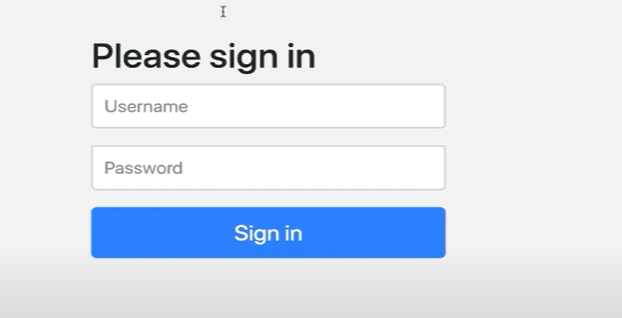
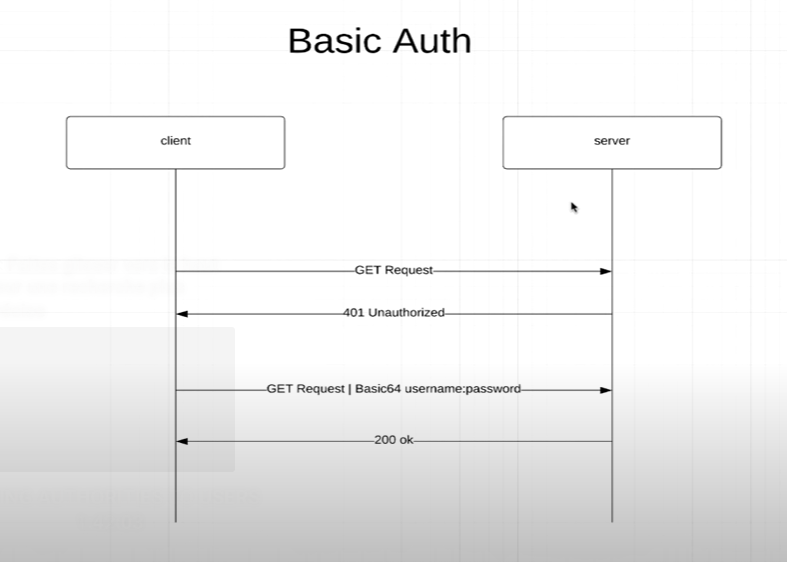
Spring boot security !

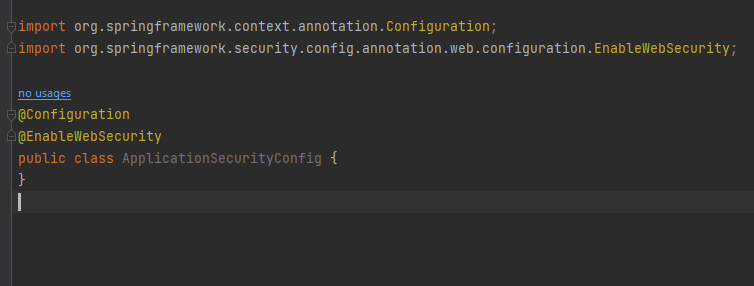
When adding spring boot dependency we get this page when consulting our rest api endpoints !





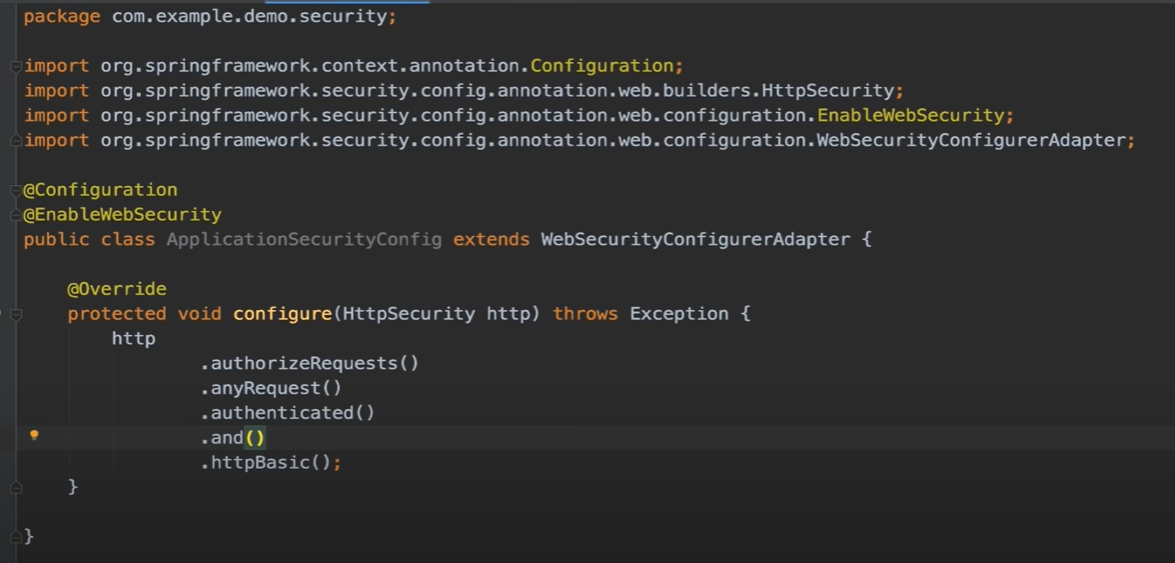
Inconvénient, le client doit envoyer le username et le password dans le header des GET request A CHAQUE FOIS qu’il veut communiquer avec un serveur .

