Western Norway
University of
Applied Sciences

DAT152 – Advanced Web Applications

Web Frameworks

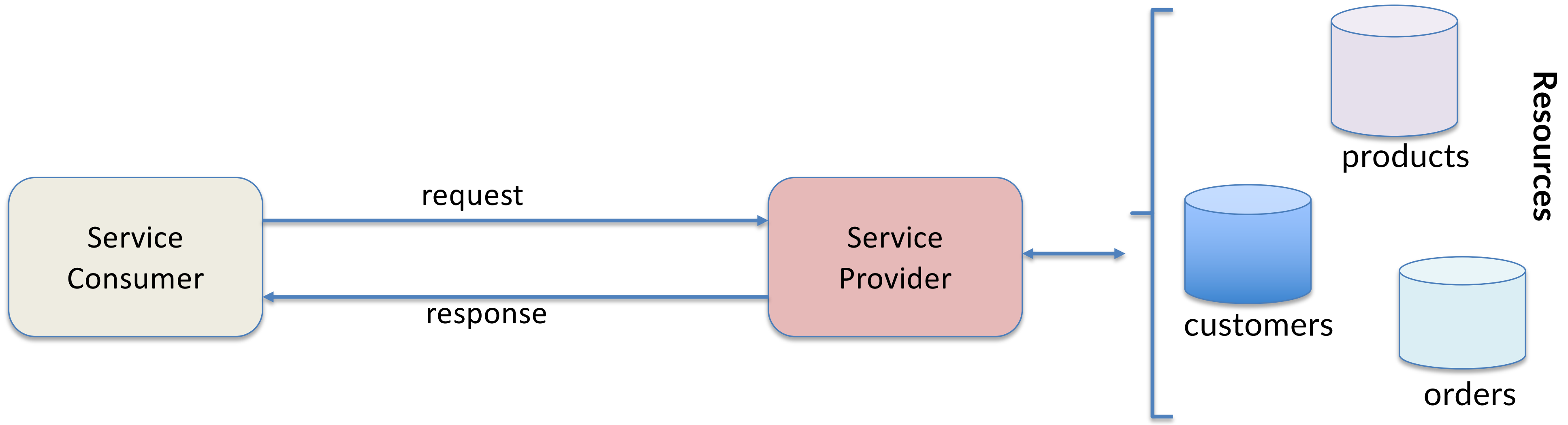
Authentication and Authorization Part 1

Today’s agenda

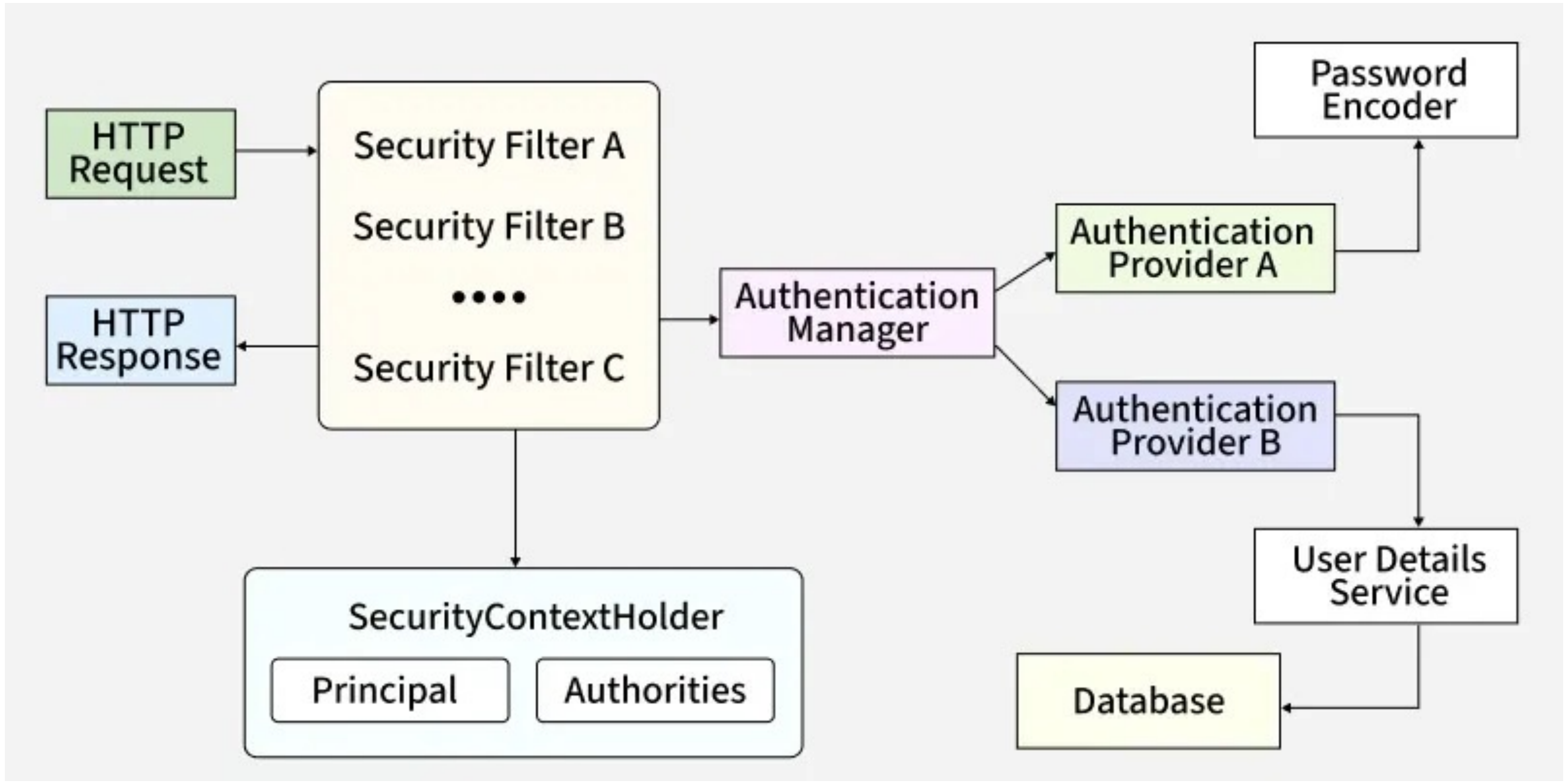
- Authentication and Authorization in Spring Web

