Colombian School of Engineering
Enterprise Architecture, 2020-2

HTTP SERVER AND SPARK IMPLEMENTATION

Sergio Alejandro Bohórquez Alzate

04/09/2020

Contents

1 Introduction

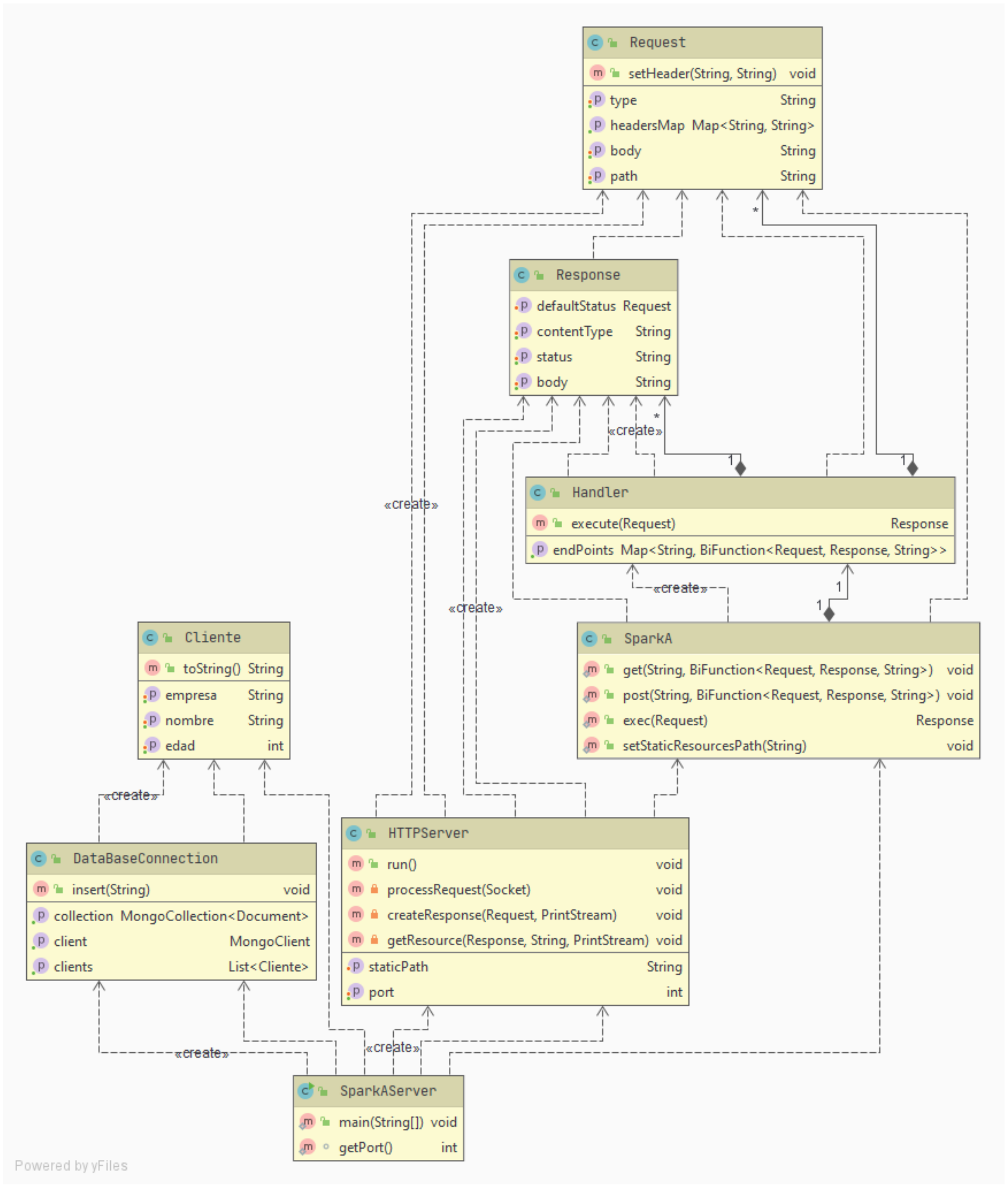
In this document we will find the description of the architecture used to solve laboratory 3 of the AREP subject, it was requested to implement a full http server that could load static resources within the context of the server and also some dynamic resources such as database and get and post requests, for which a framework similar to spark was implemented.