We create a new package called ApplicationSecurityConfig, and annotate it with @configuration et @enablewebSecurity.



We will configure here everything that has to do with security !

The old way of doing stuff !

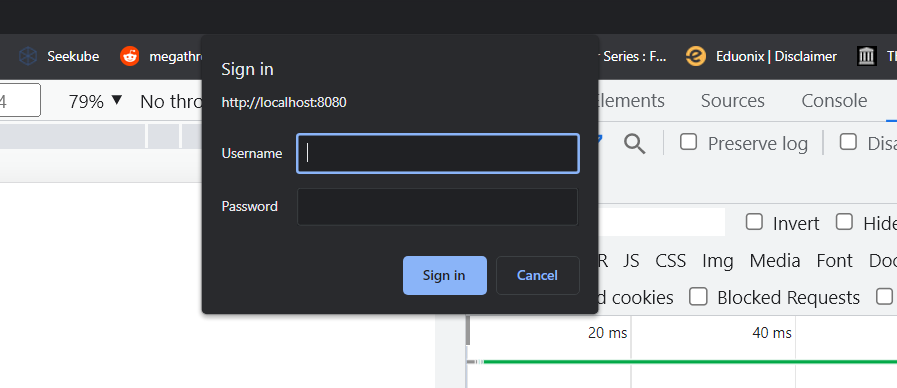


We take the http giving in the argument, and we authorizeRequests(), all request must be authenticated ie the client must specify and username and password using the mecanism Basic auth.



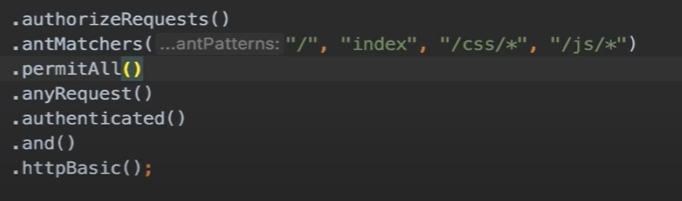
The new way of doing things require the usage of a SecurityFilter bean !

Now we git this when consulting the api endpoint !



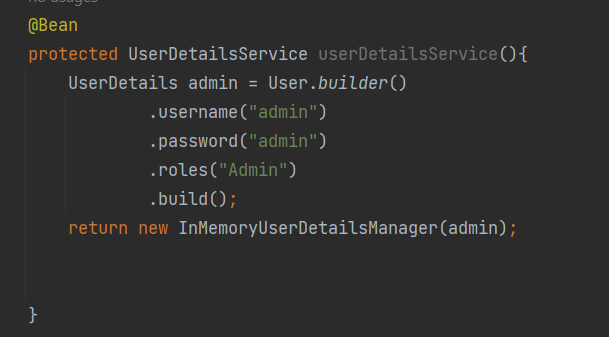
Using the basic auth, u can’t logout.

Si on veut qu’on page soit accessible par tous les utilisateurs sans avoir à se connecter on utilise ce qu’on appelle les ANT MATCHERS !



Dans antMatchers, on veut dire par ‘ /’ le root, ‘index’ tous fichiers qui contient index, et tous les fichiers css/js.

Le permitAll() permet ses patterns définis dans le antMatchers.

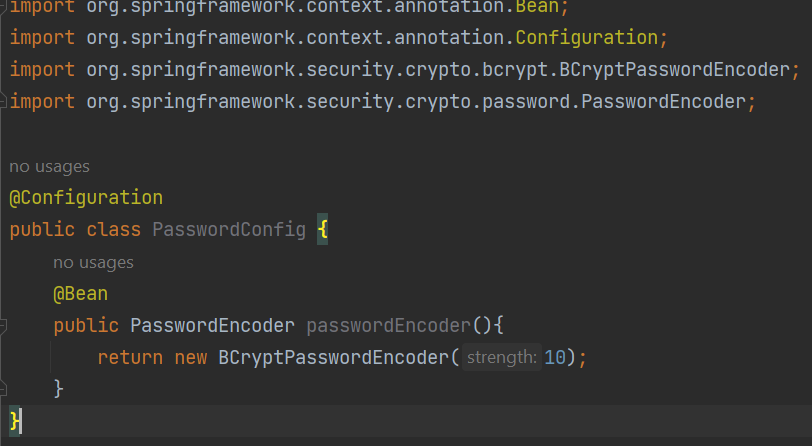
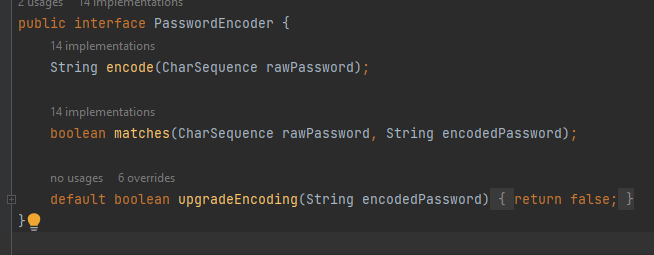


userdetailsService, is how u retrieve ur users from the database, for now i define one manually ! and call it admin, that has a username password admin and its role is Admin, then i build this user !

