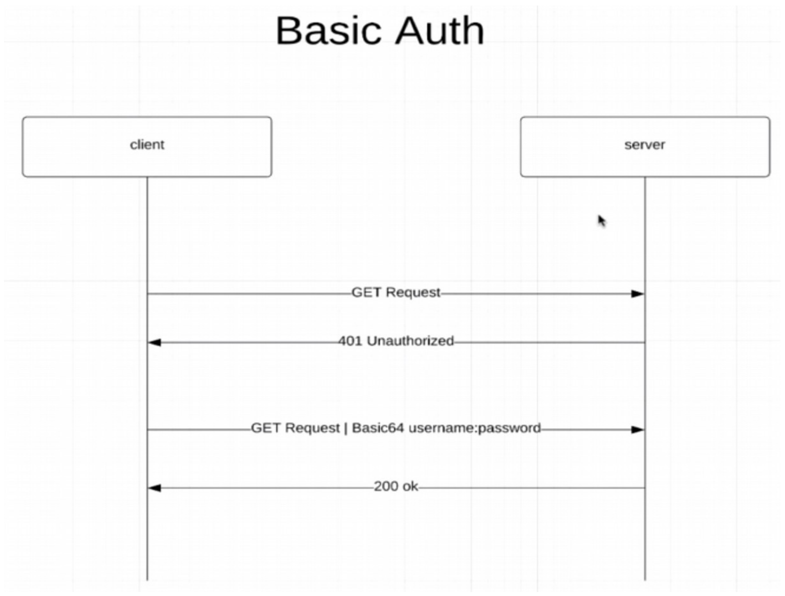
TIPS SPRING SECURITY

- Basic Auth

- By default Spring Security implements Form Base Authentication.

- With Basic Auth if you just send a request, you will get a 401 Unauthorized because with Basic Auth you need to specify the username and password inside the request header as Base64. Then the server will do some validations as if the username does exist and then checks the password and if everything is right, the server will send a 200 ok.

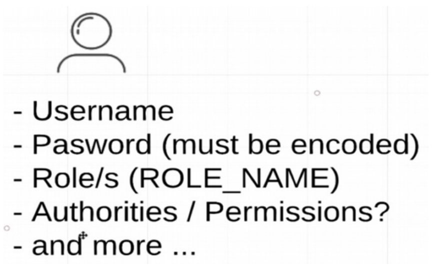
- The client has to send every single time the username and password so it's mostly used for accessing external APIs.



- With Basic Auth there is no way to log out as with Form Base Authentication in which I write http://localhost:8080/logout because username and password is sent on every single request and the server has to validate if them are correct.

- antMatchers is used to allow the access to certain paths.

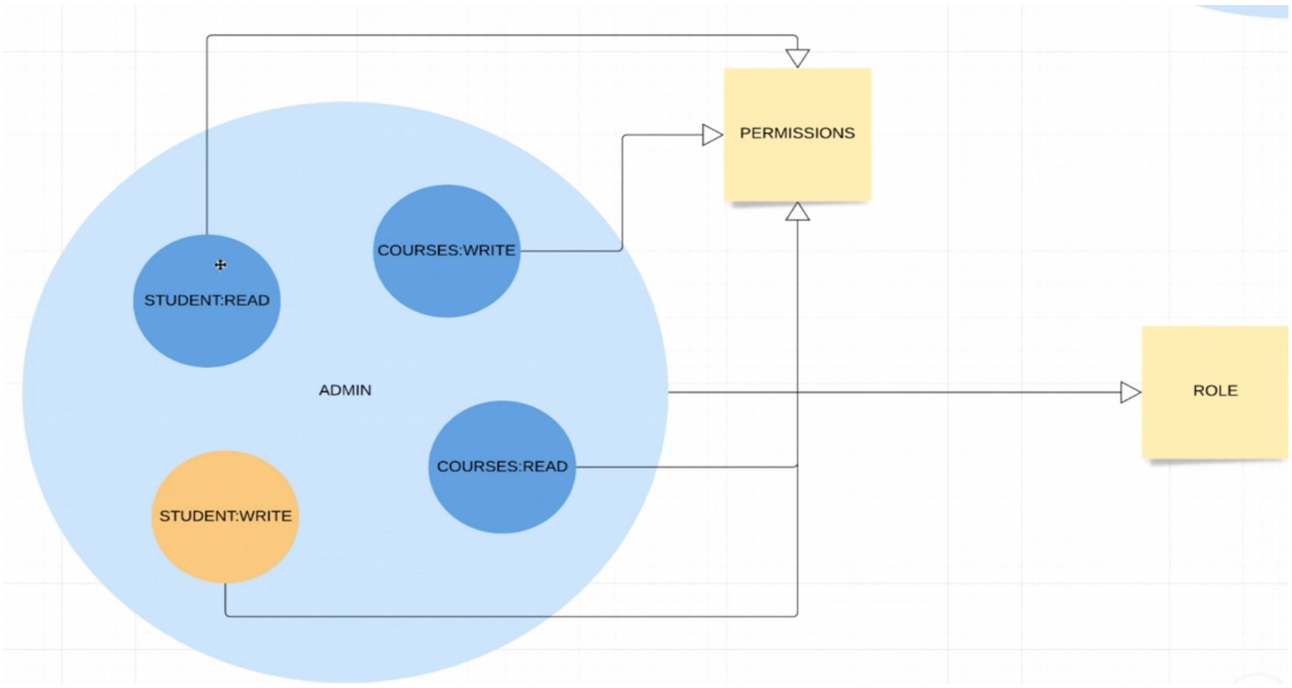
- A custom user must have a Username (must be unique); Password; Role/s (ROLE_NAME); Authorities (or Permissions); etc.

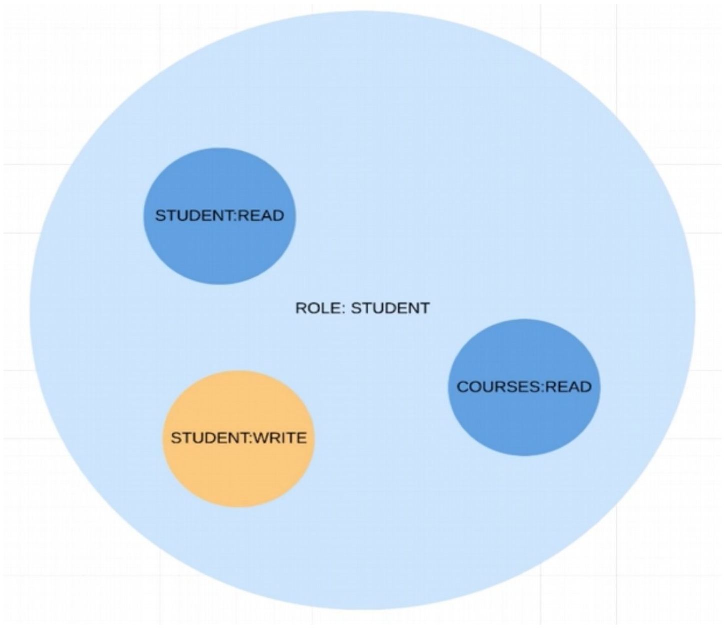


- Username
- Pasword (must be encoded)
- Role/s (ROLE_NAME)
- Authorities / Permissions?
- and more ...

