

user request -> authentication filter -> authentication manager -> authentication provider -> user details service

||

password encoder

1. end user sends req to our REST app, auth filter intercepts this request. It is

servlet filter class which will check if the user has authentication.

2. If not, the request is passed to auth mgr to check if the user details are valid.

3. auth mgr uses auth provider where login logic is there.

4. auth provider will use user details service to retrieve pwd.

5. passcode encoder is used to encrypt pwds and store.

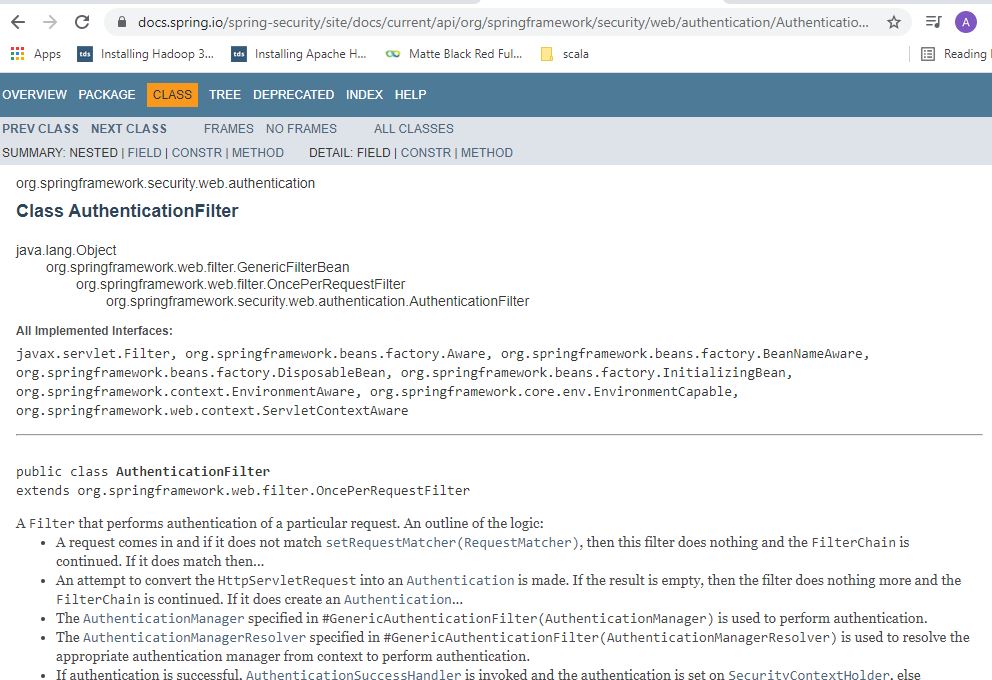
6. if the details match with user details service, auth filter will

use authenticationSuccessHandler and stores authentication info in the user

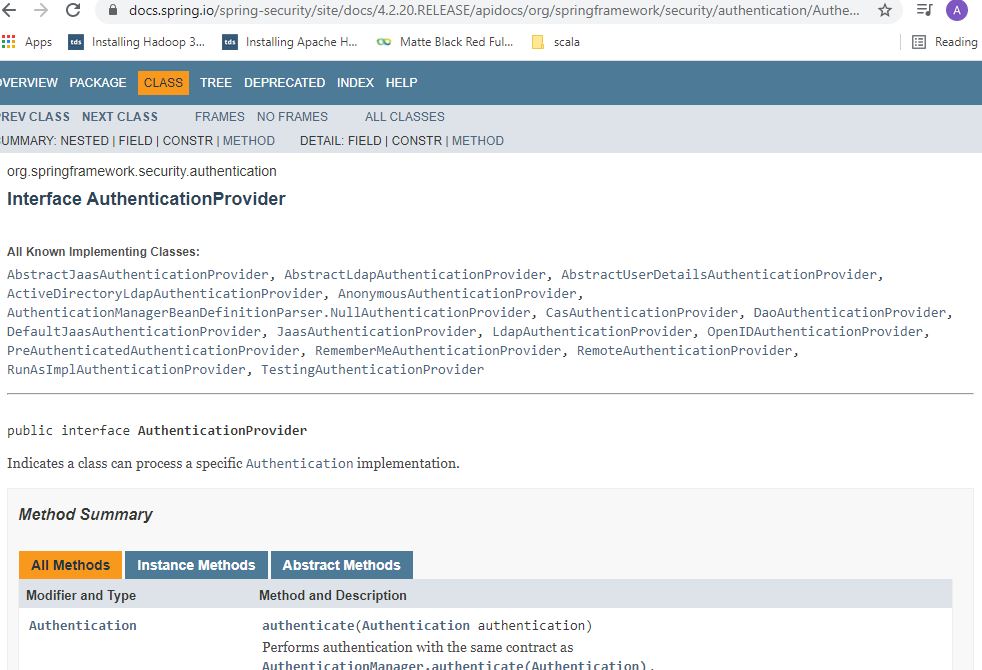
entity in security context.