In order for this to work, we must add password ENCODER ! otherwise we get this error :

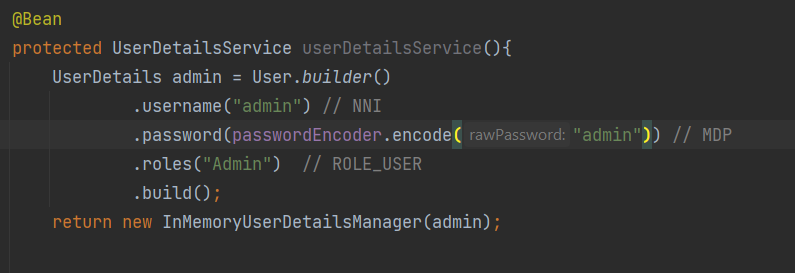


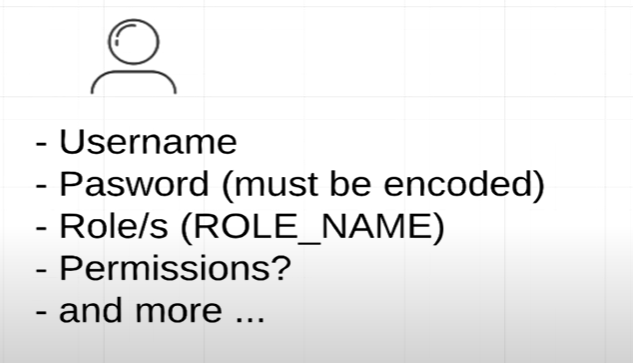
We create a new class, for the password encoder

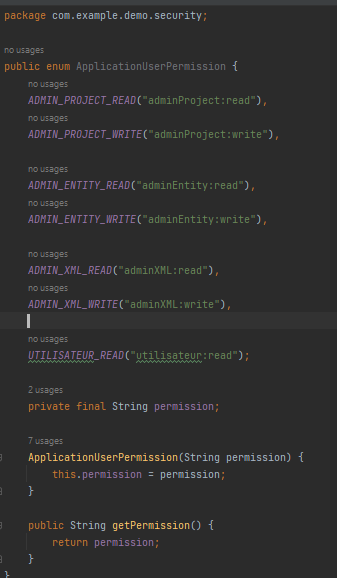
we use passwordEncoder, an interface 

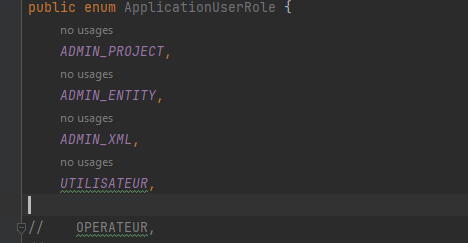
BCryptPasswordEncoder the most popular password encoder, then in Applicationsecurity, we use some dependency injeciton !



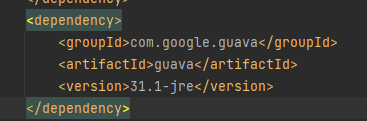
Instead of giving the raw password, we encode it !



We create 2 enums, one for the roles (admin, utilisateur,..) and the other for the permission (admin :read+write, ..) 

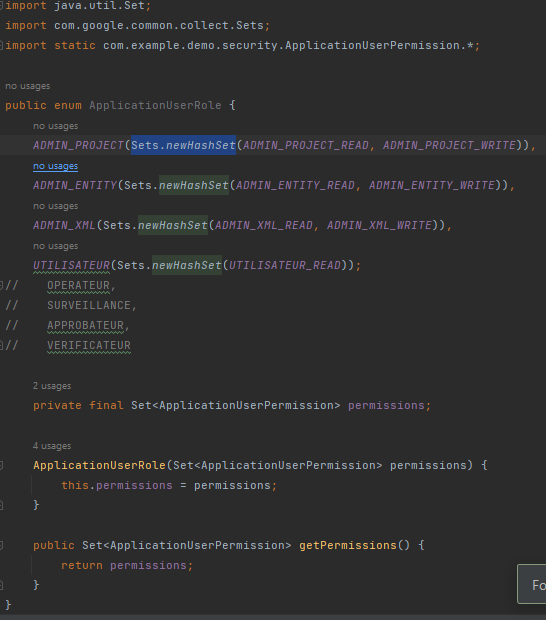


We add then a new dependency :

