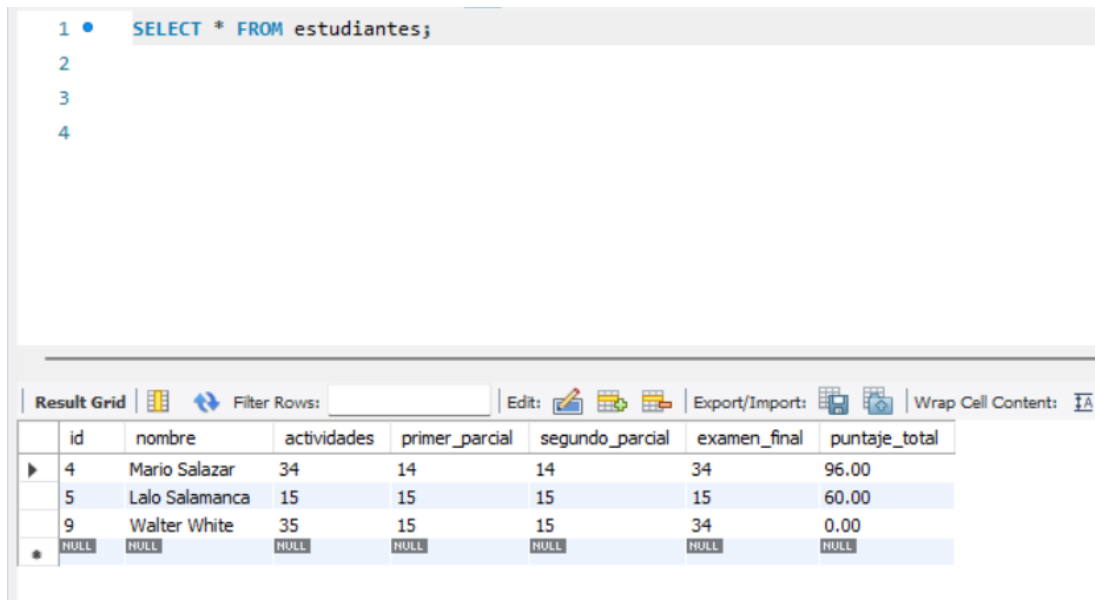


GIT HUT: <https://github.com/Egos808/Proyecto-2.git>

## IMÁGENES:



The screenshot shows a SQL query editor with the following query:

```
1 • SELECT * FROM estudiantes;
```