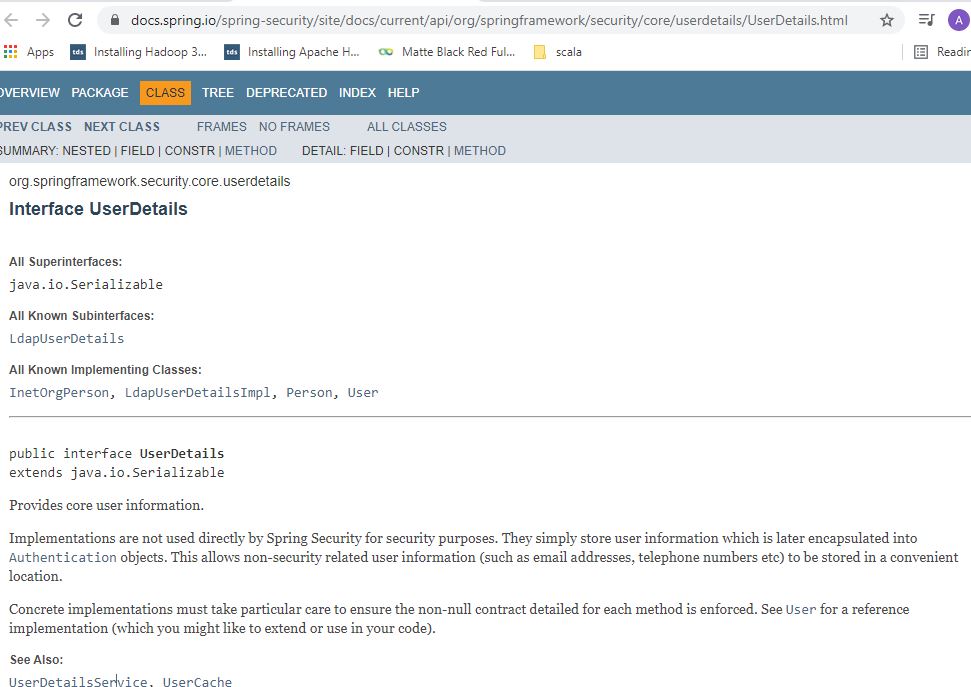
7. in case of failure, authenticationFailureHandler will be sent by auth filter

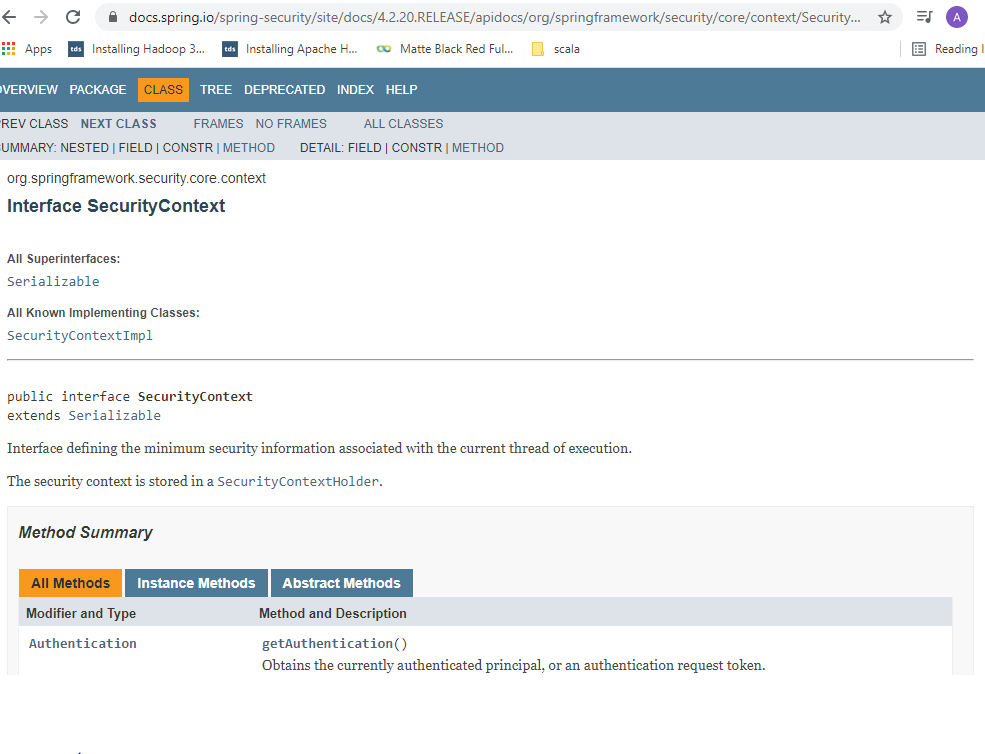
Check for Spring Authentication Filter class in google



Spring Authentication provider interface







**Important note on cookies and basic auth:**

In Basic auth, once you provide the username and pwd, it will get authenticated and send jsessionid in cookies. For subsequent requests, this jsessionid is passed and hence even if you change headers username and pwd, it will work as the Authentication filter in spring uses the jsessionid to validate request.

**OAuth2**

4 workflows/ grant types:

* **Authorization code:** (E.g. Clear Tax uses google authorization server. So client app ClearTax is pre-registered with Google and has a unique client-id, also it consists of redirect-url and client-secret. Authorization server will generate a token and send it to client app).
* **Password:** (The only difference with Authorization code approach is that user inputs username and password on client app page instead of Google page. The client app will send these details along with client-id and client-secret which it obtained when it pre-registered with Google. Disadvantage is that client-app is aware about client username on Google).
* **Client credentials:** (Here user is not in the picture. It is for one app having access to another client application. Typically used in single sign-on micro-services architecture).
* **Refresh token:** (For the above 3 approaches, tokens that are granted by Auth server have a life time. Once token expire, client-app will ask the user to again login to Google. In Refresh token method, Auth server will also return a refresh token. Client token will use refresh token and ask Auth server to send another token. So user does not have to login again to Google).

After Spring boot 2.4.0 version, the OAuth2 dependency has been removed by Spring framework and they want to merge it with spring.security package. Currently it is in spring cloud package.