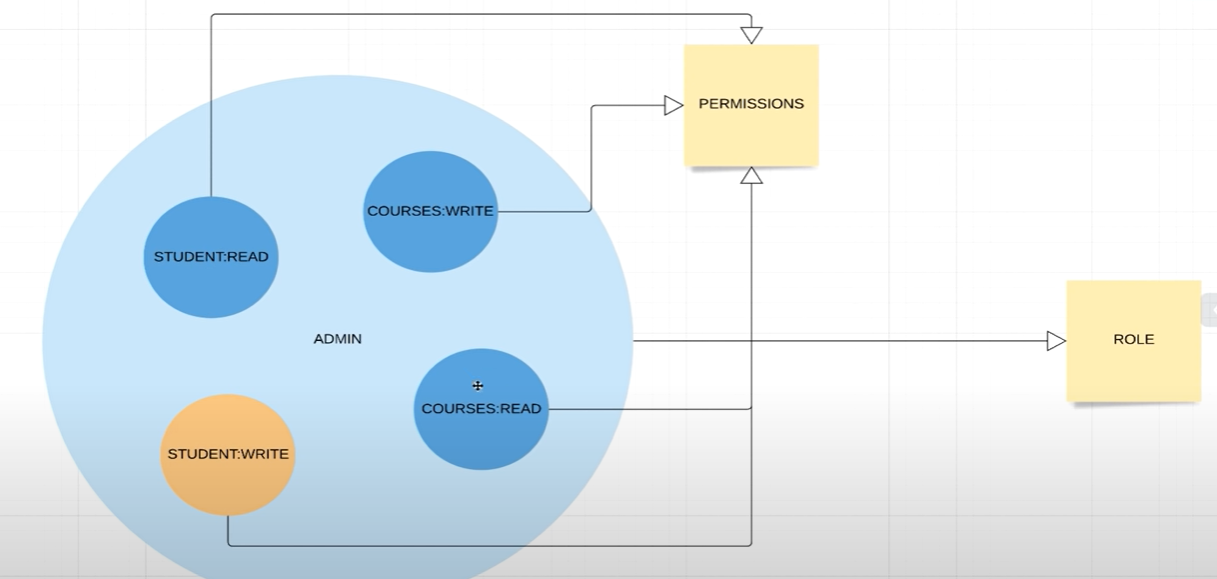


Every role can have 0 or multiple permissions !

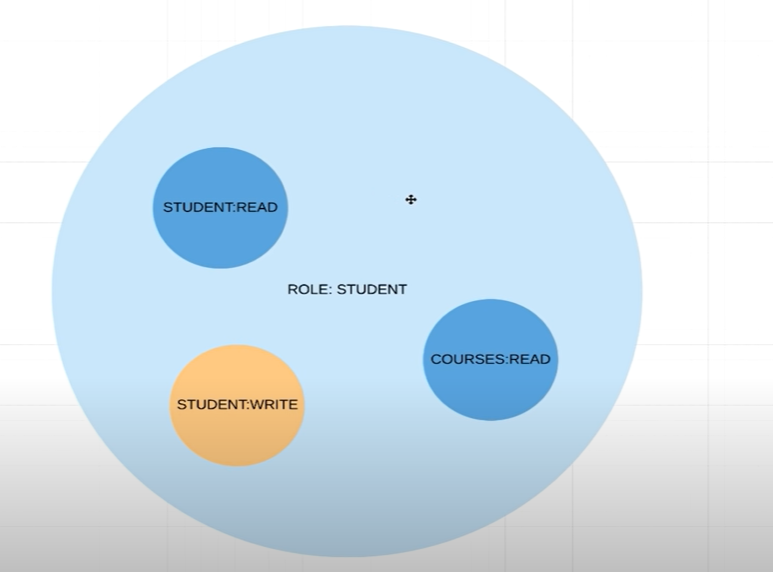
In roles enum, we link roles with permissions !

