SpringBoot Security - Part2 and Part3

Friday, 7 March 2025 3:48 PM

User Creation:

Before, we proceed with User Authentication and Authorization methods, we first need to see, User creation process because that's the first step.

Authentication and Authorization of User will happen only after User is created.

Lets see, what will happen when we add below security dependency, as seen in previous ***Architecture*** video and starts the server:

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

Provides core feature like:
- Authentication
- Authorization
- Security filters etc.

Logs when server is started:

```
2025-03-08T15:09:16.356+05:30  INFO 44103 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-03-08T15:09:16.466+05:30  WARN 44103 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database
2025-03-08T15:09:16.500+05:30  WARN 44103 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: b5341001-1e0d-4bad-9440-0f8b1c51cfa9

This generated password is for development use only. Your security configuration must be updated before running your application in production.
```

```
 main] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configured with UserDetailsService bean with name inMemoryUserDetailsManager
 main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
 main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.751 seconds (process running for 1.889)
exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring DispatcherServlet 'dispatcherServlet'
exec-1] o.s.web.servlet.DispatcherServlet       : Initializing Servlet 'dispatcherServlet'
exec-1] o.s.web.servlet.DispatcherServlet       : Completed initialization in 1 ms
```

So, what exactly happened here?

- **During server startup, user is created automatically with default username: "user"**
- **Random password is generated for testing.**
- **Each time, server is restarted, new random password will get generated.**

SecurityProperties.java

```
public static class User {

/** Default user name */
private String name = "user";

/** Password for the default username */
Private String password = UUID.randomUUID().toString();

/** Granted roles for the default username */
private List<String> roles = new ArrayList<>();

private boolean passwordGenerated = true;
                .
                .
                //getters and setters
                .
}
```

@AutoConfiguration

UserDetailsServiceAutoConfiguration.java

```
@Bean
public InMemoryUserDetailsManager inMemoryUserDetailsManager(SecurityProperties properties,
        ObjectProvider<PasswordEncoder> passwordEncoder) {
    SecurityProperties.User user = properties.getUser();
    List<String> roles = user.getRoles();
    return new InMemoryUserDetailsManager(User.withUsername(user.getName())
        .password(getOrDeducePassword(user, passwordEncoder.getIfAvailable()))
        .roles(StringUtils.toStringArray(roles))
        .build());
}
```

InMemoryUserDetailsManager.java

```
private final Map <String, Mutable UserDetails> users= new HashMap<>
();
public InMemoryUserDetailsManager(UserDetails... users) {
    for (UserDetails user : users) {
        createUser(user);
    }
}
```

```
@Override
public void createUser(UserDetails user) {
    Assert.isTrue(!userExists(user.getUsername()), message: "user should not exist");
    this.users.put(user.getUsername().toLowerCase(), new MutableUser(user));
}
```

How we can control the user creation logic?

1st: Using application.properties
(not recommended, only for development and testing)

application.properties

```
spring.security.user.name=my_username
spring.security.user.password=my_password
spring.security.user.roles=ADMIN
```

Internally, it uses reflection and calls setUserName() and setPassword() method of **SecurityProperties.java** and overrides the default values.

Now, during application startup, no default username and default password is created.

```
45512 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
45512 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rend
45512 --- [           main] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configured with UserDetailsService bean with name inMemoryUserDetailsManager
45512 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
45512 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.701 seconds (process running for 1.832)
45512 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
45512 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
45512 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
```

2nd: By creating custom InMemoryUserDetailsManager Bean
(not recommended, only for development and testing)

UserDetailsService

↓

UserDetailsManager

↓

InMemoryUserDetailsManager

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user1 = User.withUsername("my_username_1")
                .password("{noop}my_password_1") // {noop} means no encoding or hashing
                .roles("ADMIN")
                .build();

