Informe de laboratorio #4

gRPC / Protocol Buffers

KAI DAVID ALBORNOZ MADRIGAL
JEISON DAVID MONCAYO MARZOLA

Diego Jose Luis Botia Valderrama

Profesor

Arquitectura de software

Universidad de Antioquia

2022-1

Introducción

Se desarrollará una aplicación de tipo cliente-servidor para lo cual se usará el framework de comunicación gRPC (google Remote Procedure Call) que nos permitirá conectar de manera eficiente los servicios de la aplicación.

Objetivos

Objetivo general

Desarrollar una aplicación tipo cliente-servidor usando gRPC y Protocol Buffers para autorizar o no días de permiso para una persona teniendo en cuenta la cantidad de días de vacaciones acumulados y los días de permiso solicitados.

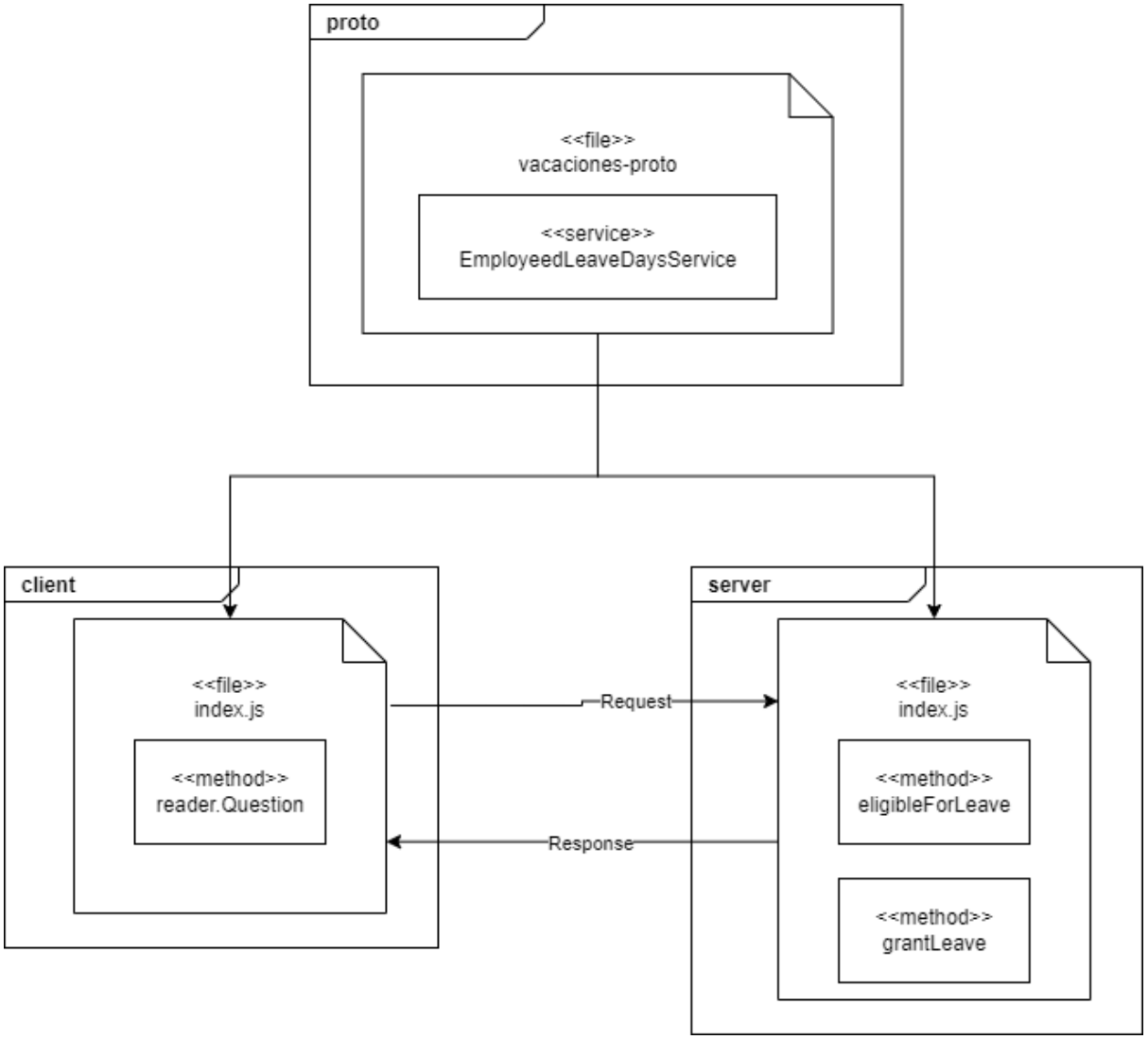
Objetivos específicos:

- Conocer y aprender gRPC
- Conocer y aprender Protocol Buffer

Herramientas empleadas

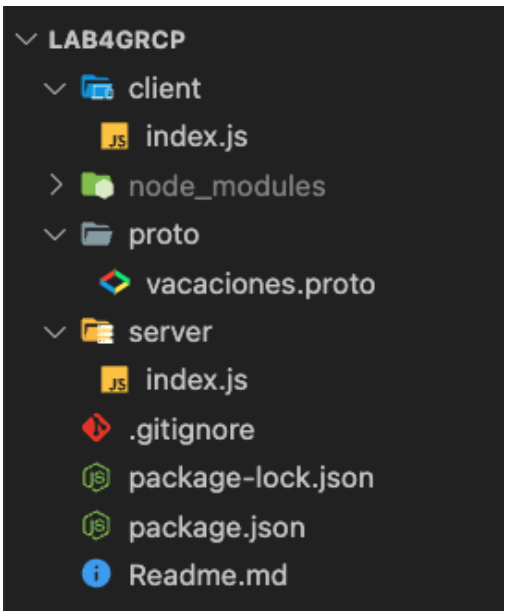
- NodeJS v16.15.1
- grpc
- @grpc/proto-loader
- Editor Visual Studio Code

Arquitectura



Procedimiento

La aplicación está dividida en 3 partes, el cliente, el servidor y proto. El cliente contiene la parte donde se interactúa con el usuario, el servidor es el que emite las respuestas al usuario y el proto es el que contiene los modelos que se enviarán entre el servidor y el usuario.



