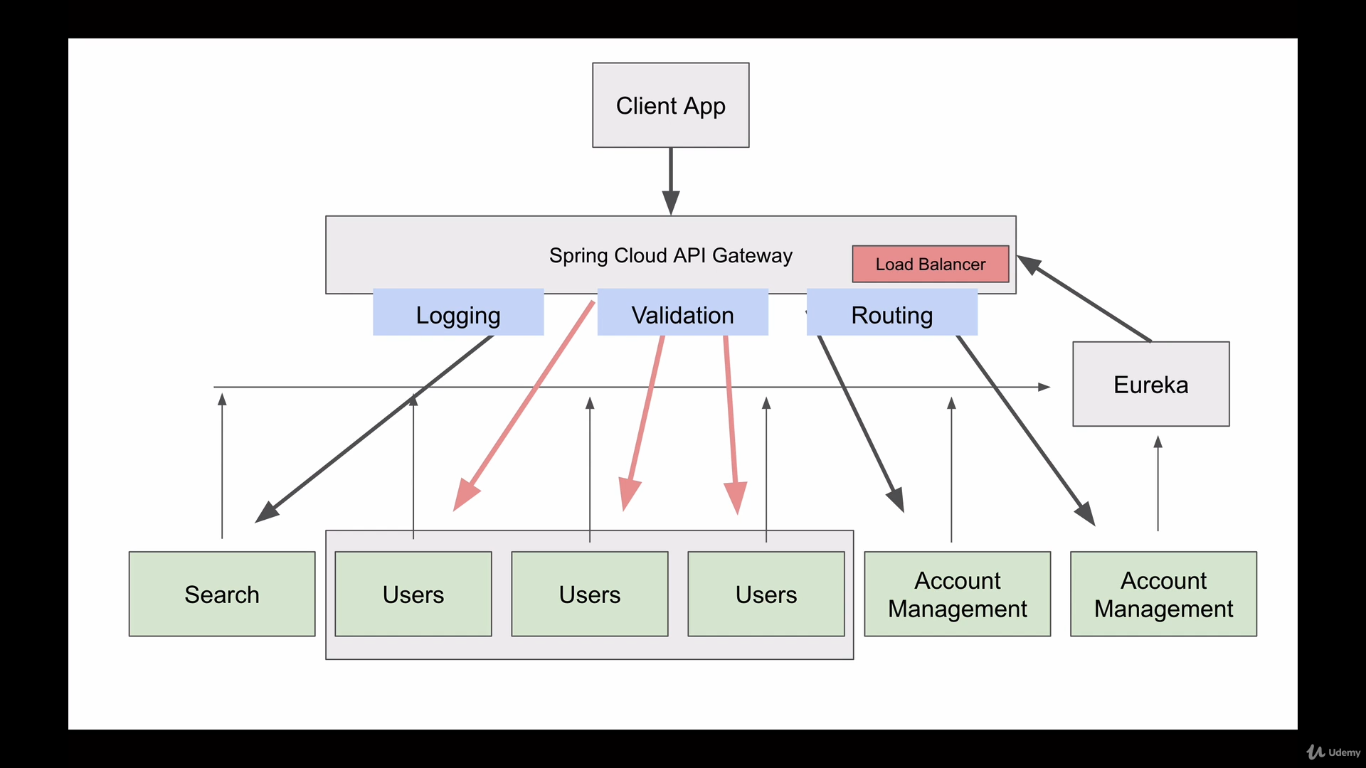
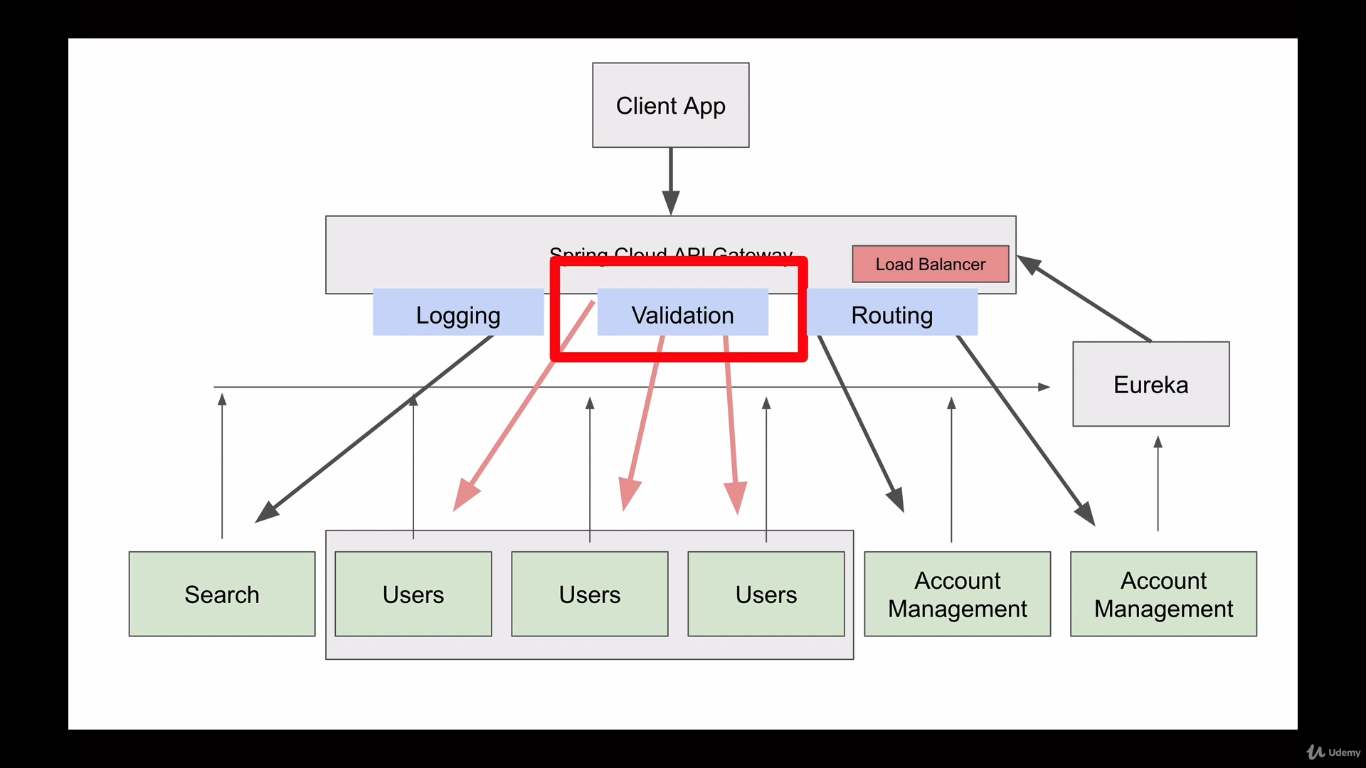
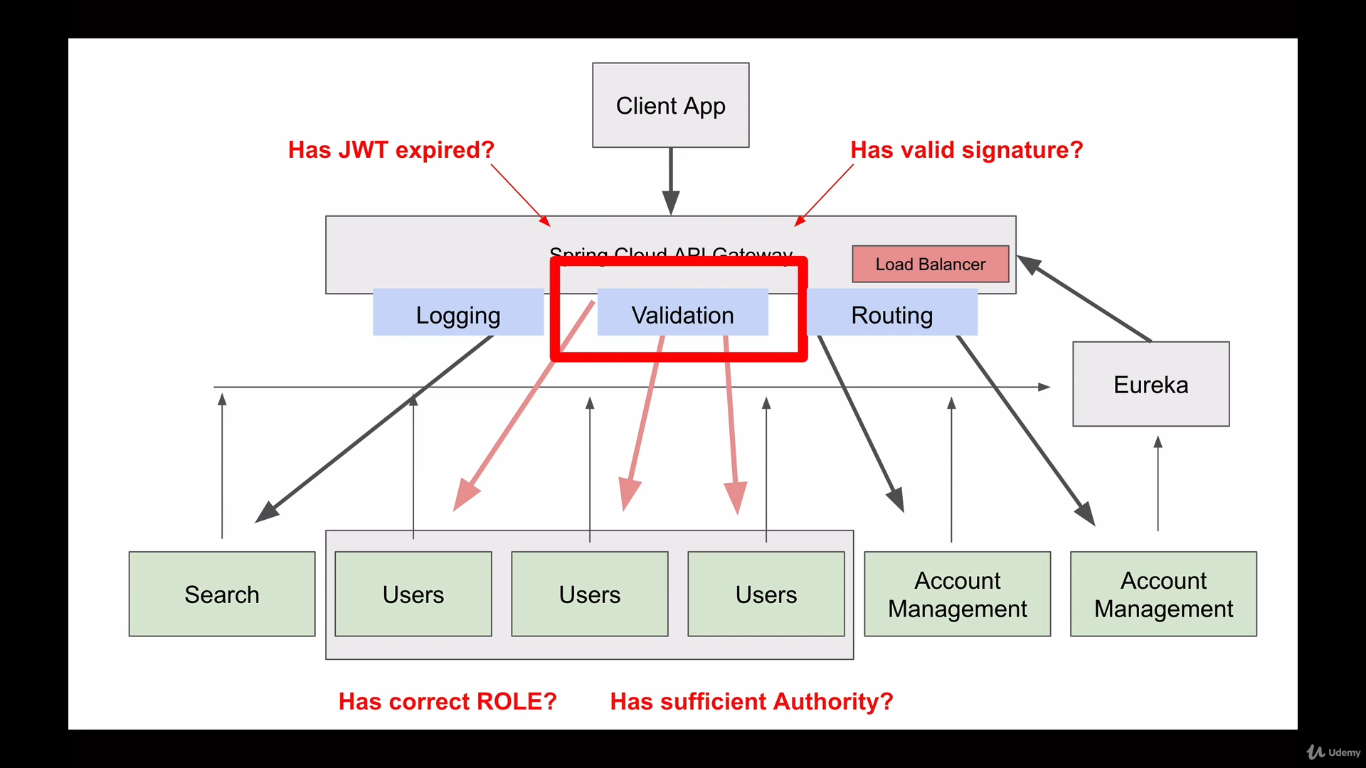