        UserDetails user2 = User.withUsername("my_username_2")
                .password("{noop}1234")  // {noop} means no encoding or hashing
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(user1, user2);
    }
}
```

why we are appending {noop} here?

The default, format for storing the password is :
{id}encodedpassword

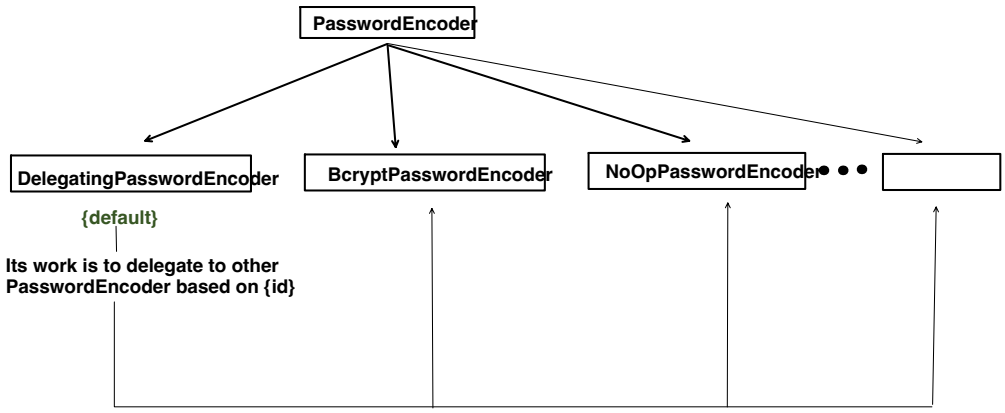
{id} can be either:
- ***{noop}***

- *{bcrypt}*
- *{sha256}*
- *Etc..*

-- During User password storing step, if we want to store user password without any encoding or hashing, then we store "{noop}plain_password"

-- Now, during authentication process:

- **1st, it will fetch the user password from inMemory.**
- **2nd, it goes for comparing logic, inMemory password and password provided for authentication.**
- **3rd, it will take out the {noop} or {bcrypt} etc. from inMemory password.**
- **4th, Then if its {noop}, it will directly compare the remaining inMemory password and provided password for authentication.**
- **5th, if say its {bcrypt}, it first do hashing of provided password using BCryptPasswordEncoder and then match it with remaining inMemory Password.**

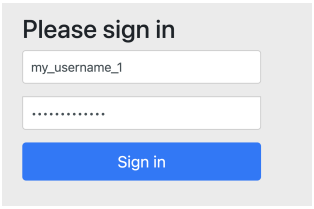


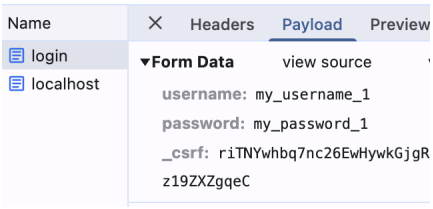
Lets say, if we want to store the hashed password (hashed using ***bcrypt*** algorithm)

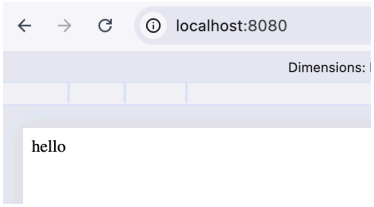
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user1 = User.withUsername("my_username_1")
                .password("{bcrypt}" + new BCryptPasswordEncoder().encode( rawPassword: "my_password_1"))
                .roles("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user1);
    }
}
```

InMemory, password is stored as : {bcrypt}hased_password
and during authentication, I am providing "my_password_1"

But still I am able to successfully authenticate because of **DelegationPasswordEncoder,** it first checks the format of stored password {id} i.e. {bcrypt}, so it passes the incoming password to BcryptPasswordEncoder, and after hashing, it has done the matching.







If, we don't want to store {bcrypt} or any other hashing algo {id} in front of password, then we can define which PasswordEncoder to use.