El archivo de proto y el archivo de server fueron proporcionados por el profesor para está práctica de laboratorio.
Nosotros implementamos la parte del cliente.
Antes de comenzar con la lógica del proyecto, se deben importar algunas librerías para el correcto funcionamiento.

```
let grpc = require("grpc");
let protoLoader = require("@grpc/proto-loader");
let readLine = require("readline");

let reader = readLine.createInterface({
  input: process.stdin,
  output: process.stdout,
});

let proto = grpc.loadPackageDefinition(
  protoLoader.loadSync("../proto/vacaciones.proto", {
    keepCase: true,
    longs: String,
    enums: String,
    defaults: true,
    oneofs: true,
  })
);

const remoteURL = "0.0.0.0:50050";

let client = new proto.lab4grcp.EmployeeLeaveDaysService(
  remoteURL,
  grpc.credentials.createInsecure()
);
```

Además de que se llama al servidor a través del URL y se utiliza el proto creado en vacaciones.proto.

En la primera parte de la ejecución, nosotros le pedimos al usuario los datos para correr la aplicación.

```
reader.question("Ingrese el Id del empleado: ", (employee_id) => {
  reader.question("Ingrese el nombre del empleado: ", (name) => {
    reader.question(
      "Ingrese el numero de dias de vacaciones que va a pedir: ",
      (requested_leave_days) => {
        reader.question(
          "Ingrese el numero de dias de vacaciones que tiene acumulados: ",
          (accrued_leave_days) => {
            client.eligibleForLeave(
              {
                employee_id: employee_id,
                name: name,
                requested_leave_days: requested_leave_days,
                accrued_leave_days: accrued_leave_days,
              },
```

Con estos datos, entramos a la función del servidor llamada “elegibleForLeave” la cual nos indica si el usuario puede o no tomar vacaciones.

```
client.eligibleForLeave(      You, hace 7 minutos
  {
    employee_id: employee_id,
    name: name,
    requested_leave_days: requested_leave_days,
    accrued_leave_days: accrued_leave_days,
  },
  (err, response) => {
    if (err) {
      console.log(err.details);
    } else if (response.eligible) {
```

Esta función nos arroja un error el cual ocurre si el usuario digitó mal la información y una respuesta, la cual es un booleano que nos indica si es elegible o no. En caso tal de que se digite bien los datos, se valida si el booleano retornado es verdadero o falso.

Si es verdadero, entra a la función “grantLeave” el cual calcula la cantidad de días que se le van a otorgar de vacaciones y cuántos le van a quedar acumulados para próximas vacaciones.

```
client.grantLeave(
  {      You, hace 8 minutos • Correcto funcionam
    employee_id: employee_id,
    name: name,
    requested_leave_days: requested_leave_days,
    accrued_leave_days: accrued_leave_days,
  },
  (err, response) => {
    if (err) {
      console.log(err);
    } else {
      console.log(
        "Se le han concedido " +
          response.granted_leave_days +
          " dias de vacaciones y le quedan " +
          response.accrued_leave_days +
          " dias de vacaciones acumulados"
      );
    }
  }
);
```

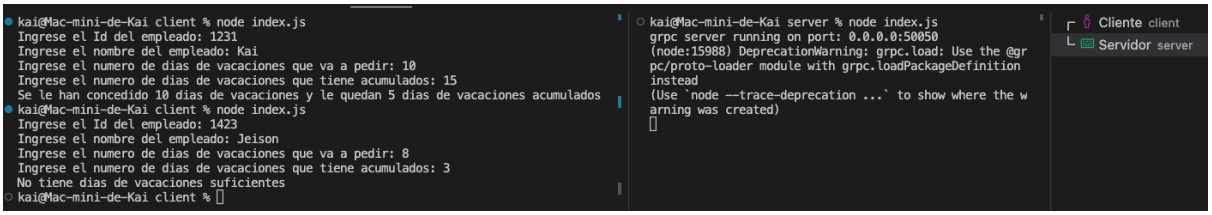
En caso tal de que la función diga que no es elegible, se rechaza la solicitud y se muestra por pantalla un mensaje de error.

```
} else {
  console.log("No tiene dias de vacaciones suficientes");
}
```

Con el reader close, cerramos la aplicación y finalizamos la ejecución.

```
    );
    reader.close();
  }
```

Al finalizar la aplicación se ve de esta manera:



Acá se ve en la parte izquierda la parte del cliente, donde se le piden los datos al usuario y se muestra por consola cuántos días se le dan de vacaciones o si se rechaza la solicitud.
En la parte derecha se ve la parte del servidor desplegado de manera local.

Conclusión

gRPC nos permite comunicar servicios de nuestras aplicaciones de forma rápida y eficiente, utiliza el protocolo HTTP/2 el cual es más flexible que HTTP/1, soporta el streaming de datos bidireccional entre cliente-servidor además del envío de mensajes en formato binario que tiene muy baja latencia, todo esto le permite obtener una óptima comunicación entre dichos servicios.

Bibliografía

- Botia, D. J. L. Arquitectura de software. Curso Arquitectura de software. Curso Arquitectura de Software. Recuperado el 8 de septiembre de 2022 de https://sites.google.com/site/cursosdiegobotia/

Anexos

Repositorio del proyecto