**Spring Security Authentication Flow - Step-by-Step**

**1. User Submits Login Request**

* The user enters email and password in the login form.
* The request is sent to Spring Security’s authentication process.

Example Request:

POST /login

email: user@example.com

password: secret

**2. UsernamePasswordAuthenticationFilter Intercepts the Request**

* This filter extracts the email and password from the request.
* It creates an UsernamePasswordAuthenticationToken containing these credentials.

Spring Security Code (internally executed):

UsernamePasswordAuthenticationToken authenticationToken =

new UsernamePasswordAuthenticationToken(email, password);

* The token is now passed to the AuthenticationManager.

**3. AuthenticationManager Calls DaoAuthenticationProvider**

* The AuthenticationManager delegates authentication to DaoAuthenticationProvider.

Spring Security Code:

Authentication authResult = authenticationManager.authenticate(authenticationToken);

**4. DaoAuthenticationProvider Calls BlogUserDetailsService**

* Calls loadUserByUsername(email) to fetch user details from the database.

Example Code in BlogUserDetailsService:

@Override

public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

return userRepo.findByEmail(email)

.map(BlogUserDetails::new)

.orElseThrow(() -> new UsernameNotFoundException("User not found: " + email));

}

* If the user exists, BlogUserDetails is returned with user object if it exist in database.

**5. Password is Verified**

* Compares the entered password with the hashed password from the database.
* Uses a PasswordEncoder (e.g., BCryptPasswordEncoder) for validation.

Spring Security Code (internally executed):

if (!passwordEncoder.matches(enteredPassword, user.getPassword())) {

throw new BadCredentialsException("Incorrect password");

}

* If the password matches, authentication continues.

**6. User Account Status is Checked**

* Spring Security calls the following methods in BlogUserDetails to check account validity:

| **Method** | **Purpose** | **Implementation** |
| --- | --- | --- |
| isAccountNonExpired() | Checks if the account is expired | true (account never expires) |
| isAccountNonLocked() | Checks if the account is locked | true (account never locks) |
| isCredentialsNonExpired() | Checks if the password is expired | true (password never expires) |
| isEnabled() | Checks if the user is active | true (user is always enabled) |

Spring Security Code (internally executed):

if (!userDetails.isAccountNonExpired() ||

!userDetails.isAccountNonLocked() ||

!userDetails.isCredentialsNonExpired() ||

!userDetails.isEnabled()) {

throw new DisabledException("User account is not active");

}

* If all checks pass, the user is fully authenticated.

**7. User Roles and Authorities are Loaded**

* Calls getAuthorities() to load user roles and permissions.

Example Code in BlogUserDetails:

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return List.of(new SimpleGrantedAuthority("ROLE\_USER"));

}

* Spring Security uses these roles to enforce access control.

**8. Authentication is Stored in SecurityContextHolder**

* Spring Security stores the authenticated user in the session.

Spring Security Code:

SecurityContextHolder.getContext().setAuthentication(authentication);

* Now, the user remains authenticated for future requests.

**9. User is Redirected or Receives a Token**

* If authentication is successful:
  + For Web Apps: Redirects to /home.
  + For REST APIs: Returns a JWT token.

Web App Redirect Example:

response.sendRedirect("/home");

REST API Response Example:

{

"token": "eyJhbGciOiJIUzI1NiIsIn..."

}

**10. Future Requests Use Authentication**

* On every request, Spring Security checks if the user is already authenticated.
* If authenticated, the request is processed normally.
* Otherwise, the user is redirected to login.

Spring Security Code (internally executed):

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

if (auth == null || !auth.isAuthenticated()) {

response.sendRedirect("/login");

}

**Final Summary (Step-by-Step Process)**

| **Step** | **Action** |
| --- | --- |
| 1. User submits login form | Sends email & password to /login |
| 2. UsernamePasswordAuthenticationFilter intercepts | Extracts email & password, creates authentication token |
| 3. AuthenticationManager processes authentication | Calls DaoAuthenticationProvider |
| 4. BlogUserDetailsService fetches user | Calls loadUserByUsername(email) |
| 5. Password is verified | Compares entered password with stored hash |
| 6. Account status is checked | Calls isEnabled(), isAccountNonLocked(), etc. |
| 7. Roles & authorities are loaded | Calls getAuthorities() (ROLE\_USER) |
| 8. Authentication is stored | SecurityContextHolder stores authentication |
| 9. User is redirected or receives JWT | Web apps → Redirect, APIs → JWT Token |
| 10. Future requests are authenticated | Checks session or JWT for authentication |

**Next Steps**

Currently, every user has only "ROLE\_USER".

* To implement **role-based authentication**, the roles can be dynamically loaded from the database.
* Additional security configurations (e.g., JWT authentication) can be added for REST APIs.