2 Objectives

- To understand the functioning and architecture of an http server
- To understand the functioning and architecture of java frameworks such as spark
- To implement an http server and its basic requests
- To implement a framework similar to spark.

3 Architecture

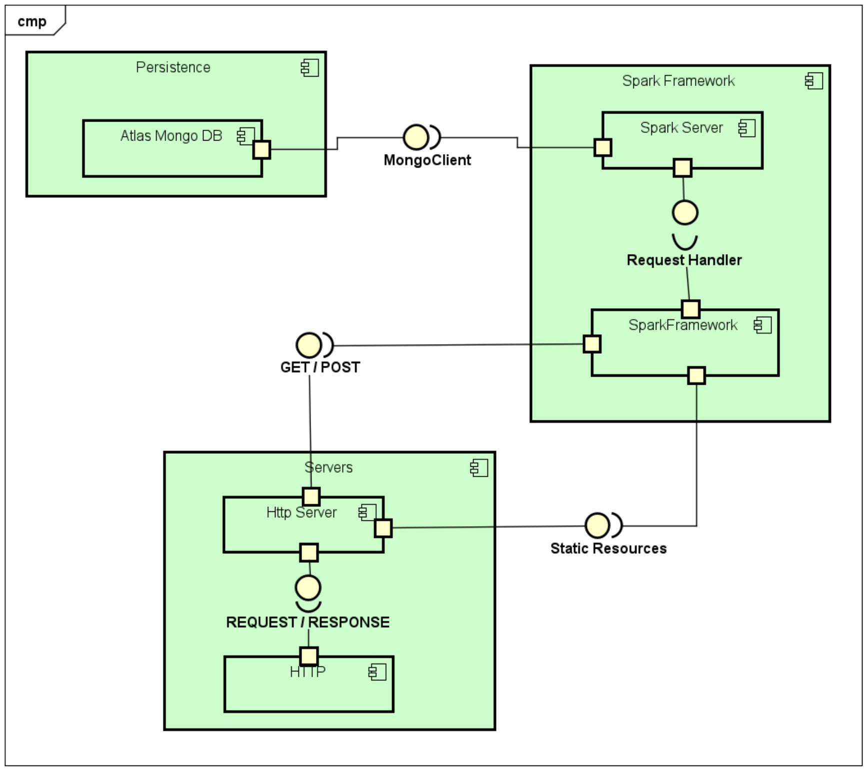
3.1 Class Diagram


Request
setHeader(String, String) void
type String
headersMap Map<String, String>
body String
path String
Response
defaultStatus Request
contentType String
status String
body String
Handler
execute(Request) Response
endPoints Map<String, BiFunction<Request, Response, String>>
SparkA
get(String, BiFunction<Request, Response, String>) void
post(String, BiFunction<Request, Response, String>) void
exec(Request) Response
setStaticResourcesPath(String) void
Cliente
toString() String
empresa String
nombre String
edad int
DataBaseConnection
insert(String) void
collection MongoCollection<Document>
client MongoClient
clients List<Cliente>
HTTPServer
run() void
processRequest(Socket) void
createResponse(Request, PrintStream) void
getResource(Response, String, PrintStream) void
staticPath String
port int
SparkAServer
main(String[]) void
getPort() int
«create»
1
*
Powered by yFiles

In the image above we see the class diagram, which presents us with a main view of the structure of our project. We will describe the classes below:

- Request http request containing the headers and body of the payload in the case of the post method, plus the type of request.
- Response http response containing the body of the response for its respective request and the type of content to be sent.
- HttpServer the http server is in charge of sending the respective necessary files and printing what is needed in the browser
- Handler the handler saves the non-static requests defined by the developer
- SparkA is in charge of defining the main behavior for the spark framework, in addition it defines the respective functional interfaces.
- SparkServer creates an http server that allows to take advantage of the functionalities of the HttpServer and the SparkA at the same time
- DatabaseConnect offers a connection to the MongoDB database
- Cliente is an example class that allows parsing database objects to java.

3.2 Internal Server Architecture



In the following component diagram we observe the main components of our spark-like framework, the main components and the interfaces they offer, in addition it gives us an idea of how the program works at runtime.

