

Spring API Rest MVC with Authentication

Design and Testing II

Group 6

Ángel Delgado Luna

Belen Garrido López

Ezequiel Portillo Jurado

Alejandro Rodríguez Díaz

M^a de Gracia Piñero Pastor

Índice

Tabla de contenido

<i>Foundations</i>	3
<i>Security</i>	4
<i>CRUD Actions</i>	5
Create	6
Edit/Get.....	7
List	8
Delete	9
<i>Bibliography</i>	10

Foundations

Nowadays, we can find a lot of platforms for data consuming. An example, google calendar. Google calendar has an API (Application Programming Interface) which lets users to consumes its calendars in any application, taking into account that you must have an account previously.

We must take into account that using a secure API will allow developers to build a comfortable architecture without entering on important details for each implementation. JSON is a data format which helps us to make this action; because, we only have to take a model that is going to be used later by our application.

In this example, we have chosen social profiles entity from any user. Let's see a short schema:

```
private String  nick;  
private String  socialNetworkName;  
private String  link;  
private Actor   actor;
```

```
[  
  {  
    "id": 716,  
    "version": 0,  
    "nick": "DP1010",  
    "socialNetworkName": "LinkedIn",  
    "link": "https://www.linkedin/DP1010"  
  },  
  {  
    "id": 717,  
    "version": 0,  
    "nick": "DP1011",  
    "socialNetworkName": "Tuenti",  
    "link": "https://www.tuenti/DP1011"  
  }  
]
```

According to this, we are going to generate our own api with this entity. Let's move on the next items from this document!

Maven Dependencies

Dependency 1 – Jackson (New on the project)

```
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-databind</artifactId>  
  <version>2.5.3</version>  
</dependency>
```

Dependency 2 – Spring Dependencies (Core – WebMVC – ORM)

Security

The first step we must do, it's to configure a security system for users login. In our case, we have chosen a basic authentication: username and password (encrypted with MD5). Having a look to the package structure on the project attached to this deliverable, you are supposed to see two packages called "api" and "api_security"; on this item we will focus on the second named.

Api_security packaged contains a class named as "SecurityConfiguration" which extends from "WebSecurityConfigurerAdapter".

```
package api_security;

import org.springframework.beans.factory.annotation.Autowired;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

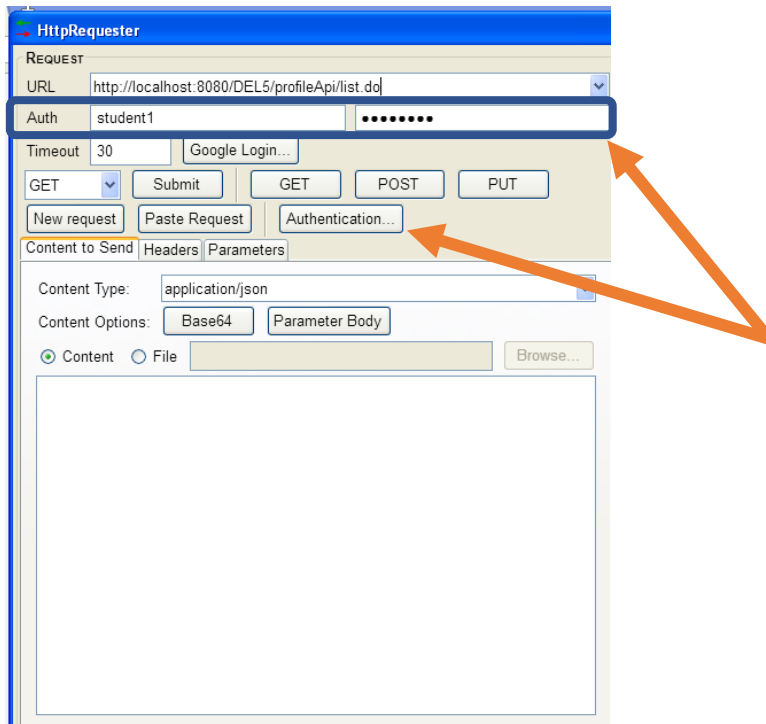
    @Autowired
    private LoginService serviceLogin;

    public void configureGlobalSecurity(final AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(this.serviceLogin).passwordEncoder(new Md5PasswordEncoder());
    }

    @Override
    protected void configure(final HttpSecurity http) throws Exception {
        http.authorizeRequests().anyRequest().authenticated().and().httpBasic().and().csrf().disable();
    }
}
```

The first method call "configureGlobalSecurity" receives as parameter an object of authentication manager. This will let us to build the authentication in our system according to the parameters receive from LoginService class. This class extends from UserDetailsService (native from Spring) which gives us the main parameter to make this configuration (username and password). As indicated, a MD5PasswordEncoder instance is given as password encoder for the auth.

In the second method, we are saying that every http call must be authenticated if it is required.



