# Aplication.properties

Contains mainly properties to control the CORS origins. In this case it’s allowing requests from every sources and only the GET method as allowed as it isn’t necessary more.

|  |
| --- |
| quarkus.http.cors=true quarkus.http.cors.origins=\* quarkus.http.cors.headers=accept, authorization, content-type, x-requested-with quarkus.http.cors.methods=GET |

There is also a small tweak in the quarkus service port as the default is 8080 and that is usually being used for something else.

|  |
| --- |
| quarkus.http.port=9000 dev.quarkus.http.port=9000 |

# LabSeqController

This is the REST API implementation. It contains the GET method getSequenceValue that is responsible to return the solution of the sequence.

It is programmed to basically return two answer types success (error code 200) and error (error code 400) for when the parameter is invalid.

|  |
| --- |
| Response.*status*(400).entity("Error!Invalid value: {" + pos + "}. Must be bigger than 0.").build();  Response.*ok*(result.toString()).build(); |

It also contains few annotations that will ease the swaggerUI documentation.

The controller class will be supported by LabSeqService which is the class that contains the logical solution to the sequence.

# LabSeqService

This class contains a method where the calculation of the requested sequence is implemented.

The method calculateSequence receives a numeric value (validated to being zero or more) and sends a solution to the sequence that is a BigInteger. This was a necessity as float or int where not big enough to hold the solution to n=10000.

|  |
| --- |
|  |

It also contains a simple cache mechanism in the form of linkedList that contains the configurable size to hold the most recent solutions so that it can speed up the process in larger numbers.

|  |
| --- |
|  |

# SwaggerUI

As said before this project contains a swaggerui where some documentation can be checked and test the endpoint.

It can be accessed by default in [http://localhost:9000/q/swagger-ui/#](http://localhost:9000/q/swagger-ui/) and the starting page looks like this:

|  |
| --- |
|  |

On the dropdown menu can be seen information about the method and it contains a “try it out” button that should be used to test the api.

|  |
| --- |
|  |

The api can also be tested by postman by importing the following:

curl --location 'http://localhost:9000/labseq/1000'

# webapp

It was also developed a very simple angular project where the api can be tested. The design is very rudimentary and simple but as a proof of concept it works.

|  |
| --- |
| Uma imagem com texto, captura de ecrã, Tipo de letra, número  Descrição gerada automaticamente |

The html code is simple and is based on a two way bind system so that the variables are linked together.

|  |
| --- |
| <input [(ngModel)]="index"> <button (click)="execute()">Execute</button> <div class="output">  {{output}} </div> |

On clicking the button the execute method will be called and a request to the api will be made:

|  |
| --- |
| execute() {  this.http.get('http://localhost:9000/labseq/' + this.index).subscribe(  (response:any) => { console.log(response);  this.output = String(response)  },  (error) => { console.log('error ', error); } );  } |

# Problems

In bigger n values the result was too big which caused some problems asserting that the returned value was in fact correct.

In angular solution I couldn’t find a way to display such big values so the solution shows an infinity value, starting around n=3560.

|  |
| --- |
|  |

The value should be a string however angular is casting it as a number. In console log the result is displayed properly.

|  |
| --- |
|  |