Restlet Framework: Design of use cases

Javier Moraga Matoque

In this document I explain the steps I took in order to deploy a simple RESTful service using the restlet framework.

The first thing I did was searching information about REST architectural style, and which were the things that made its use interesting. As I read, I understood that its basis are set like this:

* Resources pointed by URI (Uniform resource identifiers) are the main thing in here.
* Resources can be delivered in multiple ways, each of them stands for a new representation of the Resource.
* We are able to use methods that are strictly related with HTTP, and these are GET,PUT,POST or DELETE.
* We can deduce that a REST architectural style stands for a web architectural style in which the elements managed are the resources that we can obtain, store, update or delete from a server. Other behaviors are not directly contemplated.
* So REST basically defines a way of managing web resources.

## 1. Setting up the environment

After this initial research I started working to setup an IDE where I could work on this project, my choice was Eclipse, as recommended in the official restlet page. In order to start working on it I also needed to download the libraries associated to the Restlet framework from <https://restlet.com/open-source/downloads/current/>.

Having this framework downloaded I configured the build path of my project in a way that I had all the “.jar” libraries from the restlet engine that are mandatory for developing my project.

When I had this setup ready I started working in the project, the book “Restlet in action” by Jérome Louver and others was specially useful for me, there I found lots of examples that helped me to develop this Restful API based application.

## 2. Description of the intended application

### 2.1 Contacts service

Knowing that the REST architecture consist on managing resources, my first idea was to implement one application that basically allows the user to manage contacts from a server remotely. This application could be launched in an user agent capable of handling html representations, such as a web browser. In this way, the idea of the application was to be able to retrieve contacts from a restlet server instance runned inside the application, that would have the ability to manage URIs in order to route the request to the correct resources.

Also I wanted to create a restlet client for the application that allowed me to launch PUT and DELETE requests in order to create new contacts or delete them, due to the fact that I found no information about how to launch these requests using the web browser.

Having this initial service implemented I also found out that I was able to make POST requests from inside an HTML page using forms. This allowed me to change the state of my resources remotely.

### 2.2 Smarthome service

In this case I wanted to store a different kind of resources, these would be similar to the contacts because they also store information relating to different instances of a class. But the difference between contacts and these resources is that the data on the server represents the state of a sensor or actuator from a smarthome appliance.

With this service we will be able to manage the state of our smarthome equipment remotely using a normal web browser, the server part will be monitoring and supplying the different states and values for the different appliances and those values will be provided to the user accessing to the REST based web service.

We will still be able to use forms in order to perform “POST” requests using the html form so we can see the interesting part of this service. We can retrieve information about our appliances using a web browser and also we can change the state of them in “real time” by submitting those post requests.

## 3.Development of the application

### 3.1 Contact and Appliance classes

These are normal classes (”Contact.java”,”Appliance.java”) that represent contacts and appliances with their respective attributes. Contacts contain at the same time an attribute of type address that will require to develop another class (“Address.java”) representing objects of this type.

The only thing remarkable about all those classes is that we are talking about classes that need to be passed over a network, so we need their objects to be Serializable.

### 3.2 Contact and Appliance holder classes

Those are the “Appliances.java” and “Contacts.java” files. Both have inside the collection of resources mapped to their identifiers. A singleton pattern is defined in order to get sure that we have only one instance of each running inside our server at the same time. We will also ensure that their lists are correctly concurrently accessed by making all their methods synchronized. The getInstance() class method will return us the instance from where the restlet server can manage the information needed.

I also added a function called inicializa() that will initialize the list with default components to make the testing process quicker. The other methods are there in order to manipulate the lists by adding (or updating) or deleting the lists.

In the appliances case we index the Appliances by using two parameters (which are the identifier of the element and the floor in which we find the element). This adds a little extra complexity to the methods developed because we will have a “list of lists”. First list is indexed by floor names. The list for a given floor is indexed by the identifier of the appliances.

It would be possible to make the contents of these lists persistent by using local storage files to save or delete resources, but in this case I used in-memory resource storage for simplicity.

### 3.3 Resource interface

In my project I created the “ResourceApp.java” interface that will be implemented by all the Restlet resources that are going to be used. Note that above each method we have “@methodType” that indicates which method is represented by the function in the restlet server. With this interface we get sure that all our resources are able to perform the most popular (and needed) REST operations.

For example “@Get” above the method “getResource()” means that the latter stands for the GET operation in the RESTful API.