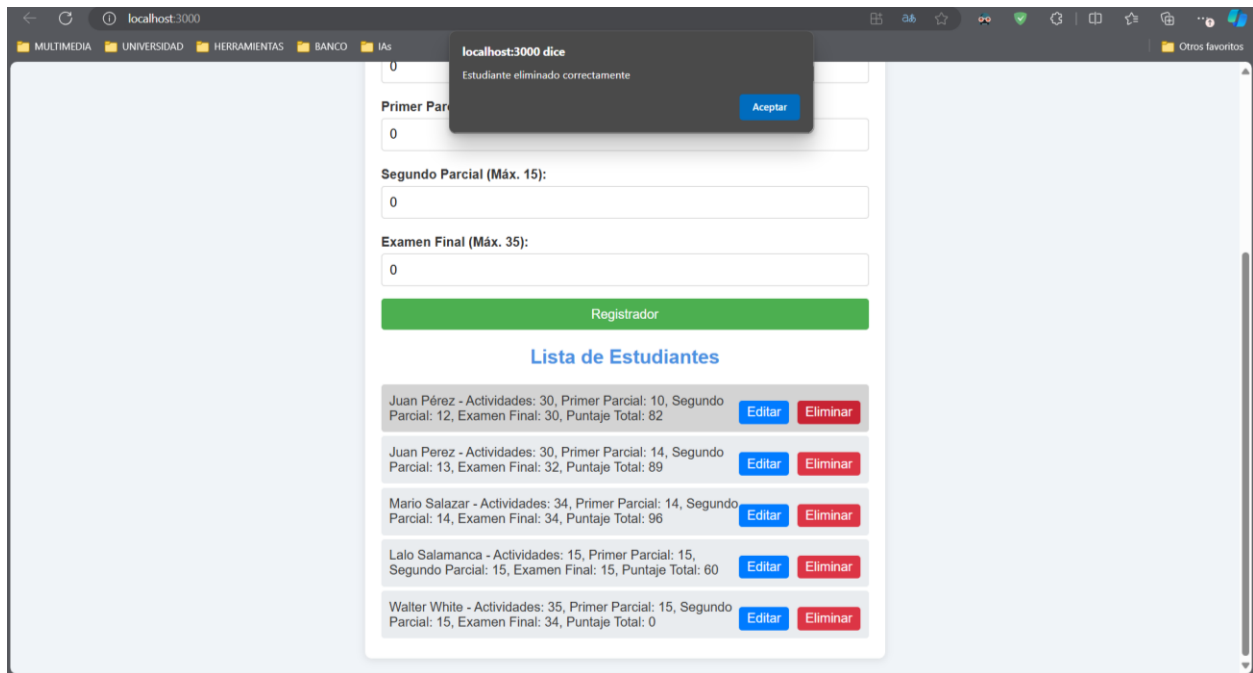
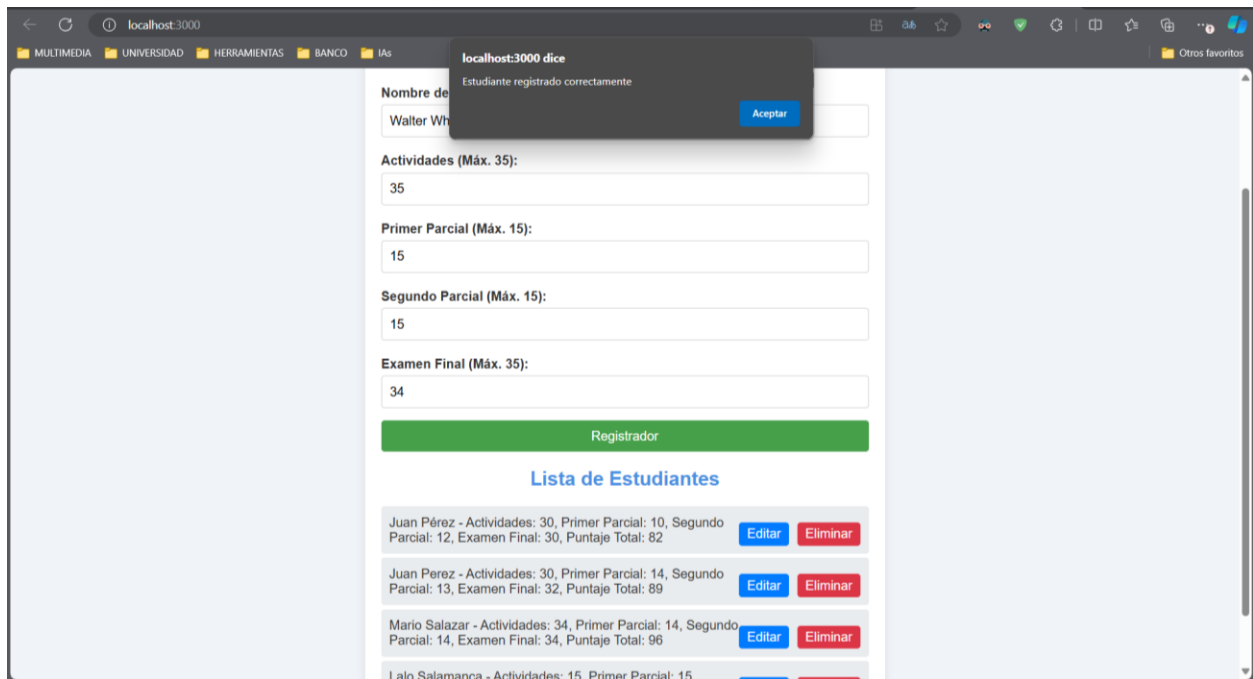
The results are displayed in a table with the following columns: id, nombre, actividades, primer\_parcial, segundo\_parcial, examen\_final, and puntaje\_total. The table contains three rows of data:

	id	nombre	actividades	primer_parcial	segundo_parcial	examen_final	puntaje_total
▶	4	Mario Salazar	34	14	14	34	96.00
	5	Lalo Salamanca	15	15	15	15	60.00
	9	Walter White	35	15	15	34	0.00
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL



