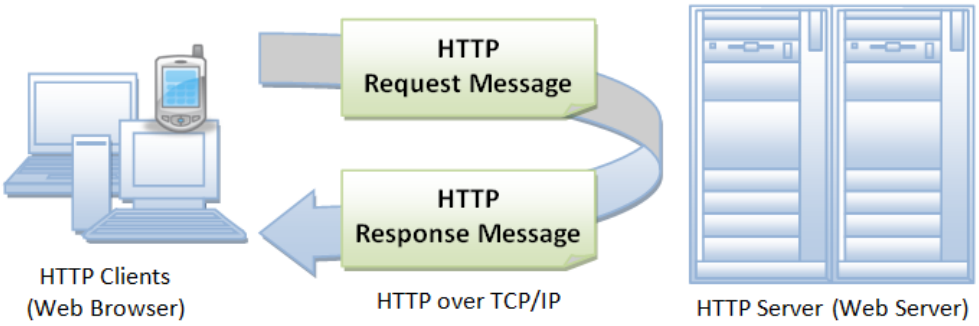
CA HTTP RESPONSE

FARM PROJECT

CLOUD BASED WEB APPLICATIONS



LECTURE: David Gonzales
STUDENT: Ayme Margot Hurtado
2019284

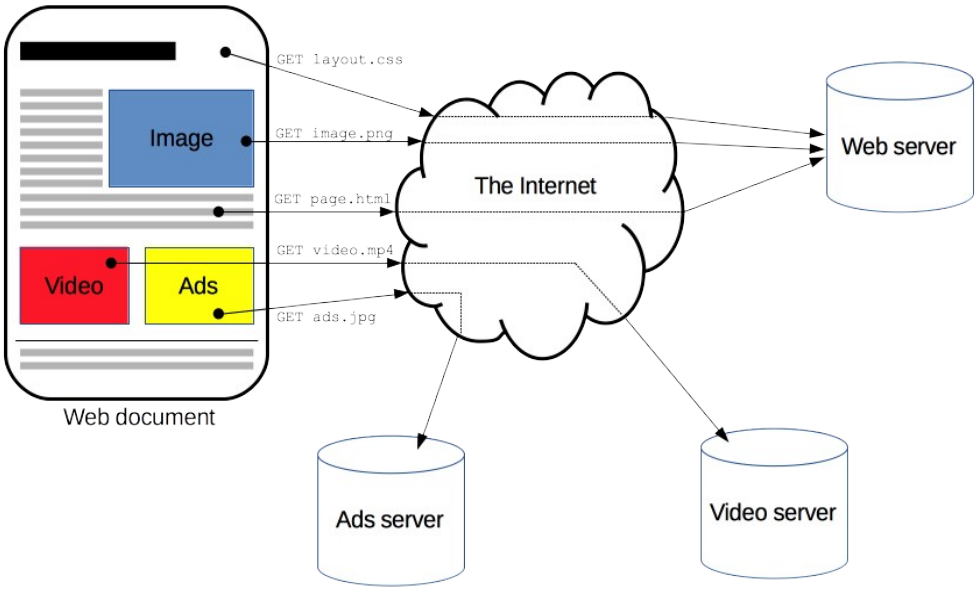
Table of content

1. Introduction

The purpose of the project is define the concepts of HTTP REQUEST, its functionality, how is it implemented, the type of requests made by the client, the server response, the REST API platforms, all exposed on a practical example.
Before to begin, some concepts are necessaries for a good understanding.

1. Http

Http is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.



2. How HTTP Request Work

A request is a way to exchange data between a client and server, the client sends the request, the server responds to the request, send a specific URL, the message returned by the server is called RESPONSE.

3. ***Data Request***

   The data can be sent through an URL parameter like

   example.com/api?data=value

   We are going to use Google Chrome, or another type of browser

   It also can be sent though Form data, in a JSON format data,

   JSON({"data":"value"})

   we are going to use postman, which will be defined later.