### 3.4 Restlet resources

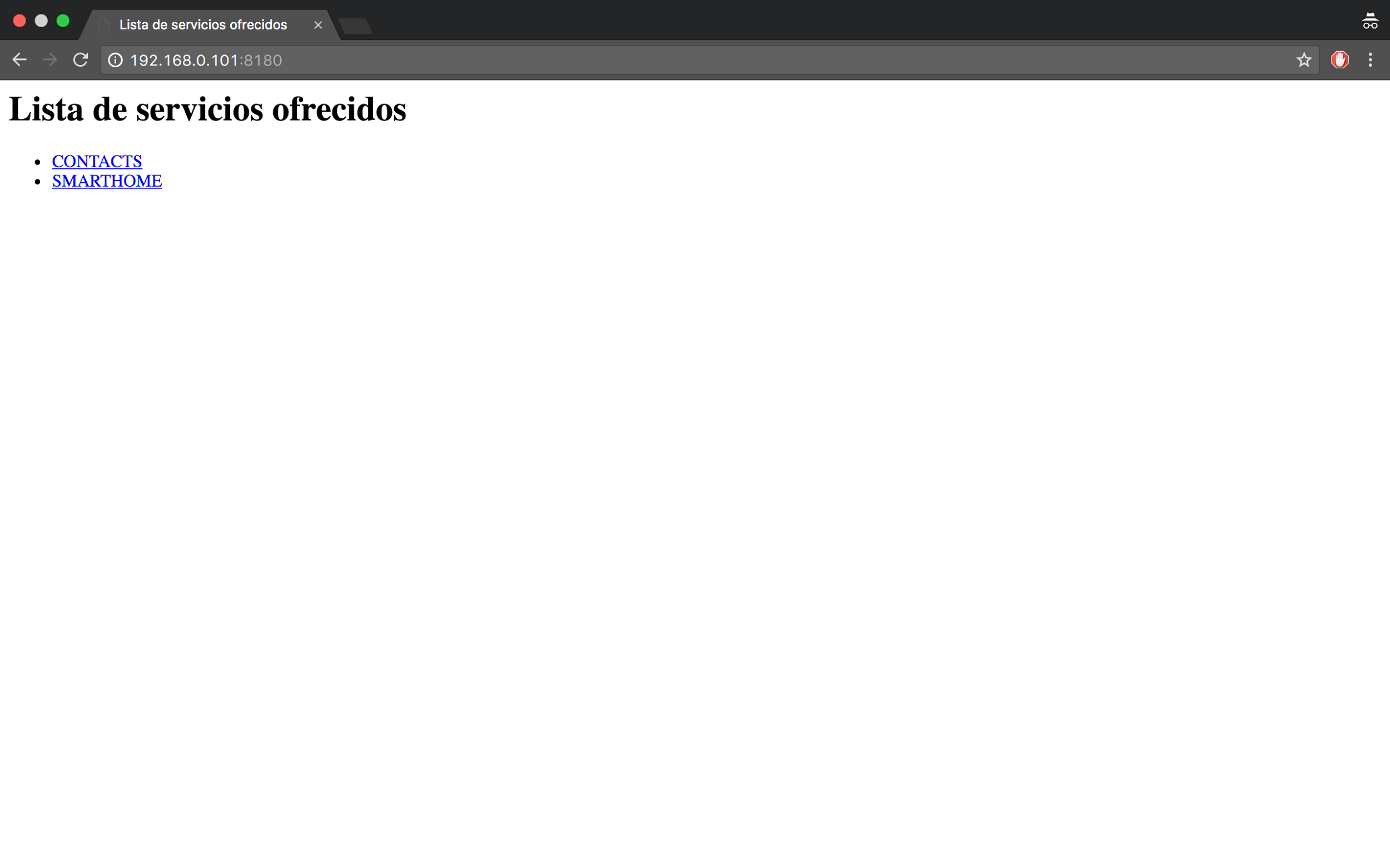
In this section I will explain which are the classes that are triggered whenever a request passes the restlet routing system (that I will cover in the explanation of another class).

All of these classes extend from the restlet “ServerResource” and implement the interface created before that I titled as “ResourceApp”. This allows us to retrieve information that can be useful for our application (such as important fields of the URI, as the identifier).

They all also have one method called “toHTML” that gets an instance of the resource we want to send and it uses it to generate an “html” representation using its data. This is possible due to the use of the extension called “freemarker” and the templates that I had to create in order to get the data represented as I wanted. In those templates I also set the forms needed to perform POST requests.