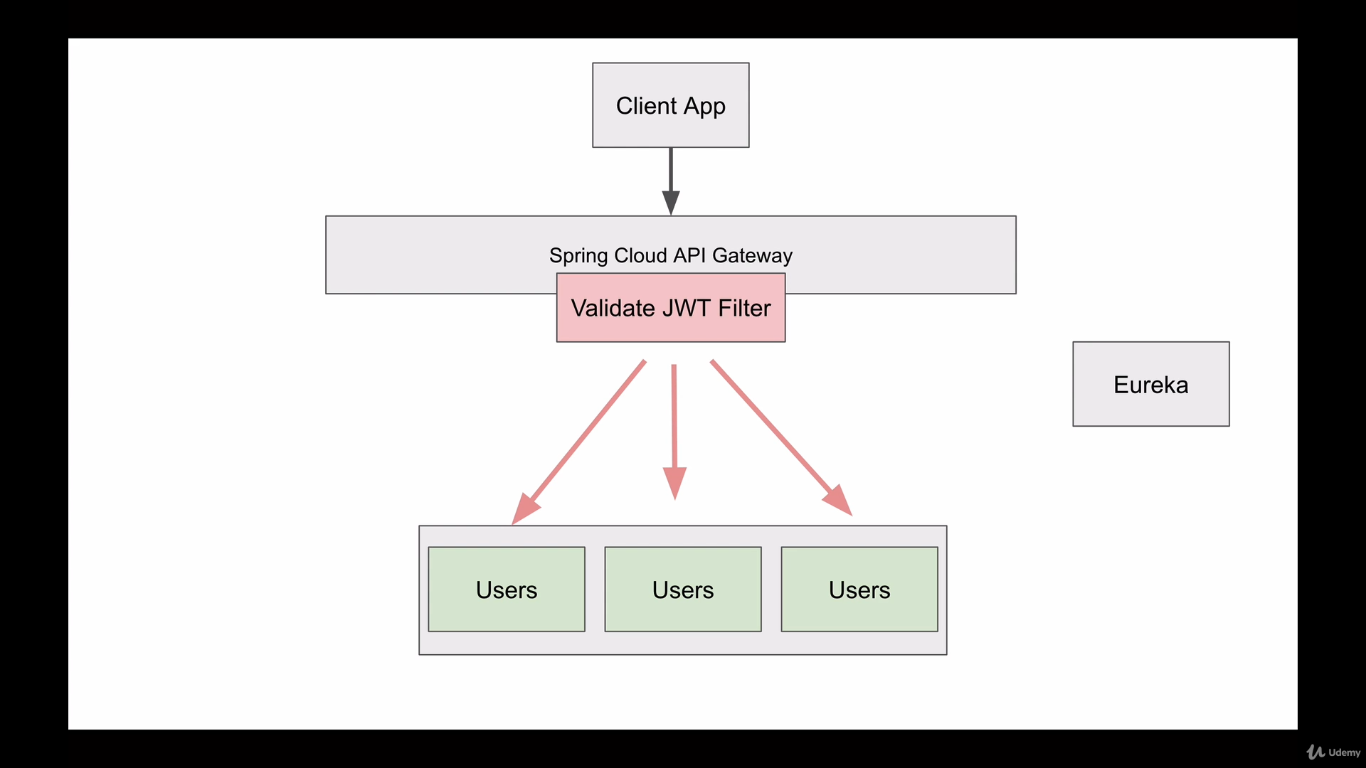
**Adding custom filter to validate JWT Token.**