CRUD Actions

In the first package called “api”, we can find our entity configured for external use. Let see first some important annotations:

- **@RestController**: With this annotation we are indicating to spring that this is a controller which uses a json data structure.
- **@ResponseBody**: The returned items will be given as JSON data.
- **@RequestBody**: The post data will have a json data structure.
- **@JsonIgnore**: Profile has an actor attribute. When we are working with social profiles, we do not need any actor data, as consequence we want that spring ignores this parameter. (Use in java model above the actor get method)

Create

1. We first make the json structure necessary.
2. We introduce as url to post: <http://localhost:8080/DEL5/profileApi/save.do>
3. We click on submit changing its param to POST action

The screenshot displays the HttpRequester application interface. The 'Request' tab is active, showing a POST request to `http://localhost:8080/DEL5/profileApi/save.do` with authentication 'student1' and a timeout of 30 seconds. The 'Content to Send' section shows the request body as a JSON object:

```
{
  "id": 0,
  "version": 0,
  "nick": "dptuenti1",
  "socialNetworkName": "Tuenti2",
  "link": "https://www.google.es"
}
```

. The 'Response' tab shows a successful 200 OK status with the response body:

```
{
  "id": 98304,
  "version": 0,
  "nick": "dptuenti1",
  "socialNetworkName": "Tuenti2",
  "link": "https://www.google.es"
}
```

. The 'Headers' section lists: Server: Apache-Coyote/1.1, Content-Type: application/json, Transfer-Encoding: chunked, and Date: Sun, 26 May 2019 14:02:09 GMT. The 'History' table at the bottom shows two requests: a successful POST and a failed POST.

Request	Response	Date	Size	Time
POST http://localhost:8080/DEL5/profileApi/save.do	200 OK	May 26 2019 - 4:02:06 PM	104 B	2430 ms
POST http://localhost:8080/DEL5/profileApi/save.do	400 Bad Requ...	May 26 2019 - 4:01:51 PM	968 B	3196 ms

Edit/Get

1. We introduce the following url:
<http://localhost:8080/DEL5/profileApi/get.do?id=98304> in order to get the object with the id requested (In our example, the one created on the item before). Id = 98304.
2. We submit the request taking into account that we must use GET HTTP Request.

The screenshot shows the HttpRequester application interface. The 'REQUEST' tab is active, displaying the URL `http://localhost:8080/DEL5/profileApi/get.do?id=98304` and the authentication 'student1'. The 'RESPONSE' tab shows a successful GET request with status '200 OK' and a JSON response: `{"id":98304,"version":0,"nick":"dptuenti1","socialNetworkName":"Tuenti2","link":"https://www.google.es"}`. The 'HEADERS' section shows 'Server: Apache-Coyote/1.1', 'Content-Type: application/json', 'Transfer-Encoding: chunked', and 'Date: Sun, 26 May 2019 14:05:54 GMT'. The 'History' table at the bottom lists three requests: a GET request for id=98304, a POST request for save.do, and another POST request for save.do.

Request	Response	Date	Size	Time
GET http://localhost:8080/DEL5/profileApi/get.do?id=98304	200 OK	May 26 2019 - 4:05:53 PM	104 B	772 ms
POST http://localhost:8080/DEL5/profileApi/save.do	200 OK	May 26 2019 - 4:02:06 PM	104 B	2430 ms
POST http://localhost:8080/DEL5/profileApi/save.do	400 Bad Requ...	May 26 2019 - 4:01:51 PM	968 B	3196 ms

List

1. We introduce the following url: `http://localhost:8080/DEL5/profileApi/list.do` in order to get the profiles from the user logged.
2. We submit the request taking into account that we must use GET HTTP Request.

The screenshot displays the HttpRequester application interface. The 'Request' tab is active, showing a GET request to `http://localhost:8080/DEL5/profileApi/list.do` with authentication 'admin1'. The 'Response' tab shows a successful 200 OK status with a JSON response containing two user profiles. The 'Headers' section shows the server is Apache-Coyote/1.1 and the content type is application/json. The 'History' tab at the bottom shows a list of recent requests and responses.

Request

URL: `http://localhost:8080/DEL5/profileApi/list.do`
Auth: `admin1`
Timeout: 30
Method: GET
Content Type: `application/json`
Content Options: Base64, Parameter Body

Response

GET on `http://localhost:8080/DEL5/profileApi/list.do`
Status: 200 OK
Pretty format

```
[
  {
    "id": 716,
    "version": 2,
    "nick": "QUETAL",
    "socialNetworkName": "LinkedIn",
    "link": "https://www.linkedin.com/DPHOLA"
  },
  {
    "id": 717,
    "version": 0,
    "nick": "QUETAL2",
    "socialNetworkName": "Facebook",
    "link": "https://www.linkedin.com/DPHOLA"
  }
]
```