Now, since we are always using 1 encoding/hashing algorithm, and control will not goes to "DelegationPasswordEncoder", and it will directly goes to specific Password Encoder, so now no need to put {id} in front of password.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user1 = User.withUsername("my_username_1")
                .password(new BCryptPasswordEncoder().encode( rawPassword: "my_password_1"))
                .roles("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user1);
    }
}
```

3rd: Storing UserName and Password (after hashed) in DB
(recommended for production)

Implements UserDetails because, During Authentication (form, basic, jwt etc..), security framework tries to fetch the user and return the object of UserDetails only, if we don't implement it, then we have to do the mapping (from UserAuthEntity to UserDetails).

UserAuthEntity.java

```java
@Entity
@Table(name = "user_auth")
public class UserAuthEntity implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    private String role;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role));
    }

    @Override
    public boolean isAccountNonExpired() { return true; }

    @Override
    public boolean isAccountNonLocked() { return true; }

    @Override
    public boolean isCredentialsNonExpired() { return true; }

    @Override
    public boolean isEnabled() { return true; }

    //getters and setters
    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
```

UserAuthEntityRepository.java

```java
@Repository
public interface UserAuthEntityRepository extends JpaRepository<UserAuthEntity, Long> {

    Optional<UserAuthEntity> findByUsername(String username);
}
```

Implements UserDetailsServic because, for the same reason, authentication, based on auth method we are using Basic, jw etc. it will try to load user first we are using DB for storing th username and password, spri security don't know how to fe we have to implement UserDetailsService and overri method "loadUserByUsernam

UserAuthEntityService.java

```java
@Service
public class UserAuthEntityService implements UserDetailsService {

    @Autowired
    private UserAuthEntityRepository userAuthEntityRepository;

    public UserDetails save(UserAuthEntity userAuth) {
        return userAuthEntityRepository.save(userAuth);
    }

    @Override
    public UserAuthEntity loadUserByUsername(String username) throws UsernameNotFoundException {

        return userAuthEntityRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    }
```

```
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
}
```

```
}
```

UserAuthController.java

```
@RestController
@RequestMapping("/auth")
public class UserAuthController {

    @Autowired
    private UserAuthEntityService userAuthEntityService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody UserAuthEntity userAuthDetails) {
        // Hash the password before saving
        userAuthDetails.setPassword(passwordEncoder.encode(userAuthDetails.getPassword()));

        // Save user
        userAuthEntityService.save(userAuthDetails);
        return ResponseEntity.ok( body: "User registered successfully!");
    }
}
```

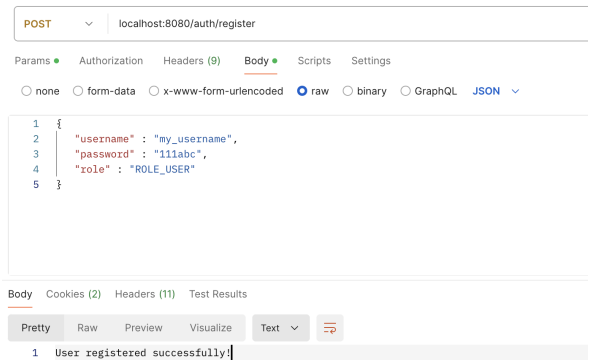
Encod passw storing

Now, by-default in spring boot security, all the endpoints are AUTHENTICATED, means we have to authenticate ourself by either username/password or JWT etc.. To access any API, so how we will access "***/auth/register" API,*** which is just a first step to create user.

Yes, we have to relax the authentication for this API and its industry standard.

SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/auth/register").permitAll()
                        .anyRequest().authenticated()
                )
                .csrf(csrf -> csrf.disable())
                .httpBasic(Customizer.withDefaults());
        return http.build();
    }
}
```



Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM USER_AUTH
```

```
SELECT * FROM USER_AUTH;
```

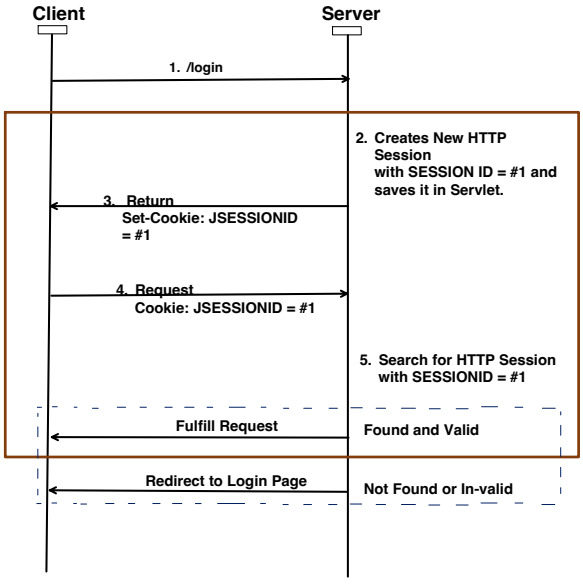
ID	PASSWORD	ROLE	USERNAME
1	$2a$10$euzihUhyp4exMejDkyDb0eK2q49sqTcG8EShOnT2GmaL7lxvIeOfO	ROLE_USER	my_username

(1 row, 3 ms)

Edit

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
```

