Chain of Responsibility

We have decided to also include the Chain of Responsibility. We include this in our system as it helps with the handling of authentication request. When someone request functionality of authentication we have to make sure that they are in fact who they say that they are otherwise they can use functionality not made for them. This is why we opt to use Chain of Responsibility so that we can check every request that comes through our microservice if they are authenticated correctly. It doesn't matter to us what kind of request it is. If you want access to our microservice you have to be authenticated first.

We implemented the chain of responsibility with the help of Spring web security. Web security allows you to implement filters. These filters are then chained together, call each other and can change the response if necessary. With our function, instead of giving an error when things go wrong, they edit the response so that you cannot access what you were trying to access. So if you try to access authentication without a valid jwt token your access shall be denied

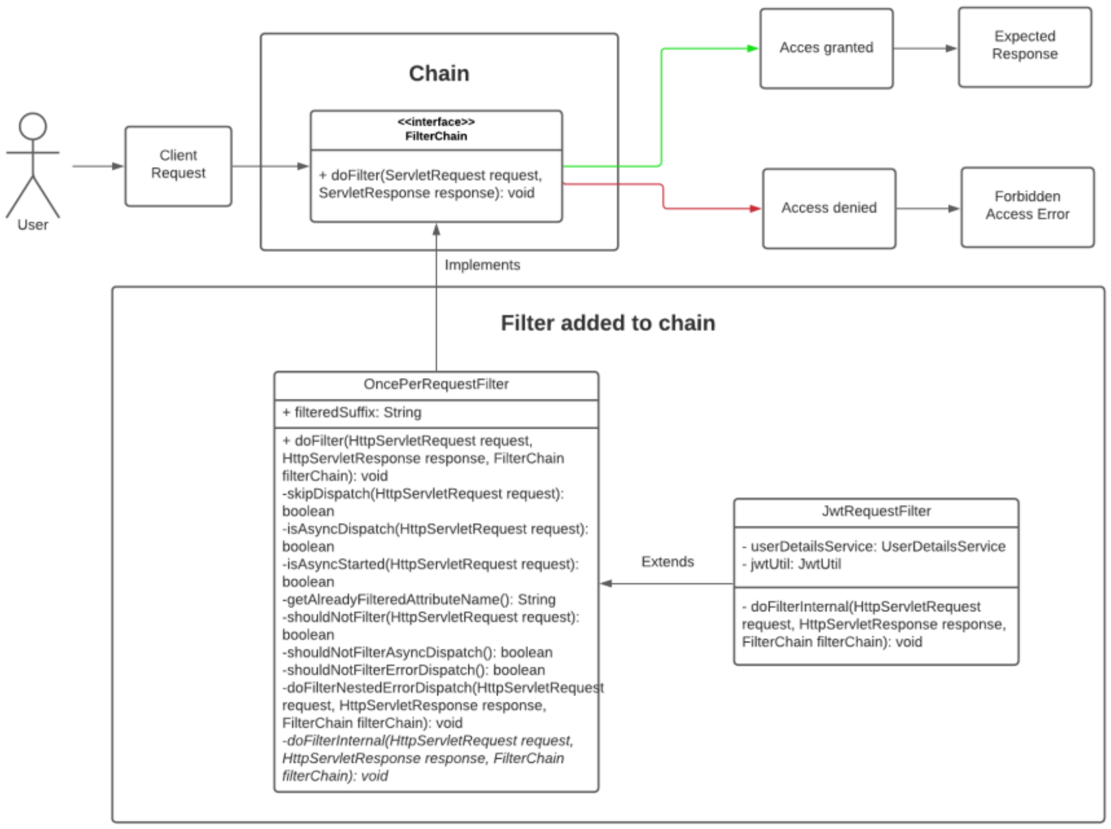
This brings a few very nice advantages and a few disadvantages. The first advantage being that whenever you request something the endpoint of that request doesn't have to bother checking if you are a valid user as it can assume it has passed the chain and therefor is rightly authenticated.

The second reason we use a chain is when in the future we might want to also test request on different things. We don't know what kind of things a request needs to be checked for in the future but if things pop up we can simple at a filter and all request will use this new filter.

There are however a few disadvantages. The first and foremost being the fact that users have to be able to authenticate themselves and if we need every request to be authenticated this obviously cannot work. As you cannot be authenticated before you have authenticated yourself. The same goes for registering. This means that we have to specify every request that doesn't have to be authenticated. Which is rather a lot of overhead.

The second disadvantage we have considered while making this chain is that request could get lost when the chain has an error and doesn't call the next method correctly. Therefor not changing the response correctly and allowing users by data they are not supposed to access. This security risk however we are aware of and we just have to make sure the chain is configured correctly when making changes to our program.

The implementation of the the chain of responsibility can be found in
sem-repo-31b/authentication/src/main/java/authentication
Where in the *filters package* you can find the code that is used within the chain. And in the *configure package* you can see that this filter is added to the code.
On the next page you can see the class diagrams for the chain of responsibility.


User
Client Request
Chain
<<interface>> FilterChain
+ doFilter(ServletRequest request, ServletResponse response): void
Acces granted
Expected Response
Access denied
Forbidden Access Error
Implements
Filter added to chain
OncePerRequestFilter
+ filteredSuffix: String
+ doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain): void
-skipDispatch(HttpServletRequest request): boolean
-isAsyncDispatch(HttpServletRequest request): boolean
-isAsyncStarted(HttpServletRequest request): boolean
-getAlreadyFilteredAttributeName(): String
-shouldNotFilter(HttpServletRequest request): boolean
-shouldNotFilterAsyncDispatch(): boolean
-shouldNotFilterErrorDispatch(): boolean
-doFilterNestedErrorDispatch(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain): void
-*doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain): void*
Extends
JwtRequestFilter
- userDetailsService: UserDetailsService
- jwtUtil: JwtUtil
- doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain): void