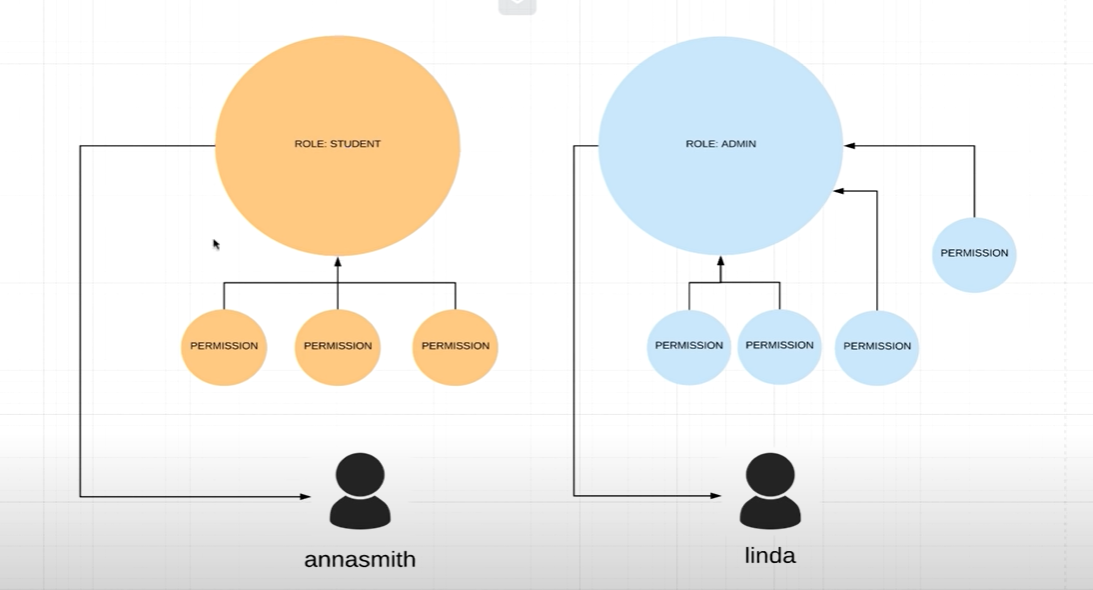


Exemple :



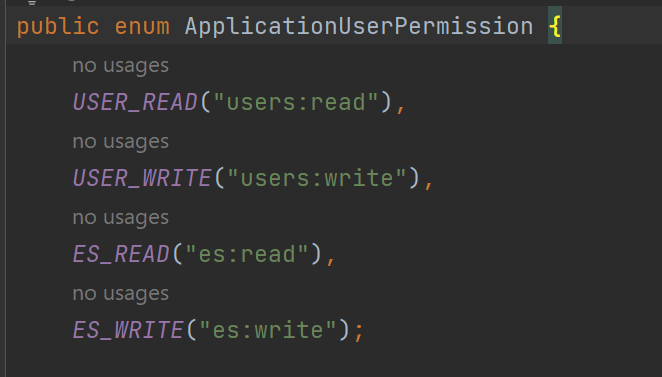
Role admin have multiple permissions, he can read student, write student, read/write courses !



Student role can view/update his student info, and read courses, but can’t write courses ! 

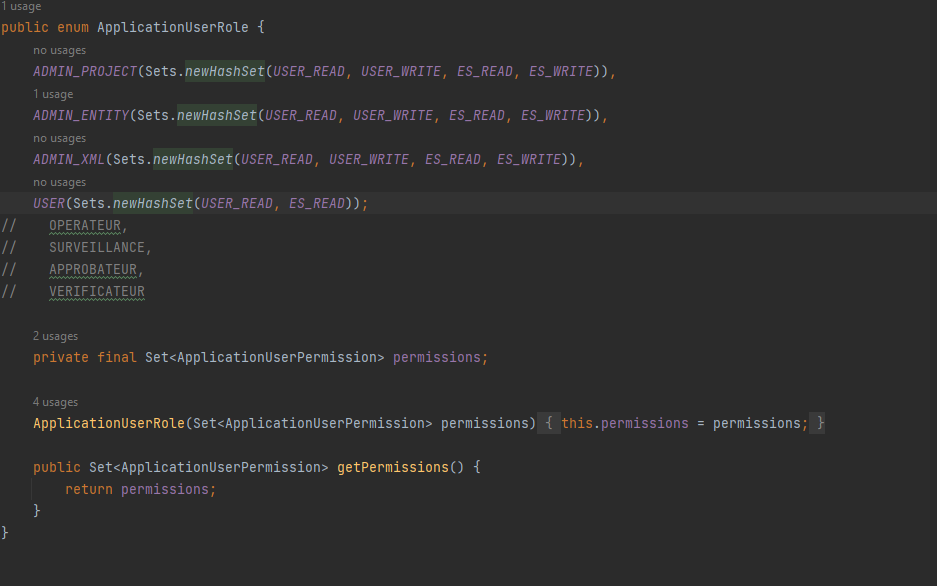
We can attach permissions to roles. You can assign more than one role to a user.

Our permission !



What we can do ? read or write users, read or write Es !

Then we attach our permissions to our roles !



Protect API with role based authentication !



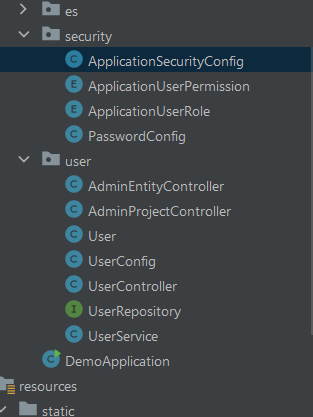
.requestMatchers allow us to secure ou api endpoints !

Api with the uri « / » root is permit to all,

Api with uri /api/ permitted only to admin\_project, so we remove the right to acces this api to other roles !