   Is possible send pictures and files even XML data, depends of the scenario, In the project is being used JSON format though postman and URL data format though Google Chrome or any browser

4. ***Request Components***

   Method Get, Post, Delete, Put

   Know as Request verbs, each one describes what should be done with the source, for example if the user wants to create, update, delete an entity

   Request Headers

   When the client send a request to the server, the request contains a header and body, the header contains additional information such as what type of response is required, for example how the server has to response, in XML, JSON or some other format

   Request body

   Contains the data to sent to the server, example, the post request contains the data for the new item the user wants to create the data format may be on XML or JSON

5. ***Response Components***

   Response body

   The response body contains the data sent as as a response from the server

   Status Code

   Provide the client, the status of the request

   Those are the status code depends of the use

   1XX Information

   2XX Success

3XX Redirect

4XX Client Error

5XX Server Error

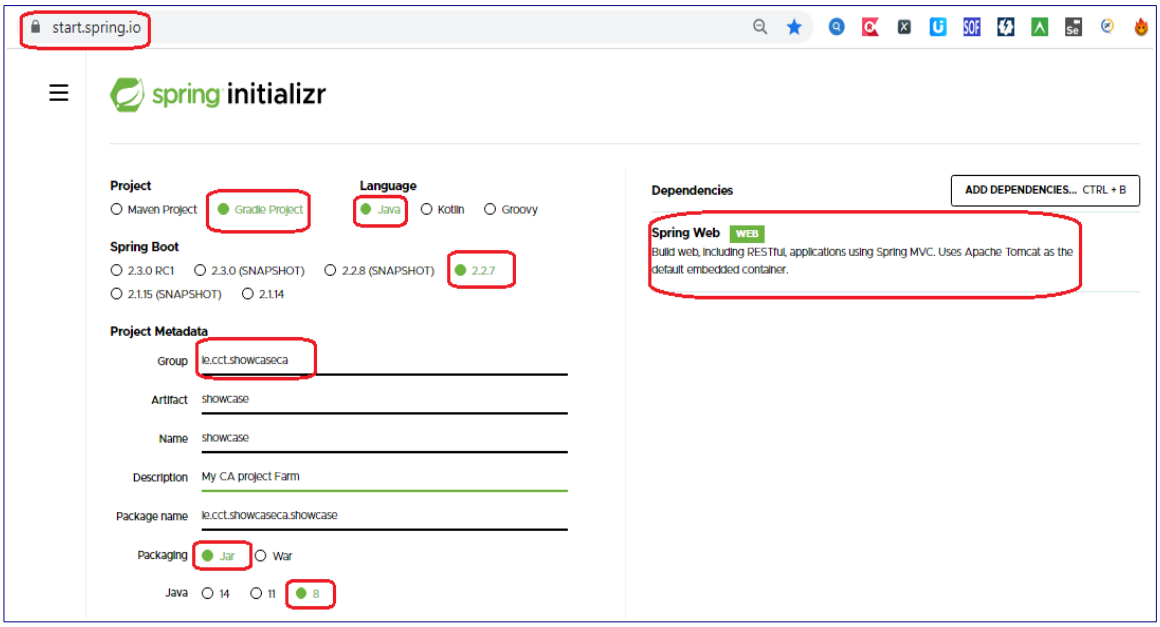
2. About the application

The server used for this purpose is Spring Framework.