The screenshot shows a web browser with the address bar displaying `localhost:8080/api/estudiantes`. The browser's address bar also shows a list of folders: MULTIMEDIA, UNIVERSIDAD, HERRAMIENTAS, and BANCO. The main content area displays a JSON array of student records:

```
1 [
2   {
3     "id": 4,
4     "nombre": "Mario Salazar",
5     "actividades": 34,
6     "primerParcial": 14,
7     "segundoParcial": 14,
8     "examenFinal": 34,
9     "puntajeTotal": 96
10  },
11  {
12    "id": 5,
13    "nombre": "Lalo Salamanca",
14    "actividades": 15,
15    "primerParcial": 15,
16    "segundoParcial": 15,
17    "examenFinal": 15,
18    "puntajeTotal": 60
19  },
20  {
21    "id": 9,
22    "nombre": "Walter White",
23    "actividades": 35,
24    "primerParcial": 15,
25    "segundoParcial": 15,
26    "examenFinal": 34,
27    "puntajeTotal": 0
28  }
29 ]
```



←localhost:3000

MULTIMEDIAUNIVERSIDADHERRAMIENTASBANCOIASOtros favoritos

### Registrador Notas

Nombre del Estudiante:

Bruce Wayne

Actividades (Máx. 35):

40

Primer Parcial (Máx. 35):

0

Segundo Parcial (Máx. 15):

0

Examen Final (Máx. 35):

0

Registrador

### Lista de Estudiantes

Mario Salazar - Actividades: 34, Primer Parcial: 14, Segundo Parcial: 14, Examen Final: 34, Puntaje Total: 96	Editar	Eliminar
Lalo Salamanca - Actividades: 15, Primer Parcial: 15, Segundo Parcial: 15, Examen Final: 15, Puntaje Total: 60	Editar	Eliminar
Walter White - Actividades: 35, Primer Parcial: 15, Segundo Parcial: 15, Examen Final: 34, Puntaje Total: 0	Editar	Eliminar

## MODEL > ESTUDIANTE:

```
package com.example.notas.model; // Especifica el paquete en el que se encuentra la clase.

import jakarta.persistence.*; // Importa las anotaciones de Jakarta Persistence para la gestión de entidades.

@Entity // Indica que esta clase es una entidad de JPA (Java Persistence API).
@Table(name = "estudiantes") // Especifica el nombre de la tabla en la base de datos que se mapea a esta entidad.
public class Estudiante {

    @Id // Indica que este campo es la clave primaria de la entidad.
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Indica que el valor de la clave se generará automáticamente.
    private Long id; // Campo que almacena el ID único del estudiante.

    private String nombre; // Campo que almacena el nombre del estudiante.
    private double actividades; // Campo que almacena la puntuación de actividades del estudiante.
    private double primerParcial; // Campo que almacena la puntuación del primer parcial.
    private double segundoParcial; // Campo que almacena la puntuación del segundo parcial.
    private double examenFinal; // Campo que almacena la puntuación del examen final.

    @Column(columnDefinition = "DECIMAL(5,2)") // Especifica que el campo se almacenará como un número decimal con 5 dígitos en total y 2 decimales.
    private double puntajeTotal; // Campo que almacena el puntaje total del estudiante.

    // Getters y Setters para acceder y modificar los campos privados.

    public Long getId() {
        return id; // Retorna el ID del estudiante.
    }

    public void setId(Long id) {
        this.id = id; // Establece el ID del estudiante.
    }

    public String getNombre() {
        return nombre; // Retorna el nombre del estudiante.
    }

    public void setNombre(String nombre) {
        this.nombre = nombre; // Establece el nombre del estudiante.
    }

    public double getActividades() {
        return actividades; // Retorna la puntuación de actividades.
    }

    public void setActividades(double actividades) {
        this.actividades = actividades; // Establece la puntuación de actividades.
    }
}
```

```
}

public double getPrimerParcial() {
    return primerParcial; // Retorna la puntuación del primer parcial.
}

public void setPrimerParcial(double primerParcial) {
    this.primerParcial = primerParcial; // Establece la puntuación del primer parcial.
}

public double getSegundoParcial() {
    return segundoParcial; // Retorna la puntuación del segundo parcial.
}

public void setSegundoParcial(double segundoParcial) {
    this.segundoParcial = segundoParcial; // Establece la puntuación del segundo parcial.
}

public double getExamenFinal() {
    return examenFinal; // Retorna la puntuación del examen final.
}

public void setExamenFinal(double examenFinal) {
    this.examenFinal = examenFinal; // Establece la puntuación del examen final.
}

public double getPuntajeTotal() {
    return puntajeTotal; // Retorna el puntaje total del estudiante.
}

public void setPuntajeTotal(double puntajeTotal) {
    this.puntajeTotal = puntajeTotal; // Establece el puntaje total del estudiante.
}
}
```