HEADERS

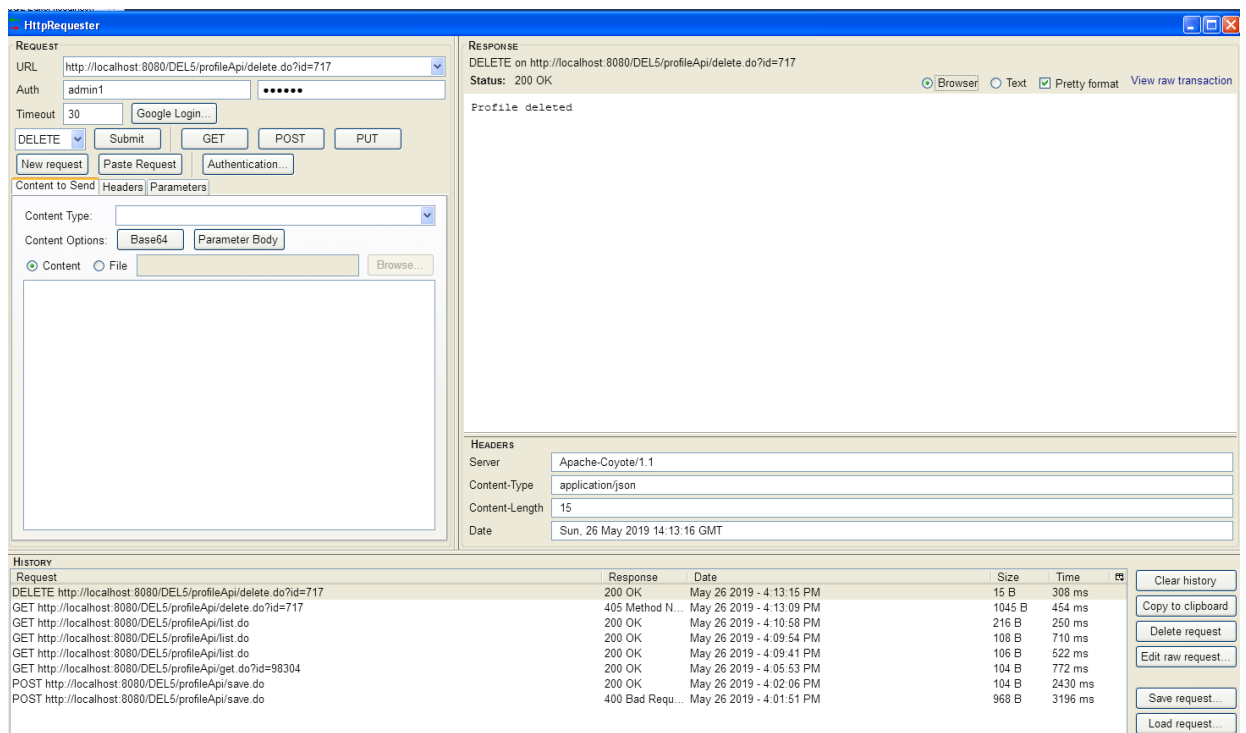
Server: Apache-Coyote/1.1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 26 May 2019 14:10:58 GMT

HISTORY

Request	Response	Date	Size	Time
GET <code>http://localhost:8080/DEL5/profileApi/list.do</code>	200 OK	May 26 2019 - 4:10:58 PM	216 B	250 ms
GET <code>http://localhost:8080/DEL5/profileApi/list.do</code>	200 OK	May 26 2019 - 4:09:54 PM	108 B	710 ms
GET <code>http://localhost:8080/DEL5/profileApi/list.do</code>	200 OK	May 26 2019 - 4:09:41 PM	106 B	522 ms
GET <code>http://localhost:8080/DEL5/profileApi/get.do?id=98304</code>	200 OK	May 26 2019 - 4:05:53 PM	104 B	772 ms
POST <code>http://localhost:8080/DEL5/profileApi/save.do</code>	200 OK	May 26 2019 - 4:02:06 PM	104 B	2430 ms
POST <code>http://localhost:8080/DEL5/profileApi/save.do</code>	400 Bad Requ...	May 26 2019 - 4:01:51 PM	968 B	3196 ms

Delete




1. We introduce the following url:
<http://localhost:8080/DEL5/profileApi/delete.do?id=717> in order to delete the profile with the id indicated.
2. We submit the request taking into account that we must use DELETE HTTP Request.



Before

Filter: <input type="text"/>						
Edit:						
File: Autosize:						
	id	version	link	nick	social_network_name	actor
▶	716	2	https://www.linkedin/DPHOLA	QUETAL	LinkedIn	709
	717	0	https://www.linkedin/DPHOLA	QUETAL2	Facebook	709
	98304	0	https://www.google.es	dptuenti1	Tuenti2	710
*	NULL	NULL	NULL	NULL	NULL	NULL

After

Filter: <input type="text"/>						
Edit: 						
File: 						
Autosize: 						
	id	version	link	nick	social_network_name	actor
▶	716	2	https://www.linkedin/DPHOLA	QUETAL	LinkedIn	709
	98304	0	https://www.google.es	dptuenti1	Tuenti2	710
*	NULL	NULL	NULL	NULL	NULL	NULL

Bibliography

- [1] <https://stackoverflow.com/questions/29838960/how-to-dynamically-remove-fields-from-a-json-response>
- [2] <https://medium.com/@ziateonlyone/spring-rest-api-part-3-spring-security-basic-authentication-3fd20342745b>