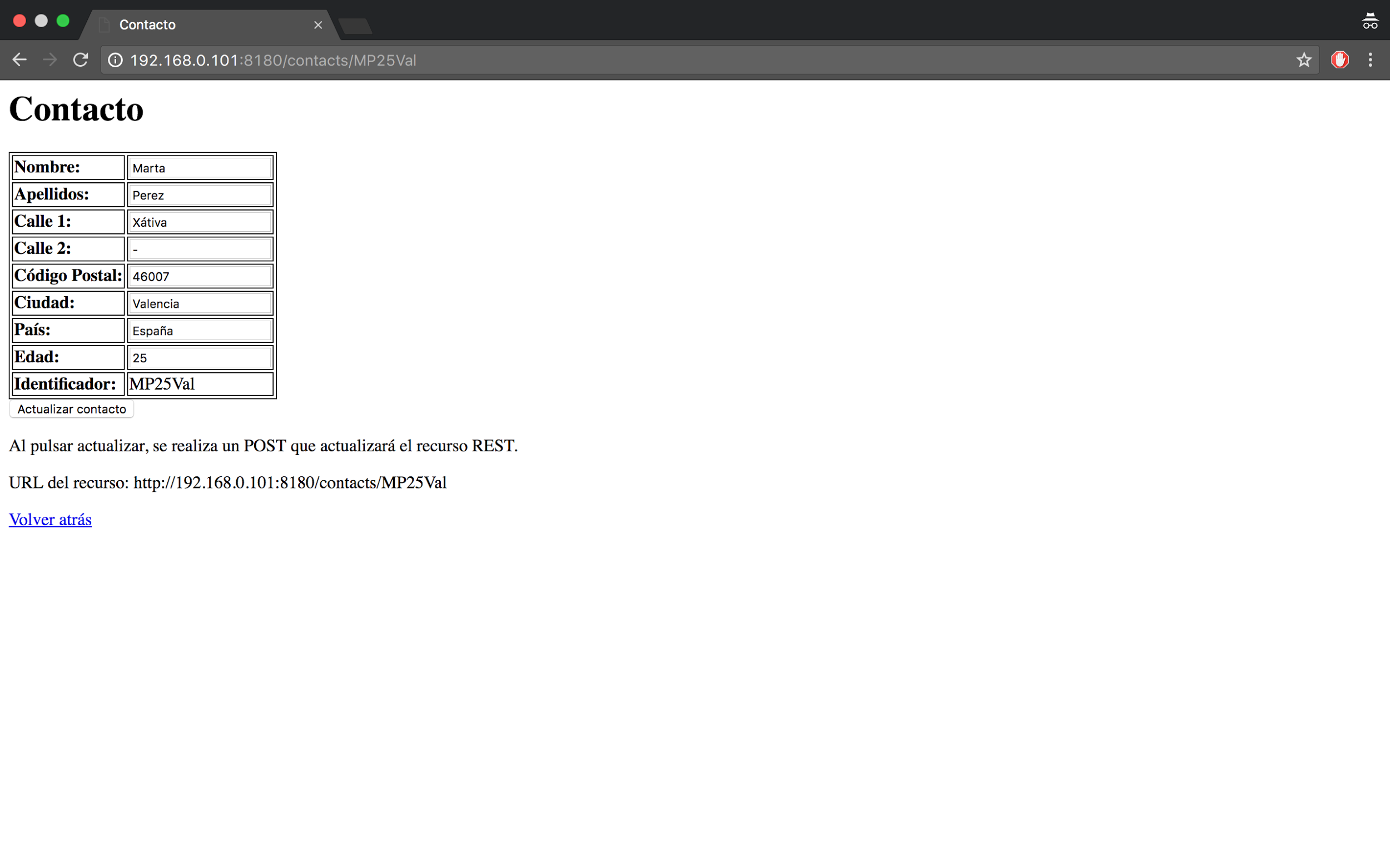
#### 3.4.1 Homepage resource

This resource (set in the file “ServidorPaginaPrincipal.java”) will only have the Get method implemented, that consists in sending the homepage representation that I have previously set in a template accessible from the server application. This homepage has inside the links needed to reach the two services offered.