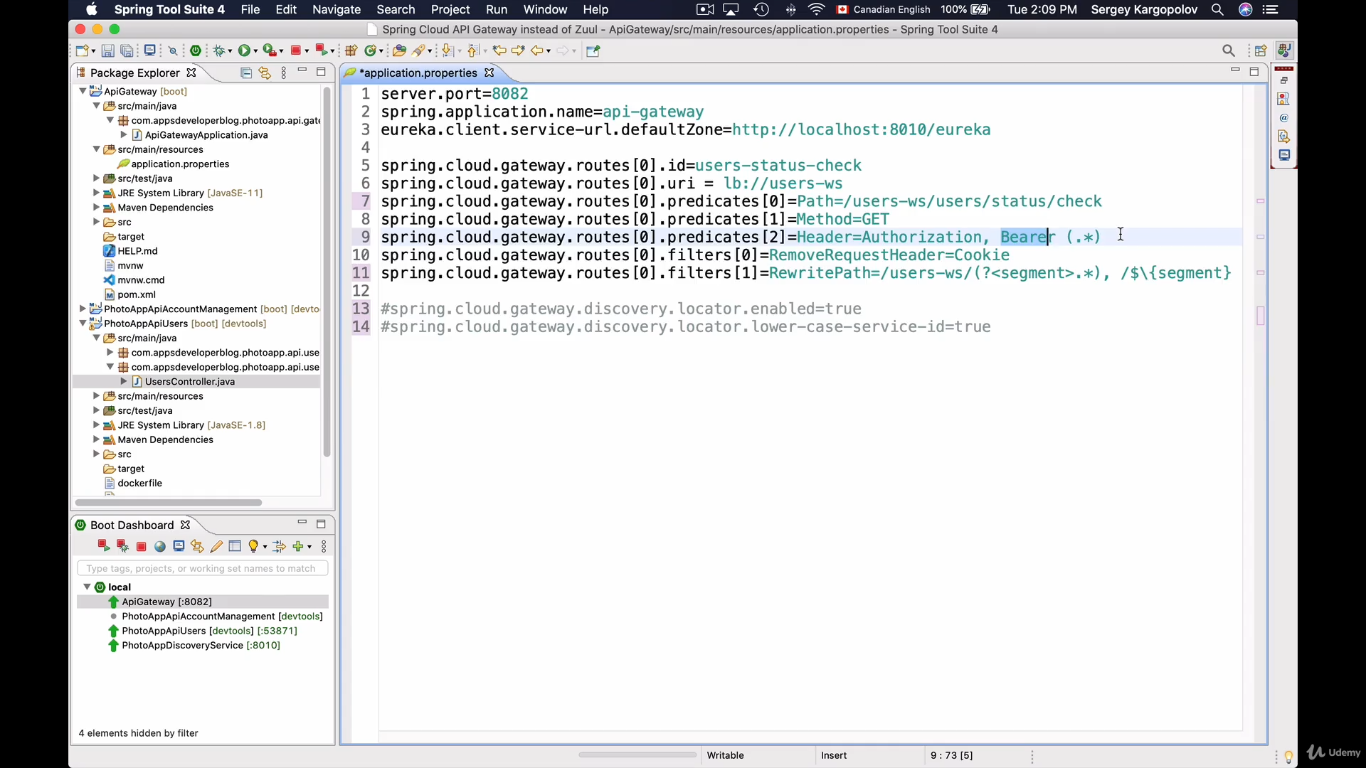
We will use gateway predicate and gateway filters to achieve the following. Some user service doesn’t need to be validated like user registration, user login.