- Roles and Authorities

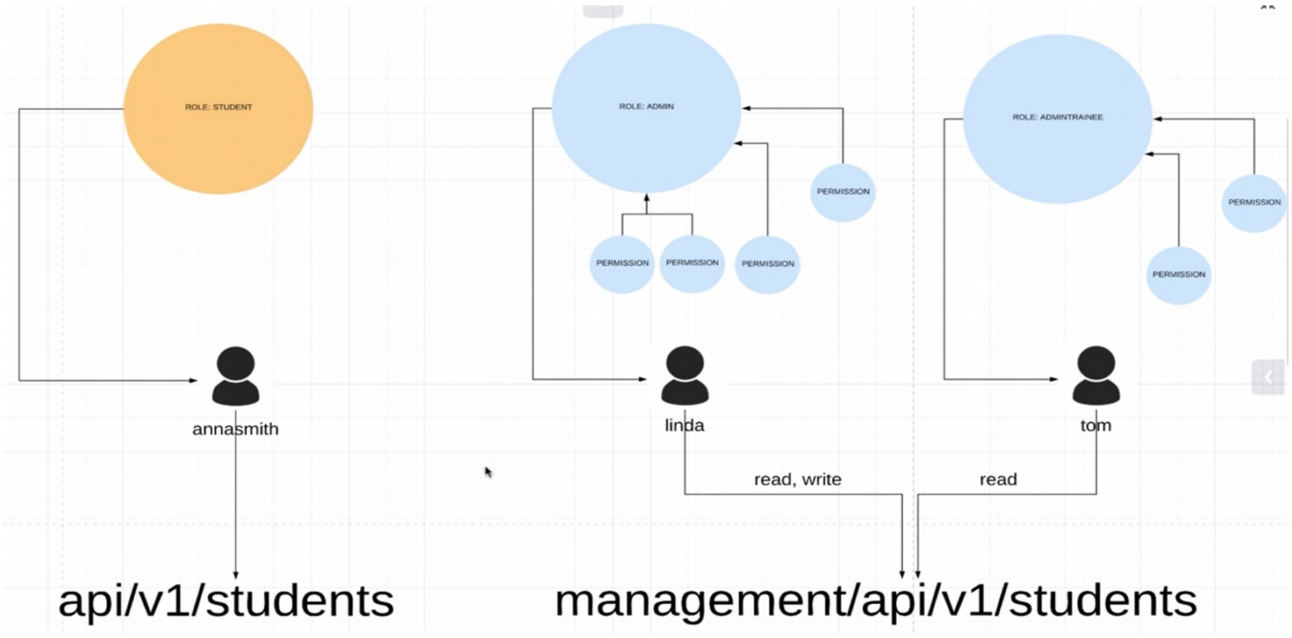
- With Roles we control which endpoints are allow.
- For a specific user you assign a role.





- Roles and permissions allows us to secure the endpoints. For example, you may have an API which is only accessible by admins or other API that's only accessible by students.

- One API should be meant to be used for only one role. For instance I should have localhost:8080/**admin**/api/v1/students



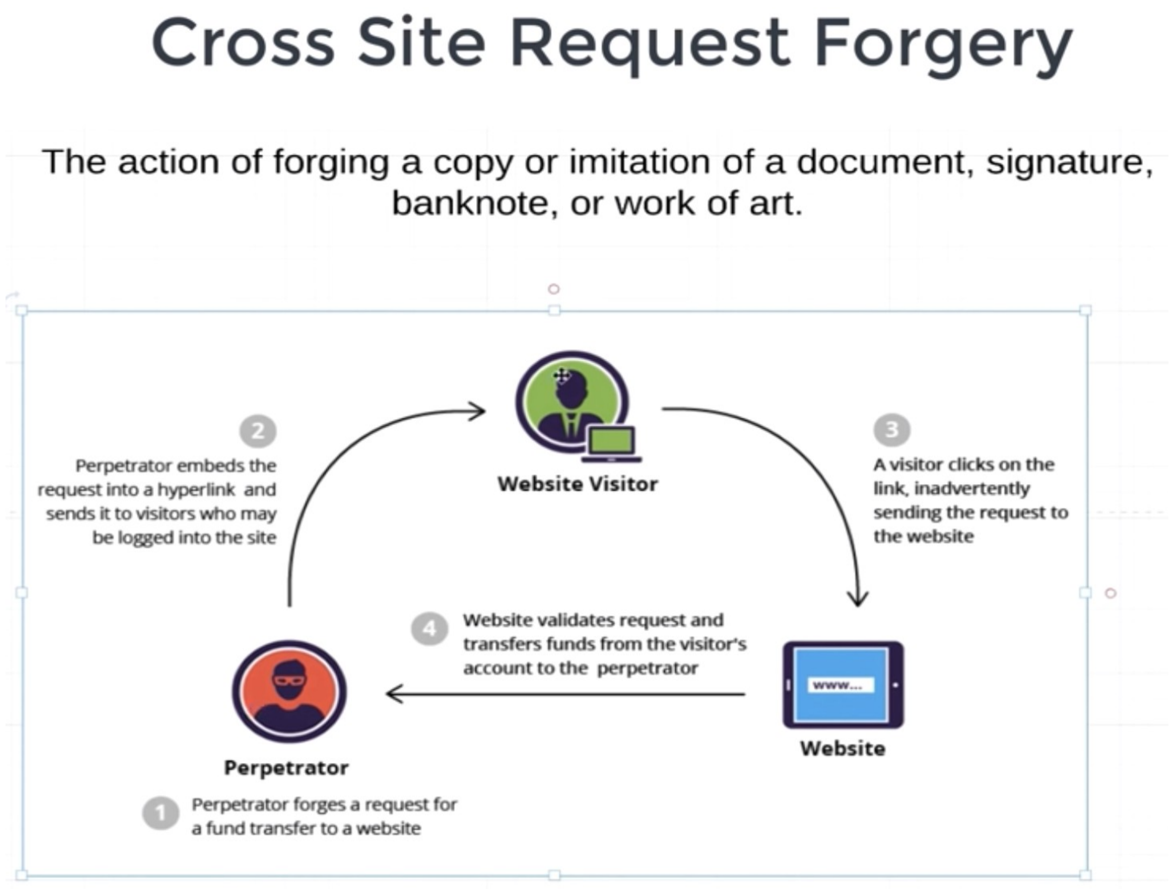
- Order of antMatchers

```
            .antMatchers(HttpMethod.DELETE,
"/management/api/**").hasAuthority(COURSE_WRITE.getPermission())

            .antMatchers(HttpMethod.POST,
"/management/api/**").hasAuthority(COURSE_WRITE.getPermission())

            .antMatchers(HttpMethod.PUT,
"/management/api/**").hasAuthority(COURSE_WRITE.getPermission())

            .antMatchers("/management/api/**").hasAnyRole(ADMIN.name(),
ADMINTRAINEE.name())
```

→ If I'd set the last one that doesn't indicate a type of request at the beginning, all the rest of them would work because I'm saying that just the URL and any user that be ADMIN or ADMINTRAINEE is able to go forward.