#### 3.4.2 Contacts

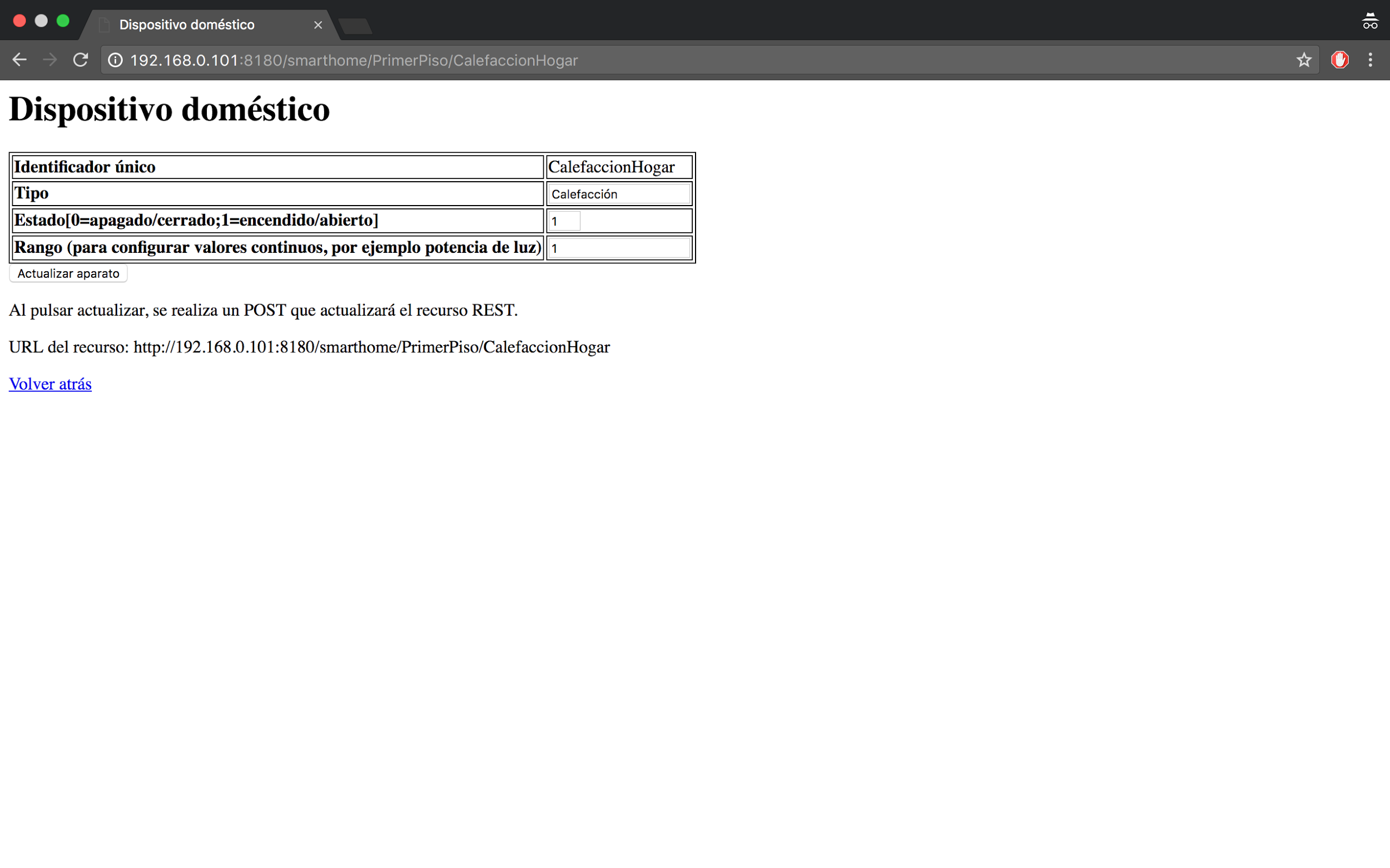
This resource (set in the file “ServidorContactos.java”) has the methods that allow us to retrieve, delete, update or store a contact into our server. Note that in this case we also have one method implemented with a “@Post” above it. This method is called postContacto(Representation entity) and that means that it will be triggered everytime we receive a POST request that updates one of our contacts.



This is the representation retrieved by a web browser for one contact resource. Note that I also added one link that allows you to get back to the contact list. All the fields of the contact are modifiable unless the identifier.