## SERVICE > ESTUDIANTESERVICE

```
package com.example.notas.service;

import com.example.notas.model.Estudiante;
import com.example.notas.repository.EstudianteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service // Indica que esta clase es un servicio de Spring.
public class EstudianteService {

    @Autowired // Inyección de dependencias para utilizar el repositorio de Estudiante.
    private EstudianteRepository estudianteRepository;

    // Obtener todos los estudiantes
    public List<Estudiante> obtenerTodos() {
        return estudianteRepository.findAll(); // Devuelve una lista de todos los estudiantes en la base de datos.
    }

    // Obtener estudiante por ID
    public Estudiante obtenerPorId(Long id) {
        Optional<Estudiante> estudiante = estudianteRepository.findById(id); // Busca un estudiante por su ID.
        return estudiante.orElse(null); // Si no se encuentra el estudiante, devuelve null.
    }

    // Guardar (crear o actualizar) estudiante
    public Estudiante guardar(Estudiante estudiante) {
        // Calcular el puntaje total antes de guardar el estudiante.
        double puntajeTotal = calcularPuntajeTotal(estudiante);
        estudiante.setPuntajeTotal(puntajeTotal); // Establece el puntaje total en el objeto estudiante.
        return estudianteRepository.save(estudiante); // Guarda o actualiza el estudiante en la base de datos.
    }

    // Eliminar estudiante
    public void eliminar(Long id) {
        estudianteRepository.deleteById(id); // Elimina un estudiante por su ID.
    }

    // Método para calcular el puntaje total
    private double calcularPuntajeTotal(Estudiante estudiante) {
        // Suma las puntuaciones de actividades, primer parcial, segundo parcial y examen final.
        return estudiante.getActividades() + estudiante.getPrimerParcial() +
            estudiante.getSegundoParcial() + estudiante.getExamenFinal();
    }
}
```

## REPOSITORY > ESTUDIANTEREPOSITORY

```
package com.example.notas.repository;

import com.example.notas.model.Estudiante;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository // Indica que esta interfaz es un componente de Spring y maneja la persistencia de datos.
public interface EstudianteRepository extends JpaRepository<Estudiante, Long> {
    // JpaRepository proporciona métodos CRUD básicos como save, findAll, findById, deleteById, etc.
}
```