- Cross Site Request Forgery (CSRF)

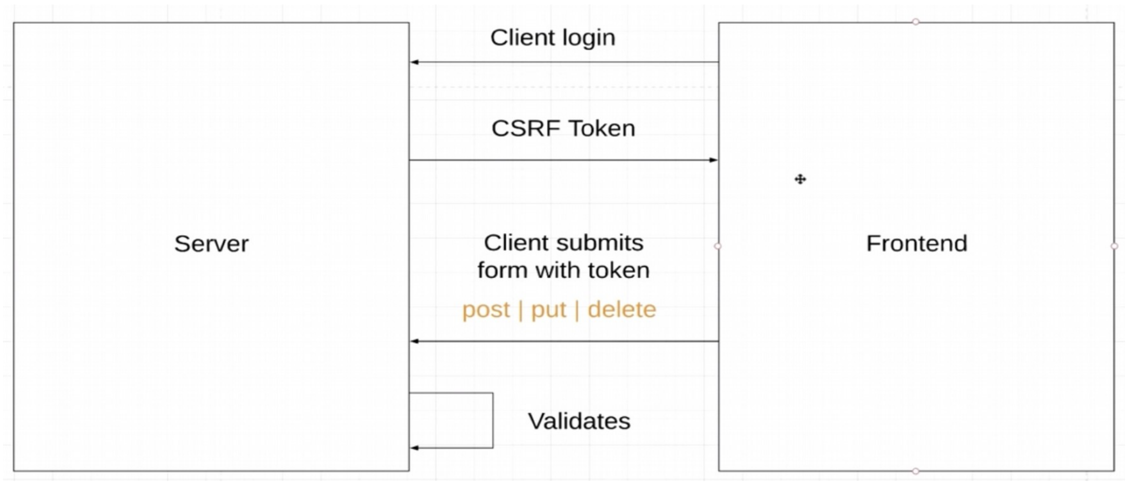
- Forge means make or shape a metal object by heating it in a fire and beating it.
- It’s the action of forging a copy or imitation of a document, signature, etc.
- Example:
    - Transaction: Transfer money from “A” account to “B” account from A account is from the victim and B account from the hacker.
    - To achieve the transaction the victim must be logged in to the home banking.
    - The hacker’s stratregy is making (or forging) the request which makes the transaction and hide it inside a hiperlink being clicked by the victim that may eventually being logged into their home banking.
    - The victim clicks the link and sends the request (not being aware) to their home banking.
    - The request is validated by the home banking and founds are transferred from the victim’s account to the hacker’s account.

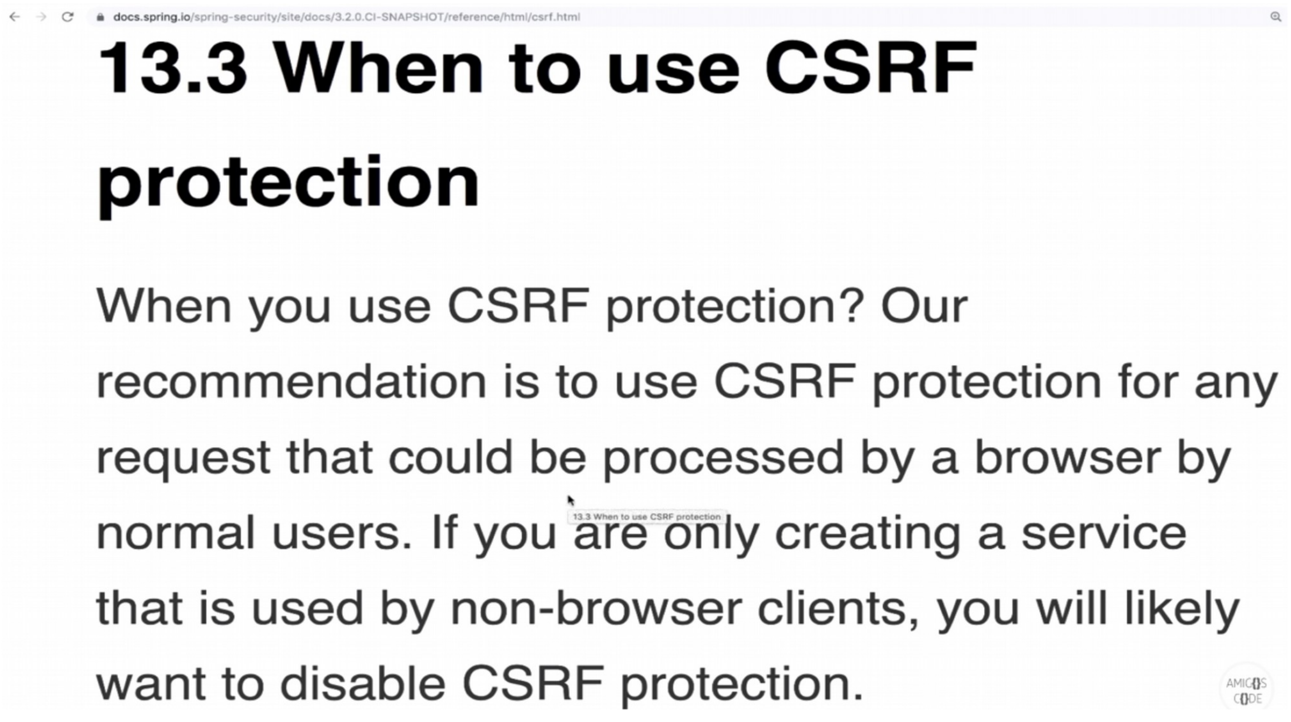

Cross Site Request Forgery

The action of forging a copy or imitation of a document, signature, banknote, or work of art.

Website Visitor
2 Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3 A visitor clicks on the link, inadvertently sending the request to the website
4 Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1 Perpetrator forges a request for a fund transfer to a website

- What Spring Security does to prevent CSRF is generating a CSRF token when the user is logged in and send it into a cookie to the user.
- Only the user knows the token and so the frontend will sent for each submit form (like post, put or delete) the request with the token attached to it. So in case of a third part (as a hacker) trying to forge a request, it will have no token and will be rejected by Spring boot.



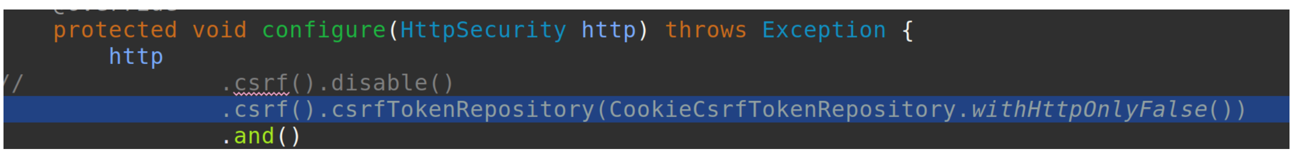

docs.spring.io/spring-security/site/docs/3.2.0.CI-SNAPSHOT/reference/html/csrf.html