Servers

The servers component is in charge of receiving several non-concurrent requests from a client, the client can be both the developer and an end user of the deployed server.

- GET / POST are functional interfaces that offer the spark server the functionality to obtain resources such as updating resources.
- Static Resources offers the possibility of delivering static resources to the
- REQUEST/RESPONSE we made the abstraction of the main objects of the http protocol in this case the use of Requests and the responses to these requests, both in the request and in the response we store useful information such as payload, headers and so on.

Spark Framework

This component is the core for the developers, it is in charge of making the connection to the database and offering the functional interfaces for get and post which can be easily used by other systems, for example a javascript frontend.

- Request Handler The request handler has the function to control the get and post requests stored in the server, it controls them with the Bifunction functional interface that offers the developer the possibility to work with an easy to understand framework and without the need to perform many configuration steps

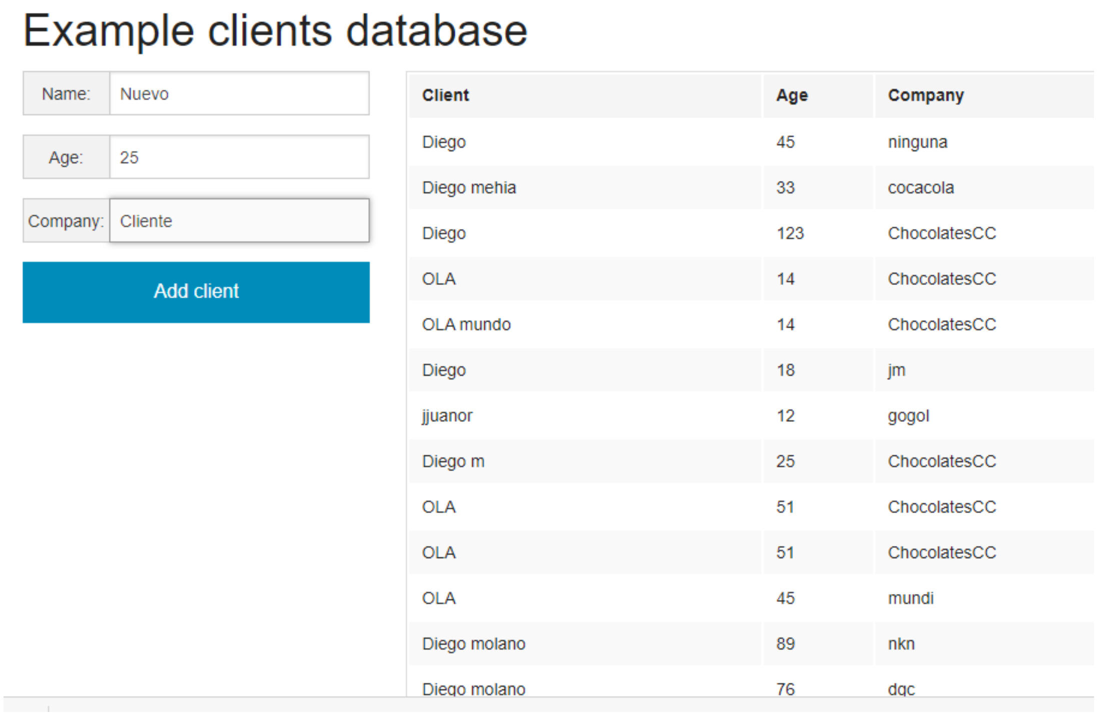
Persistence

This component offers a non-relational database client defined by documents which are very similar to JSON objects, making an analogy between relational databases, we could say that the collections could be the database varrels and the documents would do the table function.