In spring cloud API gateway we will check whether the request header contains the valid JWT token, to check this we will use header predicate.

**Spring.cloud.gateway.routes[0].predicates[2] = Header = Authorization, Bearer (.\*)**

This predicate will check whether the request header has a header called “Authorization” and value which starts from “Bearer”. This predicate does not validate the token, token will be validated by filter separately.

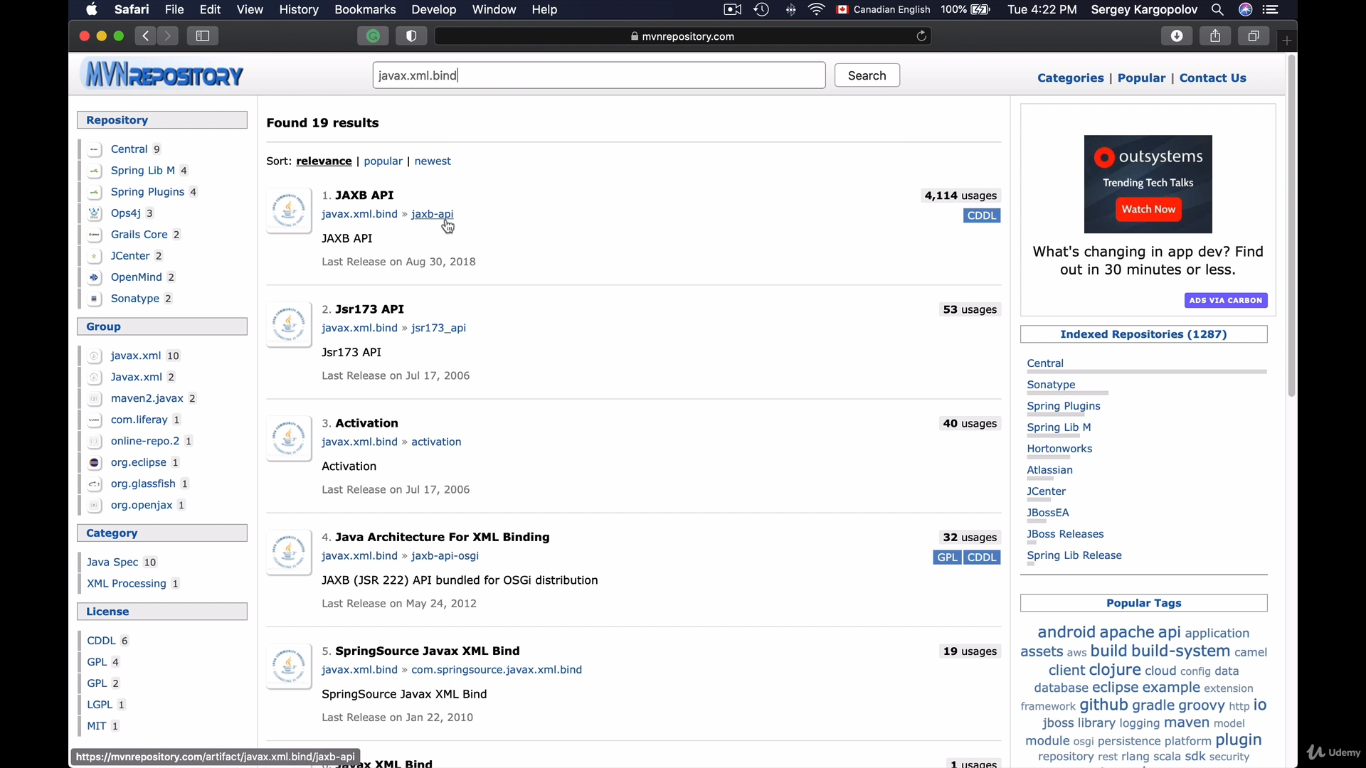


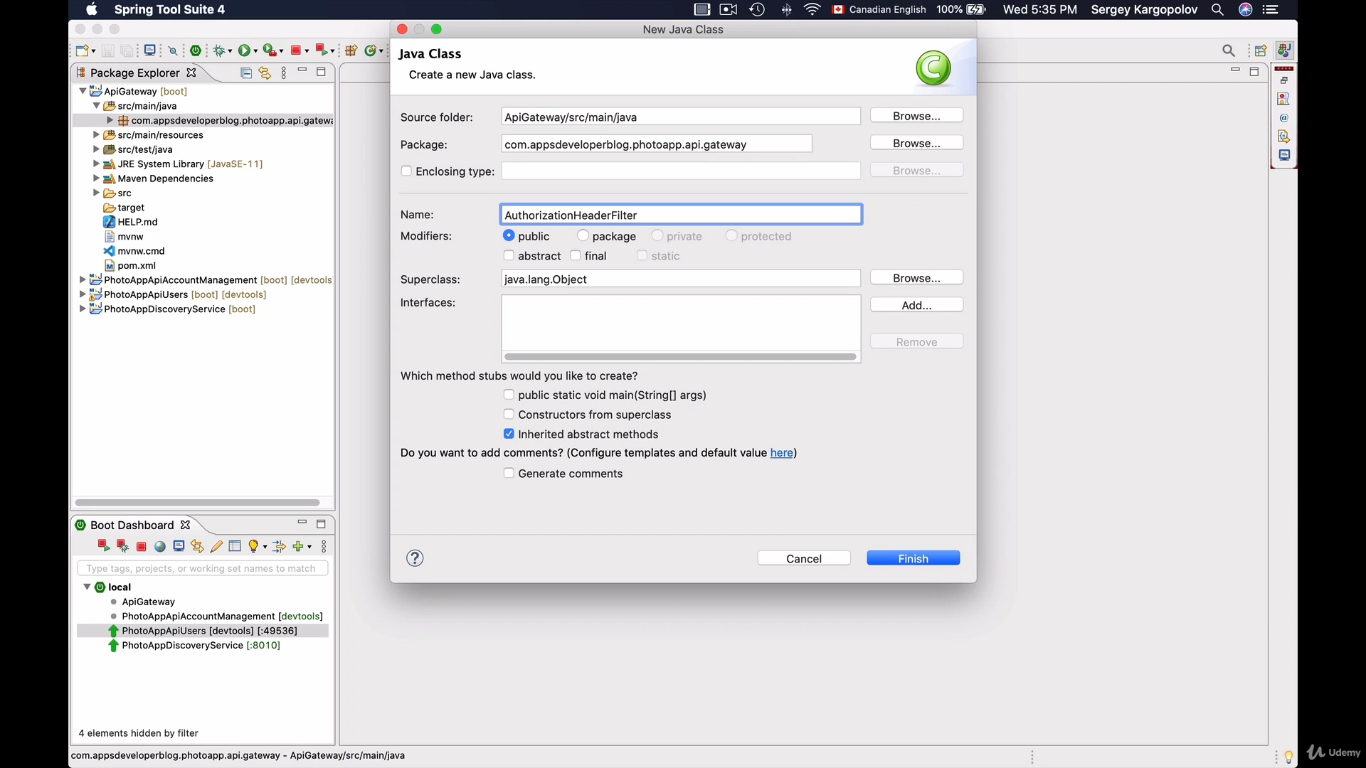
**Try to run the application at this moment, without providing the header “Authorization” and any random value start from “Bearer”.**