13.3 When to use CSRF protection

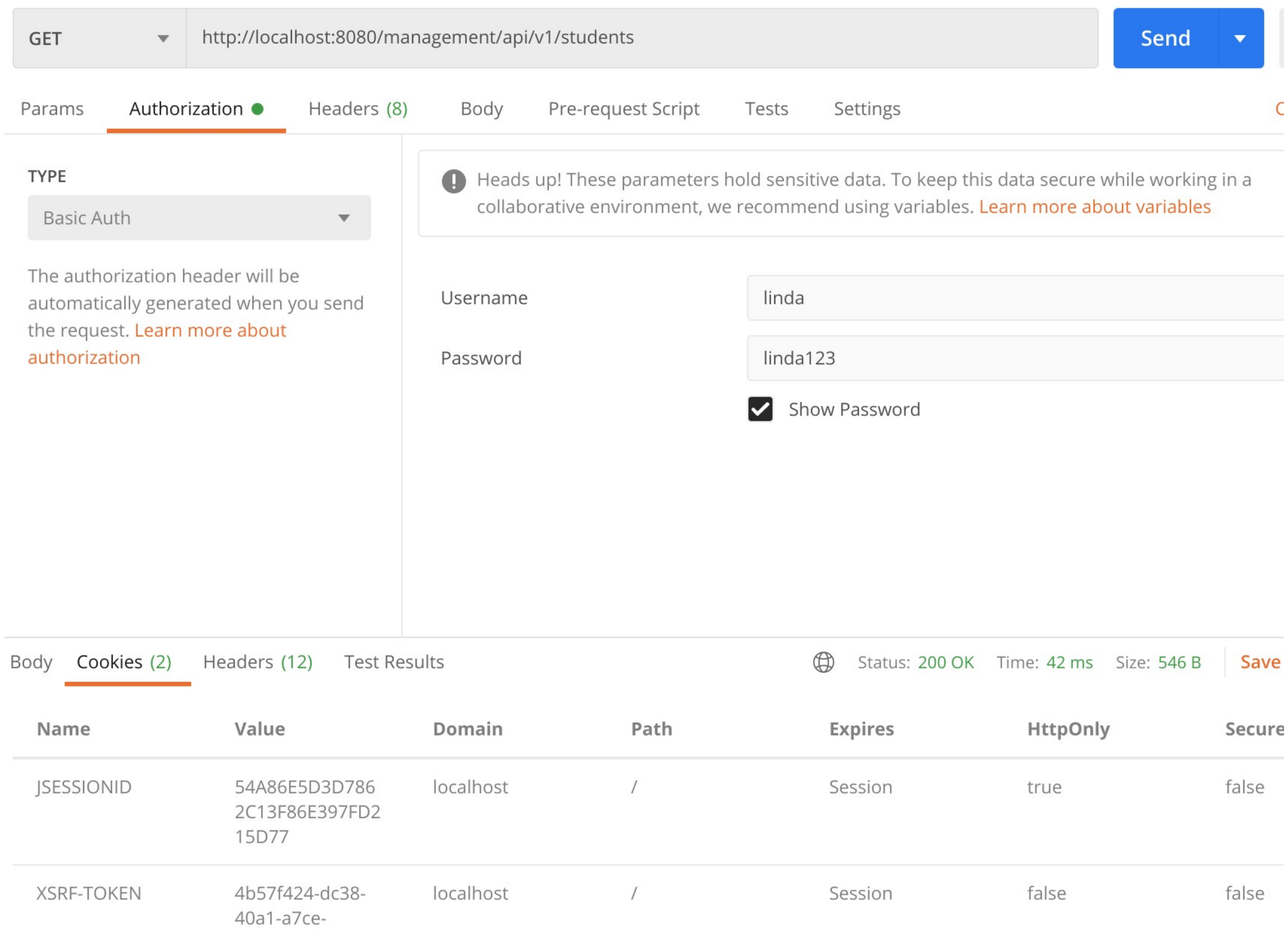
When you use CSRF protection? Our recommendation is to use CSRF protection for any request that could be processed by a browser by normal users. If you are only creating a service that is used by non-browser clients, you will likely want to disable CSRF protection.

→ Stepts to use token with Postman

- Install interceptors in Postman and download the extension for Chrome.



```
    protected void configure(HttpSecurity http) throws Exception {
        http
//              .csrf().disable()
                .csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
                .and()
```



GET http://localhost:8080/management/api/v1/students

TYPE: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Username: linda

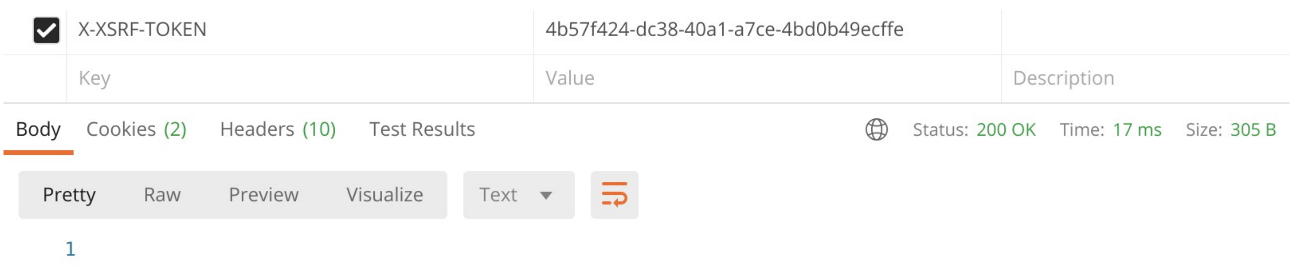
Password: linda123

☑ Show Password

Status: 200 OK Time: 42 ms Size: 546 B

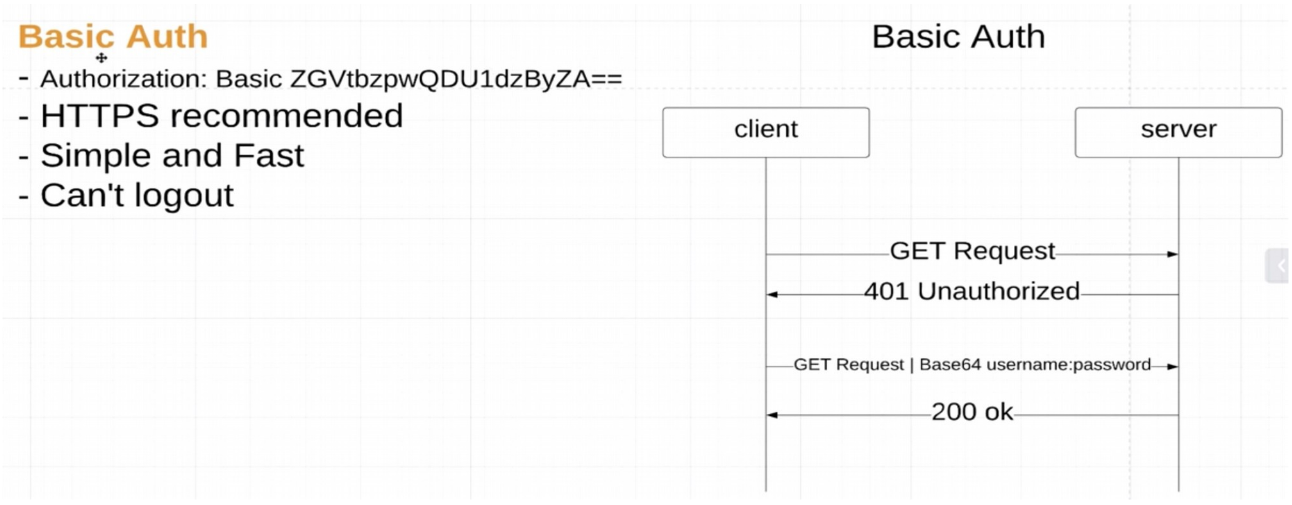
Name	Value	Domain	Path	Expires	HttpOnly	Secure
JSESSIONID	54A86E5D3D7862C13F86E397FD215D77	localhost	/	Session	true	false
XSRF-TOKEN	4b57f424-dc38-40a1-a7ce-	localhost	/	Session	false	false

→ Make a GET request in order to copy the token generated by Spring Security and then add it (in headers) to the next request that requires a token like DELETE.



