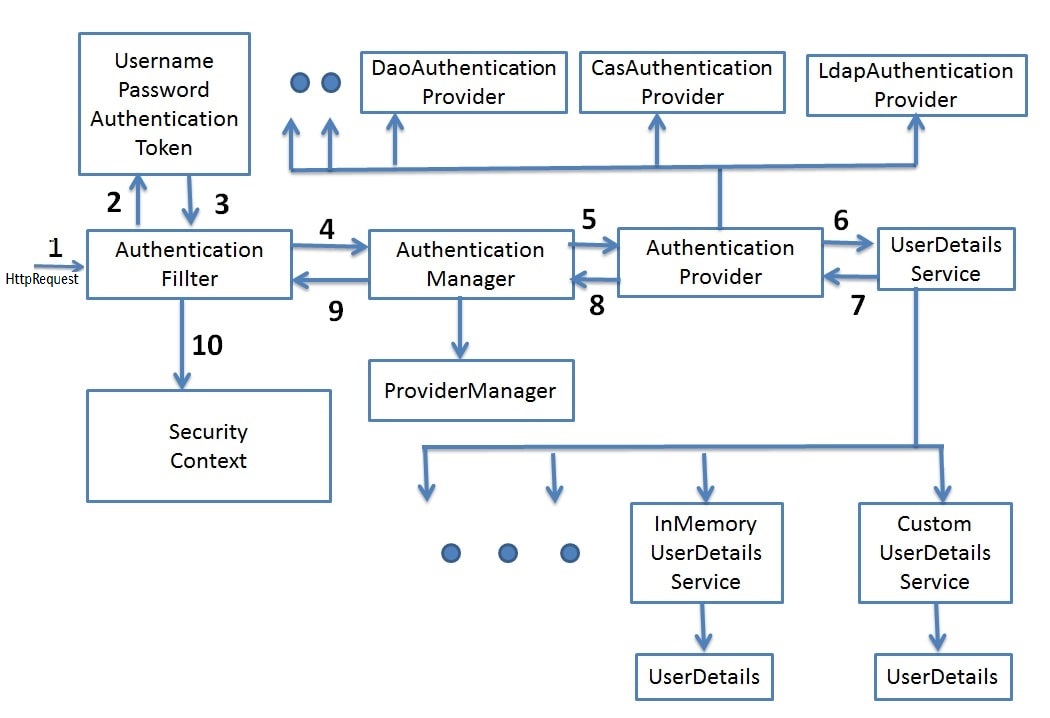
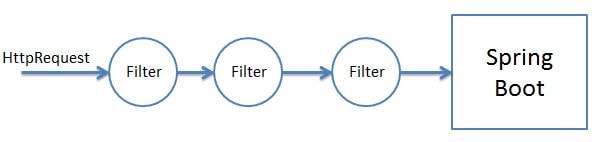
# **Understanding Spring Security Architecture**

From: <https://www.javainuse.com/webseries/spring-security-jwt/chap3>

Let us understand how Spring Security Works.

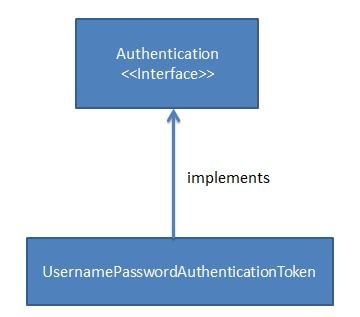


* **Filters -** Before the request reaches the Dispatcher Servlet, it is first intercepted by a chain of filters.



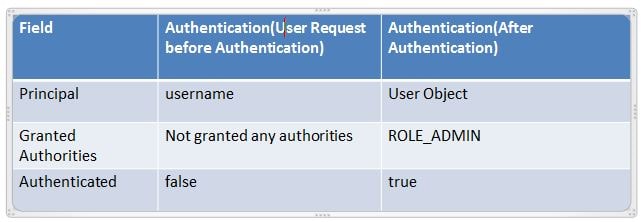
These filters are responsible for Spring Security. So any incoming request will go through these filters and it is here that authentication and authorization takes place. Based on the type of requests there are different Authentication Filters like the **BasicAuthenticationFilter**, **UsernamePasswordAuthenticationFilter** etc

* **Authentication Object Creation -** When the request is intercepted by the appropriate AuthenticationFilter it retrieves the username and password from the request and creates the **Authentication Object**. If the extracted credentials are username and password, then **UsernamePasswordAuthenticationToken** is created.

