SpringBoot Security - Part4 (Basic Authentication)

Monday, 24 March 2025 12:01 PM

Basic Authentication

- It's a Stateless Authentication method.
  - Stateless authentication means, server do not maintains the user authentication state

- In this, client has to pass the username and password with every request using Authorizati

  Authorization: Basic <base64(username:password)>

- These Credentials are encoded using Base64 (not encrypted), making it insecure over HTTF

Let's go step by step:

1st: Lets create a user

- Already discussed, all possible ways to create user and how we can create users dynamically approach.
- Also how to create and store it in DB or in-memory. So kindly check that out if there is any d

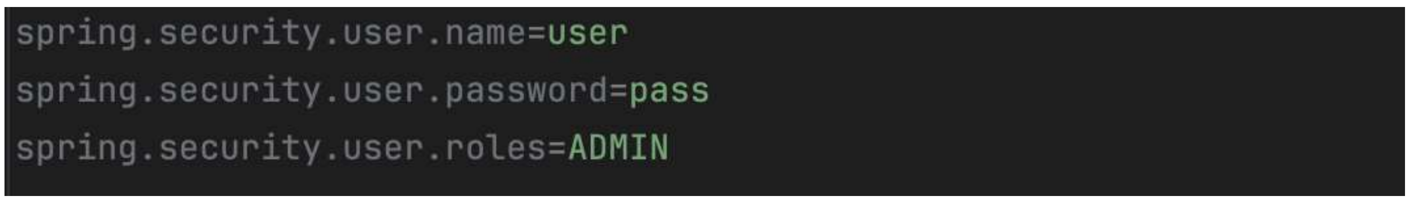
SELECT * FROM USER_AUTH;

ID	PASSWORD	ROLE	USERNAME
1	$2a$10$euzihUhyp4exMejDkyDb0eK2q49sqTcG8EShOnT2GmaL7lxvIeOfO	ROLE_USER	my_username

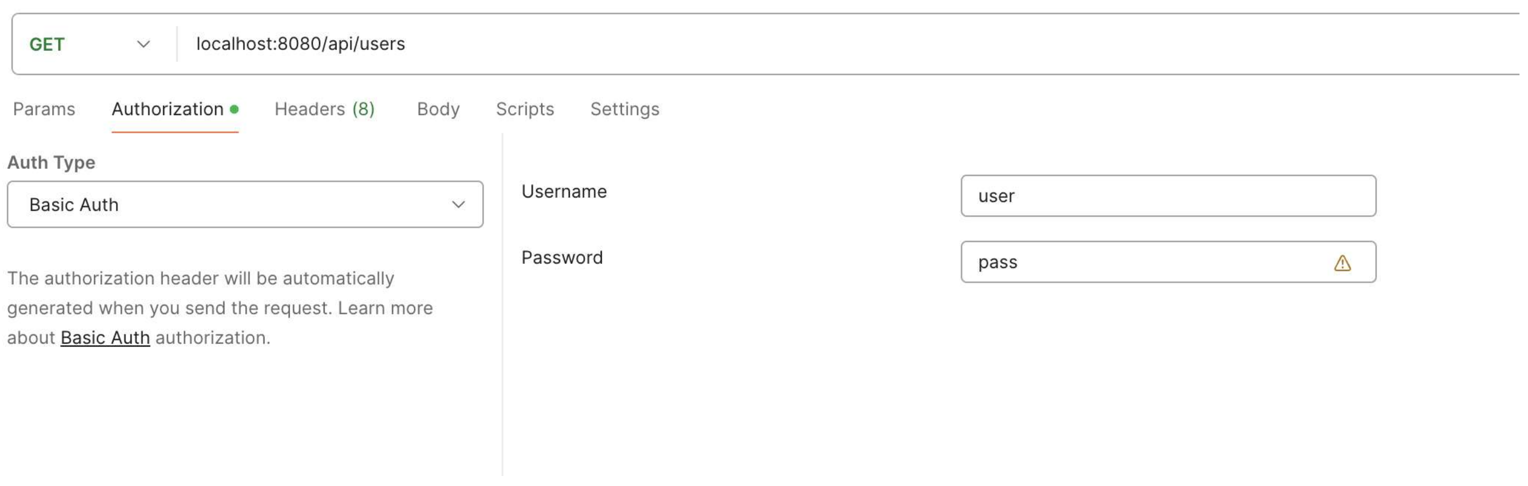
Lets create a user for testing purpose:

Application.properties

```
#creating username and password and assigning the ROLE to the user
```



```
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=ADMIN
```



GET localhost:8080/api/users

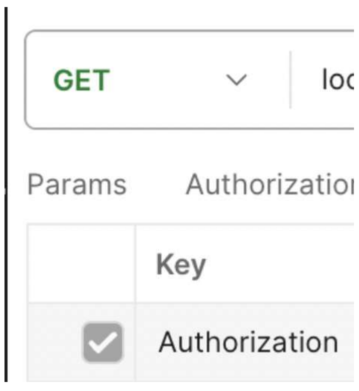
Params | Authorization ● | Headers (8) | Body | Scripts | Settings

Auth Type: Basic Auth

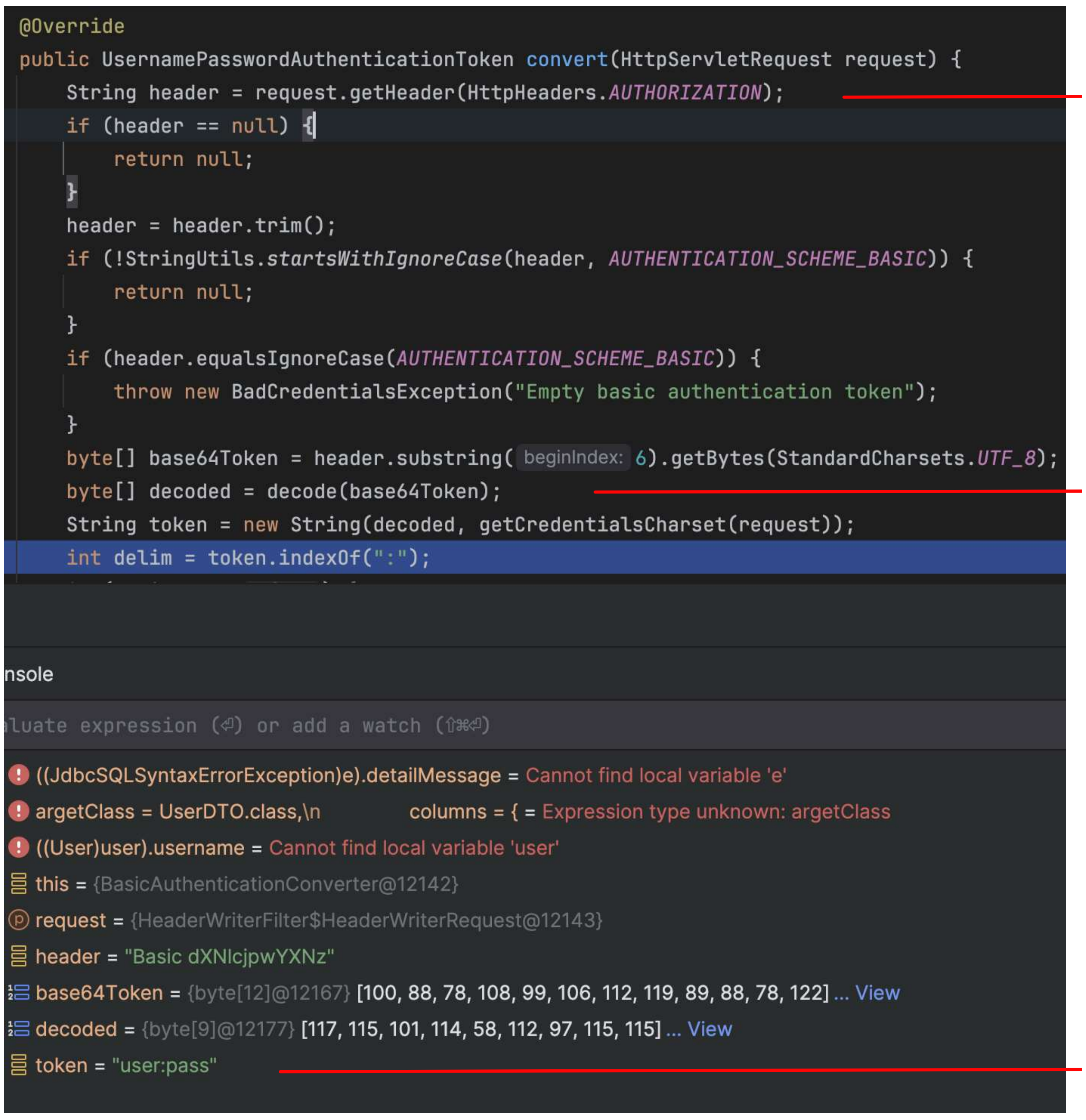
The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: user