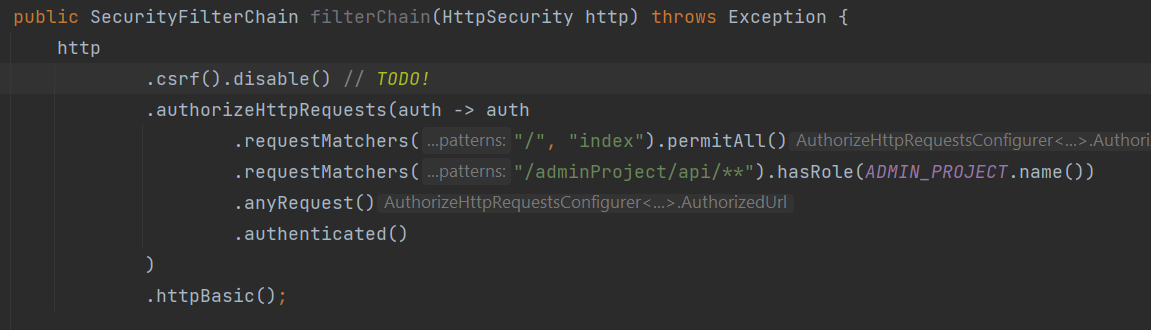
Protect API with permission based authentication !

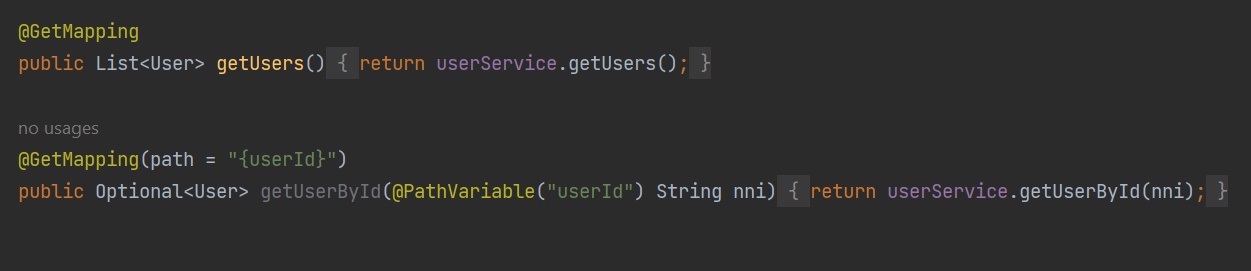
We create more restful controllers 

One for each type of user !

If we try to send post/delete/put request, it will fail !

We need to disable the csrf !

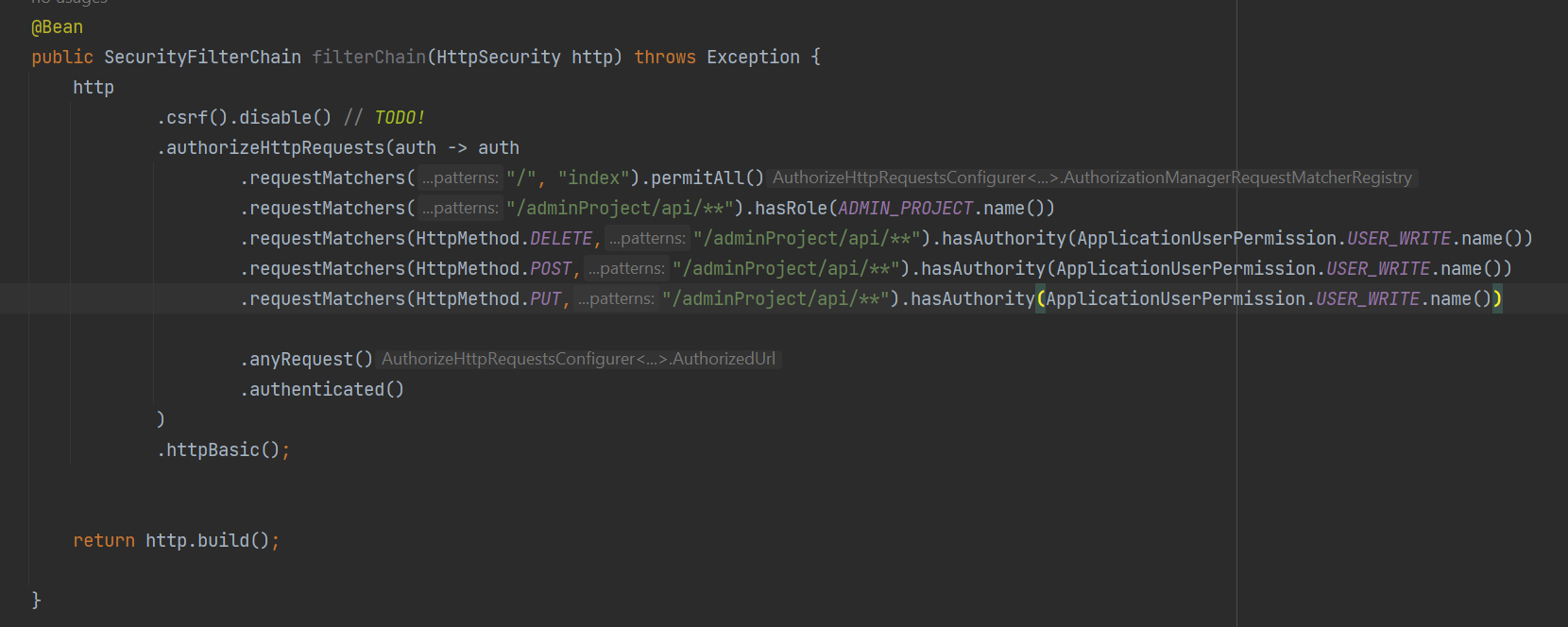




The user type USER can now only get the data,

Later on he will only get the data of the same entity type as him !