**Creating a new Filter and adding support for JWT Validation.**

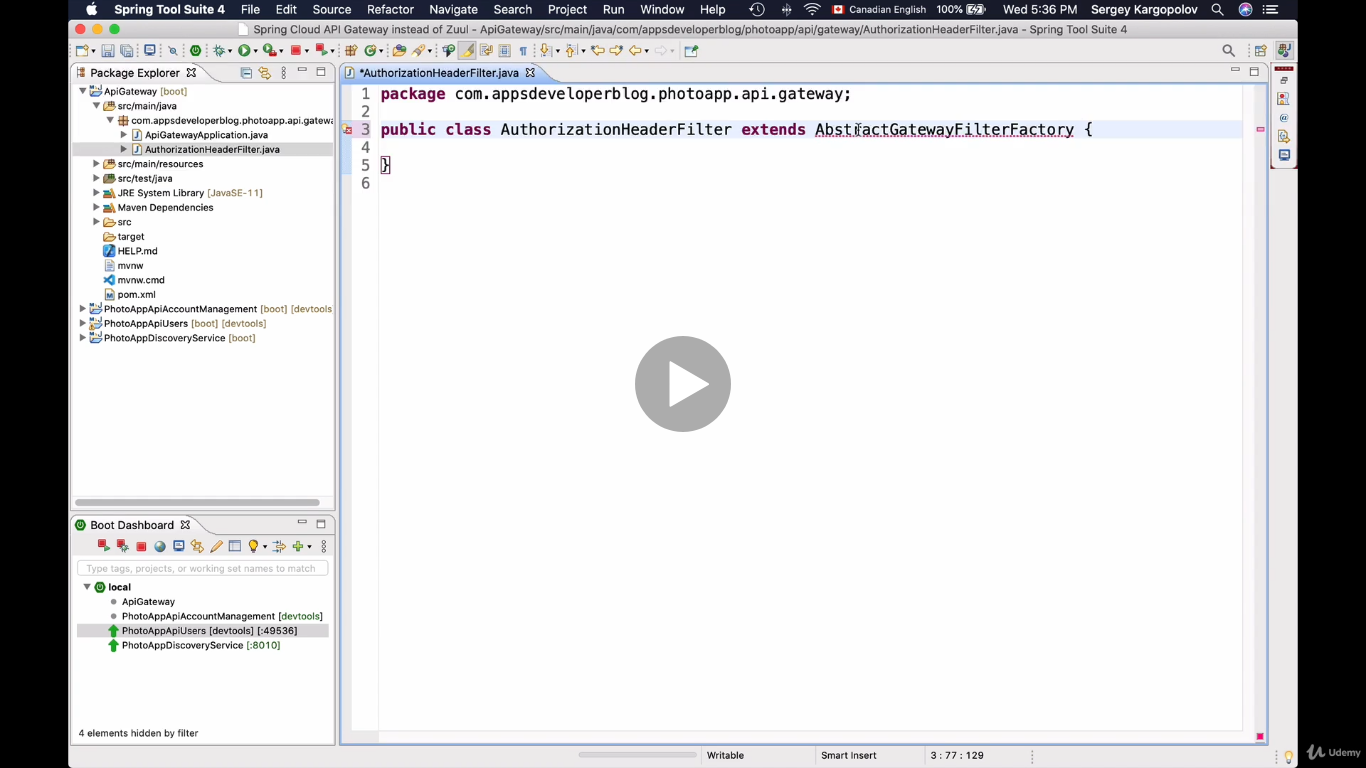


In this place we can either choose Jackson dataformat api