Password: pass



BasicAuthenticationFilter.java



```
@Override
public UsernamePasswordAuthenticationToken convert(HttpServletRequest request) {
    String header = request.getHeader(HttpHeaders.AUTHORIZATION);
    if (header == null) {
        return null;
    }
    header = header.trim();
    if (!StringUtils.startsWithIgnoreCase(header, AUTHENTICATION_SCHEME_BASIC)) {
        return null;
    }
    if (header.equalsIgnoreCase(AUTHENTICATION_SCHEME_BASIC)) {
        throw new BadCredentialsException("Empty basic authentication token");
    }
    byte[] base64Token = header.substring( beginIndex: 6).getBytes(StandardCharsets.UTF_8);
    byte[] decoded = decode(base64Token);
    String token = new String(decoded, getCredentialsCharset(request));
    int delim = token.indexOf(":");
```

```
nsole
luate expression (⏎) or add a watch (⇧⌘⏎)
((JdbcSQLSyntaxErrorException)e).detailMessage = Cannot find local variable 'e'
argetClass = UserDTO.class,\n          columns = { = Expression type unknown: argetClass
((User)user).username = Cannot find local variable 'user'
this = {BasicAuthenticationConverter@12142}
request = {HeaderWriterFilter$HeaderWriterRequest@12143}
header = "Basic dXNlcjpwYXNz"
base64Token = {byte[12]@12167} [100, 88, 78, 108, 99, 106, 112, 119, 89, 88, 78, 122] ... View
decoded = {byte[9]@12177} [117, 115, 101, 114, 58, 112, 97, 115, 115] ... View
token = "user:pass"
```

One question comes to mind is, why we need to send username and password in what not in Request body or any other way?

Possible reasons:

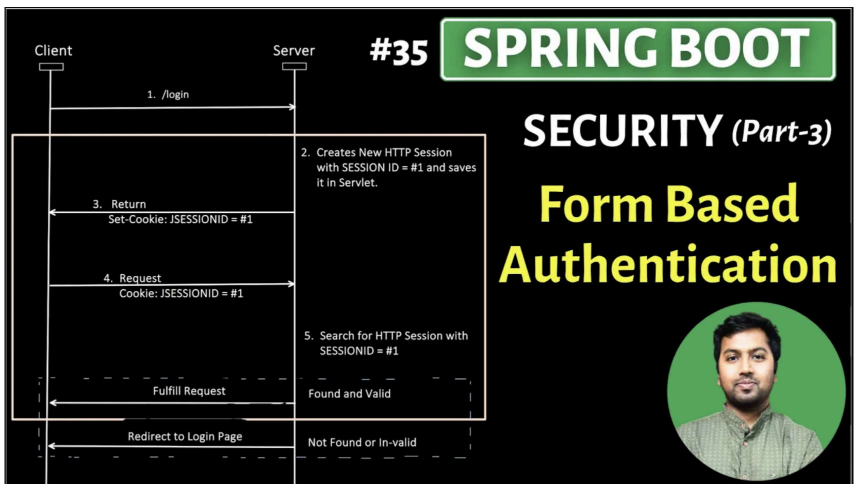
1. Standard:
As per HTTP Standardization (RFC 7617), this format is accepted, in order to make
standard across APIs and Clients.
Otherwise, if there is no Standard follows and some send in Headers, some in boc
when need to deal with multiple APIs and Clients.