1. What is Spring

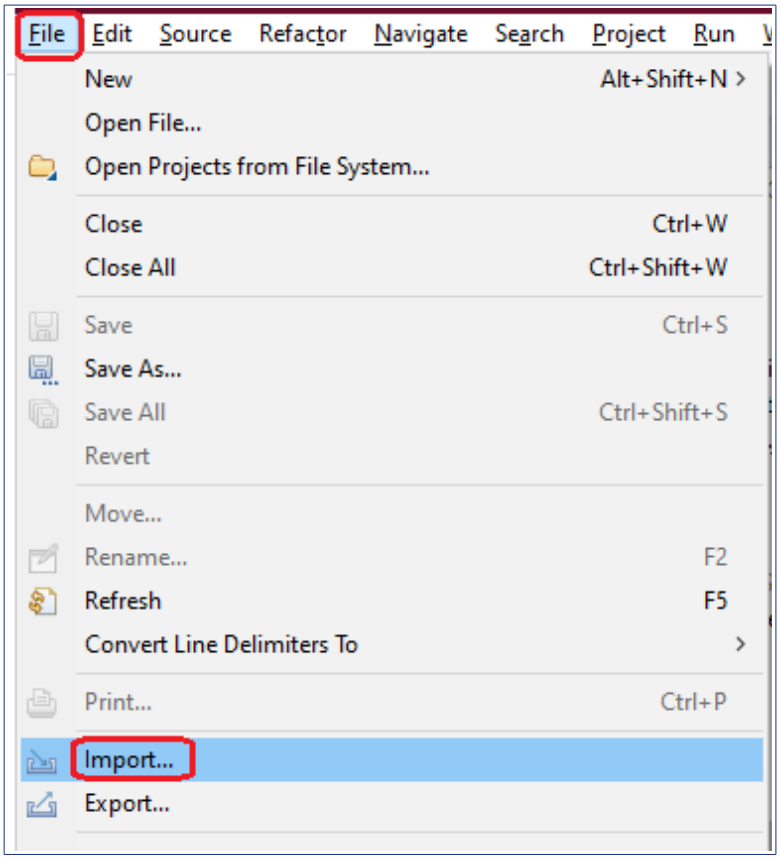
Spring is an enterprise Java framework. It was designed to simplify Java EE development and make developers more productive.

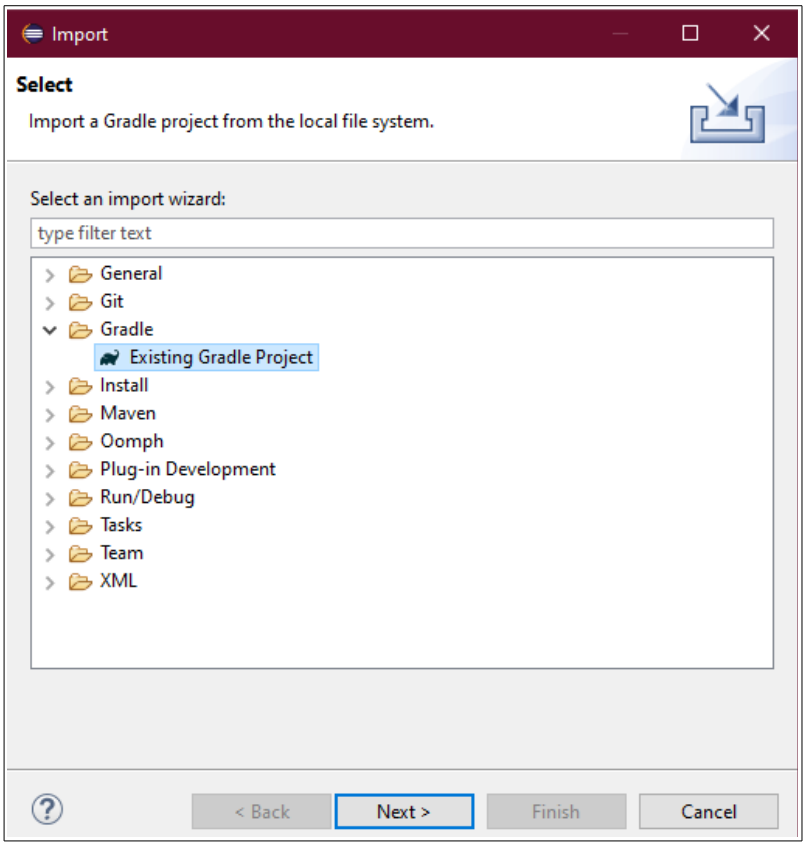
In this case, we are going to generate a Spring boot Project Structure using the spring initializr

