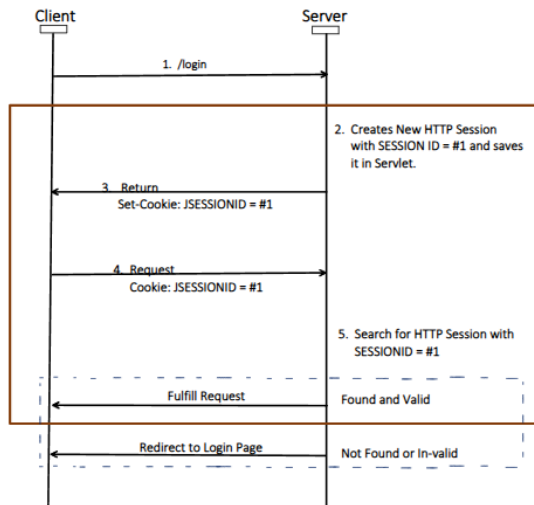


Form Login Authentication:

- It's a stateful authentication method.
 - Stateful authentication means, server maintains the user authentication state (aka Session).
 - So that user don't have to provide username/password every time with each request.
- User enters their credentials (i.e. username/password) in an HTML login form.
- On successful authentication, a session (JSESSIONID) is created to maintain the user authentication state across different requests.
- Now, with subsequent request, client only passes JSESSIONID and not username/password. And server validates it with stored JSESSIONID.
- It's a Default Authentication Method of Springboot Security.
- Default Login URL: /login
- Default Logout URL: /logout



- By default, Time to live for the HTTP Session is 30mins (depends on servlet container). But we can configure it too.
- Yes, we can store the HTTP Session in DB too.

application.properties

```
server.servlet.session.timeout=1m
```

- Now, after 1 minute of inactivity, makes the session expires.

Note: if user activity keeps on happening, it will keep on re-authenticating and after 1 min session will not get expired.

We can also store the session in the DB

Add below dependency in Pom.xml

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

Add below config in application.properties

```
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m
```

SpringBoot (ConceptAndCoding) xSpringBoot (ConceptAndCoding) xSpringBoot (ConceptAndCoding) x

onedrive.live.com/view.aspx?resid=6B364628C29FEB52!sb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHZJ2Lm1zL28vYy82YjM2NDYyOGMyOWZlYjUyL0Vrc0xiYk11UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkM...

SpringBoot (ConceptAndCoding) v

RunRun SelectedAuto completeClearSQL statement:
SELECT * FROM SPRING_SESSION

SELECT * FROM SPRING_SESSION;

PRIMARY_ID	SESSION_ID	CREATION_TIME	LAST_ACCESS_TIME	MAX_INACTIVE_INTERVAL	EXPIRY_TIME	PRINCIPAL_NAME
308b0ed3-98fe-4d29-b8d60b711dc	a3d2b136-57c8-4211-983e-8cd887138598	1742579548528	1742579548542	300	1742579848542	user

(1 row, 5 ms)

This expiry time, will keep on increasing, if user keep on sending request.

Flow diagram for Form based Authentication Method:

1. User is log-in for the first time, means Session does not exist at this point of time, only user exists.

"/login" api is invoked

Please sign in

user

.....

Sign in

35°C Haze

Search

5:22 PM 6/2/2025

SpringBoot (ConceptAndCoding) x

SpringBoot (ConceptAndCoding) x

SpringBoot (ConceptAndCoding) x

+

onedrive.live.com/view.aspx?resid=6B364628C29FE521sb06D0b4b3c6e43888c560d8ed130ea41&migratedto=...&redeem=aHR0cHM6Ly8xZHI2Lm1zL28vYy82YjM2NDYyOGMyOWZiYUyL0Vrc0xiYk1UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkm...