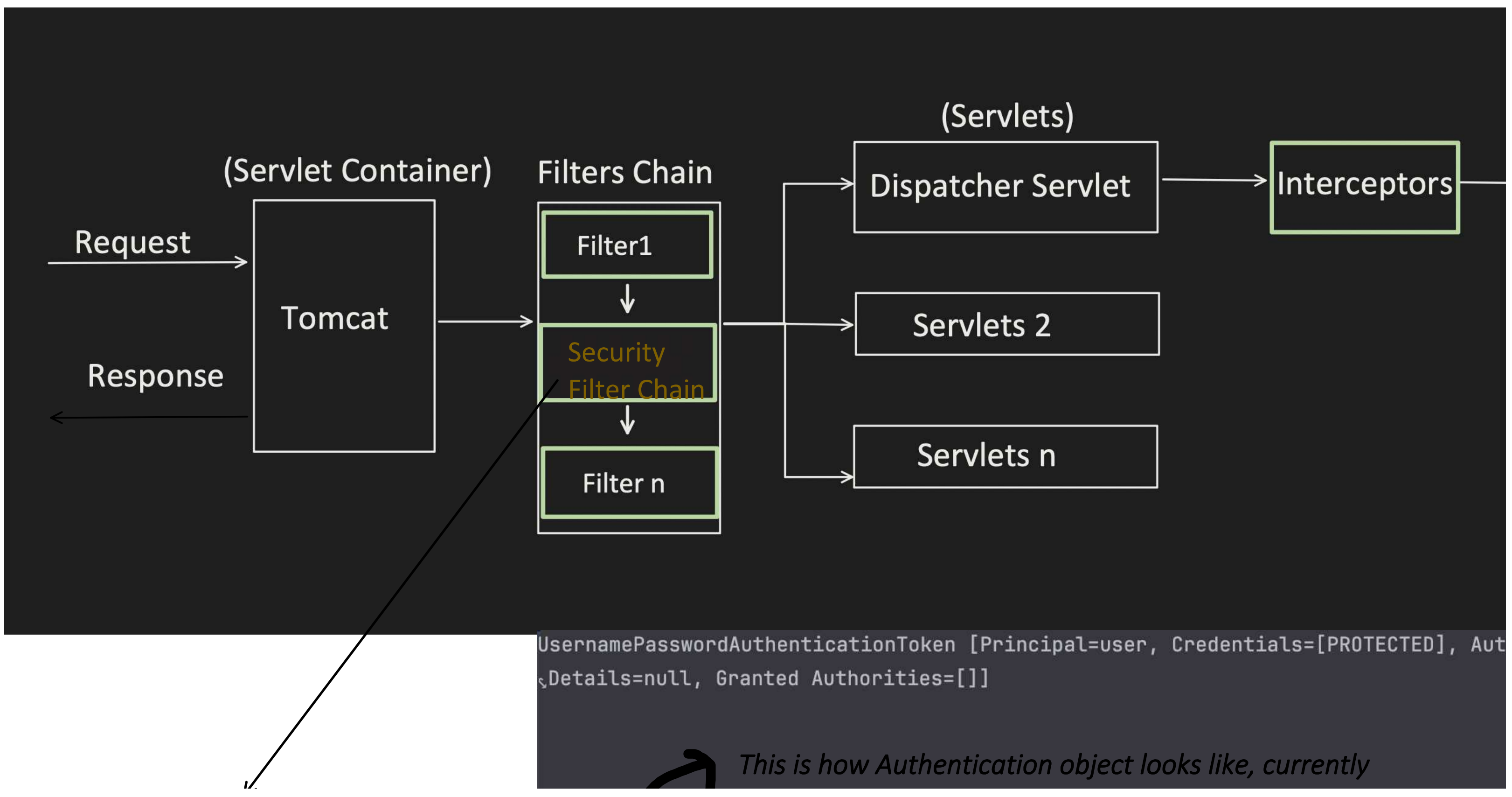
2. Security:
Web servers sometimes log the request body for debugging or analytics purpose.
But Headers are typically not logged. So this reduce the risk of exposures of client
password.

3. Support for all HTTP request:
Apart from POST & PUT, there are HTTP requests like GET which do not accept
with Headers for such APIs too, credentials can be sent consistently.