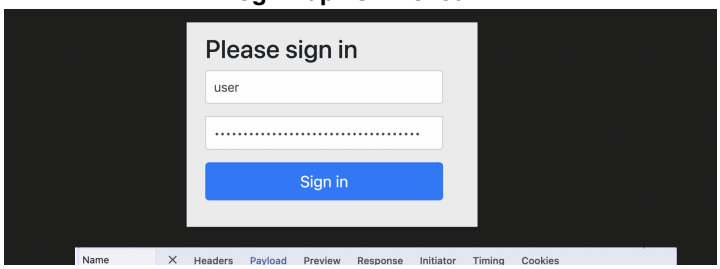
- **SpringBoot, will automatically create and manage "SPRING_SESSION" table for us.**

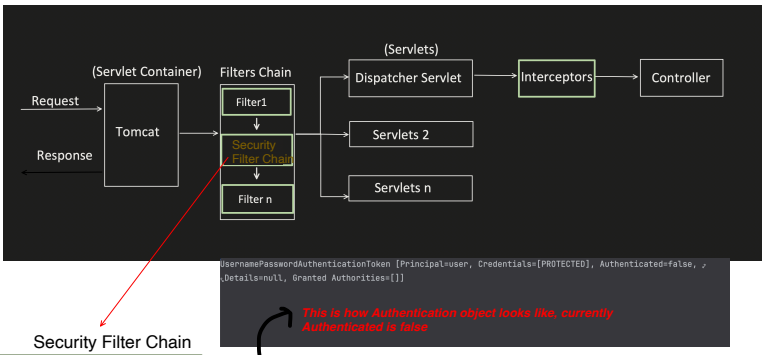
This expiry time, will keep on increasing, if user keep on sending request.

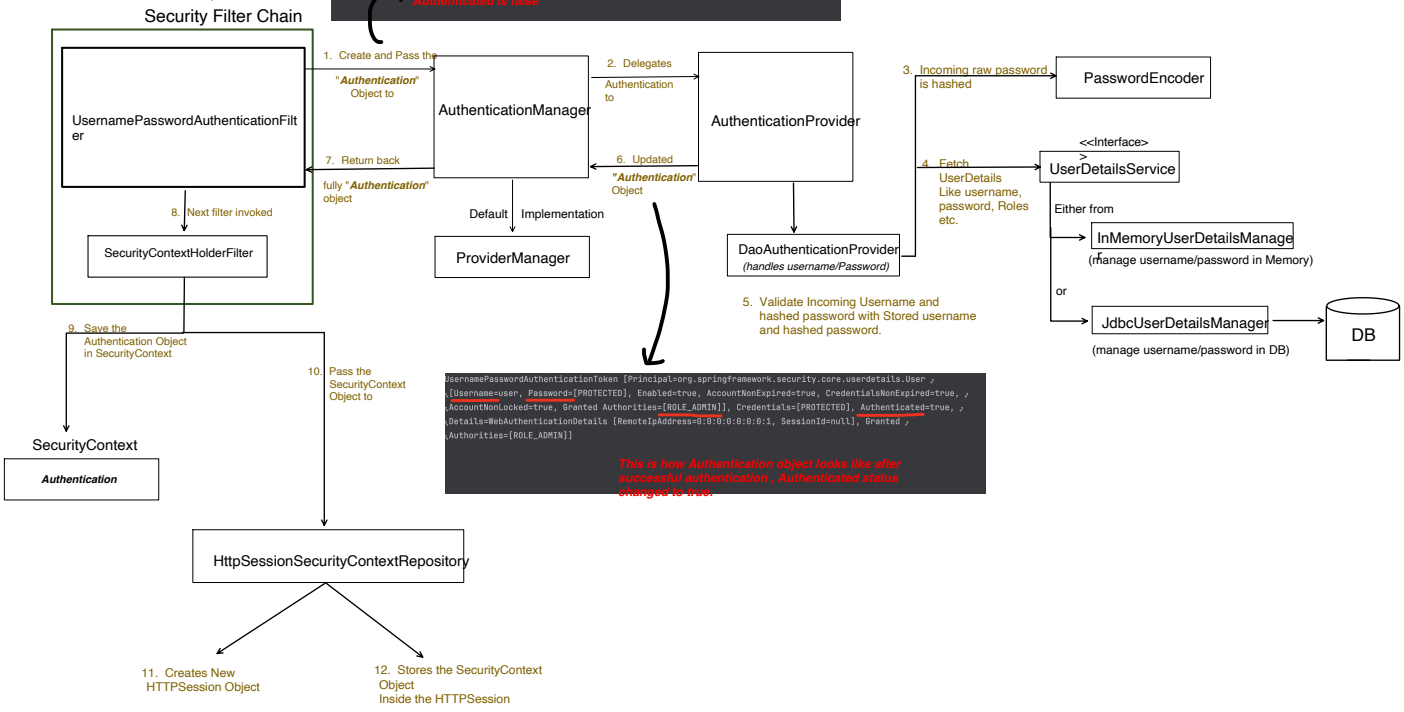
Flow diagram for Form based Authentication Method:

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

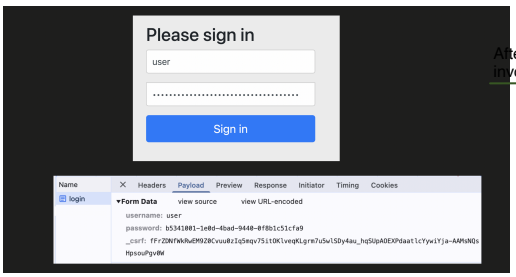






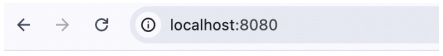
- After successful authentication :
  - If */login* endpoint was used, then it will try to hit default endpoint.
  - If any specific endpoint was used, then after successful authentication, that specific endpoint will only get invoked.

/login

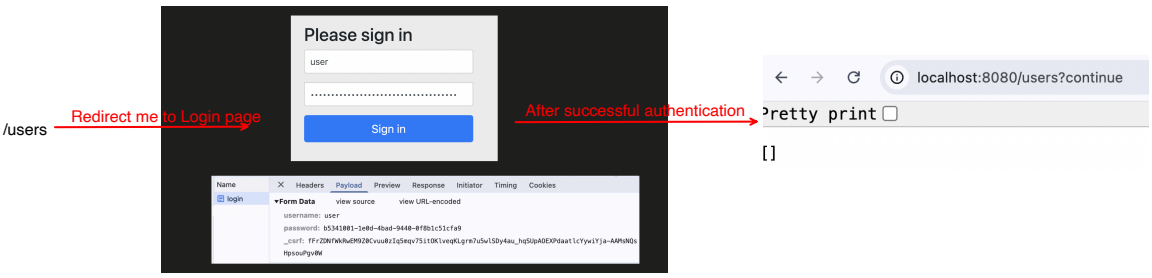


After successful default endpoint got invoked

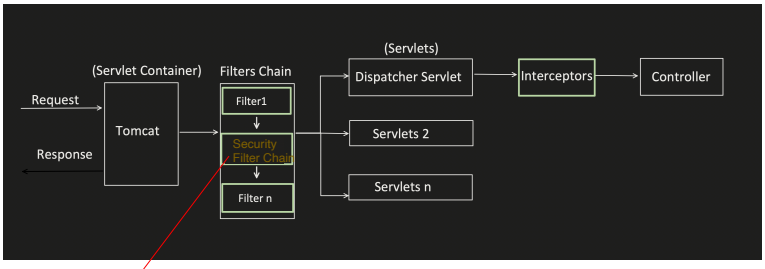
```
@GetMapping("/")
Public String hello() {
        return"hello";
    }
```