	Key	Value	Description
✓	X-XSRF-TOKEN	4b57f424-dc38-40a1-a7ce-4bd0b49ecffe	

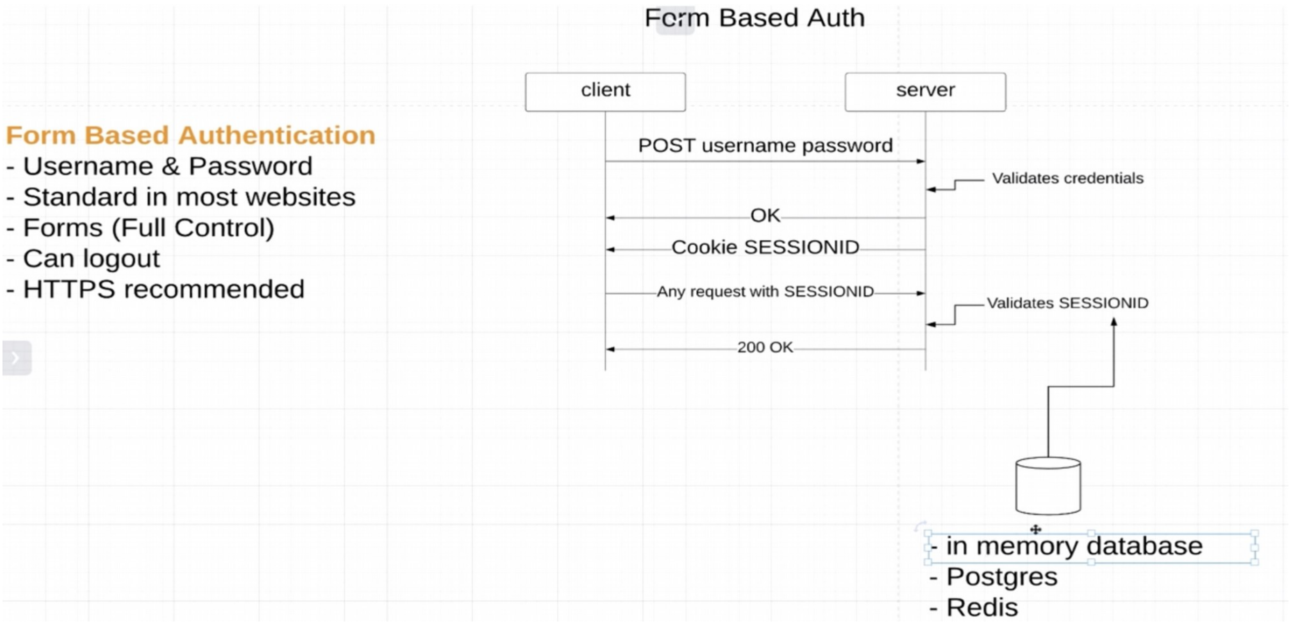
Status: 200 OK Time: 17 ms Size: 305 B


Basic Auth

- Authorization: Basic ZGVtbzpwQDU1dzByZA==
- HTTPS recommended
- Simple and Fast
- Can't logout

→ Remember that with Basic Authentication we have to include the ***authentication header*** in every single request.

- Form Base Authentication


Form Based Authentication

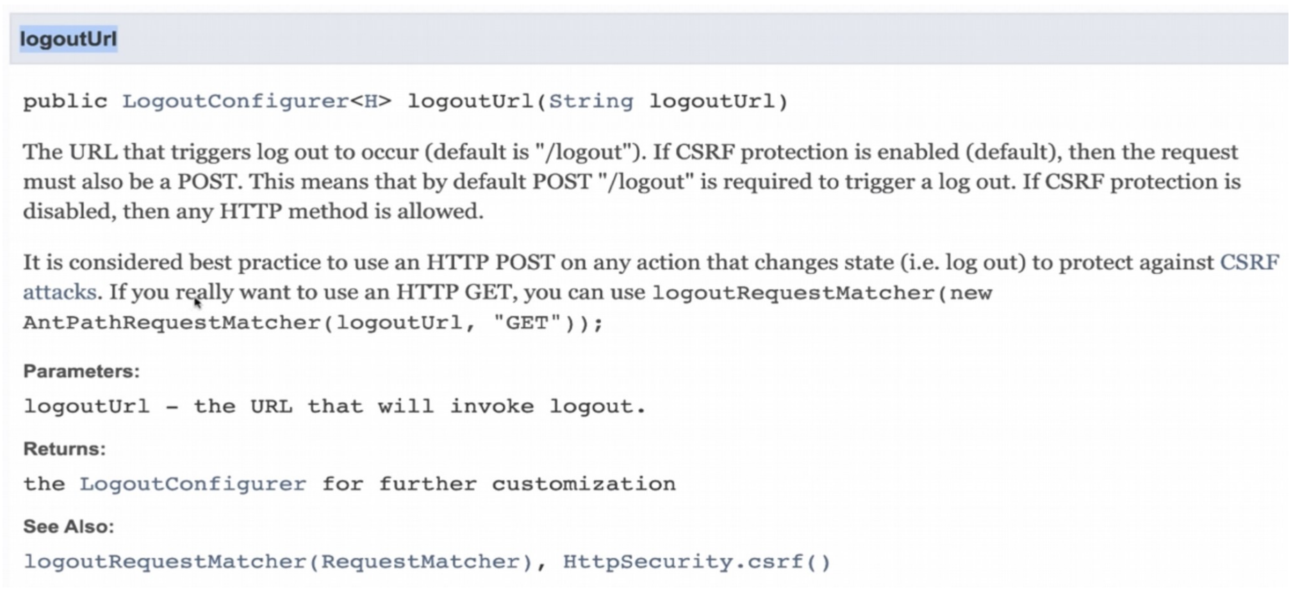
- Username & Password
- Standard in most websites
- Forms (Full Control)
- Can logout
- HTTPS recommended

- By default Spring Security uses in memory database to store the session id.
- In case the server is restarted the session id will be lost, so it’s best practice to store it in some kind of database.

1- I don't understand why if Basic Auth doesn't generate a cookie (that I know it doesn't) when I refresh the browser it doesn't ask me again for the login.
2- Working with Basic Auth if I stop the server and start it again, and press f5 in the browser, then it doesn't ask me for the credentials again (in the video I saw in min. 34 that for you it asked again for credentials).

3- Why using Basic Auth when I login, a cookie appears in Cookies section from Application tab in chromme.
4- In case of Form Based unlike 2:22:25 when I restart the server and then refresh the browser, I don't have to enter back again the credentials.