Now, lets understand the flow of Basic Authentication:

- If Form Based Authentication flow is understood properly, then this flow is very similar t

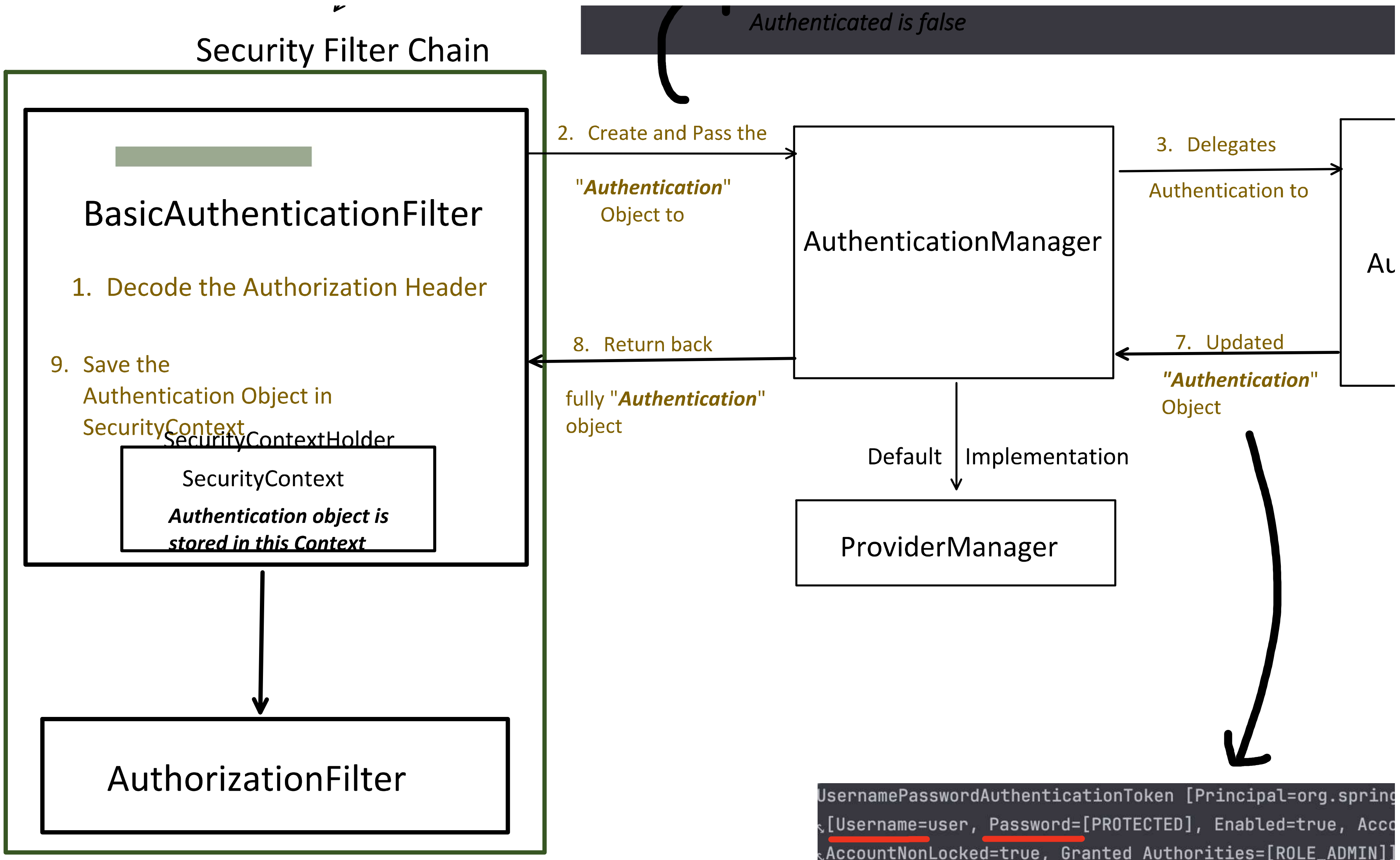

Client
Server
1. /login
2. Creates New HTTP Session with SESSION ID = #1 and saves it in Servlet.
3. Return Set-Cookie: JSESSIONID = #1
4. Request Cookie: JSESSIONID = #1
5. Search for HTTP Session with SESSIONID = #1
Fulfill Request
Found and Valid
Redirect to Login Page
Not Found or In-valid
#35 SPRING BOOT
SECURITY (Part-3)
Form Based Authentication