#### 3.4.3 Appliances

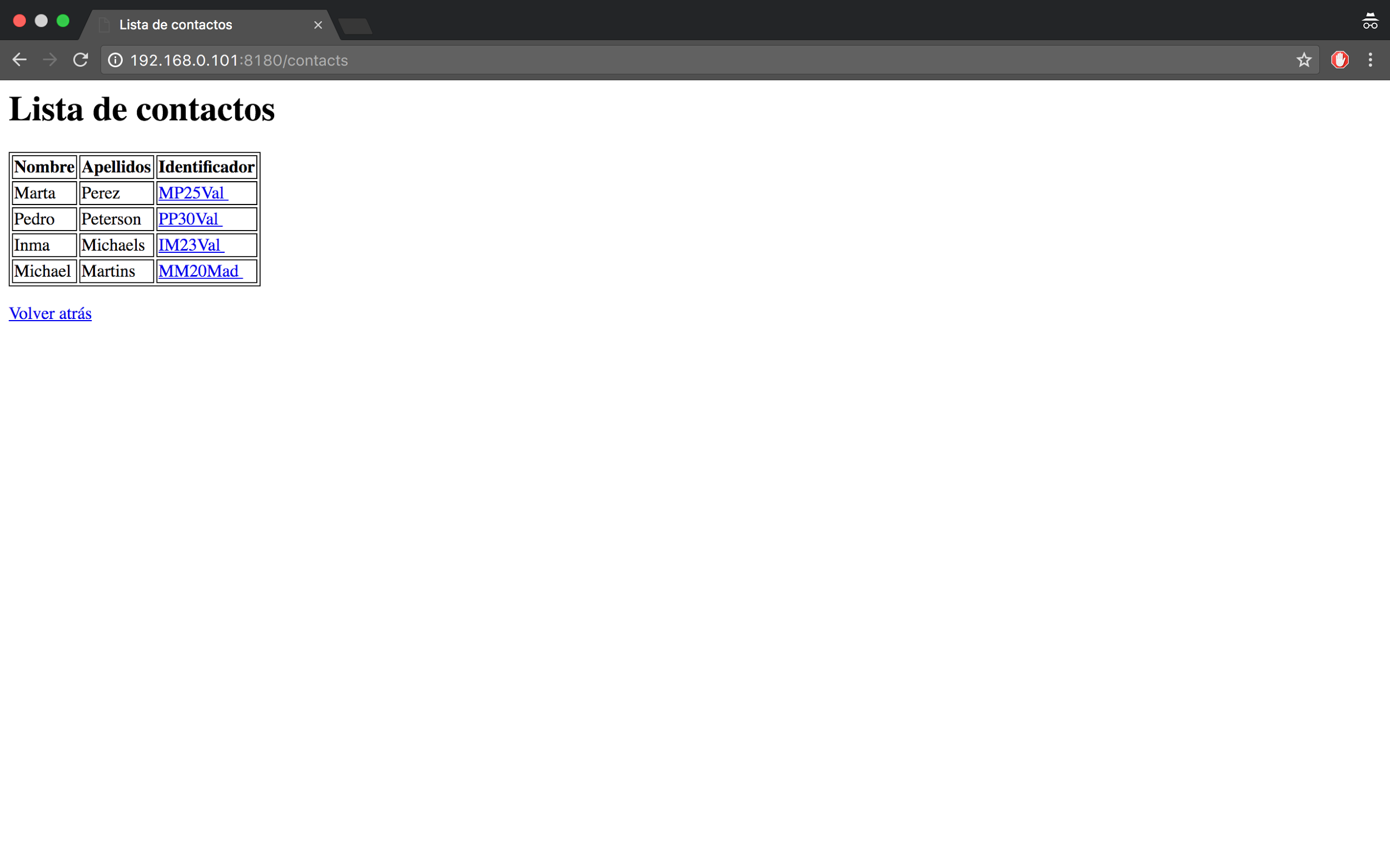
Similarly to the contact resource, this one (set in the file “ServidorAparatos.java”) has the methods that allows us to retrieve, delete, update or store an appliance (given some floor too) into our server. Again, we have a post method that will allow us to change the state of the appliances, and in this case, in a real implementation the server data would be directly connected with the smarthome equipment that it represents, so it would have a “real” effect into the environment.



This is the representation retrieved for a given appliance. As we can see in the URI this one is placed in the first floor. Again, its identifier remains unmodifiable.