3.3 Architecture Description

4 Testing the Server

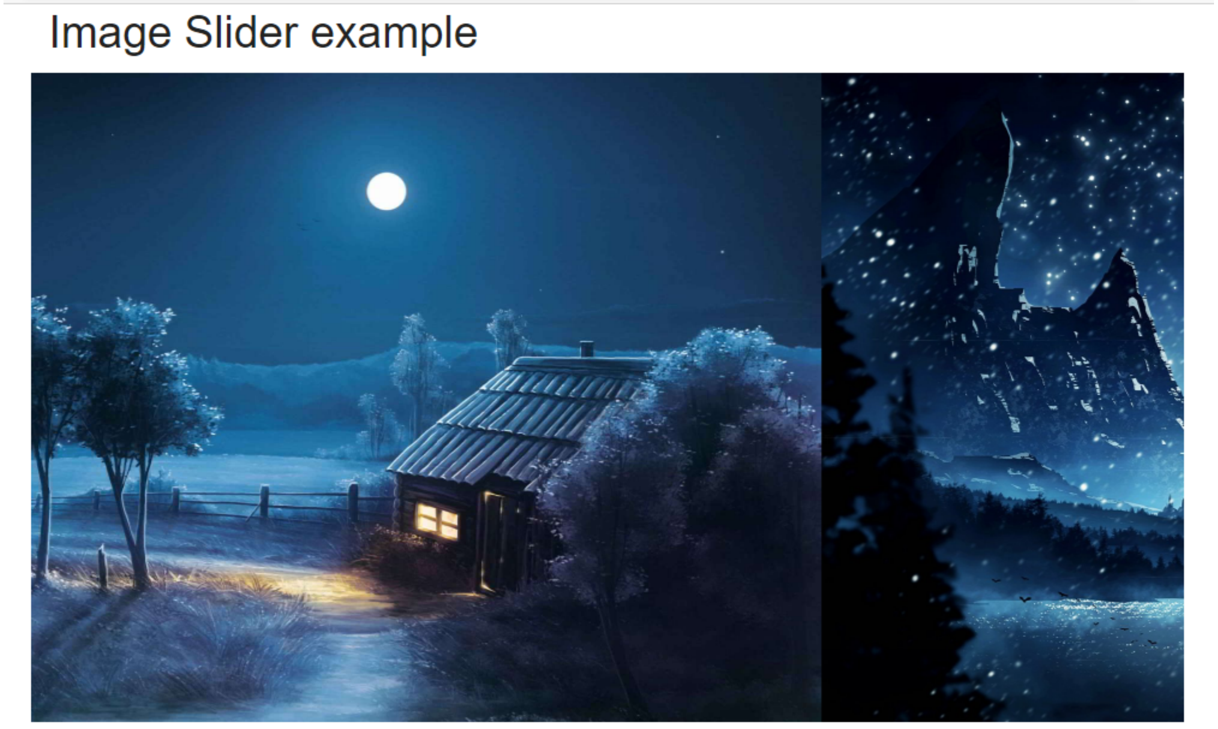
In the deployment to heroku we observed a functional demo of the https://spark-implementation-alejandro.herokuapp.com/clients endpoint in which we can make a POST request and add clients to the database.


Example clients database

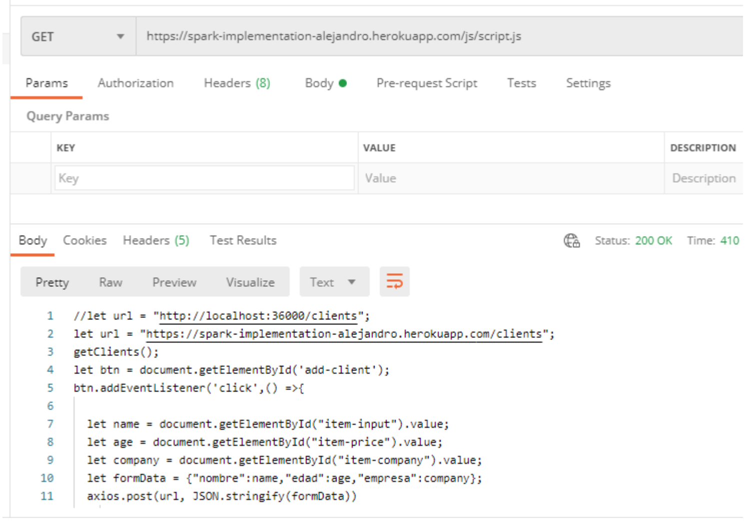
Name:	Nuevo
Age:	25
Company:	Cliente

Client	Age	Company
Diego	45	ninguna
Diego mehia	33	cocacola
Diego	123	ChocolatesCC
OLA	14	ChocolatesCC
OLA mundo	14	ChocolatesCC
Diego	18	jm
jjuanor	12	gogol
Diego m	25	ChocolatesCC
OLA	51	ChocolatesCC
OLA	51	ChocolatesCC
OLA	45	mundi
Diego molano	89	nkn
Diego molano	76	dgc

In addition we will see a slide of images with some animations.