SpringBoot (ConceptAndCoding) v

Request

Response

Tomcat

(Servlet Container)

Filters Chain

Filter1

Security Filter Chain

Filter n

(Servlets)

Dispatcher Servlet

Servlets 2

Servlets n

Interceptors

Controller

```
UsernamePasswordAuthenticationToken [Principal=user, Credentials={PROTECTED}, Authenticated=false,
details=null, Granted Authorities=[]]
```

This is how Authentication object looks like, currently Authenticated is false

Security Filter Chain

UsernamePasswordAuthenticationFilter

SecurityContextHolderFilter

SecurityContext

Authentication

1. Create and Pass the "Authentication" Object to

AuthenticationManager

2. Delegates Authentication to

AuthenticationProvider

3. Incoming raw password is hashed

PasswordEncoder

4. Fetch User Details Like username, password, Roles etc.

<<Interface>>

UserDetailsService

Either from

InMemoryUserDetailsManager (manage username/password in Memory)

or

JdbcUserDetailsManager (manage username/password in DB)

DB

5. Validate Incoming Username and hashed password with Stored username and hashed password.

DaoAuthenticationProvider (handles username/Password)

6. Updated "Authentication" Object

AuthenticationManager

7. Return back fully "Authentication" object

UsernamePasswordAuthenticationFilter

8. Next filter invoked

SecurityContextHolderFilter

9. Saves the Authentication Object in SecurityContext

SecurityContext

Authentication

10. Pass the SecurityContext Object to

HttpSessionSecurityContextRepository

11. Creates New HTTPSession Object

12. Stores the SecurityContext Object inside the HTTPSession

```
UsernamePasswordAuthenticationToken [Principal=user, Password={PROTECTED}, Enabled=true, AccountNonExpired=true, CredentialsNonExpired=true,
AccountNonLocked=true, Granted Authorities=[ROLE_ADMIN], Credentials={PROTECTED}, Authenticated=true,
details=WebAuthenticationDetails [RemoteIpAddress=0:0:0:0:0:1, SessionId=null], Granted
Authorities=[ROLE_ADMIN]]
```

This is how Authentication object looks like after successful authentication, Authenticated status changed to true.