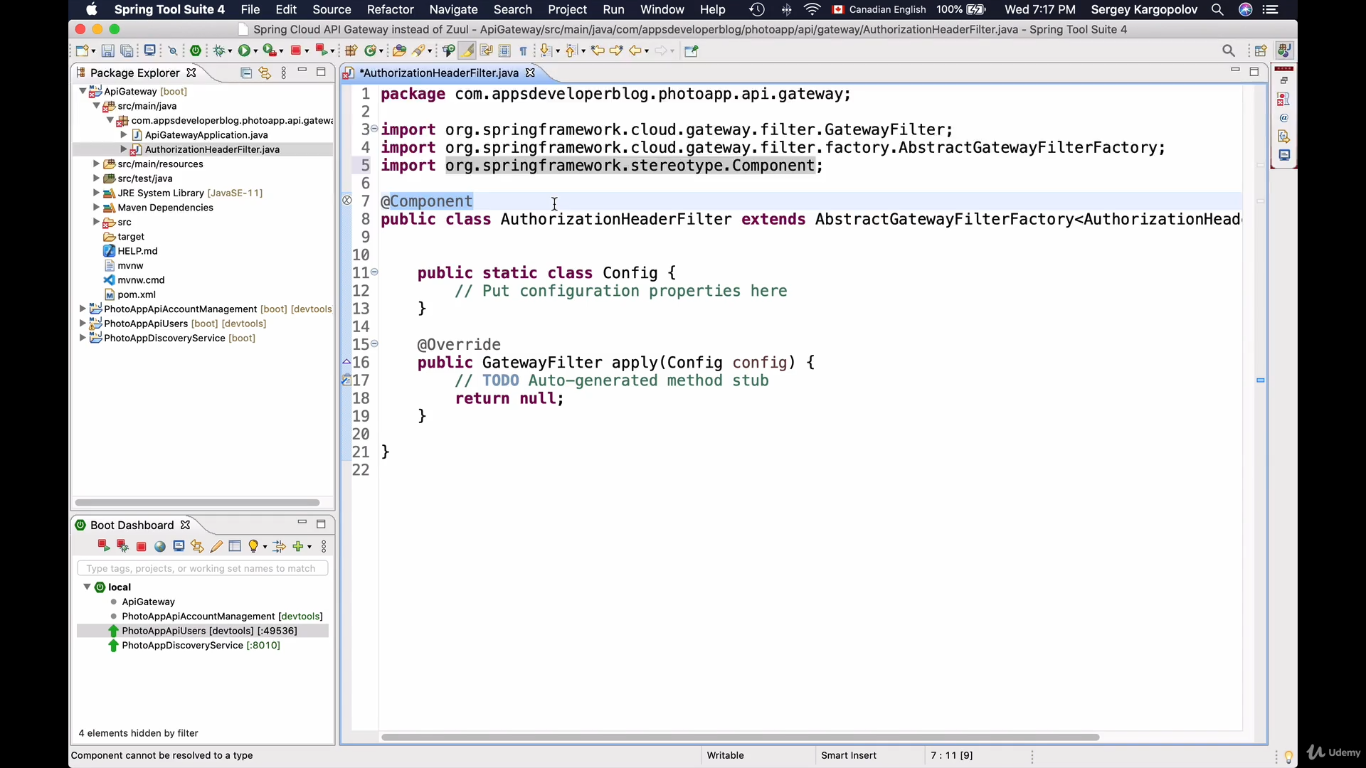




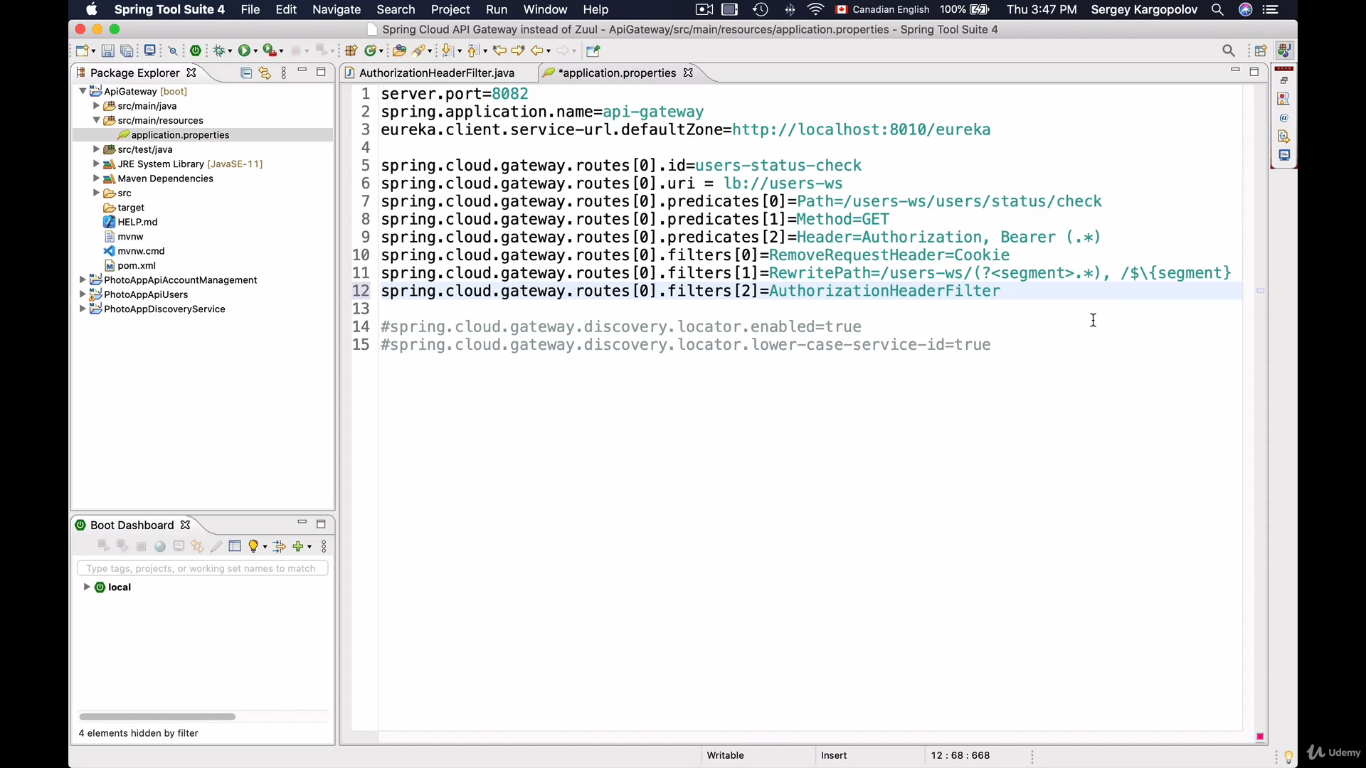
**“Abstract gateway filter factory” is used to intercept before a request to the destination is made.**



