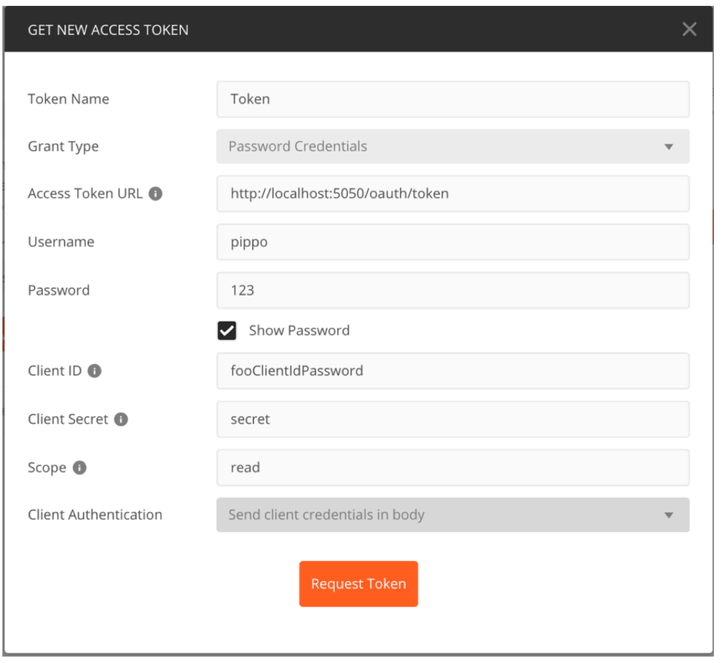
Send login credentials via JSON (Password grant) - Authorization Server

In the doFilter method, within JsonToUrlEncodedAuthenticationFilter, we rely on the request field, which is private, in order to convert the json object sent from our client. This scenario is what we tipically have when we send an Ajax call where a JSON object is sent. To do so we use reflection and set the field as accessible.
The request field at this stage should contain data in the form of a hashmap instead of JSON data. The reason behind is that the Oauth2 original standard requires to use application/x-www-form-urlencoded and Spring Security does not allow other possibilities contrary to Postman for example. At this stage the urlencoded data in the request are expected to be in the form of a HashMap. Therefore what we need to do is to convert the data from JSON (as sent by our Javascript client) to HashMap.

```
    @Override
    @SneakyThrows
    public void doFilter(ServletRequest request, ServletResponse response, FilterChai
n chain) {
        Field f = request.getClass().getDeclaredField("request");
        f.setAccessible(true);
        Request realRequest = (Request) f.get(request);

       //Request content type without spaces (inner spaces matter)
       //trim deletes spaces only at the beginning and at the end of the string
        String contentType = realRequest.getContentType().toLowerCase().chars()
                .mapToObj(c -> String.valueOf((char) c))
                .filter(x->!x.equals(" "))
                .collect(Collectors.joining());

        if ((contentType.equals(MediaType.APPLICATION_JSON_UTF8_VALUE.toLowerCase())|
|
                contentType.equals(MediaType.APPLICATION_JSON_VALUE.toLowerCase()))
                        && Objects.equals((realRequest).getServletPath(), "/oauth/tok
en")) {

            InputStream is = realRequest.getInputStream();
            try (BufferedReader br = new BufferedReader(new InputStreamReader(is), 16
384)) {
                String json = br.lines()
                        .collect(Collectors.joining(System.lineSeparator()));
                HashMap<String, String> result = mapper.readValue(json, HashMap.class
);
                HashMap<String, String[]> r = new HashMap<>();

                for (String key : result.keySet()) {
                    String[] val = new String[1];
                    val[0] = result.get(key);
                    r.put(key, val);
                }
                String[] val = new String[1];
                val[0] = (realRequest).getMethod();
                r.put("_method", val);

                HttpServletRequest s = new MyServletRequestWrapper(((HttpServletReque
st) request), r);
                chain.doFilter(s, response);
            }

        } else {
            chain.doFilter(request, response);
        }
    }
```

All of those cerimonies allow us to send some data inside the body field of our request object with application/json content type instead of application/x-www-form-urlencoded.
This implies that I will be able to send this data more conveniently when using Javascript as I will be able to use the http client of choice (for example fetch) in straightforward menner, meaning I won't have to use hidden form inputs to hold the contents of my object.



Example of Oauth2 Password grant with Postman