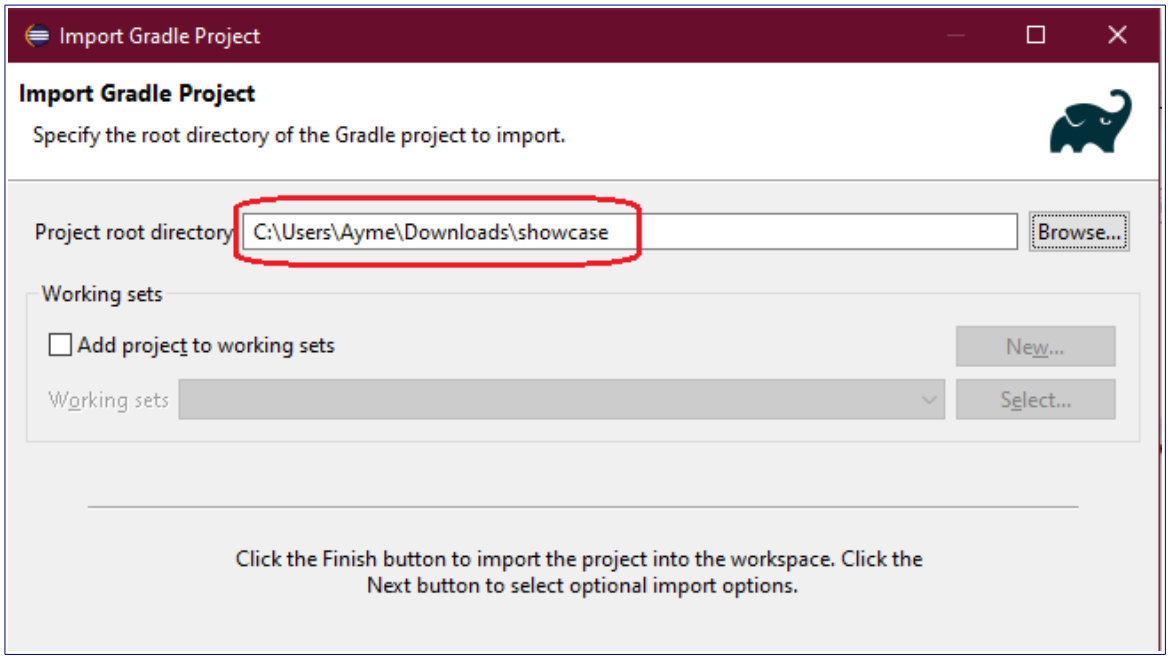


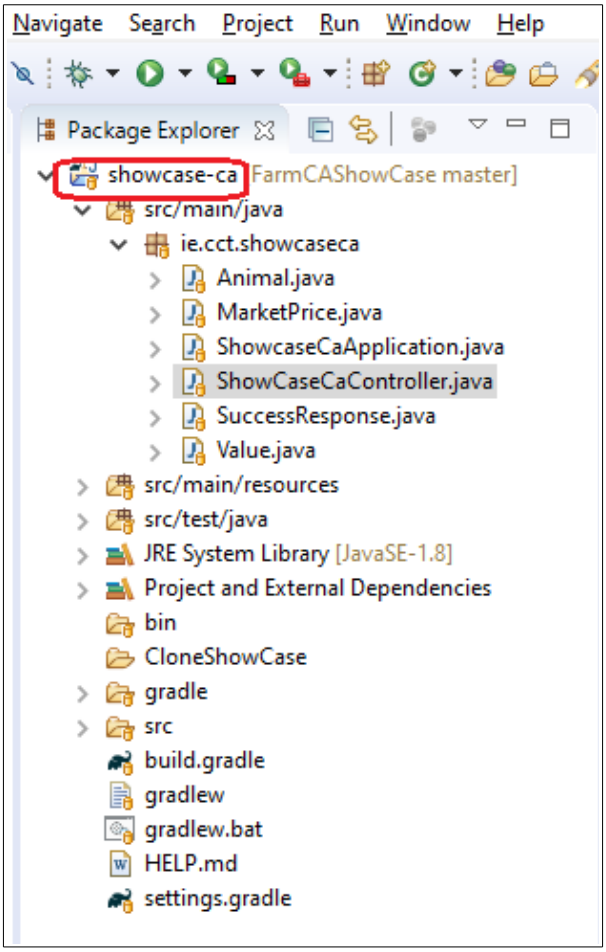
After press GENERATE button to download the folder with the spring package

As a second step, the folder uploaded has to be import in Eclipse, as a Gradle Project


File
Edit
Source
Refactor
Navigate
Search
Project
Run
New
Alt+Shift+N
Open File...
Open Projects from File System...
Close
Ctrl+W
Close All
Ctrl+Shift+W
Save
Ctrl+S
Save As...
Save All
Ctrl+Shift+S
Revert
Move...
Rename...
F2
Refresh
F5
Convert Line Delimiters To
Print...
Ctrl+P
Import...
Export...