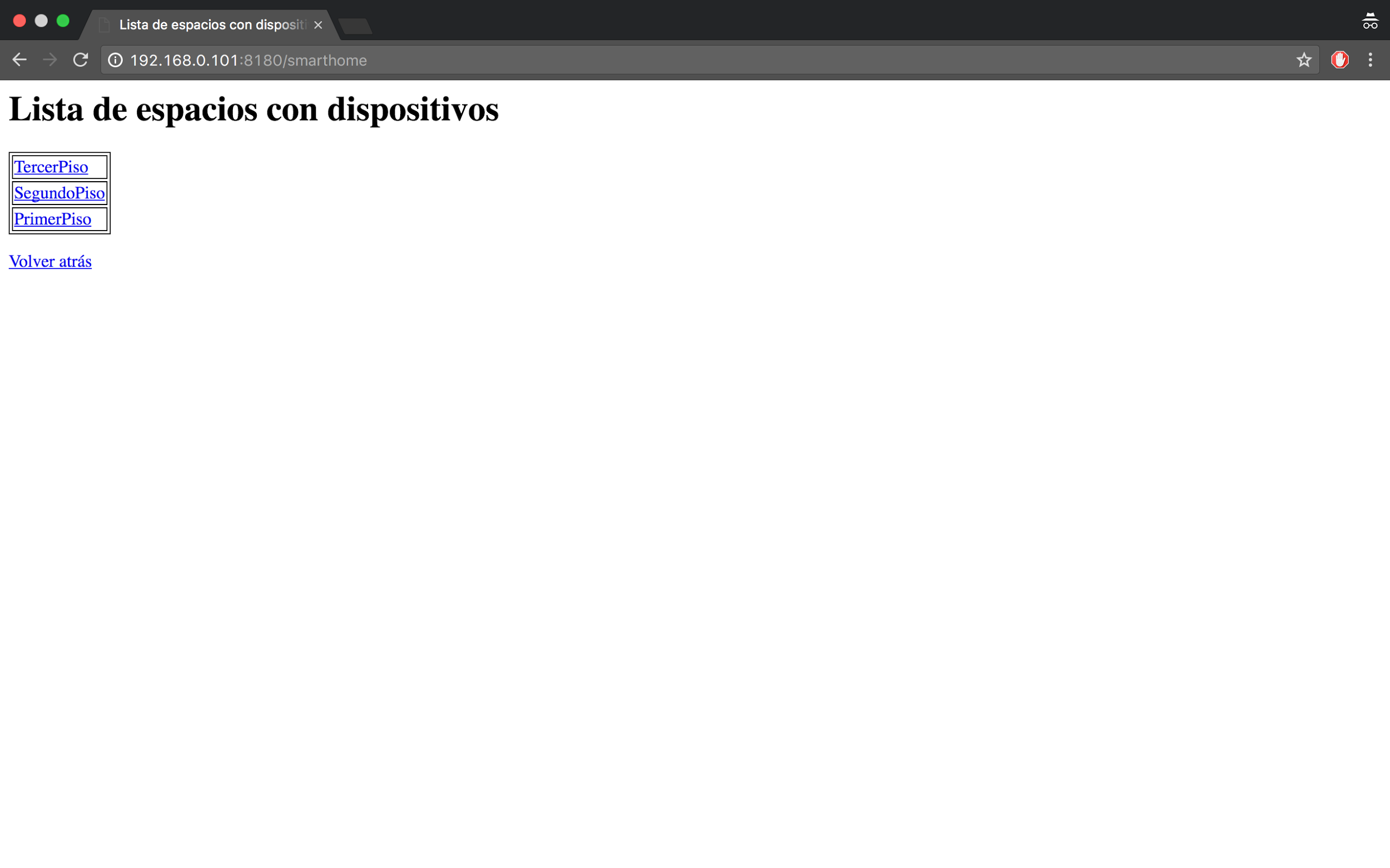
#### 3.4.4 Contact list

This resource is only capable of responding to a get request, in which we find the html representation that shows a list of the current contacts stored in the system.