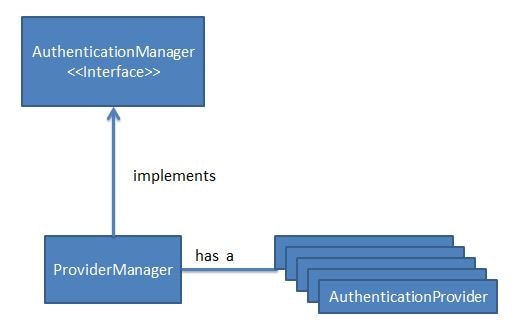


* **AuthenicationManager -** Using the Authentication Object created the filter will then call the authenticate method of the Authentication Manager. The Authentication Manager is only an interface and actual implementation of the authenticate method is provided by the **ProviderManager**.   
  

Important point to note here is that the Authentication Manager takes an Authentication object as input and after successful authentication again returns an object of type Authentication.



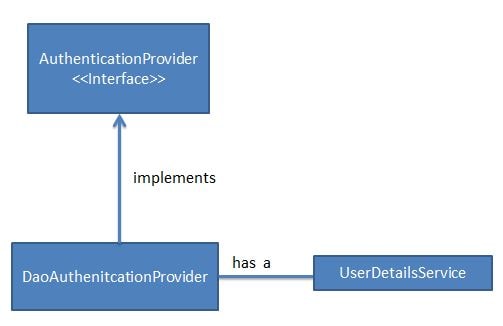
The ProviderManager has a list of **AuthenticationProviders**. From it's authenticate method it calls the authenticate method of the appropriate AuthenticationProvider. In response, it gets the **Principal Authentication Object** if the authentication is successful. Note that **Principal** means in here the current authenticated user.



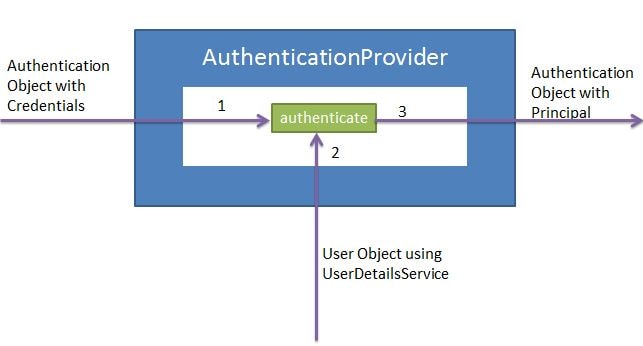
* **AuthenticationProvider -** The AuthenicationProvider is an interface with a single authenticate method.



It has various implementations like **CasAuthenticationProvider**, **DaoAuthenticationProvider**. Depending on the implementation an appropriate AuthenicationProvider implementation is used. It is in the AuthenticationProvider Implementation authenticate method where all the actual authentication takes place.

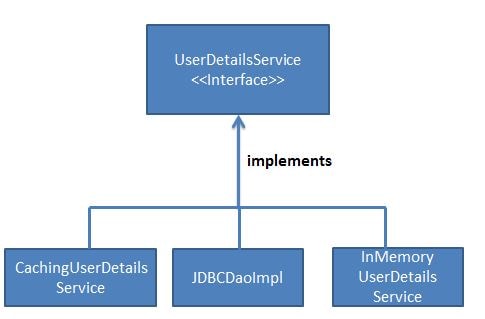


Using the UserDetails service the AuthenticationProvider fetches the User Object corresponding to the username. It fetches this User Object from either a database, internal memory or other sources. This User object credentials are then compared with the incoming Authentication Object credentials. If Authentication is successful then the Principal Authentication Object is returned in response.



* **UserDetailsService -** The UserDetailsService is an interface having a single method named loadUserByUsername.



It has various implementations CachingUserDetailsService, JDBCDaoImpl etc. Based on the implementation an appropriate UserDetailsService is called.   


It is responsible for fetching the User Object with username and password against which the incoming User Object will be compared.

/\*

To implement authentication (login) feature, we need to create a classes of

subtypes of the interfaces "UserDetails" and "UserDetailsService" (defined by Spring Security)

to represent an user authentication.

Spring Security will invoke methods in this class during the authentication process.

"UserDetailsService" interface:

- Purpose: The UserDetailsService interface is responsible for retrieving user details from

a data store (such as a database or an in-memory data structure) based on a username.

It is a part of the Spring Security framework and is used to load user-specific data during

the authentication process.

- Methods: The primary method in UserDetailsService is loadUserByUsername(String username),

which returns a UserDetails object containing information about the user, including the

user's username, password, and granted authorities.

"UserDetails" interface:

- Purpose: The UserDetails interface represents the core user information returned by

the UserDetailsService. It contains essential information about a user, such as the

username, password (or a password-encoded representation), and a collection of

granted authorities (roles).

- Methods: The UserDetails interface defines several methods, including getUsername(),

getPassword(), getAuthorities(), and others.

\*/