## CONTROLLER > ESTUDIANTECONTROLLER

```
package com.example.notas.controller;

import com.example.notas.model.Estudiante;
import com.example.notas.service.EstudianteService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/estudiantes")
@CrossOrigin(origins = "http://localhost:3000") // Permite solicitudes CORS desde el frontend en localhost:3000
public class EstudianteController {

    @Autowired
    private EstudianteService estudianteService;

    // Obtener todos los estudiantes
    @GetMapping
    public List<Estudiante> obtenerTodos() {
        // Llama al servicio para obtener la lista de todos los estudiantes
        return estudianteService.obtenerTodos();
    }

    // Obtener estudiante por ID
    @GetMapping("/{id}")
    public ResponseEntity<Estudiante> obtenerPorId(@PathVariable Long id) {
        // Busca el estudiante por ID. Si no se encuentra, devuelve 404 Not Found
        Estudiante estudiante = estudianteService.obtenerPorId(id);
        if (estudiante == null) {
            return ResponseEntity.notFound().build(); // Si no existe, responde con 404
        }
    }
}
```

```

    return ResponseEntity.ok(estudiante); // Devuelve el estudiante si existe con 200 OK
}

// Crear nuevo estudiante
@PostMapping
public Estudiante crearEstudiante(@RequestBody Estudiante estudiante) {
    // Llama al servicio para guardar un nuevo estudiante
    return estudianteService.guardar(estudiante);
}

// Actualizar estudiante existente
@PutMapping("/{id}")
public ResponseEntity<Estudiante> actualizarEstudiante(@PathVariable Long id,
    @RequestBody Estudiante estudianteDetails) {
    // Busca el estudiante por ID. Si no se encuentra, devuelve 404 Not Found
    Estudiante estudiante = estudianteService.obtenerPorId(id);
    if (estudiante == null) {
        return ResponseEntity.notFound().build();
    }

    // Actualiza los detalles del estudiante con los nuevos valores proporcionados
    estudiante.setNombre(estudianteDetails.getNombre());
    estudiante.setActividades(estudianteDetails.getActividades());
    estudiante.setPrimerParcial(estudianteDetails.getPrimerParcial());
    estudiante.setSegundoParcial(estudianteDetails.getSegundoParcial());
    estudiante.setExamenFinal(estudianteDetails.getExamenFinal());

    // Calcular el puntaje total antes de guardar los cambios
    double puntajeTotal = calcularPuntajeTotal(estudiante);
    estudiante.setPuntajeTotal(puntajeTotal);

    // Guarda el estudiante actualizado y devuelve la respuesta con el estudiante modificado
    Estudiante estudianteActualizado = estudianteService.guardar(estudiante);
    return ResponseEntity.ok(estudianteActualizado);
}

// Eliminar estudiante por ID
@DeleteMapping("/{id}")
public ResponseEntity<Void> eliminarEstudiante(@PathVariable Long id) {
    // Busca el estudiante por ID. Si no se encuentra, devuelve 404 Not Found
    Estudiante estudiante = estudianteService.obtenerPorId(id);
    if (estudiante == null) {
        return ResponseEntity.notFound().build(); // Si no existe, responde con 404
    }

    // Elimina el estudiante por ID
    estudianteService.eliminar(id);
    return ResponseEntity.noContent().build(); // Devuelve 204 No Content si la eliminación es exitosa
}

```



```

// Método privado para calcular el puntaje total del estudiante
// (Podría estar en EstudianteService si lo prefieres)
private double calcularPuntajeTotal(Estudiante estudiante) {
    return estudiante.getActividades() + estudiante.getPrimerParcial() +
        estudiante.getSegundoParcial() + estudiante.getExamenFinal();
}
}

```

## APPLICATION.PROPERTIES

```

spring.application.name=notas
# Configuración de la base de datos
spring.datasource.url=jdbc:mysql://localhost:3306/Notas
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Dialecto de Hibernate para MySQL
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Mostrar las consultas SQL en la consola
spring.jpa.show-sql=true

# Configuración de Hibernate: actualizar la estructura de la base de datos automáticamente
spring.jpa.hibernate.ddl-auto=update

```

## FRONTEND-NOTAS>APP.JS

```

import React, { useState, useEffect } from 'react'; // Importación de React y hooks para manejar estados y efectos
import './App.css'; // Importación del archivo CSS para estilos

function App() {
    // Definición de estados para almacenar los datos de los formularios y la lista de estudiantes
    const [nombre, setNombre] = useState(""); // Estado para almacenar el nombre del estudiante
    const [actividades, setActividades] = useState(0); // Estado para almacenar la calificación de actividades
    const [primerParcial, setPrimerParcial] = useState(0); // Estado para la calificación del primer parcial
    const [segundoParcial, setSegundoParcial] = useState(0); // Estado para la calificación del segundo parcial
    const [examenFinal, setExamenFinal] = useState(0); // Estado para la calificación del examen final
    const [estudiantes, setEstudiantes] = useState([]); // Estado para almacenar la lista de estudiantes registrados
    const [estudianteld, setEstudianteld] = useState(null); // Estado para guardar el ID del estudiante que se está editando
    const [puntajeTotal, setPuntajeTotal] = useState(0); // Estado para almacenar el puntaje total calculado

    // Función para calcular el puntaje total basado en las notas ingresadas
    const calcularPuntajeTotal = () => {
        return (