Import
Select
Import a Gradle project from the local file system.
Select an import wizard:
type filter text
General
Git
Gradle
Existing Gradle Project
Install
Maven
Oomph
Plug-in Development
Run/Debug
Tasks
Team
XML
< Back
Next >
Finish
Cancel


Import Gradle Project
Import Gradle Project
Specify the root directory of the Gradle project to import.
Project root directory
C:\Users\Ayme\Downloads\showcase
Browse...
Working sets
Add project to working sets
New...
Working sets
Select...
Click the Finish button to import the project into the workspace. Click the
Next button to select optional import options.


Navigate
Search
Project
Run
Window
Help
Package Explorer
showcase-ca [FarmCAShowCase master]
src/main/java
ie.cct.showcaseca
Animal.java
MarketPrice.java
ShowcaseCaApplication.java
ShowCaseCaController.java
SuccessResponse.java
Value.java
src/main/resources
src/test/java
JRE System Library [JavaSE-1.8]
Project and External Dependencies
bin
CloneShowCase
gradle
src
build.gradle
gradlew
gradlew.bat
HELP.md
settings.gradle

2. *Codding*

Declaring variables

The variable ***farmStock*** is a list of animals in which will be stored every data enter by the user.

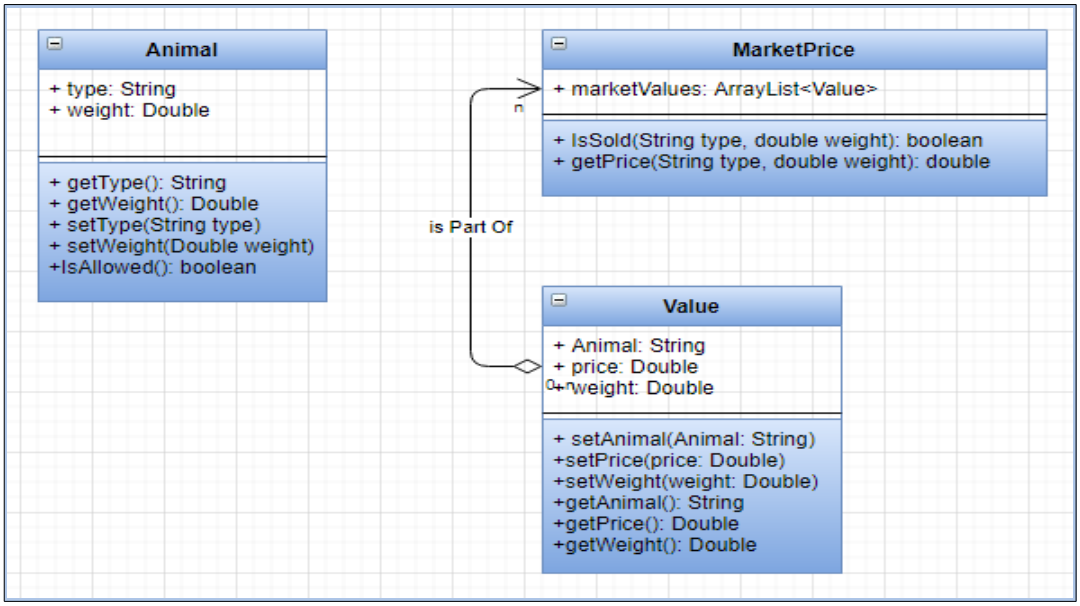
The class ***animal*** has 2 attributes ***type***, to define the type of animal, which will be cow, pig or chicken, only 3 types are allowed, also the ***weight*** of the animal will be stored in this class, About the methods in order to manipulate the class we have the setters and getters

