Práctica Guiada: gRPC con Node.js – Servicio de

Estudiantes 1. Preparación del entorno

Necesitas tener instalado:

- Node.js (>=16).
- npm (viene con Node.js).

Crea un proyecto:

mkdir grpc-estudiantes
cd grpc-estudiantes
npm init -y
Instala dependencias:
npm install @grpc/grpc-js @grpc/proto-loader

2. Definir el contrato .proto

Crea una carpeta proto y dentro un archivo estudiantes.proto:

```
syntax = "proto3";
package estudiantes;

// Mensaje que representa un estudiante
message Estudiante {
 string ci = 1;
 string nombres = 2;
 string apellidos = 3;
 string carrera = 4;
}

// Solicitud para obtener un estudiante por CI
message EstudianteRequest {
 string ci = 1;
}

// Respuesta con un estudiante
message EstudianteResponse {
 Estudiante estudiante = 1;
}

// Lista de estudiantes
message EstudiantesList {
 repeated Estudiante estudiantes = 1;
```

```
}
// Servicio
service EstudianteService {
 rpc AgregarEstudiante (Estudiante) returns (EstudianteResponse);  rpc
ObtenerEstudiante (EstudianteRequest) returns (EstudianteResponse);
rpc ListarEstudiantes (google.protobuf.Empty) returns (EstudiantesList); }

import "google/protobuf/empty.proto";
```
Nota: usamos google.protobuf.Empty para cuando no se necesita enviar parámetros. **3. Implementar el Servidor**

Crea un archivo server.js:

```
import grpc from "@grpc/grpc-js";
import protoLoader from "@grpc/proto-loader";
const PROTO_PATH = "./proto/estudiantes.proto";
// Cargar el proto
const packageDefinition = protoLoader.loadSync(PROTO_PATH, {});
const proto =
grpc.loadPackageDefinition(packageDefinition).estudiantes; // Base de
datos en memoria
const estudiantes = [];
// Implementación de los métodos
const serviceImpl = {
 AgregarEstudiante: (call, callback) => {
 const nuevo = call.request;
 estudiantes.push(nuevo);
 callback(null, { estudiante: nuevo });
 },
 ObtenerEstudiante: (call, callback) => {
 const { ci } = call.request;
 const est = estudiantes.find(e => e.ci === ci);
 if (est) {
 callback(null, { estudiante: est });
 } else {
 callback({
 code: grpc.status.NOT_FOUND,
 message: "Estudiante no encontrado"
 });
 }
 },
 ListarEstudiantes: (call, callback) => {
 callback(null, { estudiantes });
 }
};
// Crear servidor
```

```
const server = new grpc.Server();
server.addService(proto.EstudianteService.service,
serviceImpl); const PORT = "50051";
server.bindAsync(
 `0.0.0.0:${PORT}`,
 grpc.ServerCredentials.createInsecure(),
 (err, bindPort) => {
 if (err) {
 console.error(err);
 return;
 }
 console.log(`Servidor gRPC escuchando en
${bindPort}`);  server.start();
 }
);
```

4. Implementar el Cliente

Crea un archivo client.js:

```
import grpc from "@grpc/grpc-js";
import protoLoader from "@grpc/proto-loader";
const PROTO_PATH = "./proto/estudiantes.proto";
// Cargar el proto
const packageDefinition = protoLoader.loadSync(PROTO_PATH, {});
const proto =
grpc.loadPackageDefinition(packageDefinition).estudiantes;

// Crear cliente
const client = new proto.EstudianteService(
 "localhost:50051",
 grpc.credentials.createInsecure()
);

// 1. Agregar estudiante
client.AgregarEstudiante(
 { ci: "12345", nombres: "Carlos", apellidos: "Montellano", carrera:
"Sistemas" },  (err, response) => {
 if (err) return console.error(err);
 console.log("Estudiante agregado:", response.estudiante);

 // 2. Obtener estudiante por CI
 client.ObtenerEstudiante({ ci: "12345" }, (err, response) =>
{  if (err) return console.error(err);
 console.log("Estudiante obtenido:", response.estudiante);
```

```
// 3. Listar todos los estudiantes
client.ListarEstudiantes({}, (err, response) => {
if (err) return console.error(err);
console.log("Lista de estudiantes:", response.estudiantes);
});
});
}
);
```

5. Ejecutar y Probar

En dos terminales diferentes:

Terminal 1 – Levantar el servidor:
node server.js
Terminal 2 – Ejecutar el cliente:
node client.js
Deberías ver en consola algo como:
Servidor gRPC escuchando en 50051
Y en el cliente:
Estudiante agregado: { ci: '12345', nombres: 'Carlos', apellidos: 'Fernandez', carrera: 'Sistemas' } Estudiante obtenido: { ci: '12345', nombres: 'Carlos', apellidos: ‘Fernandes', carrera: 'Sistemas' } Lista de estudiantes: [
{ ci: '12345', nombres: 'Carlos', apellidos: 'Fernandez, carrera: 'Sistemas' }

Enunciado de Práctica: Sistema de Gestión de Cursos y Estudiantes con gRPC

La **Universidad X** necesita un sistema distribuido para administrar la relación entre **estudiantes** y **cursos**. El sistema debe construirse utilizando **gRPC con Node.js**.

Requisitos del sistema

1. **Definir dos entidades principales:**
    - **Estudiante**: contiene ci, nombres, apellidos, carrera.
    - **Curso**: contiene codigo, nombre, docente.
2. **Implementar los siguientes servicios gRPC:**
    - AgregarEstudiante(Estudiante) → EstudianteResponse
      Agregar un nuevo estudiante a la lista en memoria.
    - AgregarCurso(Curso) → CursoResponse
      Agregar un nuevo curso.

- InscribirEstudiante(InscripcionRequest) → InscripcionResponse
  Permitir que un estudiante se inscriba en un curso (usando ci del estudiante y codigo del curso).
- ListarCursosDeEstudiante(EstudianteRequest) → CursosList
  Dado un estudiante, listar todos los cursos en los que está inscrito.
- ListarEstudiantesDeCurso(CursoRequest) → EstudiantesList
  Dado un curso, listar todos los estudiantes inscritos.

3. **Reglas adicionales:**
   - Un estudiante puede estar inscrito en **varios cursos**.
   - Un curso puede tener **varios estudiantes**.
   - Si se intenta inscribir un estudiante en un curso en el que ya está, el sistema debe devolver un error ALREADY_EXISTS.
4. **Cliente gRPC**:
   - El cliente debe demostrar el uso de **todos los servicios**, por

     ejemplo: 1. Registrar un estudiante.

     2. Registrar dos cursos.

     3. Inscribir al estudiante en ambos cursos.

     4. Consultar los cursos del estudiante.

     5. Consultar los estudiantes de un curso.

Objetivo

El estudiante deberá implementar **el archivo .proto**, el **servidor gRPC** y un **cliente** en Node.js que permita gestionar toda esta lógica.