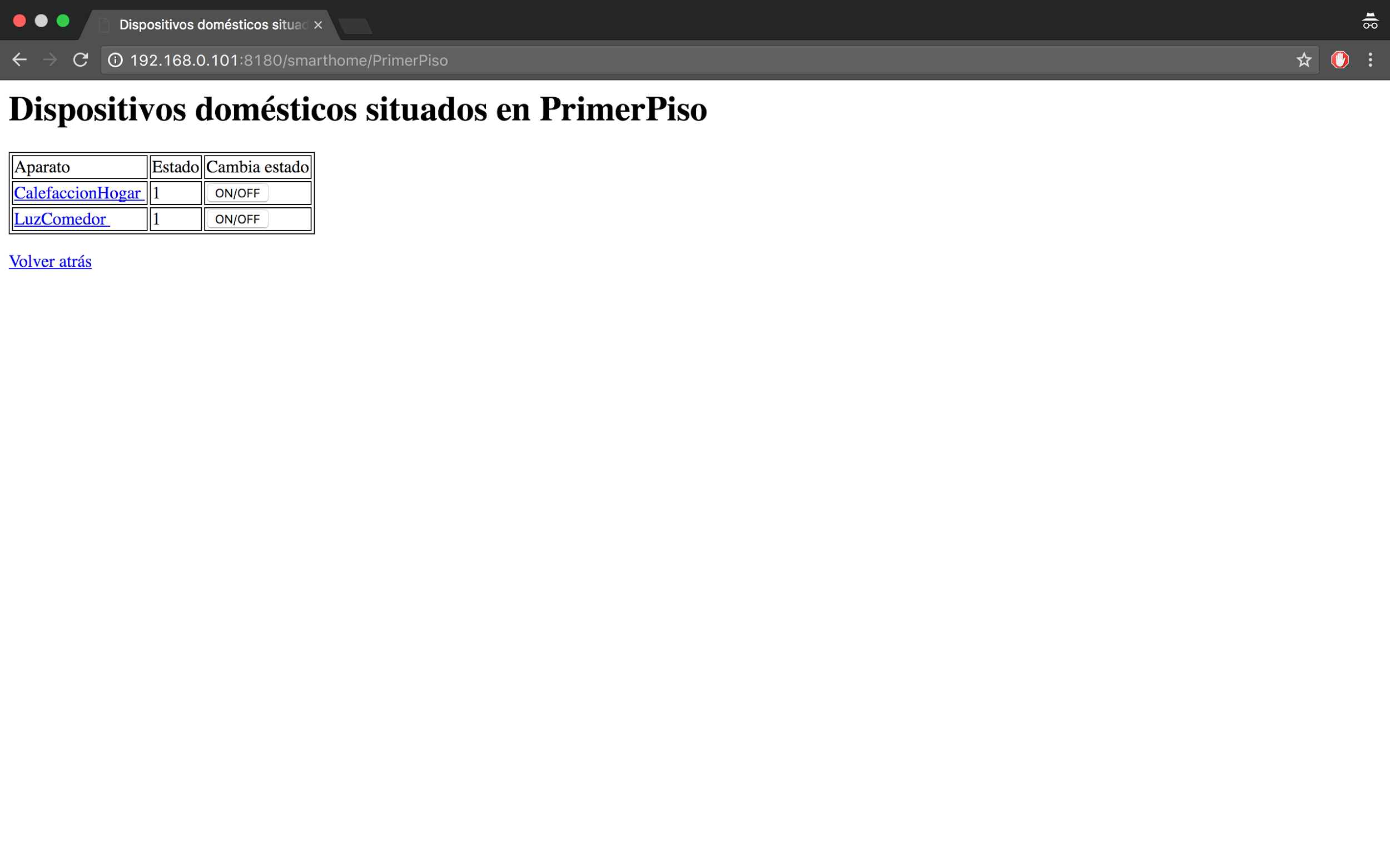
#### 3.4.5 Floor list

Similarly to the contact list resource, with this resource (placed at file “ServidorListaPlantas.java”) we will be only capable to respond to a get request. The response given will contain the html representation corresponding to the floors that have any appliance as a resource of our system.



#### 3.4.6 List of appliances on a given floor

Again with this resource (placed at file “ServidorListaAparatosPlanta.java”) we will be only able to respond to a get request. The response will contain the html representation for the current appliances on a given floor that are inside our restlet system.



### 3.5 Restlet main application

This is the application that starts the execution flow for our server program, it will request for a port where to bound the restlet service, as the host address will be automatically gotten by the use of “Inet4Address.getLocalHost().getAddress()” method.

Once we have this started and that data obtained it will start a restlet server bounded to our “host:port” and it will show the result by using standard output. Looking at this we are able to see at which initial URI we have to point. For example: server prompts “Restlet server initiated in: 192.168.0.101:8180” and we are able to reach it at “<http://192.168.0.101:8180>” because I set the routing system to respond to that URI.

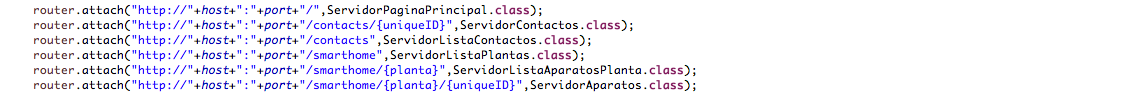
Testing has been only made in a local network but it should be doable also between two machines that can reach each other over the Internet by making some changes in the way that the server obtains its address.

After having set the server parameters and started it, I initialize the list of contacts and appliances to have some set of initial resources in the system.

In this class we have the method “createInboundRoot()” that will set the behaviour of our restlet server. We can implement different things in there (mostly with filtering or security purposes, such as access control, privacy of data...) but in my case I just set a routing system and also an authentication process.

#### 3.5.1 Routing

This restlet will be in charge of redirecting requests to their appropriate resources.