→ Logout



logoutUrl

`public LogoutConfigurer<H> logoutUrl(String logoutUrl)`

The URL that triggers log out to occur (default is "/logout"). If CSRF protection is enabled (default), then the request must also be a POST. This means that by default POST "/logout" is required to trigger a log out. If CSRF protection is disabled, then any HTTP method is allowed.

It is considered best practice to use an HTTP POST on any action that changes state (i.e. log out) to protect against CSRF attacks. If you really want to use an HTTP GET, you can use `logoutRequestMatcher(new AntPathRequestMatcher(logoutUrl, "GET"));`

Parameters:

`logoutUrl - the URL that will invoke logout.`

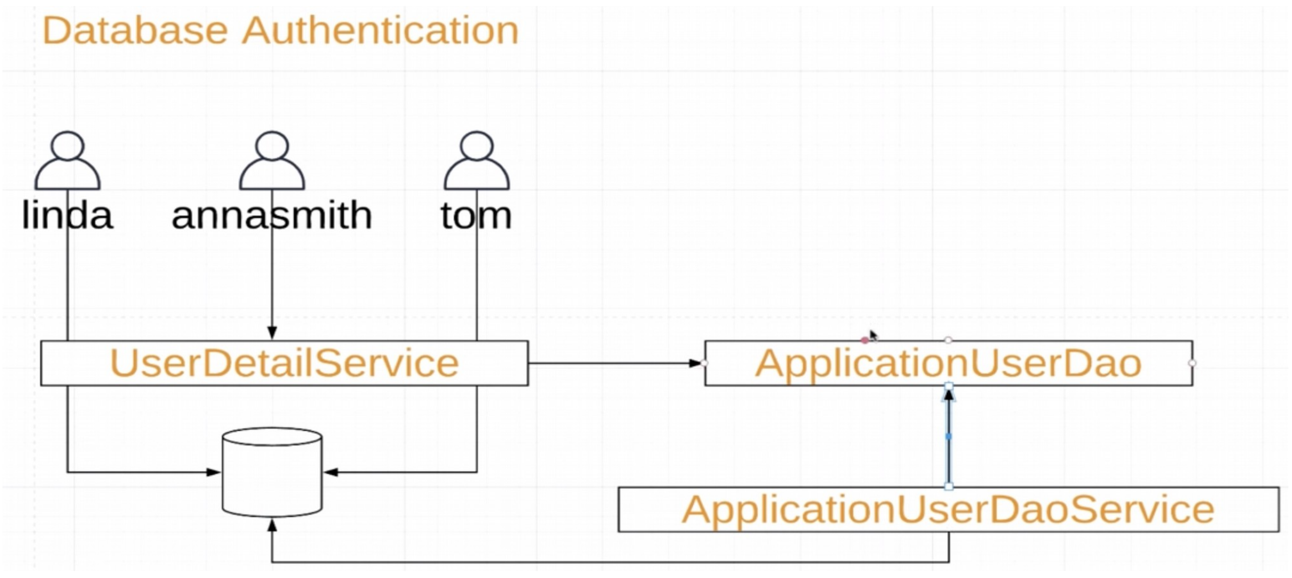
Returns:

`the LogoutConfigurer for further customization`

See Also:

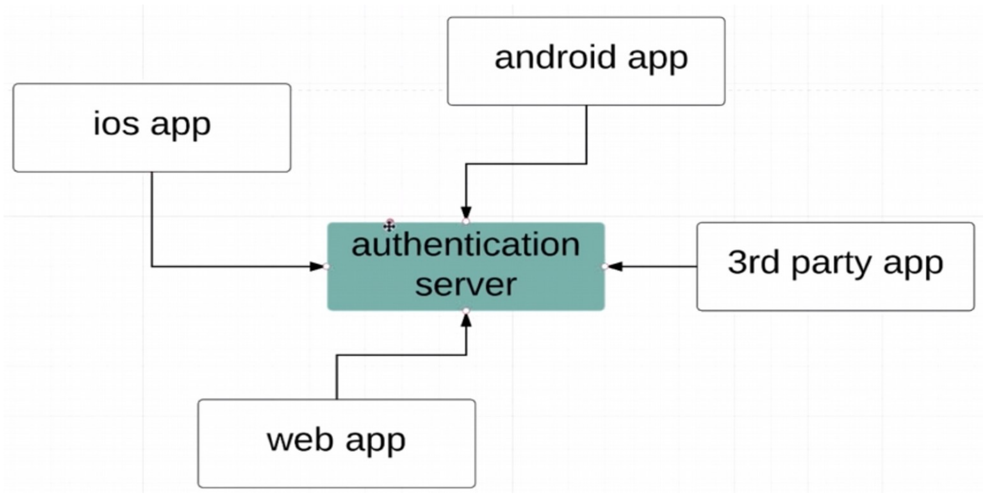
`logoutRequestMatcher(RequestMatcher), HttpSecurity.csrf()`

- Database



- JWT

- In case I have different platforms consuming services from my backend app implementing Authentication, it wouldn't be possible to implement Basic Auth or Form Based because I need to have a common way of access for the different platforms to my app.

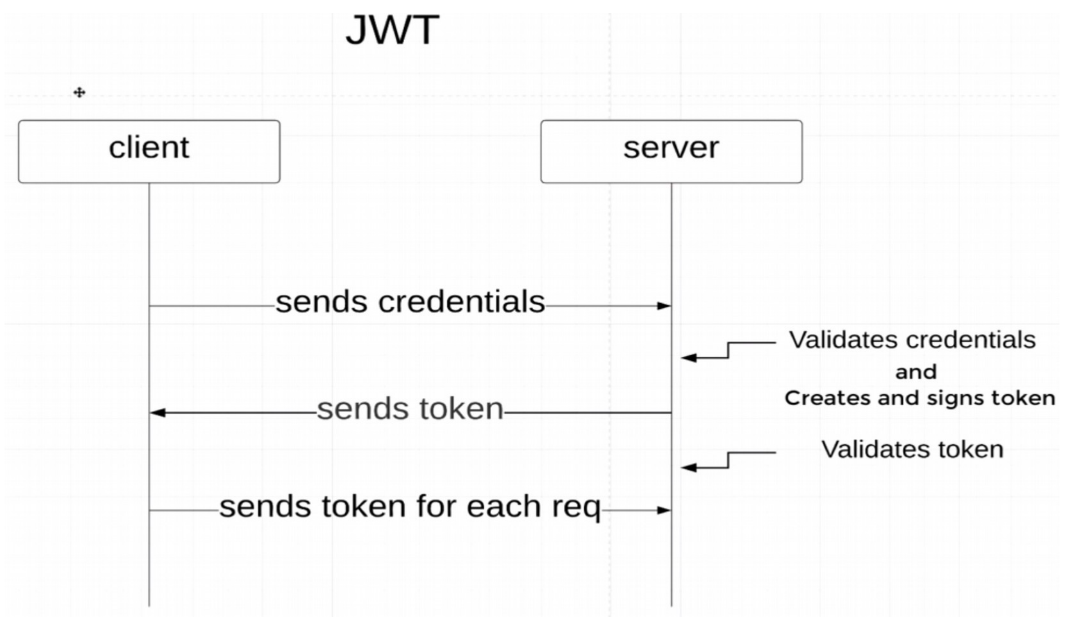


- It's stateless because the session doesn't need to be stored (there is no database) because everything is embebed in a token.