**Apply() method is where we have to enter our logic.**

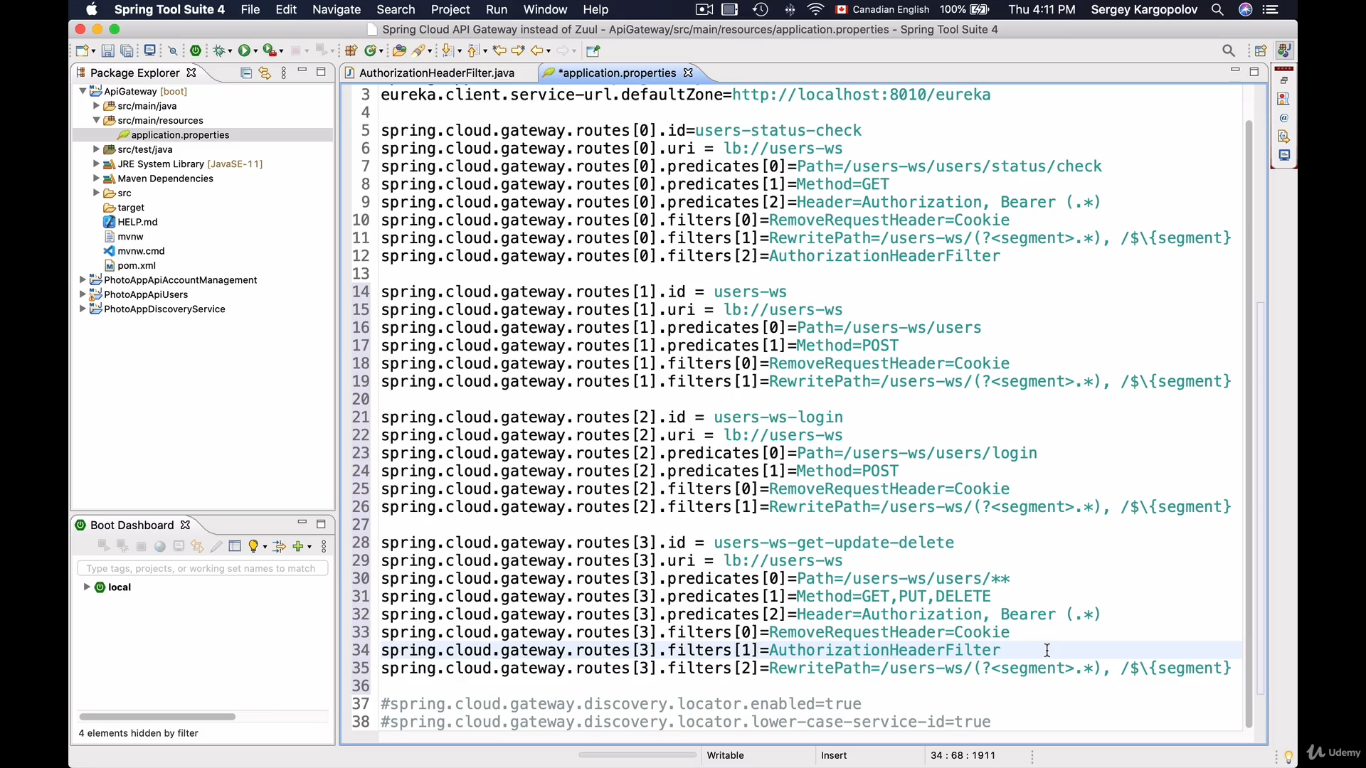


**Now we have custom filter class created, all we have to do is to assign it to a specific route for which I want this filter to be executed.**



**Spring.cloud.gateway.routes[0].filters[2]=AuthorizationHeaderFilter.**

**AuthorizationHeaderFilter class is our custom filter class name, now this filter will particularly executes only when request is routed to this path. If we have other routes configured in this properties file then we have to explicitly mention this filter over there as well.**



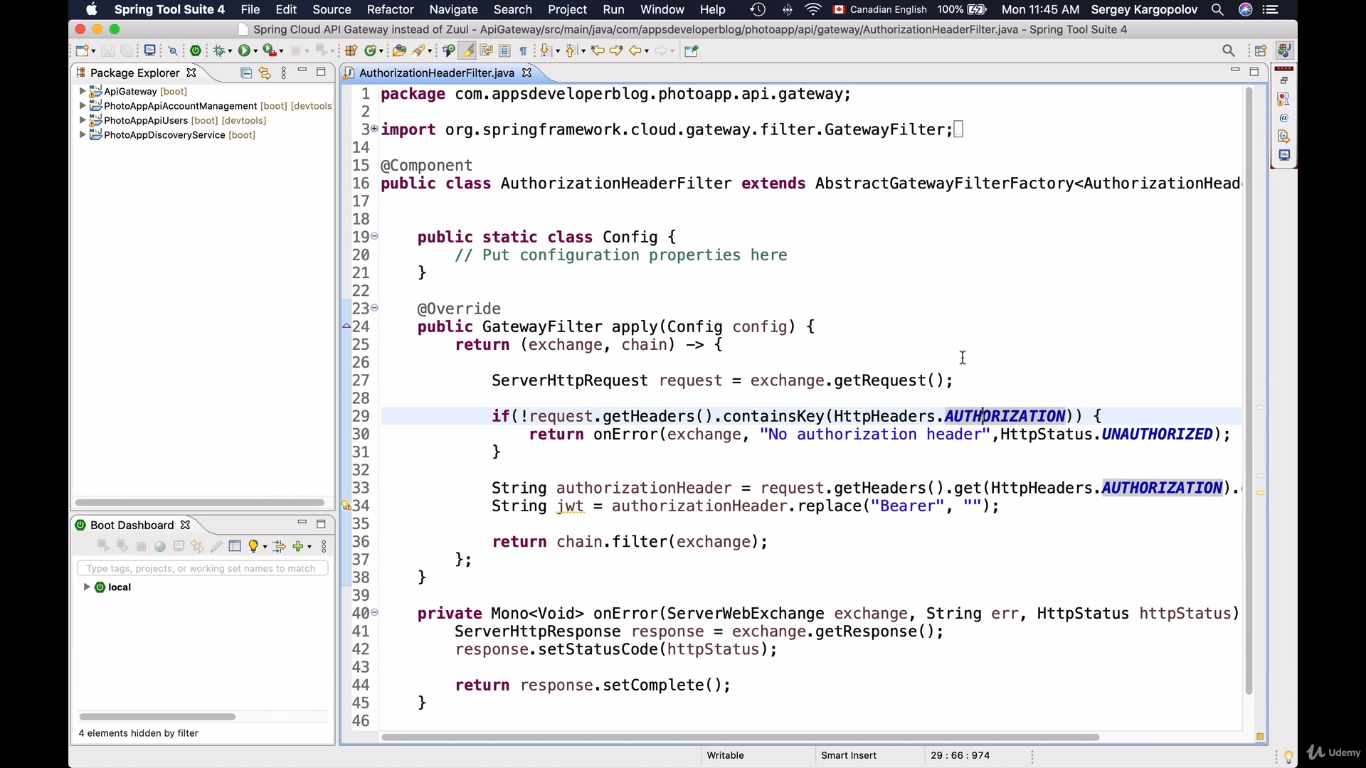
For example, we have added few new routes, add added the filters explicitly wherever is required.