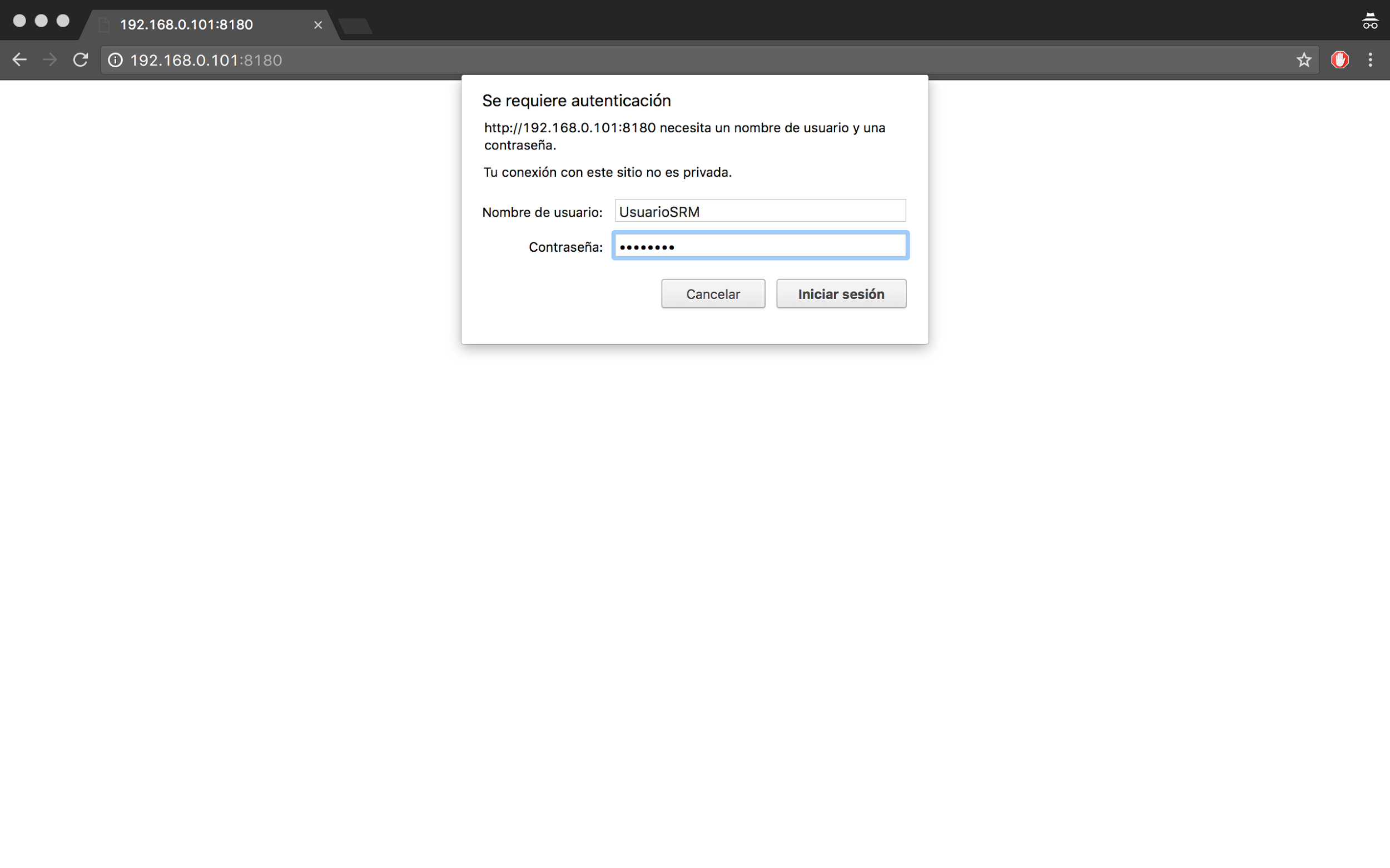


As we can see in the image, we have pointed all the possible URIs in our system to their corresponding resources. Note that the parts that are between {} symbols are variables that can be retrieved from the requests. With this simple system we will associate the URIs to the resources of our system.

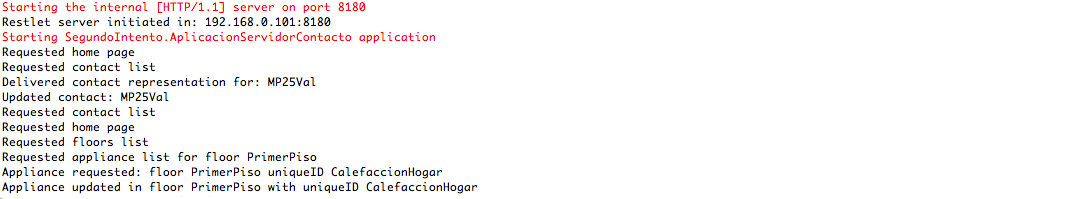
#### 3.5.2 Authenticator

Before being granted with access to the resources we will need to introduce some credentials that are set inside our server. We use “ChallengeAuthenticator“ class to set an authenticator that in my case will use an “HTTP\_basic” scheme. The credentials I have set are “UsuarioSRM” as the login and “password” as the password.

In the following image we can see what happens now when you try to access the system.



#### 3.5.3 Server logging



As we can see in this image the implementation made prompts every action done with the resources.

### 3.6 Restlet client

In the file “ClienteRestlet.java” I created one application that allows us to store or delete new resources. It has to receive the correct parameters in order to introduce new resources inside the system.

As I have set an authenticator in the server part I need to create a “ChallengeResponse” in order to authenticate in front of it. The credentials needed will be requested using the Console methods called readLine() and readPassword(). The last one won’t show what you are typing on console. If the authentication is successful and the parameters are correct the application will perform the requested action. I exported this client in a “ClienteRestlet.jar” file.

## 4. Execution flow

I will present now the steps performed to get this system working. I have generated two .jar files corresponding to the client part (able to store or delete resources) and also the server part.

1. Start the server part using “java -jar servidorRestlet.jar <port>”.
2. Access from your web browser to “http://<host:portPromptedInServerPart>”.
3. Authenticate with the username “UsuarioSRM” and password “password”.
4. Navigate by the system resources and try to modify them.
5. Use the client part correctly to put or delete a resource.
   1. You will need to authenticate to perform this.
   2. You should be able to see the result in your web browser.

I attach a video link where you can see all of this being used: [Use of this application](https://drive.google.com/open?id=0B9T1_eE3-EUNdV84SWJIb1Z6VUk)

Although in this video the server is accessed from the same machine, I tested it with other hosts that were able to communicate in the same network and It worked as well.

## 5. Conclusions

The use of this framework can produce interesting applications. Nevertheless there exist web sites such as <https://studio.restlet.com> that are intended to simplify the process of developing a restful API, saving a lot of effort from the developers. I will try to implement similar features with that facility in order to compare the effort made to develop them.