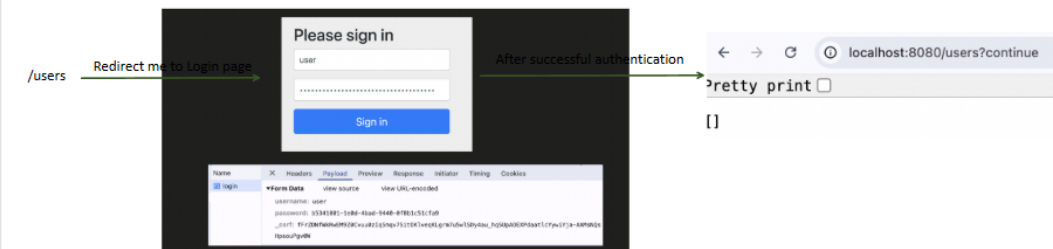
35°C

Haze

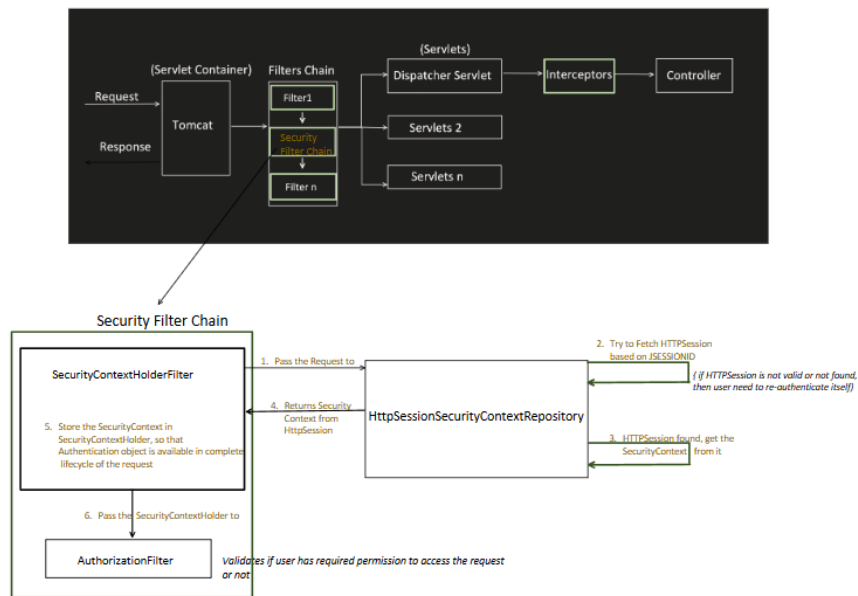
Search

5:22 PM

6/2/2025



2. After Authentication, User is invoking any subsequent APIs



If we just see, we don't have to write a single line of code, its all handled via framework only.
As it's a default authentication method of Springboot security.

SpringBootWebSecurityConfiguration

```
@Bean
@Order(SecurityProperties.BASIC_AUTH_ORDER)
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    http.authorizeHttpRequests((requests) -> requests.anyRequest().authenticated());
    http.formLogin(withDefaults());
    http.httpBasic(withDefaults());
}
```

All I have added is dependency and config.

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

```
spring.security.user.name=user
spring.security.user.password=pass
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m
```

Instead, of generating random password every time, I have hardcoded it. Already discussed in previous video

Added, just want to store the session in DB

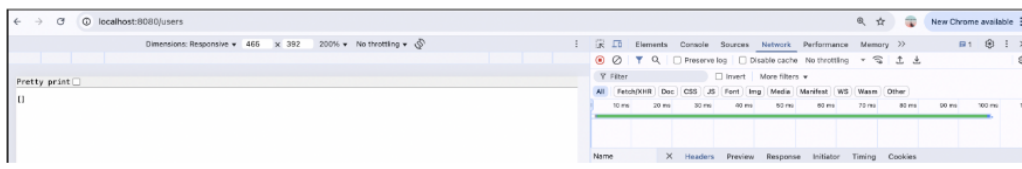
localhost:8080/login

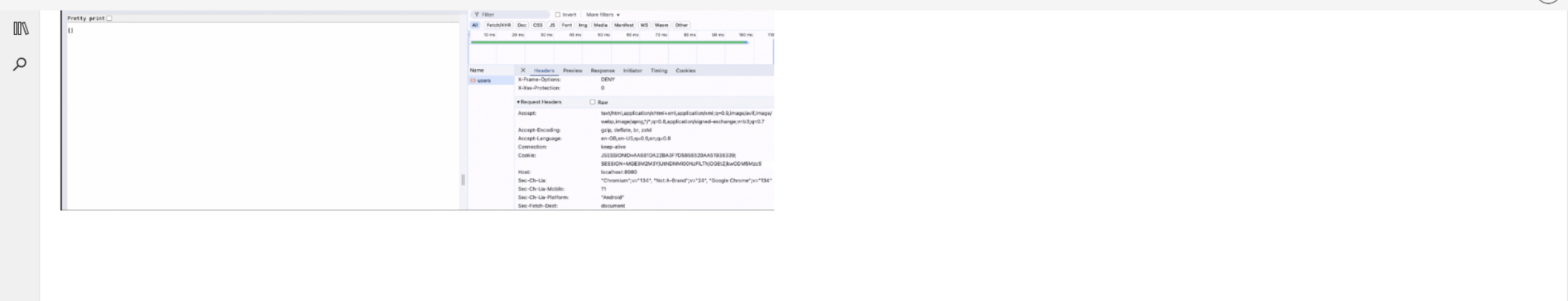
Please sign in

user

Sign in

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
login	General						
localhost	Request URL:	http://localhost:8080/login					
	Request Method:	POST					
	Status Code:	302 Found					
	Remote Address:	[::1]:8080					
	Referrer Policy:	strict-origin-when-cross-origin					
	Response Headers	Raw					
	Cache-Control:	no-cache, no-store, max-age=0, must-revalidate					
	Connection:	keep-alive					
	Content-Length:	0					
	Date:	Sat, 22 Mar 2025 09:29:31 GMT					
	Expires:	0					
	Keep-Alive:	timeout=60					
	Location:	http://localhost:8080/					
	Pragma:	no-cache					
	Set-Cookie:	SESSION=MGE3M2M3YjU0NDhmMD00ZjltHjOGETZjkwODM5Mzc5YzBi; Path=/; HttpOnly; SameSite=Lax					





- Default login and logout page
- Need to relax authentication on few endpoints
- Etc..