We move the post/request/delete request to the admin project controller !

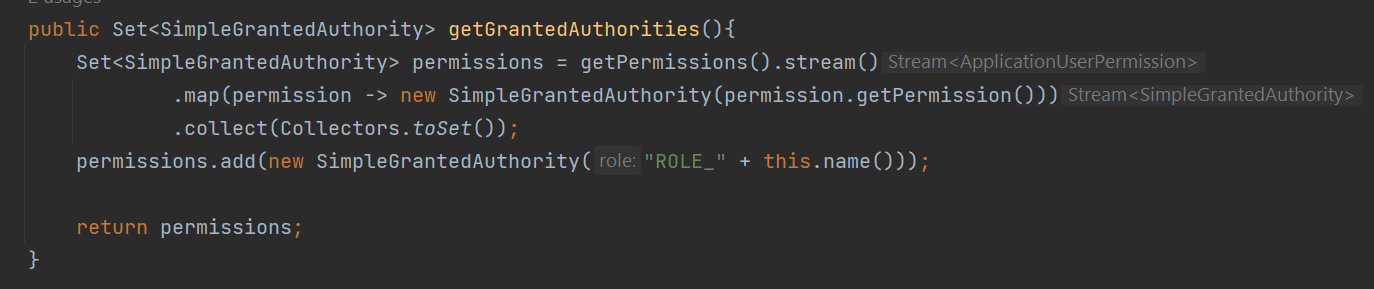




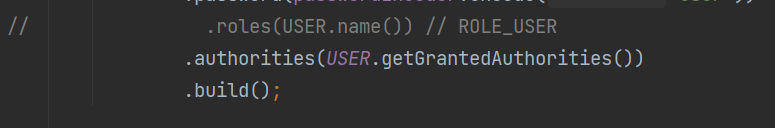
http METHOD delete, from the api uri, can only be executed if the user has the permission given.

ADDING AUTHORITIES TO USERS !

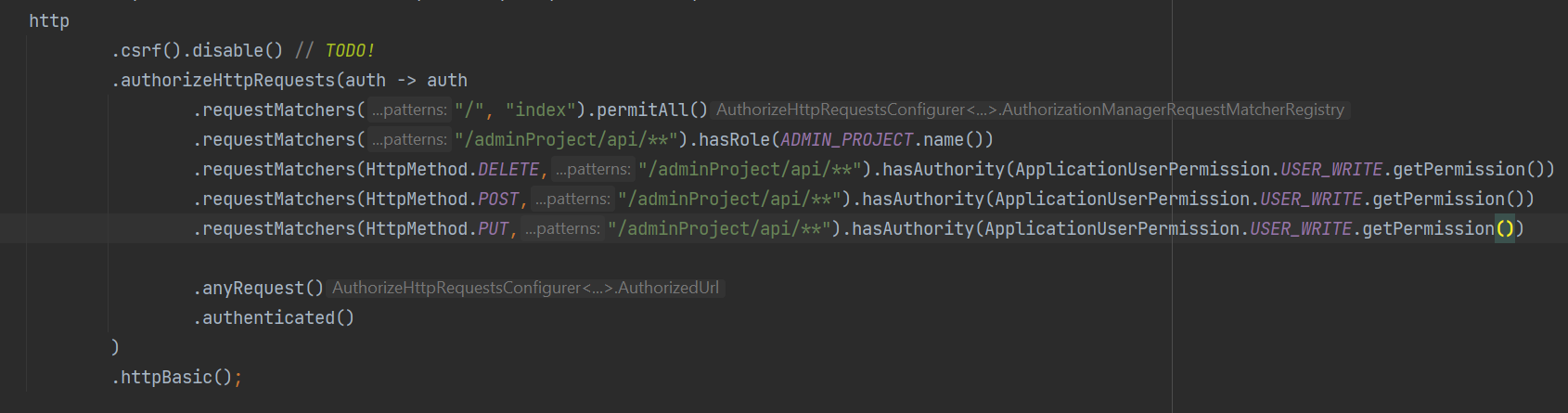
IN role enum we define !



And we change role into authorities :



And we change .name to get permission !



Somthing to note is that the order of the matchers is important ! it goes through the matchers one by one !