ShowcaseCaApplicationTests.java | ShowCaseCaController.java | Animal.java

```java
package ie.cct.showcaseca;
/*
 * This class contains the animals will be added into the farm stock
 */
public class Animal {

    private String type;
    private double weight;

    public Animal(String type, double weight) {
        this.type = type;
        this.weight = weight;
    }

    public Animal() {
    }

    // Declaring setters and getters
    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(float weight) {
        this.weight = weight;
```

The variable ***priceList*** is the market list of prices, in which every animal according its type and weight

1. Data Structure



ShowcaseCaApplicationTests.java | ShowCaseCaController.java

```java
package ie.cct.showcaseca;

import java.util.ArrayList;

@RestController
public class ShowCaseCaController {

    // Declaring my animal list
    ArrayList<Animal> farmStock;
    // Declaring my list price will define the price of each animal according to the animal weight
    MarketPrice priceList;

    public ShowCaseCaController() {
        farmStock = new ArrayList<Animal>();
        priceList = new MarketPrice();
    }

```

Both ***priceList*** and ***farmStock*** are initialize in the constructor of the class ***ShowCaseController,*** this class is the main class, in which are implemented the methods invoked by the client

@Controller Annotation

Spring MVC provides annotation based approach where you don't need to extend any base class to express request mappings, request input parameters, exception handling, and more. @Controller is similar annotation which mark a class as request handler.

In order to send error or success messages is being use the ***SuccessResponse*** Class

ShowcaseCaApplicationTests.java | ShowCaseCaController.java | Animal.java | SuccessResponse.java

```java
package ie.cct.showcaseca;

public class SuccessResponse {

    private String message;

    public SuccessResponse(String message){
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }


}
```

Adding data

@PostMapping – Handle HTTP POST Requests

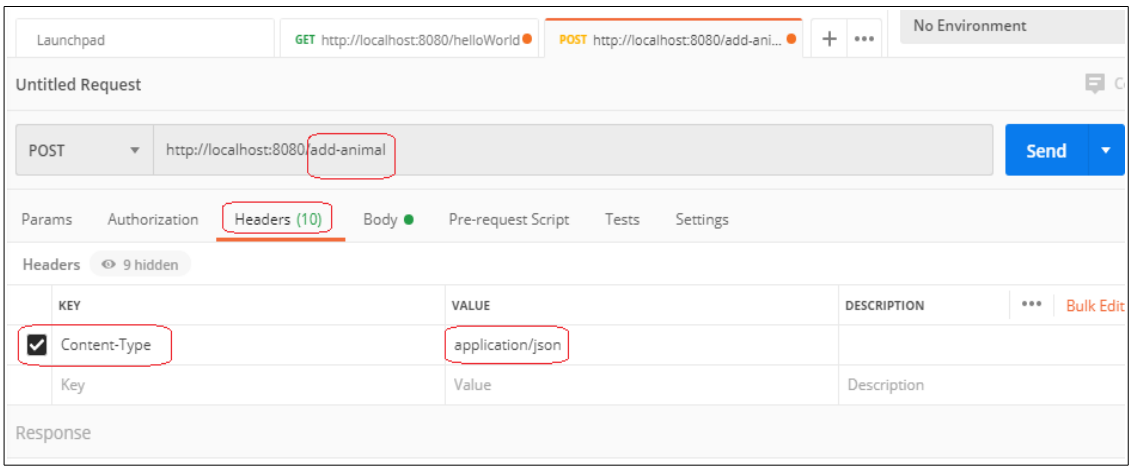
will send and argument RequestBody an animal object

RequestBody: Annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request. Optionally, automatic validation can be applied by annotating the argument with @Valid.