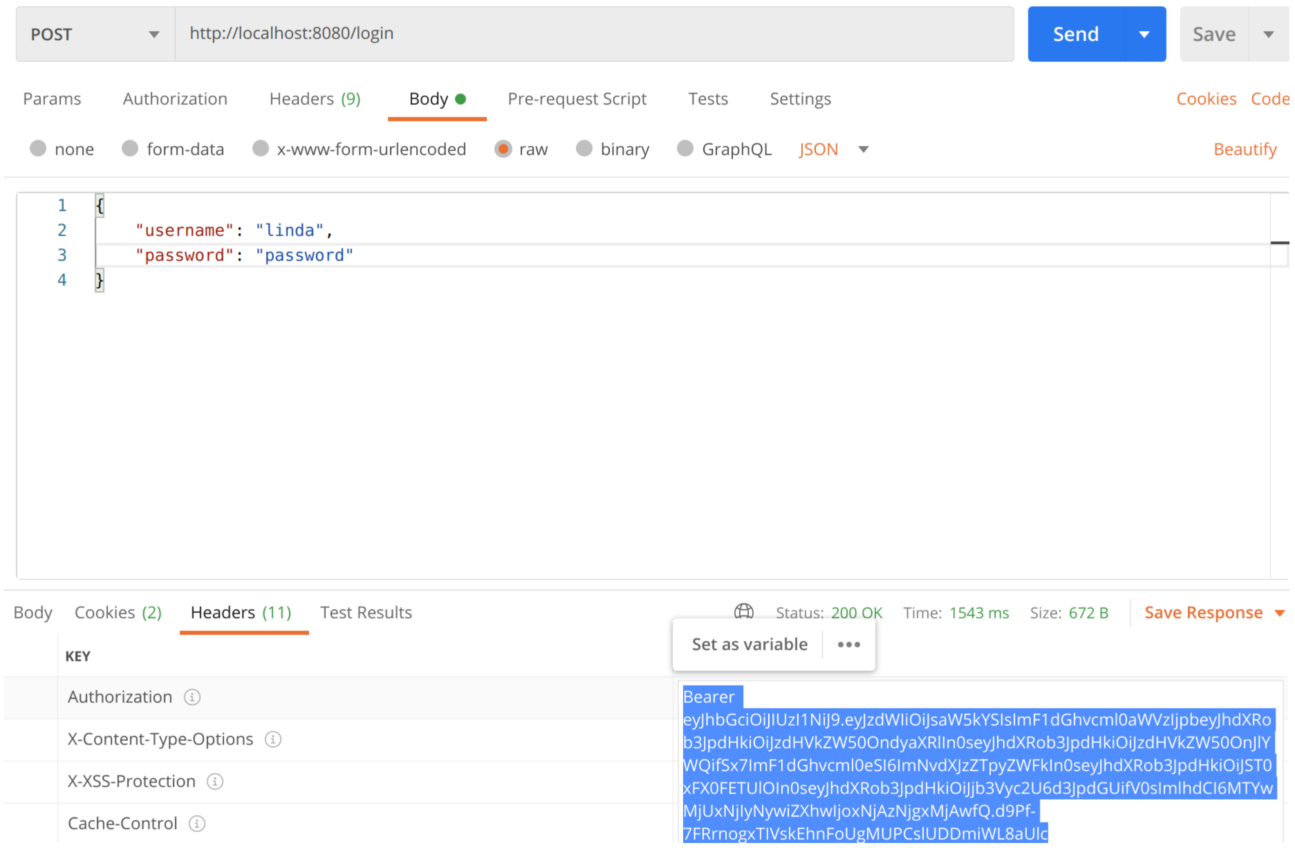
- It may be used across many services.

- If the token is stolen a hacker can pretend to be the real user in my app.

- There is no visibility to logged in users.



→ Once implemented I check it by postman:



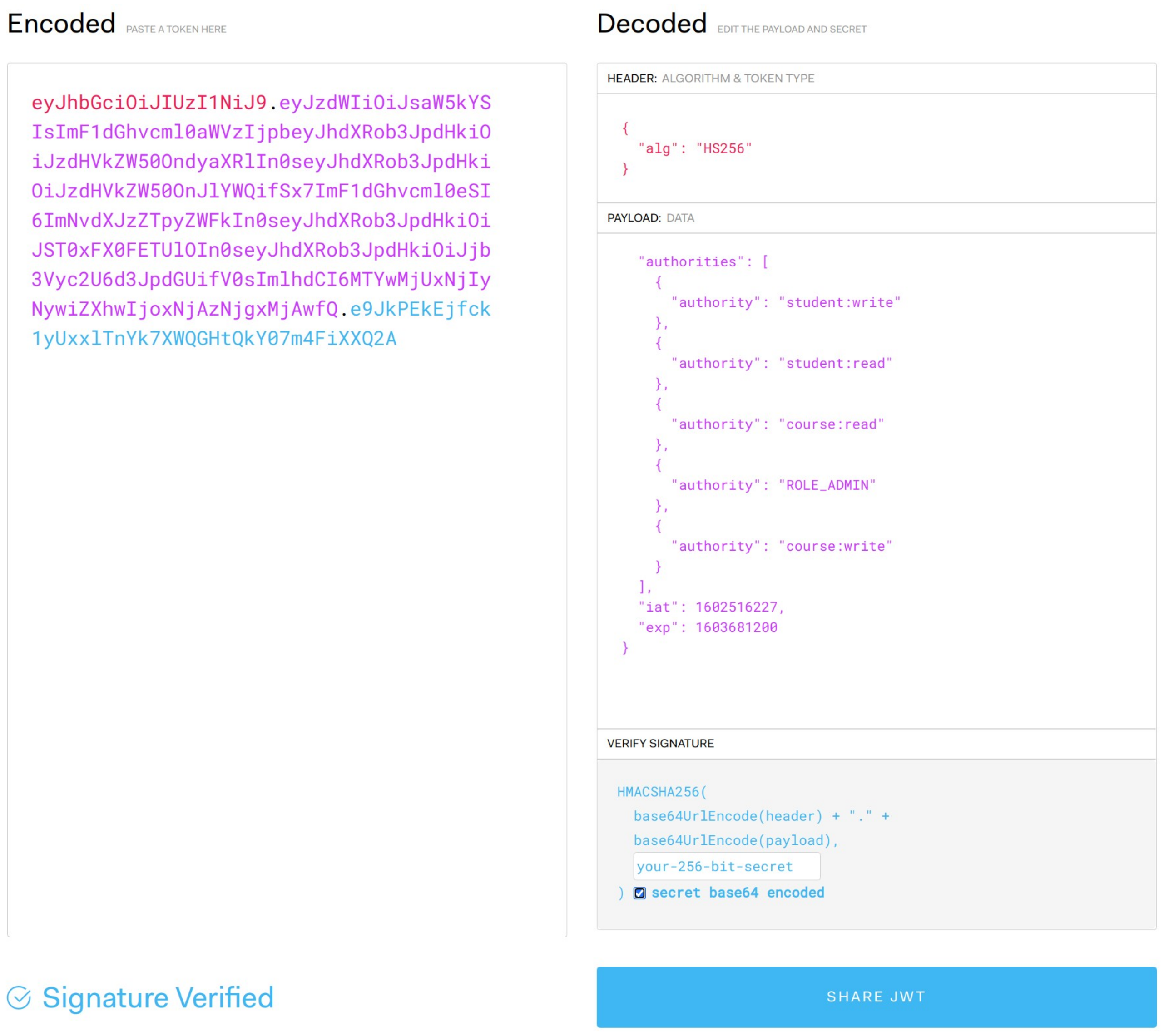
http://localhost:8080/login

```
{
    "username": "linda",
    "password": "password"
}
```