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/users").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}

```


Now, form based Authentication is clear, but still one thing is left i.e. **AuthorizationFilter**

- Once the user is Authenticated, and when user is trying to access any resource, authorization check is mandatory.
- It is done to make sure, User has the permission to access it.
- By-default, SpringBoot Security do not put any restriction on any resource, we have to do it manually.

Authorization has 2 phases

Authorization check as part of
SecurityFilter

Authorization check after Request passes
the SecurityFilter and reaches the
Controller.

*(this we will cover later as it common for all
different authentication methods either
Form Based, Basic or JWT)*

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(pattern: "/users").hasRole("USER")
                .anyRequest().authenticated()
            )
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

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=USER

#just for storing the session details in DB
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m
```

```

        .anyRequest().authenticated()
    )
    .formLogin(Customizer.withDefaults());
    return http.build();
}

```

```
#just for storing the session details in DB
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m
```

- Now I am manually restricting that any user trying to access `/users` endpoint, should have `ROLE_USER` role.
- While using `hasRole`, we don't need to add `"ROLE_"` it get appended automatically.
- Now in `AuthorizationFilter`, it will validate does endpoint has any restriction (i.e. user should have any specific role), if Yes, then it matches the role present in `SecurityContext` and what is required for the endpoint.

If required role is missing, it will throw FORBIDDEN exception.

Sat Mar 22 16:48:44 IST 2025

There was an unexpected error (type=Forbidden, status=403).

- Generally, we can give any name to the Role like ADMIN, USER, ANONYMOUS etc.. As its just a String. But should follow the proper meaning.
- Also more than 1 roles can be assigned to a User.

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/pattern: /users").hasAnyRole(ROLE "USER", "ADMIN")
                .anyRequest().authenticated()
            )
            .formLogin(Customizer.withDefaults());

        return http.build();
    }
}

```

```

#creating user and password and assigning the ROLE to the user
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=USER,ADMIN

#just for storing the session details in DB
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m

```

How to control the Sessions per user?

- 1 user can keep on login in different browsers, so how to restrict per user session limit.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(...patterns: "/users").hasRole("USER")
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .maximumSessions(1)
                .maxSessionsPreventsLogin(true))
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

- I am already log-in in 1 browser.
- And when I am trying to log-in via different browser but for the same user.

Please sign in

Maximum sessions of 1 for this principal exceeded

Sign in

What are Session Creation Policies?

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(...patterns: "/users").hasRole("USER")
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED))
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

IF_REQUIRED

HttpSession is only created when needed(DEFAULT).
For example:

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Markup and share

IF_REQUIRED	HttpSession is only created when needed(DEFAULT). <i>For example: public api for which authentication is not required, HttpSession will not be created if this policy will be chosen.</i>
ALWAYS	HttpSession is always created. If already present then use it. <i>For example: even for public api for which authentication is not required, HttpSession will be created if this policy will be chosen.</i>
NEVER	Do not creates a Session, but use if present.
STATELESS	No Session is created, used for Stateless applications.

Disadvantages of Form based authentication:

1. Vulnerable to Security issues like CSRF and Session hijacking :
Be default, CSRF is enabled for form based login and we should not disable it.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .anyRequest().authenticated()
            )
            .csrf(csrf -> csrf.disable()) //we should not do this for form based authentication
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

2. Session Management is big overhead and in case of distributed system it can lead to scalability issue.
3. Database load: If there are multiple servers then we might need to store the session in DB or Cache, which again required memory and lookup time. Which can cause latency issue.