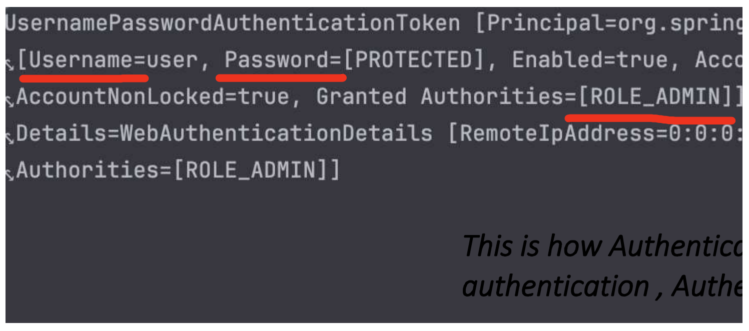
```
UsernamePasswordAuthenticationToken [Principal=user, Credentials=[PROTECTED], Aut
Details=null, Granted Authorities=[]]
```

This is how Authentication object looks like, currently

Authenticated is false



Security Filter Chain
BasicAuthenticationFilter
1. Decode the Authorization Header
9. Save the Authentication Object in SecurityContext
SecurityContextHolder
SecurityContext
Authentication object is stored in this Context
AuthorizationFilter
2. Create and Pass the "Authentication" Object to
AuthenticationManager
3. Delegates Authentication to
Au
8. Return back fully "Authentication" object
7. Updated "Authentication" Object
Default Implementation
ProviderManager

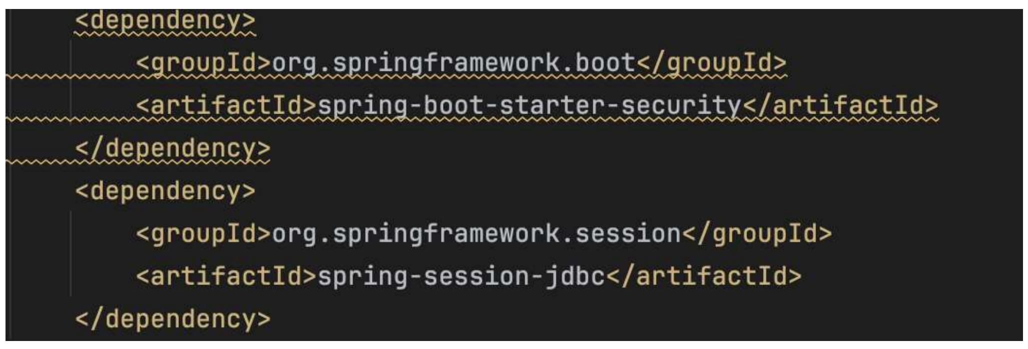


```
UsernamePasswordAuthenticationToken [Principal=org.spring
[Username=user, Password=[PROTECTED], Enabled=true, Acc
AccountNonLocked=true, Granted Authorities=[ROLE_ADMIN]]
Details=WebAuthenticationDetails [RemoteIpAddress=0:0:0:
Authorities=[ROLE_ADMIN]]
```

This is how Authentic
authentication , Authe

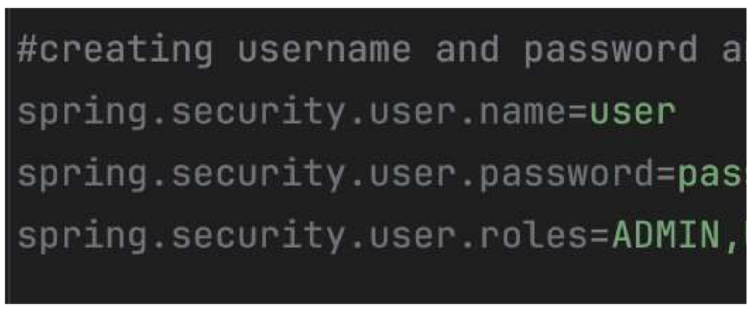
So, what we need to implement it?