Status: 200 OK Time: 1543 ms Size: 672 B

KEY	
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsaW5kYSIsImF1dGhvcml0aWVzIjpbeyJhdXRob3JpdHkiOiJzdHVkZW50OndyaXRlIn0seyJhdXRob3JpdHkiOiJzdHVkZW50OnJlYWQifSx7ImF1dGhvcml0eSI6ImNvdXJzZTpyZWFkIn0seyJhdXRob3JpdHkiOiJST0xFX0FETUlOIn0seyJhdXRob3JpdHkiOiJjb3Vyc2U6d3JpdGUifV0sImlhdCI6MTYwMjUxNjIyNywiZXhwIjoxNjAzNjgxMjAwfQ.d9Pf-7FRrnogxTIVskEhnFoUgMUPCslUDDmiWL8aUlc
X-Content-Type-Options	
X-XSS-Protection	
Cache-Control	

→ I grab the token from the headers response and paste it on jwt.io:



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsaW5kYS
IsImF1dGhvcml0aWVzIjpbeyJhdXRob3JpdHkiO
iJzdHVkZW50OndyaXRlIn0seyJhdXRob3JpdHki
OiJzdHVkZW50OnJlYWQifSx7ImF1dGhvcml0eSI
6ImNvdXJzZTpyZWFkIn0seyJhdXRob3JpdHkiOi
JST0xFX0FETUlOIn0seyJhdXRob3JpdHkiOiJjb
3Vyc2U6d3JpdGUifV0sImlhdCI6MTYwMjUxNjIy
NywiZXhwIjoxNjAzNjgxMjAwfQ.e9JkPEkEjfck
1yUxxlTnYk7XWQGHtQkY07m4FiXXQ2A
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
  "authorities": [
    {
      "authority": "student:write"
    },
    {
      "authority": "student:read"
    },
    {
      "authority": "course:read"
    },
    {
      "authority": "ROLE_ADMIN"
    },
    {
      "authority": "course:write"
    }
  ],
  "iat": 1602516227,
  "exp": 1603681200
}
```

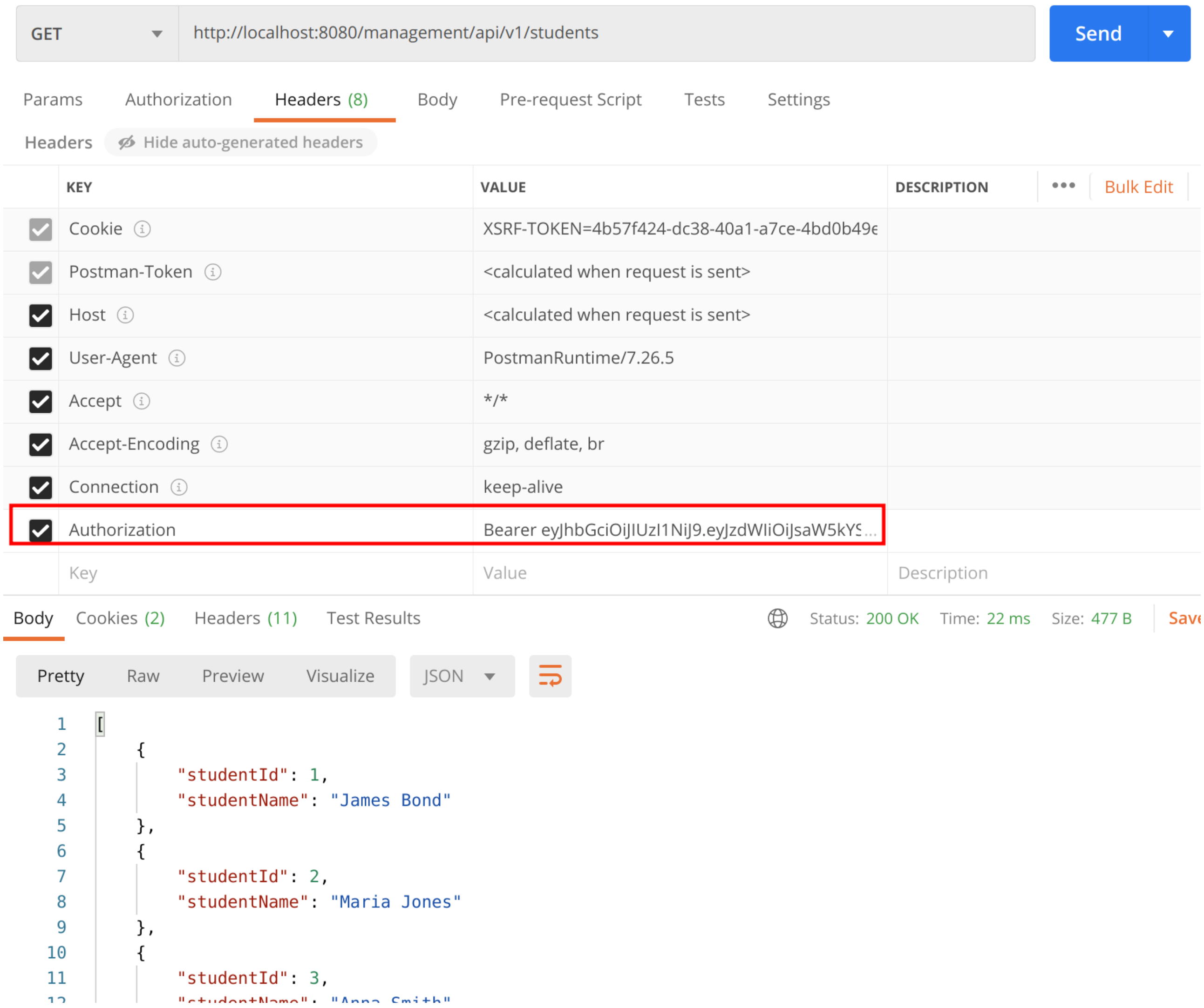
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☑ secret base64 encoded
```

Signature Verified

SHARE JWT

→ After implementing the second filter that validates the request, I copy the token and use it in the next request.



GET http://localhost:8080/management/api/v1/students

Headers (8)

KEY	VALUE	DESCRIPTION
Cookie	XSRF-TOKEN=4b57f424-dc38-40a1-a7ce-4bd0b49e	
Postman-Token	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.26.5	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsaW5kYS…	

Status: 200 OK Time: 22 ms Size: 477 B

```
[
    {
        "studentId": 1,
        "studentName": "James Bond"
    },
    {
        "studentId": 2,
        "studentName": "Maria Jones"
    },
    {
        "studentId": 3,
```