**UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO**

ELTON PENICELA

eltonpenicela@gmail.com

Mestrado em Engenharia Informática e Tecnologia Web

**INTEGRAÇÃO DE SISTEMAS**

Link das do projecto completo no github

<https://github.com/epenicela/Sis_ApostasEuroMil.git>

**APLICAÇÕES CLIENTE REST | GRPC**

As aplicações cliente REST e gRPC são duas tecnologias amplamente utilizadas na construção de sistemas distribuídos e comunicação entre serviços. REST (Representational State Transfer) é um estilo arquitetônico que define um conjunto de princípios para o desenvolvimento de APIs (Application Programming Interfaces) baseadas em HTTP. Por outro lado, gRPC é um protocolo de comunicação remota desenvolvido pelo Google que utiliza o Protocol Buffers para serialização de dados e suporta diversas linguagens de programação.

Ambas as tecnologias possuem suas vantagens e desvantagens, e a escolha entre elas dependerá dos requisitos específicos do projeto. O REST é amplamente utilizado em sistemas web, enquanto o gRPC é mais indicado para aplicações que exigem alta performance e escalabilidade.

Entretanto na presente, pretende-se criar dois projectos de integração de sistemas exemplificando as duas tecnologias, usando nodejs para backend (usando como tecnologias REST, e Grpc), vuejs para frontend.

**Aplicações REST**

Nsta estapa é criado um servidor Node.js que usa o framework Express para criar uma API RESTful que fornece acesso a recursos específicos através de endpoints HTTP. Incluindo as dependências necessárias, que são o *Axios* para fazer chamadas *HTTP* e o *Express* para criar o servidor.

Em seguida, há três rotas diferentes definidas com o método app.get(). Cada rota tem um endpoint diferente, com dois parâmetros de consulta: credit\_account\_id e value. As rotas fornecem acesso a diferentes recursos:

/check/:credit\_account\_id/ammount/:value: Esta rota é responsável por verificar se um cheque digital pode ser autenticado com um determinado credit\_account\_id e value. Ele usa o pacote Axios para fazer uma chamada para outro servidor que possui uma API para verificar a autenticidade do cheque. A resposta é analisada para verificar se o cheque digital é autêntico ou não e retorna a resposta correspondente como JSON.

/lua/:credit\_account\_id/ammount/:value: Esta rota é um endpoint de teste que sempre retorna um objeto JSON específico.

/lista/:credit\_account\_id/ammount/:value: Esta rota é responsável por retornar a primeira entrada em uma lista de cheques digitais autenticados. A resposta é novamente analisada para recuperar o primeiro cheque digital autenticado e é retornada como JSON.

A última parte do código usa o método app.listen() para iniciar o servidor na porta 3000 e exibe uma mensagem no console informando que o servidor está em execução.

**Aplicações GRPC**

Nesta tecnologia foram executados os seguintes passos:

1. Instalar o pacote grpc

npm install grpc

1. Criando um arquivo server.js

Criar um arquivo server.js que define a implementação do serviço Euromil. Neste arquivo, foi criada uma instância do servidor gRPC e registrar a implementação do serviço Euromil.

const euroMilService = grpc.load('euromil.proto').euromil.Euromil;

function registerEuroMil(call, callback) {

const key = call.request.key;

const checkId = call.request.checkid;

// TODO: implementar a lógica de registro do Euromilhões

const message = 'Aposta registrada com sucesso';

callback(null, { message });

}

const server = new grpc.Server();

server.addService(euroMilService, { RegisterEuroMil: registerEuroMil });

server.bind('0.0.0.0:50051', grpc.ServerCredentials.createInsecure());

console.log('Servidor gRPC iniciado na porta 50051');

server.start();

Neste exemplo, a função registerEuroMil é chamada sempre que o método RegisterEuroMil é invocado pelo cliente. A função recebe um objeto call que contém as informações da chamada (como os dados da mensagem de requisição) e um callback que deve ser invocado com a resposta do serviço.

1. Criar um arquivo client.js

Criando um arquivo client.js que define um cliente para acessar o serviço gRPC. Neste arquivo, foi criada uma instância do cliente gRPC e invocar o método RegisterEuroMil do serviço Euromil.

const grpc = require('grpc');

const euroMilService = grpc.load('euromil.proto').euromil.Euromil;

const client = new euroMilService('localhost:50051', grpc.credentials.createInsecure());

const key = '1234567890';

const checkId = 'a1b2c3d4e5';

client.RegisterEuroMil({ key, checkid: checkId }, (err, response) => {

if (err) {

console.error('Erro ao registrar aposta:', err);

} else {

console.log('Resposta do servidor:', response.message);

}

});

Neste exemplo, o cliente foi criada uma instância do serviço Euromil apontando para o endereço do servidor (no caso, localhost:50051) e invoca o método RegisterEuroMil, passando o key e o checkid da aposta. O callback é invocado quando o servidor enviar a resposta.

**Referências**

Richardson, L., Amundsen, M., & Ruby, S. (2013). RESTful web APIs: Services for a changing world. O'Reilly Media, Inc.

Sampaio, A., & Pinto, M. (2019). A Comparative Analysis of REST and gRPC for Microservices. In Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (pp. 307-314).

gRPC documentation: <https://grpc.io/docs/>

Conceitos básicos sobre gRPCs, https://grpc.io/docs/guides/concepts/