```

```
(parseInt(actividades) || 0) + // Sumar la nota de actividades
(parseInt(primerParcial) || 0) + // Sumar la nota del primer parcial
(parseInt(segundoParcial) || 0) + // Sumar la nota del segundo parcial
(parseInt(examenFinal) || 0) // Sumar la nota del examen final
);
};
```

```
// useEffect para recalcular el puntaje total cada vez que cambian las notas
useEffect(() => {
  setPuntajeTotal(calcularPuntajeTotal()); // Actualizar el puntaje total cuando cambian los valores de las notas
}, [actividades, primerParcial, segundoParcial, examenFinal]); // Lista de dependencias para el efecto (se ejecuta cuando estas cambian)
```

```
// Función para registrar un nuevo estudiante enviando los datos al servidor
```

```
const handleRegistrar = async (e) => {
  e.preventDefault(); // Prevenir la acción por defecto del formulario
  const nuevoEstudiante = {
    nombre,
    actividades,
    primerParcial,
    segundoParcial,
    examenFinal,
    puntajeTotal, // Incluir el puntaje total calculado
  };
};
```

```
// Realizar una solicitud POST al servidor para registrar el estudiante
```

```
const response = await fetch('http://localhost:8080/api/estudiantes', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' }, // Especificar que el contenido es JSON
  body: JSON.stringify(nuevoEstudiante), // Convertir el objeto a JSON para enviarlo
});
```

```
// Comprobar si la respuesta fue exitosa
```

```
if (response.ok) {
  alert('Estudiante registrado correctamente'); // Mostrar un mensaje de éxito
  fetchEstudiantes(); // Actualizar la lista de estudiantes
  resetForm(); // Limpiar el formulario después de registrar
} else {
  alert('Error al registrar el estudiante'); // Mostrar un mensaje de error si la solicitud falla
}
};
```

```
// Función para obtener la lista de estudiantes desde el servidor
```

```
const fetchEstudiantes = async () => {
  const response = await fetch('http://localhost:8080/api/estudiantes'); // Realizar una solicitud GET
  const data = await response.json(); // Parsear la respuesta a JSON
  setEstudiantes(data); // Actualizar el estado con la lista de estudiantes obtenida
};
```

```
// useEffect para cargar la lista de estudiantes cuando se carga el componente
useEffect(() => {
  fetchEstudiantes(); // Llamar a la función para obtener estudiantes cuando el componente se monta
}, []); // Este efecto solo se ejecuta una vez, cuando el componente se monta
```

```
// Función para rellenar el formulario con los datos del estudiante a editar
const handleEdit = (estudiante) => {
  setNombre(estudiante.nombre); // Establecer el nombre del estudiante a editar
  setActividades(estudiante.actividades); // Establecer la nota de actividades
  setPrimerParcial(estudiante.primerParcial); // Establecer la nota del primer parcial
  setSegundoParcial(estudiante.segundoParcial); // Establecer la nota del segundo parcial
  setExamenFinal(estudiante.examenFinal); // Establecer la nota del examen final
  setEstudianteld(estudiante.id); // Guardar el ID del estudiante a editar
  setPuntajeTotal(estudiante.puntajeTotal); // Establecer el puntaje total al editar
};
```

```
// Función para actualizar los datos de un estudiante existente
const handleUpdate = async (e) => {
  e.preventDefault(); // Prevenir la acción por defecto del formulario
  const updatedPuntajeTotal = calcularPuntajeTotal(); // Recalcular el puntaje total
  const updatedEstudiante = {
    nombre,
    actividades,
    primerParcial,
    segundoParcial,
    examenFinal,
    puntajeTotal: updatedPuntajeTotal, // Incluir el puntaje total actualizado
  };
};
```

```
// Realizar una solicitud PUT al servidor para actualizar el estudiante
const response = await fetch(`http://localhost:8080/api/estudiantes/${estudianteld}`, {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' }, // Especificar que el contenido es JSON
  body: JSON.stringify(updatedEstudiante), // Convertir el objeto a JSON para enviarlo
});
```

```
// Comprobar si la respuesta fue exitosa
if (response.ok) {
  alert('Estudiante actualizado correctamente'); // Mostrar un mensaje de éxito
  fetchEstudiantes(); // Actualizar la lista de estudiantes
  resetForm(); // Limpiar el formulario después de actualizar
} else {
  const errorMessage = await response.text(); // Obtener el mensaje de error
  alert(`No se pudo actualizar el estudiante: ${errorMessage}`); // Mostrar el mensaje de error
}
};
```

```
// Función para eliminar un estudiante del servidor
const handleDelete = async (id) => {
```

```

const response = await fetch(`http://localhost:8080/api/estudiantes/${id}`, {
  method: 'DELETE', // Especificar el método DELETE para eliminar
});

// Comprobar si la respuesta fue exitosa
if (response.ok) {
  alert('Estudiante eliminado correctamente'); // Mostrar un mensaje de éxito
  fetchEstudiantes(); // Actualizar la lista de estudiantes
} else {
  alert('Error al eliminar el estudiante'); // Mostrar un mensaje de error si la solicitud falla
}
};

// Función para limpiar el formulario
const resetForm = () => {
  setNombre(''); // Limpiar el campo del nombre
  setActividades(0); // Resetear la nota de actividades
  setPrimerParcial(0); // Resetear la nota del primer parcial
  setSegundoParcial(0); // Resetear la nota del segundo parcial
  setExamenFinal(0); // Resetear la nota del examen final
  setEstudiantId(null); // Eliminar el ID del estudiante en edición
  setPuntajeTotal(0); // Resetear el puntaje total
};

return (
  <div className="App">
    <h1>Sistema de Registro de Notas</h1>

    {/* Formulario para registrar o actualizar un estudiante */}
    <form onSubmit={estudiantId ? handleUpdate : handleRegistrar}>
      <h2>{estudiantId ? 'Actualizar Notas' : 'Registrar Notas'}</h2>
      <label>Nombre del Estudiante:</label>
      <input type="text" value={nombre} onChange={(e) => setNombre(e.target.value)} required />

      <label>Actividades (Máx. 35):</label>
      <input type="number" value={actividades} onChange={(e) => setActividades(e.target.value)} max="35"
      required />

      <label>Primer Parcial (Máx. 15):</label>
      <input type="number" value={primerParcial} onChange={(e) => setPrimerParcial(e.target.value)} max="15"
      required />

      <label>Segundo Parcial (Máx. 15):</label>
      <input type="number" value={segundoParcial} onChange={(e) => setSegundoParcial(e.target.value)} max="15"
      required />

      <label>Examen Final (Máx. 35):</label>
      <input type="number" value={examenFinal} onChange={(e) => setExamenFinal(e.target.value)} max="35"
      required />