**routes[0] is welcome page, requires validation**

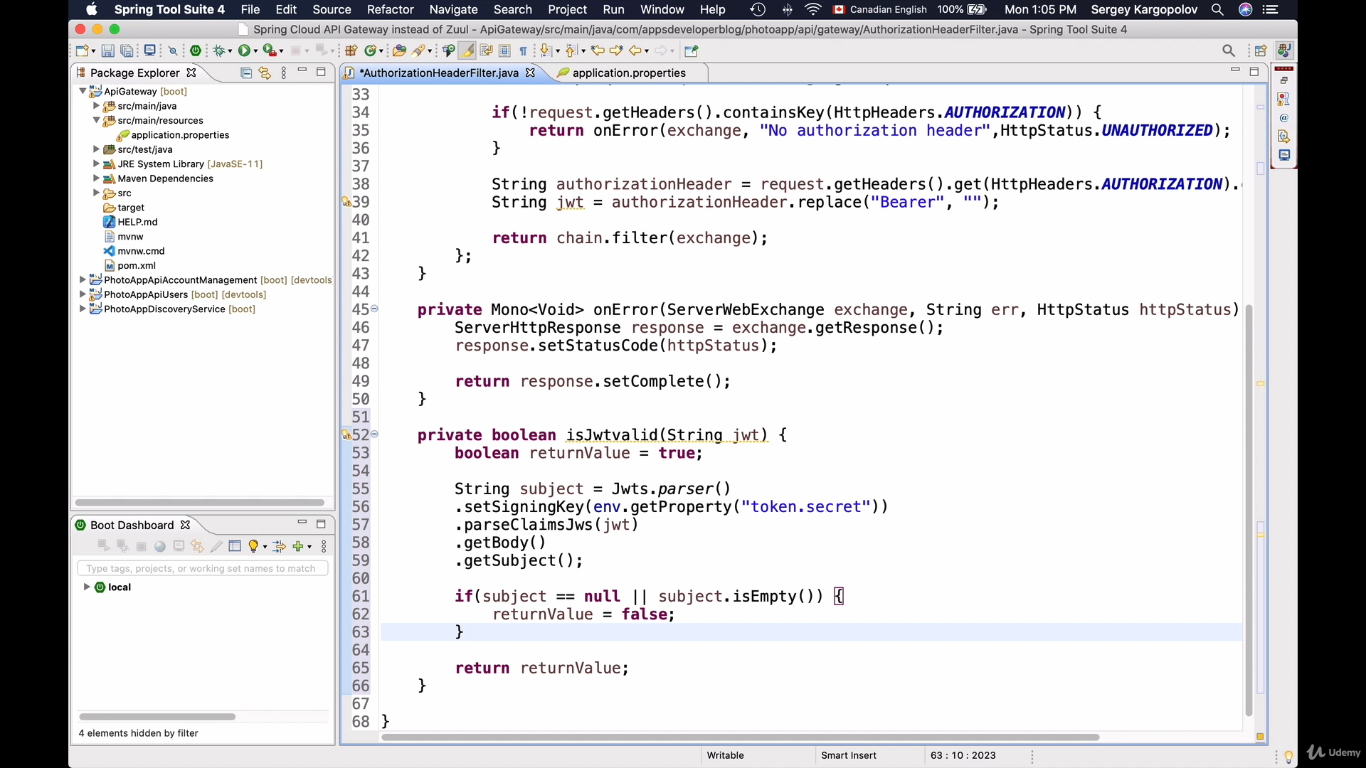
**routes[1] is register user page, doesn’t require validation**

**routes[2] is login page, doesn’t require validation**

**routes[3] is read, update, delete user, requires validation**



In above image we are implementing the logic to read the jwt token passed request. “GatewayFilter” is an interface contains one method with 2 parameters exchange and chain. We are using lambda functions to implement this method. Exchange parameter contains the request content and chain parameter is passes the request to next filter chain. We are getting the request, read it contents, if request doesn’t contain a header “Authorization” then we will return the error else we will check the jwt token by removing the work bearer from it.



In the above image, isJwtvalid() is validating the jwt key received in the request with the setSigning key and subject which is nothing but the userId.

Here we are not validating the userid that is subject. We can validate it using the method -level security rather than calling DB to validate.

Run the application at this point and test the validation of JWT.