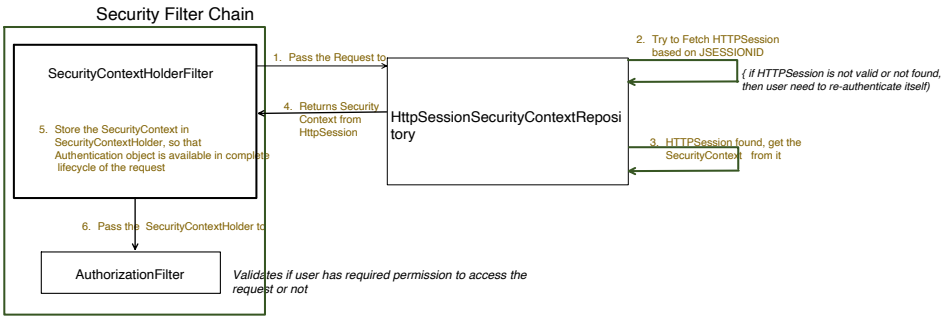


hello



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

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>
```

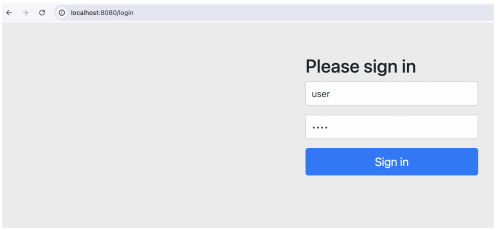
Application.properties

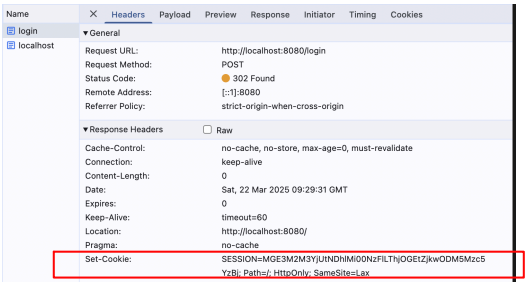
```
spring.security.user.name=user
spring.security.user.password=pass
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
server.servlet.session.timeout=5m
```

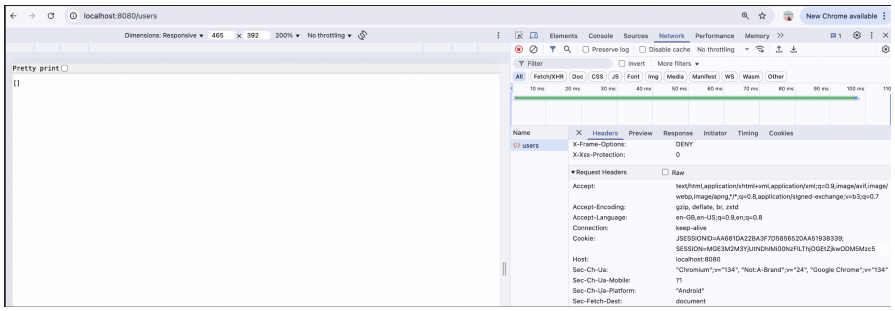
Instead, of generating random password every time, I have hardcoded it. Already discussed in previous video

Added, just want to store the session in DB

```
</dependency>
```







Now, lets say, I want to change few things like:

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

Etc..

Then we can override above default SecurityFilterChain method

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/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}

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/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
```

- **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.**

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

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

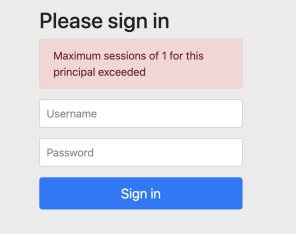
```
#creating username and password and assigning the ROLE to the user
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=USER,ADMIN

#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.**

```java
@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.**



What are Session Creation Policies?

```java
@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:* *public api for which authentication is not required, HTTPSession will not be created if this policy will be chosen.*
ALWAYS	HttpSession is always created. If already present then use it. *For example:* *even for public api for which authentication is not required, HTTPSession will be created if this policy will be chosen.*
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.

```java
@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 their 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.