```

```

    <button type="submit">{estudianteId ? 'Actualizar' : 'Registrar'}</button>
  </form>

  <h2>Lista de Estudiantes</h2>
  { /* Mostrar la lista de estudiantes registrados */ }
  <ul>
    {estudiantes.map((estudiante) => (
      <li key={estudiante.id}>
        {estudiante.nombre} - Total: {estudiante.puntajeTotal}
        <button onClick={() => handleEdit(estudiante)}>Editar</button>
        <button onClick={() => handleDelete(estudiante.id)}>Eliminar</button>
      </li>
    ))}
  </ul>
</div>
);
}

export default App; // Exportar el componente para que pueda ser usado en otros archivos

```

## FRONTEND-NOTAS>FORMULARIONOTAS.JS

```

import React, { useState } from 'react'; // Importación de React y del hook useState para manejar el estado
import axios from 'axios'; // Importación de axios para hacer solicitudes HTTP

const FormularioNotas = ({ onNotaAgregada }) => {
  // Definición de los estados locales para manejar los campos del formulario
  const [nombre, setNombre] = useState(""); // Estado para almacenar el nombre del estudiante
  const [actividades, setActividades] = useState(""); // Estado para almacenar la nota de actividades
  const [primerParcial, setPrimerParcial] = useState(""); // Estado para la nota del primer parcial
  const [segundoParcial, setSegundoParcial] = useState(""); // Estado para la nota del segundo parcial
  const [examenFinal, setExamenFinal] = useState(""); // Estado para la nota del examen final
  const [mensajeExito, setMensajeExito] = useState(""); // Estado para manejar el mensaje de éxito

  // Función que maneja el envío del formulario
  const manejarSubmit = (e) => {
    e.preventDefault(); // Evita que el formulario recargue la página por defecto

    // Crear un objeto con los datos del estudiante y convertir las notas a enteros
    const nuevaNota = {
      nombre,
      actividades: parseInt(actividades), // Convierte la nota de actividades a entero
      primerParcial: parseInt(primerParcial), // Convierte la nota del primer parcial a entero
      segundoParcial: parseInt(segundoParcial), // Convierte la nota del segundo parcial a entero
      examenFinal: parseInt(examenFinal), // Convierte la nota del examen final a entero
    };
  };