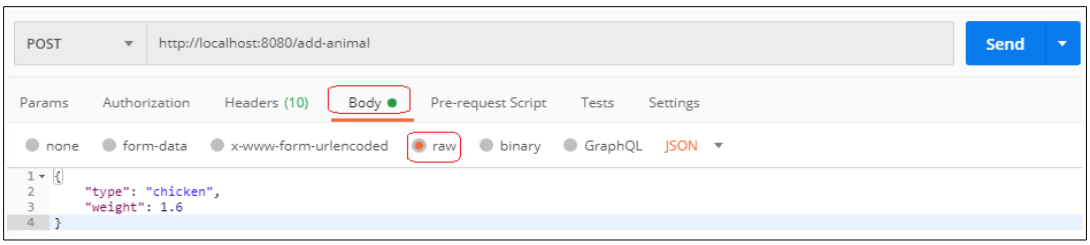
```java
    /*
     *  1.- Add Animal into the list
     */

    @PostMapping("add-animal")
    public SuccessResponse addAnimal(@RequestBody Animal animal) {

        // Validating the data enter by the user, only cow, pig and chicken type are allowed
        if(animal.isAllowed()) {

            farmStock.add(animal);  // Adding animal on the farm stock
            return new SuccessResponse("Animal "+ animal.getType() + ", "+ animal.getWeight()+ " Added");
        }
        else
            return new SuccessResponse("Only pig, cow and chicken are allowed into the farm, please verify type");
    }

```

Testing on the client side will use postman

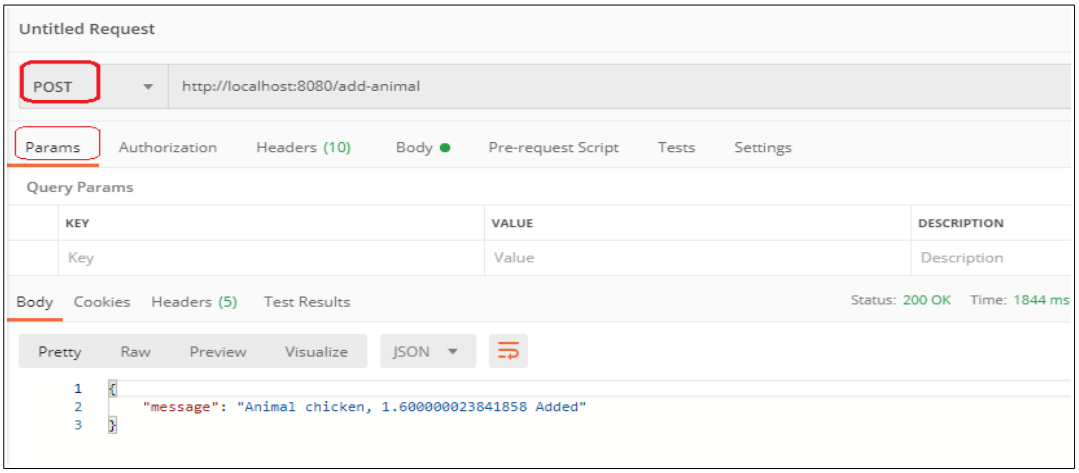
Postman is a popular API client that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses. The result - more efficient and less tedious work.



As the data is a class with 2 attributes we will send them thought JSON format, setting it on Headers option.



In Body option, select raw, and register the data, in this case we will store an Animal type chicken with 1.6 Weight, then press send button.



In Params option well get the success ***successresponse*** message, as the type entered is correct the object is added. Otherwise the response will throw an error

Body Cookies Headers (5) Test Results Status

Pretty Raw Preview Visualize JSON

```
{
    "message": "Only pig, cow and chicken are allowed into the farm, please verify type"
}
```

The following data were added in order to test every functionality

```
{
    "type": "cow",
    "weight": 530
}
{
    "type": "cow",
    "weight": 600
}
{
    "type": "cow",
    "weight": 140
}
{
    "type": "chicken",
    "weight": 0.5
}
{
    "type": "chicken",
    "weight": 1
}
{
    "type": "chicken",
    "weight": 0.9
}
{
    "type": "parrot",
    "weight": 0.30
}
{
    "type": "pig",
    "weight": 300
}
{
    "type": "pig",
    "weight": 250
}
```

There are pigs, cows, and chicken also a parrot just to test the validation, Is not allowed to enter different animals

***The average by type** is:*

localhost:8080/average-by-type?type=chicken

```
{"message":"The average weight of chicken is 0.29999999701976776"}
```

localhost:8080/average-by-type?type=pig

```
{"message":"The average weight of pig is 68.75"}
```

localhost:8080/average-by-type?type=cow

```
{"message":"The average weight of cow is 158.75"}
```

