Mestrado em Engenharia Informática e Tecnologia Web

Integração Sistemas

Tarefa 4 – Atividade II: aplicações cliente REST | gRPC

Nome do autor: Tiago Miguel Lima Romão

Nº aluno UTAD: 75309

Repositório: https://github.com/TiagoRomao-MEIW/Atividade2

REST

O cliente REST criado em Javascript devolve o resultado do método GET à API existente em *ken.utad.pt:8181/check/{credit_account_id}/amount/{value}* em que as variáveis **NumConta** e **Valor** correspondem aos atributos *{credit_account_id}* e *{value}* correspondentemente exibindo-o numa página HTML.

```
const URL = 'https://cors-anywhere.herokuapp.com/http://ken.utad.pt:8181/check/';
const NumConta = '12345678';
const Valor = '1000';
const response = await fetch(URL + NumConta + '/amount/' + Valor);
const data = await response.json();
const {date, checkID} = data;
```

O cliente envia o pedido à rota composta pelas variáveis **URL**, **NumConta** e **Valor** e o resultado devolvido, data do cheque (**date**) e número do cheque (**checkID**), atualiza o texto dos elementos HTML existentes na página de exibição de resultados.

```
<body>
    <script src="REST.js"></script>REST
    <p id="pResultado">
        <div>Data Cheque: <span id="GetDate" style="color: goldenrod;"></span></div>
        <div>Número do Cheque: <span id="GetCheckID" style="color: goldenrod;"></span></div>
    </p>
</body>
```

A página de HTML ao ser carregada chama o ficheiro JS que contém o cliente REST que irá atualizar os valores dos elementos HTML para exibir a resposta da API.

REST

Resultado do método Get para a conta 12345678 e valor 1000

Data Cheque: 2022-01-16T15:10:03
Número do Cheque: 6690732236553087

gRPC

O cliente gRPC construído em Javascript, utiliza um ficheiro de Protocol Buffers (proto3) com a seguinte definição:

syntax define a versão do ficheiro;

service Euromil define o serviço a ser executado;

```
syntax = "proto3";

service Euromil {
  // Sends a greeting
  rpc RegisterEuroMil (RegisterRequest) returı
}
```


Tiago Romão - 75309
Mestrado em Engenharia Informática e Tecnologia Web

message RegisterRequest é o pedido a ser enviado e que tem que estar corretamente formatada;

message RegisterReply é a resposta da API ao pedido enviado

```
message RegisterRequest {
  string key = 1;
  string checkid = 2;
}
message RegisterReply {
  string message = 1;
}
```

O cliente começa por carregar os pacotes necessários para a comunicação gRPC;

```
//Pacotes necessários
const grpc = require('@grpc/grpc-js')
const protoLoader = require('@grpc/proto-loader')
//Caminho do ficheiro Protocol Buffers (versão proto3)
const PROTO_PATH = "./EuroMil.proto";
```

De seguida, o ficheiro *proto3* é carregado numa variável que será utilizada como definição na comunicação gRPC utilizando o endpoint *ken.utad.pt:8282* e, no caso deste API, uma ligação não segura (sem SSL).

```
var protoObj = protoLoader.loadSync(PROTO_PATH);
const EMilDefinition = grpc.loadPackageDefinition(protoObj).Euromil;
const client = new EMilDefinition('ken.utad.pt:8282', grpc.credentials.createInsecure());
```

Finalmente o pedido envia mensagem *RegisterRequest* esperando o retorno da mensagem *RegisterReply*.

```
// Envio do pedido
client.RegisterEuroMil({
  "key": "10 20 30 40 50 60 70",
  "checkID": "1234567812345678"
}, (err, message) => {
  if (err) throw err;
  console.log(message)
});
```

Bibliografia

1. Medium. *Introdução ao gRPC com Node.JS* https://medium.com/xp-inc/introdu%C3%A7%C3%A3o-ao-grpc-com-node-js-98f6a4ede11
2. Lucas Santos. *O guia do gRPC parte 2: Mãos à obra com JavaScript* https://blog.lsantos.dev/o-guia-do-grpc-2/