[[1]](#footnote-1)

Web framework Implementation using maven JAVA(September 2019)

David Daniel A. Ramirez Moreno, Systems engineering student

# INTRODUCTION

T

HIS document is going to show a brief approximation on the realization of a basic web framework that can handle petitions such as images, web pages in html format and also is able to execute methods defined via POJOs and their respective annotation.

Across this document information about the architecture, deployment and the operation of the application can be found and explained the easiest way possible.

Finally, there will be a functional section where the application will be tested an showed to the user working with images that explain the application and the different petitions that the framework support.

# Software architecture

Imagen que contiene captura de pantalla

Descripción generada automáticamente

The framework follow a simple architecture which is displayed before, first of all we notice that the client access the application via web on the browser of its preference, as the application is hosted on Heroku, the client enters the Heroku link where the app is deployed (<https://arep-project-david-ramirez.herokuapp.com/hola.html>) this connection is made with https. After the connection is done the client can access to 3 resources available at the moment, those are .jpg images available on the server, .html files available on the server and an specific method from a POJO correctly defined on the URL.

To access an image the petition must finish with .jpg e.g.,/resource/image.jpg,image.jpg.

To access to an html file the request must end with the characters .html e.g., hello.html, /resorce/hello.html.

Finally, to access to a method from a POJO the next structure must be followed :

/app/{name of the POJO}/{name of the method}[?param1={param}&param2={param2}…]

e.g. /app/POJO/saludar?name=david

/app/POJO/ejecutarClasePrueba

All of these petitions are handled from the app server and if something goes wrong a web page is displayed showing the error to the user.

# Deployment Diagram

Imagen que contiene captura de pantalla

Descripción generada automáticamente

As we can see in the diagram the framework is based on the server-client architecture where the connection is direct and every petition from the client will be managed by the server, first of all, a web socket is opened from where the server will receive the petitions, also it will keep listening through this port until a connection is made, when it is done the server proceed to wait from a petition coming from the port.

each petition will travel from the client´s web browser through the Heroku server and then it will arrive to the AppServer class present in the application, after arriving, the AppServer class will read and interpret the petition, after it manage to understand what is being asked, it proceed to search for the file following the rules previously explained, if the file is found it is returned to the user, if no such file is found the application will send a web page displaying an error 404 not found.

The images and the html files are stored on a file located on the server at /recursos, on the other hand the POJOs, by its nature are java classes so those are stored at /src/main/java/edu/escuelaing/arep/

# Testing the application

The following chapter will be divided into 5 parts, in the first part it will be shown how to bring an html from the server, the second part will show how to import an image, on the third part it will be displayed how to use methods without parameters and the next part will contain the methods with parameters and finally we will see what happens when a resource that does not exist is being asked, all of this request will be made on the application deployed on the Heroku server following the next link: <https://arep-project-david-ramirez.herokuapp.com/>

1. The first test consists on bringing an html file, first we check that the file is present on the server, for this purpose we are going to navigate through the server files:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Then we ask for it on the application:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

And we notice that it worked correctly.

1. The second test consist on an image being asked from the server, so we proceed to see that the image is present on the server files:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

We will be asking to the lion image, so we proceed to write it on the url:

Imagen que contiene animal, mamífero, foto, hierba

Descripción generada automáticamente

1. Now we will use the POJO defined for the project so it goes as follows:

Imagen que contiene captura de pantalla, monitor

Descripción generada automáticamente

The method that does not contains parameters is “ejecutarClasePrueba” so we follow the notation defined previously:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

1. Now we will try a method with parameters, for this we will use the “saludar” method:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

1. Our final test consists on the file not found error so we will ask for a resource that does not exist to see what happens:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Conclusion

The project was a nice approximation to what the creation of a framework is, with the implementation of the resources folder, the handlerMethod,the web annotation and the App server, a library can be created and anyone can create an unique and independent POJO to display a basic web app without noticing what is happening behind, just using the annotation @web and calling it from the browser.

1. [↑](#footnote-ref-1)