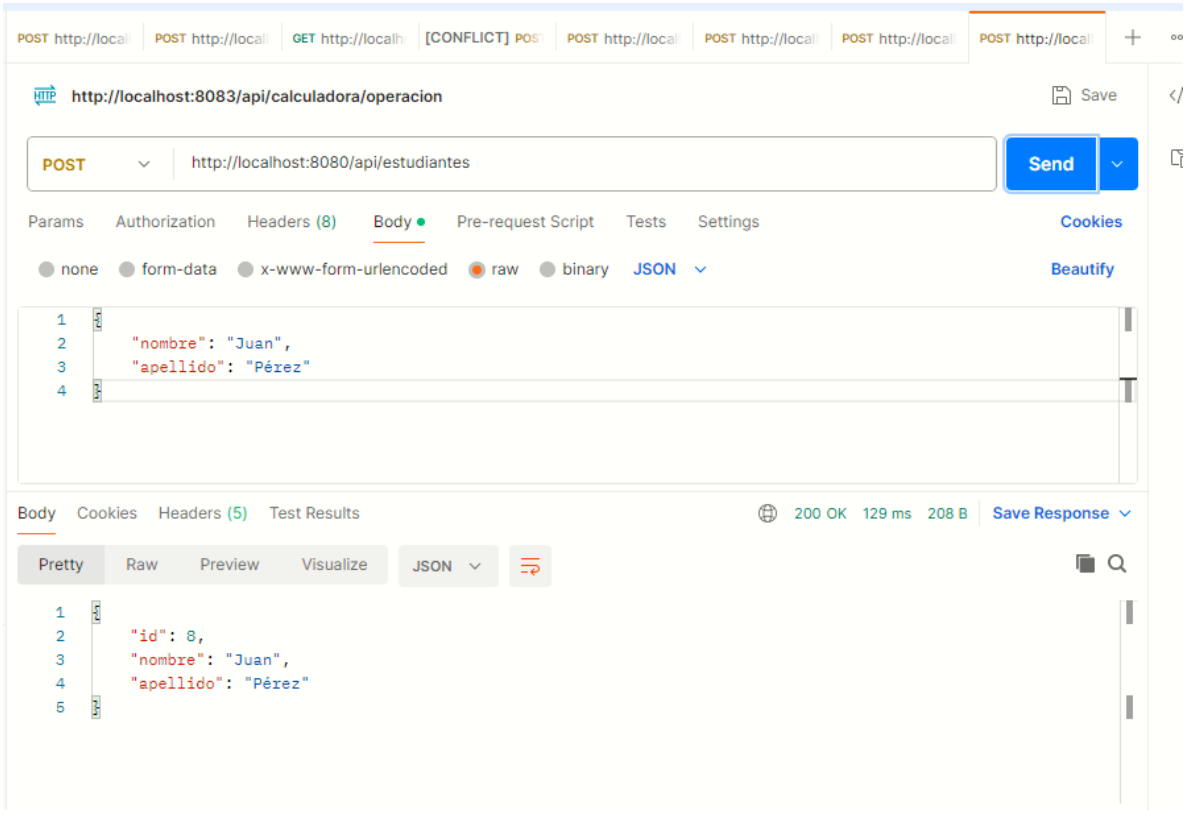
```java
package com.umg.sistemanotas.service;

import ...

@Service  no usages  SANDO-07 *
public class NotaService {
    @Autowired
    private NotaRepository notaRepository;

    public List<Nota> getAllNotas() { return notaRepository.findAll(); }

    public Nota saveNota(Nota nota) {  no usages  SANDO-07
        nota.calcularPuntajeTotal();
        return notaRepository.save(nota);
    }


}
```

```java
package com.umg.sistemanotas;

import ...

@SpringBootApplication  SANDO-07
public class SistemaNotasApplication {
    public static void main(String[] args) { SpringApplication.run(SistemaNotasApplication.class, args); }
}
```

- En la parte de aplication.properties agregamos el puerto el cual utiliza nuestra base de datos al igual que colocamos el nombre de nuestra base de datos, agregamos el nombre y la contraseña la cual utilizamos cuando instalamos MYSQL en nuestro equipo.