```

```

// Enviar los datos al servidor mediante una solicitud POST usando axios
axios.post('http://localhost:8080/api/estudiantes', nuevaNota)
  .then(response => {
    onNotaAgregada(response.data); // Actualiza el estado en el componente padre (App.js)
    // Limpiar los campos del formulario después de enviar
    setNombre("");
    setActividades("");
    setPrimerParcial("");
    setSegundoParcial("");
    setExamenFinal("");
    // Mostrar mensaje de éxito
    setMensajeExito('¡Nota registrada con éxito!');
    // Limpiar el mensaje de éxito después de 3 segundos
    setTimeout(() => setMensajeExito(""), 3000);
  })
  .catch(error => {
    console.error('Error al agregar el estudiante:', error); // Muestra un error en la consola si algo falla
  });
};

// Retorna el formulario con sus respectivos campos y el botón para enviar
return (
  <form onSubmit={manejarSubmit}>
    <h2>Registrar Notas</h2>
    {/* Mostrar mensaje de éxito si existe */}
    {mensajeExito && <p style={{ color: 'green' }}>{mensajeExito}</p>}
    <div>
      <label>Nombre del Estudiante:</label>
      <input type="text" value={nombre} onChange={(e) => setNombre(e.target.value)} required />
    </div>
    <div>
      <label>Actividades (Máx. 35):</label>
      <input type="number" value={actividades} onChange={(e) => setActividades(e.target.value)} required />
    </div>
    <div>
      <label>Primer Parcial (Máx. 15):</label>
      <input type="number" value={primerParcial} onChange={(e) => setPrimerParcial(e.target.value)} required
    />
    </div>
    <div>
      <label>Segundo Parcial (Máx. 15):</label>
      <input type="number" value={segundoParcial} onChange={(e) => setSegundoParcial(e.target.value)}
    required />
    </div>
    <div>
      <label>Examen Final (Máx. 35):</label>
      <input type="number" value={examenFinal} onChange={(e) => setExamenFinal(e.target.value)} required />
    </div>
    <button type="submit">Registrar</button> {/* Botón para enviar el formulario */}
  </form>
);

```

```

    </form>
  );
};

export default FormularioNotas; // Exportar el componente para ser usado en otros archivos

```

## FRONTEND-NOTAS>LISTAESTUDIANTES.JS

```

import React from 'react'; // Importación de React

// Componente que recibe la lista de estudiantes como una prop
const ListaEstudiantes = ({ estudiantes }) => {
  return (
    <div>
      <h2>Lista de Estudiantes</h2>
      {/* Condición para mostrar un mensaje si no hay estudiantes registrados */}
      {estudiantes.length === 0 ? (
        <p>No hay estudiantes registrados.</p>
      ) : (
        // Si hay estudiantes, se muestra una lista desordenada (ul)
        <ul>
          {/* Mapea la lista de estudiantes y crea un <li> para cada uno */}
          {estudiantes.map((estudiante) => (
            // Se utiliza el id del estudiante como clave única para cada elemento de la lista
            <li key={estudiante.id}>
              {/* Se muestran los datos del estudiante */}
              <p>Nombre: {estudiante.nombre}</p>
              <p>Actividades: {estudiante.actividades}</p>
              <p>Primer Parcial: {estudiante.primerParcial}</p>
              <p>Segundo Parcial: {estudiante.segundoParcial}</p>
              <p>Examen Final: {estudiante.examenFinal}</p>
              <p>Puntaje Total: {estudiante.puntajeTotal}</p>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default ListaEstudiantes; // Exportar el componente para su uso en otros archivos

```