Testing with other type not allowed

localhost:8080/average-by-type?type=parrot

```
{"message":"Only pig, chicken and cow are allowed, please try again"}
```

Animals sold

localhost:8080/animals-sold

```
{"message":"cows: 2 pigs: 2 chicken: 3"}
```

Full price

localhost:8080/full-price-farm

{"message":"The current value of the full farm stock is: 1515.0includes only cows,pigs and chickens"}

@GetMapping – Handle HTTP Get Requests

Average by type

ShowcaseCaApplicationTests.java | ShowCaseCaController.java | Animal.java | SuccessResponse.java

```java

    /*
     *  2.- Calculate the average weight of each type of animal
     */

    @GetMapping("average-by-type")
    public SuccessResponse averageByType(@RequestParam(required=true) String type) {

        // Validating the data entered by the user
        if(!type.equals("cow") && !type.equals("pig") && !type.equals("chicken"))
            return (new SuccessResponse("Only pig, chicken and cow are allowed, please try again"));

        double sumWeight=0; // in this variable will store the sum of each animal weight
        int qty=0; // this variable will count the animals with the type the user entered
        if(farmStock.size() == 0) // when the farm stock is empty
            throw new RuntimeException("No animal in the farm stock"); // throw an exception error

        else {

            for(int i=0;i< farmStock.size();i++) {//Verifying every animal into the farm stock
                // comparing if there are animals into the farm with the same type the user entered
                if(type.equals(farmStock.get(i).getType())) // get a type the user entered from the farm
                    sumWeight += farmStock.get(i).getWeight(); // adding the weight whenever the animal is the s
                    qty++; // getting the amount of animals
                }
            }

            if(qty == 0)// in case of any animal the same type was found
                throw new RuntimeException("No animal "+ type+" in the stock");
            else
                return (new SuccessResponse("The average weight of "+ type + " is "+ sumWeight/qty));
    }
```

Animals can be sold

```java
     * 3.- How many animals of each type can be sold
     */

    @GetMapping("animals-sold")
    public SuccessResponse AnimalsToBeSold() {

        if(farmStock.size() == 0) // when the farm stock is empty
            throw new RuntimeException("No animal in the list");

        // declaring  counters for every type of animal
        int cows =0;
        int pigs = 0;
        int chickens = 0;

        for(int i=0;i < farmStock.size();i++) {

            // checking if the animal has the right weight to be consider on sealing
            if(priceList.isSold(farmStock.get(i).getType(),farmStock.get(i).getWeight())) {// isSold is a function wh

                if("cow".equals(farmStock.get(i).getType())) // detecting the type of animal in order to increase the
                    cows++;
                else if("pig".equals(farmStock.get(i).getType()))
                        pigs++;
                    else if("chicken".equals(farmStock.get(i).getType()))
                            chickens++;

            }
        }
        return (new SuccessResponse("cows: "+cows+" pigs: "+pigs+" chicken: "+chickens));
    }
```

Full price

```java
    /*
     * 4.- What is the current value of the full farm stock: That is, the price of all the animals
that can be sold right now.
     */

    @GetMapping("full-price-farm")
    public SuccessResponse fullValue() {

        if(farmStock.size() == 0) // when the list is empty
            throw new RuntimeException("No animal in the farm");
        // defining counters for every type of animal
        double totalPrice = 0;

        for(int i=0;i < farmStock.size();i++) {
            // checking if the animal has the right weight to be consider on sealing, getPrice event gets the price i
            totalPrice += priceList.getPrice(farmStock.get(i).getType(), farmStock.get(i).getWeight());
        }
        return (new SuccessResponse("The current value of the full farm stock is: "+totalPrice+ "includes only cows,p
    }
```

Current Value

```java
    /*
     * 5.- What is the current value of the farm assuming the price of each animal is set by a
parameter in the HTTP request. This is an example:
- http://localhost:8080/currentValue?cow=350&pig=120&chicken=1
     */
    @GetMapping("currentValue")
    public SuccessResponse fullValueFarm(@RequestParam(required=true) double cow, @RequestParam(required=true) double
        return (new SuccessResponse("The current value of the full farm stock is: "+ (cow+pig+chicken)));
    }

}
```