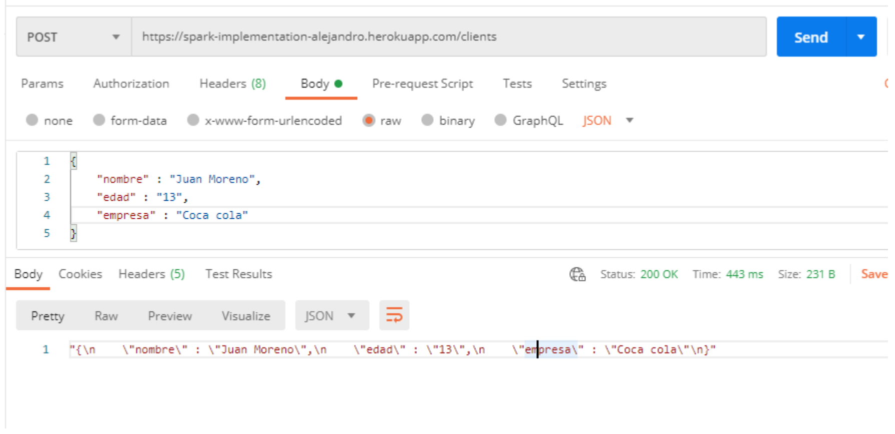
In the following example we will see the use of postman to display a static resource, we will do a get to the main url of the page and observe the response in plain text format.



```
//let url = "http://localhost:36000/clients";
let url = "https://spark-implementation-alejandro.herokuapp.com/clients";
getClients();
let btn = document.getElementById('add-client');
btn.addEventListener('click',() =>{

  let name = document.getElementById("item-input").value;
  let age = document.getElementById("item-price").value;
  let company = document.getElementById("item-company").value;
  let formData = {"nombre":name,"edad":age,"empresa":company};
  axios.post(url, JSON.stringify(formData))
```

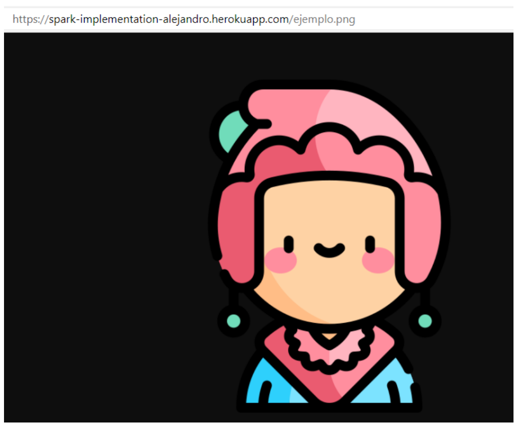
Sent data to the database via postman to the endpoint https://spark-implementation-alejandro.herokuapp.com/clients



```
{
    "nombre" : "Juan Moreno",
    "edad" : "13",
    "empresa" : "Coca cola"
}
```