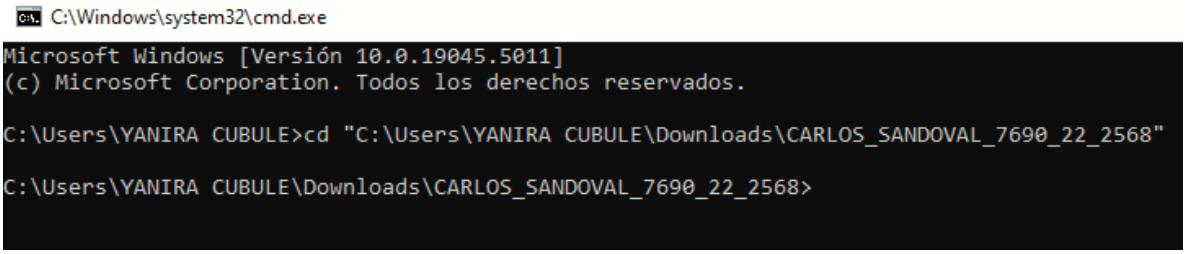
```
spring.datasource.url=jdbc:mysql://localhost:3306/sistema_notas_registro
spring.datasource.username=root
spring.datasource.password=carlos
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

12. Realizamos pruebas del BACKEND en el programa POSTMAN para validar si las consultas que se realizan funcionan correctamente:

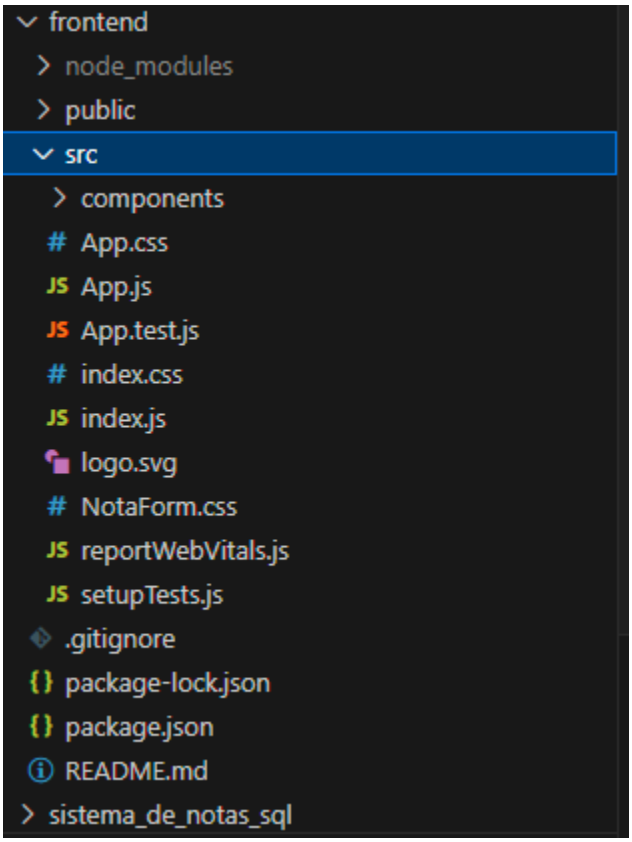


- Realizando pruebas y viendo que funciona correctamente pasamos a crear nuestro frontend.