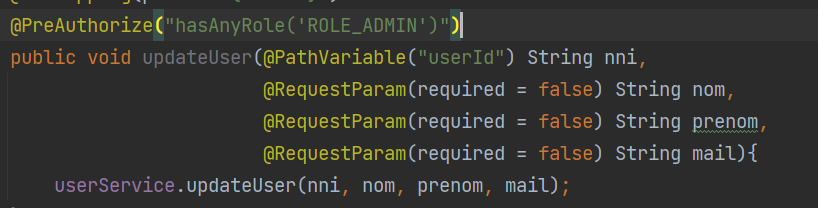
If u access the admin/project/api/\* ? if yes, then do u have the authorities needed ? if no, it blocks immediately the access and so on.

We can secure specific methodes using annotations !

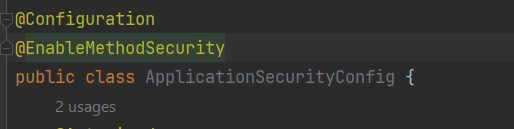
@PreAuthorize()

We control what roles or permissions can access the methods !





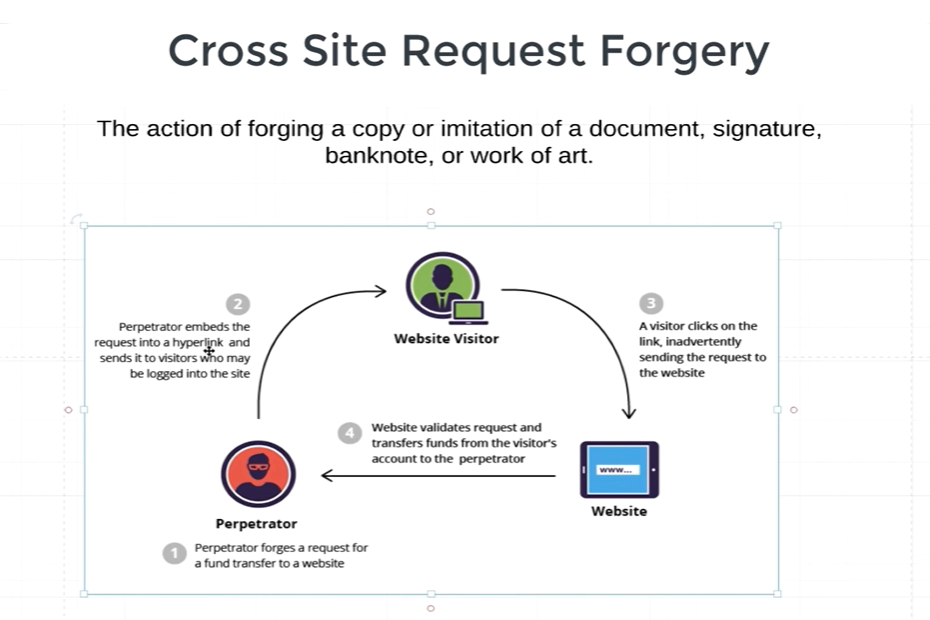
We replace all the antMatchers with these annotations !

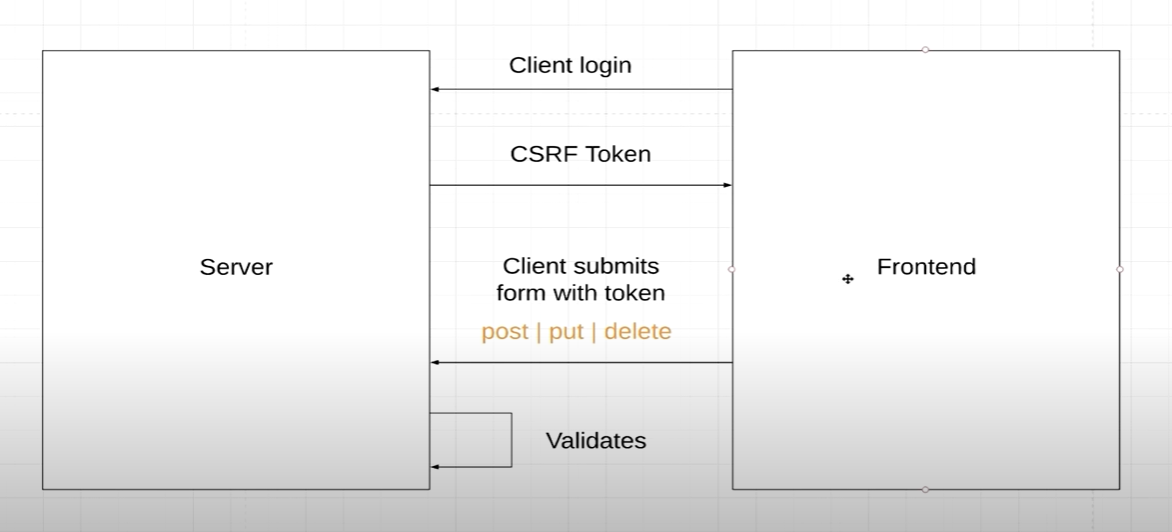


We add the @enbalemethodsecurity annotation to activate the preAuthorize annotation !

UNDERSTANDING CSRF !

Cross Site Request Forgery !





How to get the backend to send the request to the frontend !

FORM BASED AUTH :