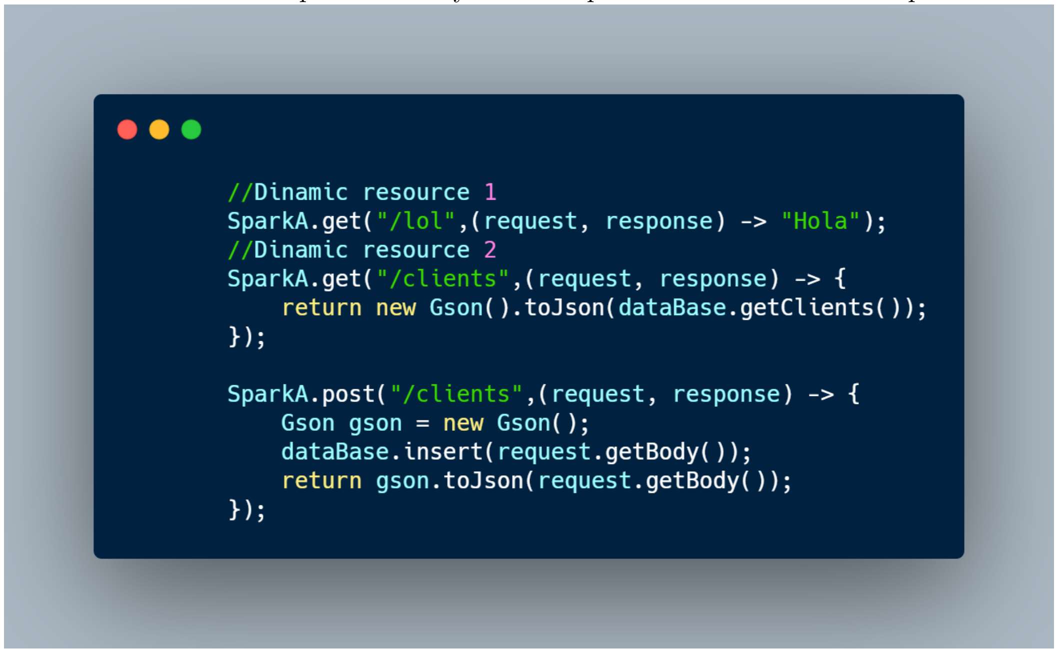
```
"{\n    \"nombre\" : \"Juan Moreno\",\n    \"edad\" : \"13\",\n    \"empresa\" : \"Coca cola\"\n}"
```

We can also observe the reading of images, we will see that we can obtain other type of static resources, to visualize it correctly we will use the browser.


https://spark-implementation-alejandro.herokuapp.com/ejemplo.png

El servidor soporta recursos estáticos, sin embargo también soporta recursos dinámicos, o creados por parte del desarrollador, anteriormente observamos el uso del servidor por parte del usuario, sin embargo, para explotar todo el potencial del servidor y del framework debemos clonar nuestro repositorio y crear los endpoints necesarios para aprovechar el uso de los métodos GET y POST implementados.

Next we will see an example of some dynamic endpoints used from the developer side.

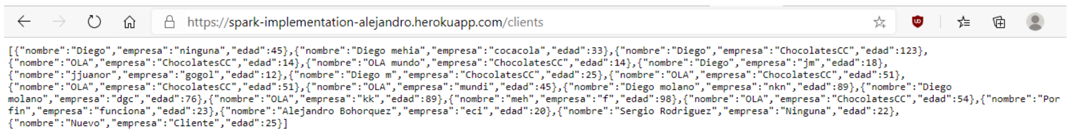


```
//Dinamic resource 1
SparkA.get("/lol",(request, response) -> "Hola");
//Dinamic resource 2
SparkA.get("/clients",(request, response) -> {
    return new Gson().toJson(dataBase.getClients());
});

SparkA.post("/clients",(request, response) -> {
    Gson gson = new Gson();
    dataBase.insert(request.getBody());
    return gson.toJson(request.getBody());
});
```

We observe the use of these endpoints in the following images:

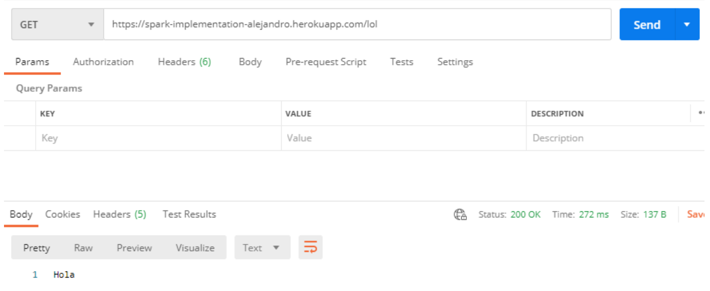
Use of the database:



https://spark-implementation-alejandro.herokuapp.com/clients

```
[{"nombre":"Diego","empresa":"ninguna","edad":45},{"nombre":"Diego mehia","empresa":"cocacola","edad":33},{"nombre":"Diego","empresa":"ChocolatesCC","edad":123},
{"nombre":"OLA","empresa":"ChocolatesCC","edad":14},{"nombre":"OLA mundo","empresa":"ChocolatesCC","edad":14},{"nombre":"Diego","empresa":"jm","edad":18},
{"nombre":"jjuanor","empresa":"gogol","edad":12},{"nombre":"Diego m","empresa":"ChocolatesCC","edad":25},{"nombre":"OLA","empresa":"ChocolatesCC","edad":51},
{"nombre":"OLA","empresa":"ChocolatesCC","edad":51},{"nombre":"OLA","empresa":"mundi","edad":45},{"nombre":"Diego molano","empresa":"nkn","edad":89},{"nombre":"Diego
molano","empresa":"dgc","edad":76},{"nombre":"OLA","empresa":"kk","edad":89},{"nombre":"meh","empresa":"f","edad":98},{"nombre":"OLA","empresa":"ChocolatesCC","edad":54},{"nombre":"Por
fin","empresa":"funciona","edad":23},{"nombre":"Alejandro Bohorquez","empresa":"eci","edad":20},{"nombre":"Sergio Rodriguez","empresa":"Ninguna","edad":22},
{"nombre":"Nuevo","empresa":"Cliente","edad":25}]
```

Plain text resource using postman



GET https://spark-implementation-alejandro.herokuapp.com/lol

Status: 200 OK Time: 272 ms Size: 137 B

```
Hola
```

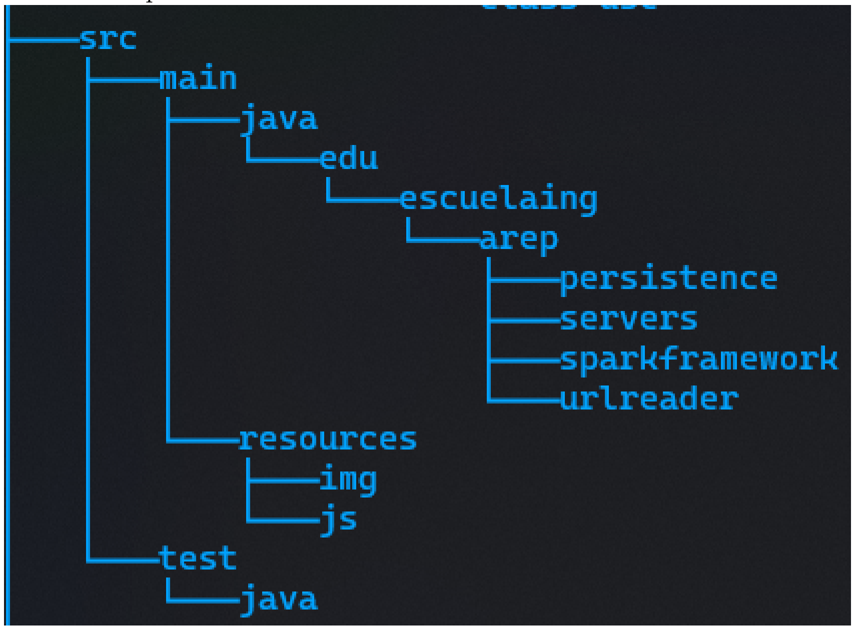
Some of the endpoints require connection to a database, in this case we use the MongoDb database, which offers us a database cluster with a non-document-based database.

4.1 Technology Stack

- **Development environment**
  - Java Development Kit 8.0
  - Java Runtime Environment 8.2
  - IntelliJ IDEA
- **Dependency Manager**
  - Maven for build the project,
- **Version control**
  - Git for changes and version control.
- **Deployment**
  - Heroku
  - CircleCI.
- **Tests**
  - JUnit
  - PostMan

4.2 Project tree

Next it will be shown the structure of the project generated by the maven dependency manager, with the objective of giving more understanding to the reader about how the solution was implemented.



```
│
├───src
│   ├───main
│   │   ├───java
│   │   │   └───edu
│   │   │       └───escuelaing
│   │   │           └───arep
│   │   │               ├───persistence
│   │   │               ├───servers
│   │   │               ├───sparkframework
│   │   │               └───urlreader
│   │   └───resources
│   │       ├───img
│   │       └───js
│   └───test
│       └───java
│
```

4.3 Documentation

If you want to see more information about the implementation of this program, you can go to the following link `https://alejandrobohal.github.io/Spark-Implementation-AREP/` in which you will find a specification for each class found in the class diagram shown above.

5 Conclusions

- We implemented an http server as well as a framework similar to Spark that offers both user and developer functionality.
- The architecture of an http server is based on client-server, the request and response represent the types of messages that can be sent
- A server allows access to both static and dynamic resources defined in the development.