Pom.xml

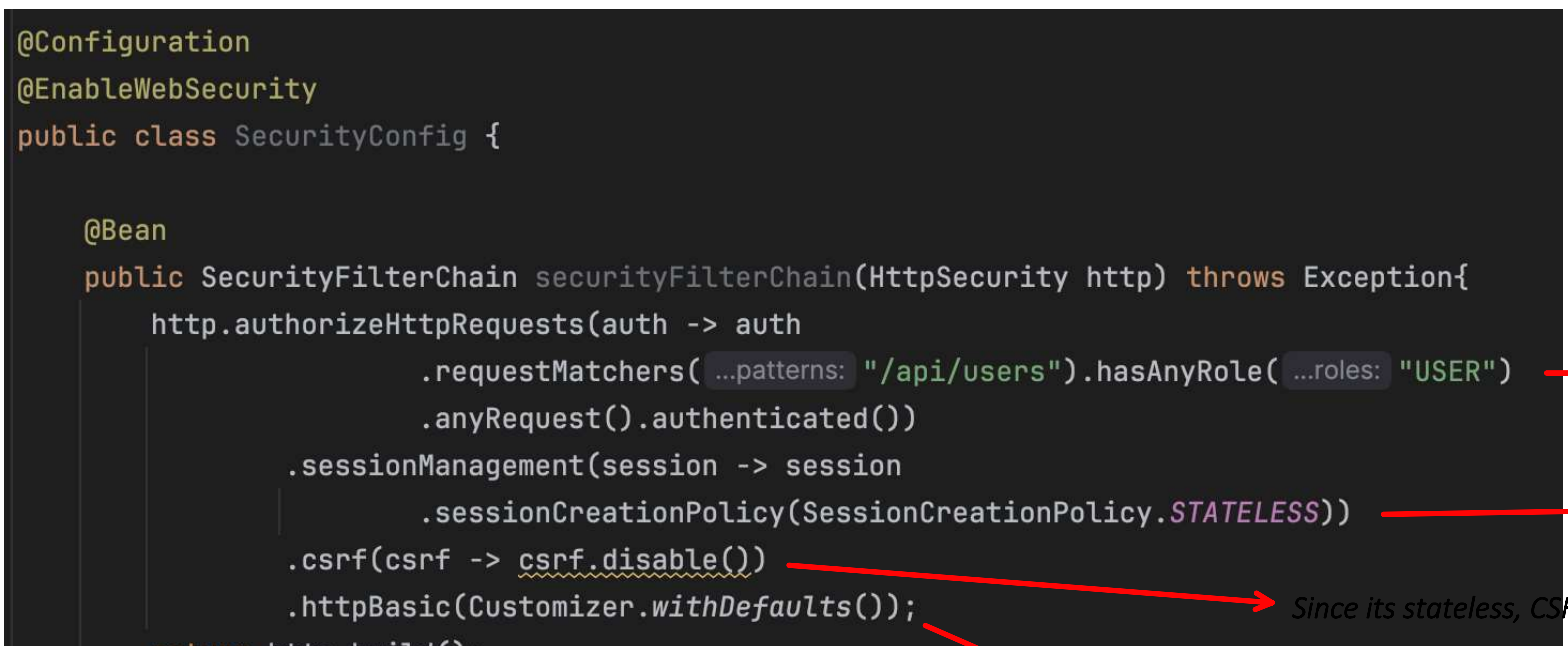


```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-jdbc</artifactId>
    </dependency>
```

Application.properties



```
#creating username and password a
spring.security.user.name=user
spring.security.user.password=pas
spring.security.user.roles=ADMIN,
```



```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
        http.authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/api/users").hasAnyRole( ...roles: "USER")
                        .anyRequest().authenticated())
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .httpBasic(Customizer.withDefaults());
```

Since its stateless, CSRF is not r

```
        return http.build();
    }
}
```

Basic authentication method to be used

Disadvantages of Basic Authentication:

1. Credentials sent in every request, if HTTPS is not enforced, then it can be intercepted and then decoded.

2. If Credentials are compromised, then only way is to change the credentials.

3. Not suitable for large scale application, as sending credentials with every request is an extra overhead.
   i. As request size increases because of authorization header.
   ii. Extra work like decoding, hashing of incoming password, fetching username and password from DB, comparing etc..
   iii. DB lookup to fetch user details which increase latency too.