13. Abrimos la consola CMD y agregamos el comando para poder crear “FRONTED”, instalar axios y otras dependencias de nuestro programa:



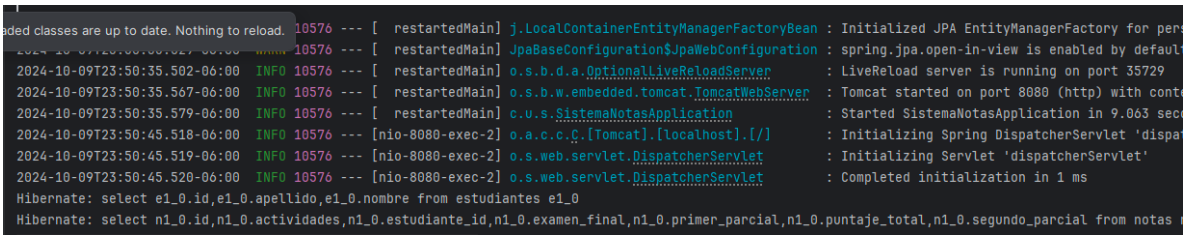
14. Realizando esto crearemos nuestras carpetas del FRONTED.



Realizando la creación de cada archivo CSS, JS, JSon creamos básicamente nuestro FRONTED.

15. Ejecutamos nuestro BACKEND y nuestro FRONTED.

BACKEND:

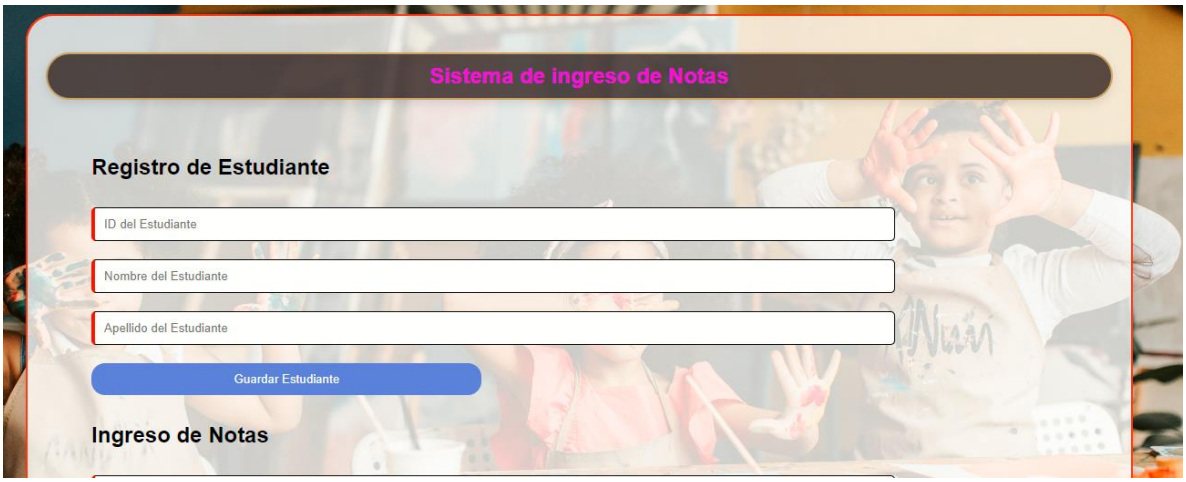


FRONTED:

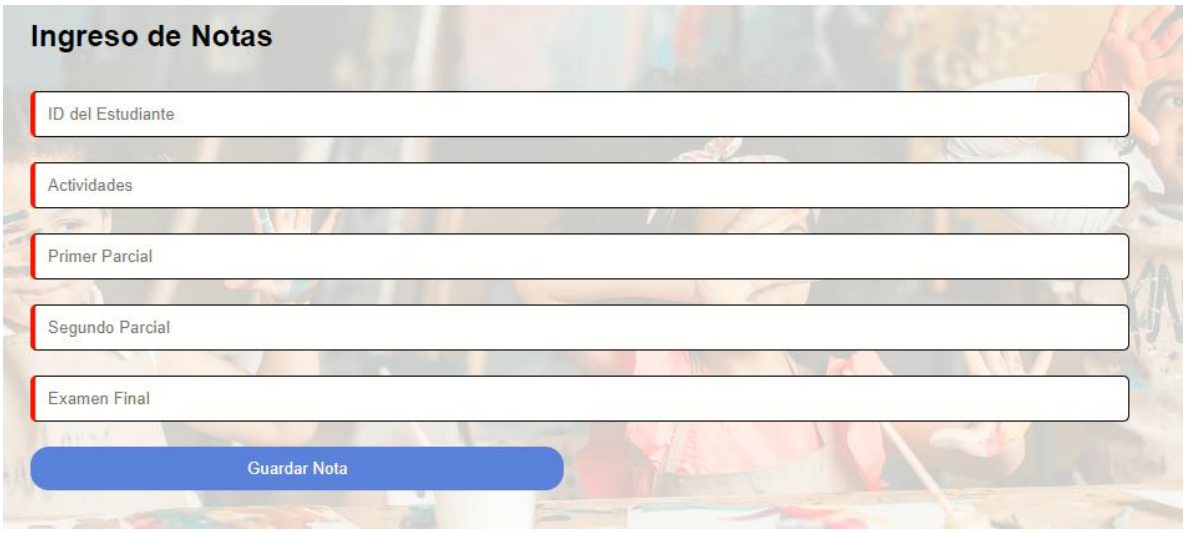
```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Validamos que nuestra aplicación se encuentre corriendo correctamente:

En este apartado tenemos la opción para poder registrar nuestros estudiantes.



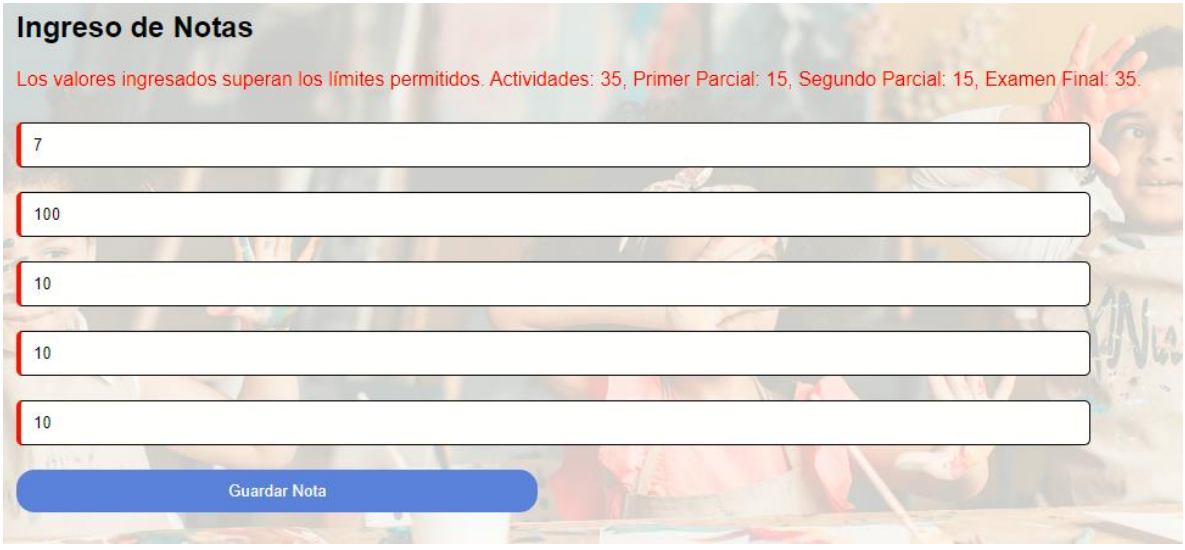
Tenemos el ingreso de notas para ingresar manualmente nuestras notas:



En esta opción podemos ver la lista de nuestros estudiantes:

Aquí podemos visualizar la lista de nuestros estudiantes, editar o eliminar el registro:

Alertas si agregamos valores no validos:



Conclusión:

El sistema de ingreso de notas nos ofrece una solución eficiente para la gestión académica, integrando un frontend intuitivo y un backend.

A través de la automatización de cálculos y la validación de entradas, se garantiza la precisión en el registro de puntajes. Además, su diseño escalable y modular lo convierte en una herramienta adaptable a futuros requerimientos educativos.