What is Authentication and Authorization?

- Authentication
  - Proof of identity – Verifies the identity of a user
- Authorization
  - Level of privilege – What are the access rights/permission for this user?



Spring Security Architecture



https://www.geeksforgeeks.org/springboot/spring-security-architecture/

Spring Security Configuration

- Requires the '<u>spring-boot-starter-security</u>' library in the classpath

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

- For Thymeleaf integration, requires the '<u>thymeleaf-extras-springsecurity6</u>' library in the classpath

```xml
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
```

Spring Security Architecture

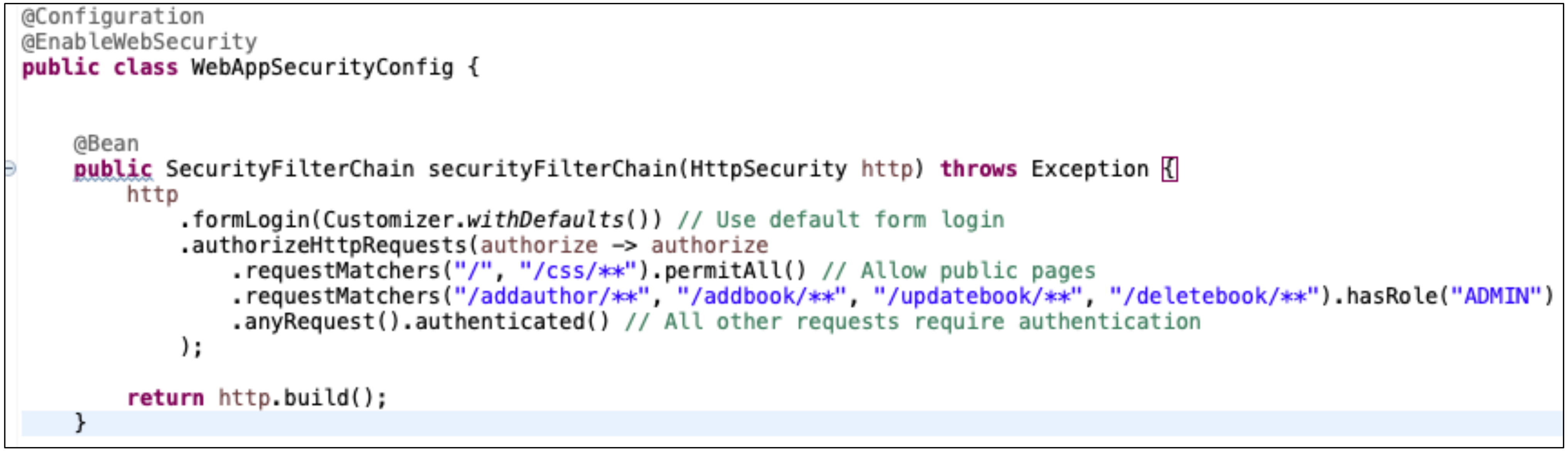
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
      http
         .csrf(Customizer.withDefaults())
         .httpBasic(Customizer.withDefaults())
         .formLogin(Customizer.withDefaults())
         .authorizeHttpRequests(authorize -> authorize
            .anyRequest().authenticated()
         );

      return http.build();
   }

}
```

- Results in the following order
  - CsrfFilter is invoked
  - Authentication filters are invoked
  - Authorization filters are invoked

https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-architecture

Spring Security Architecture

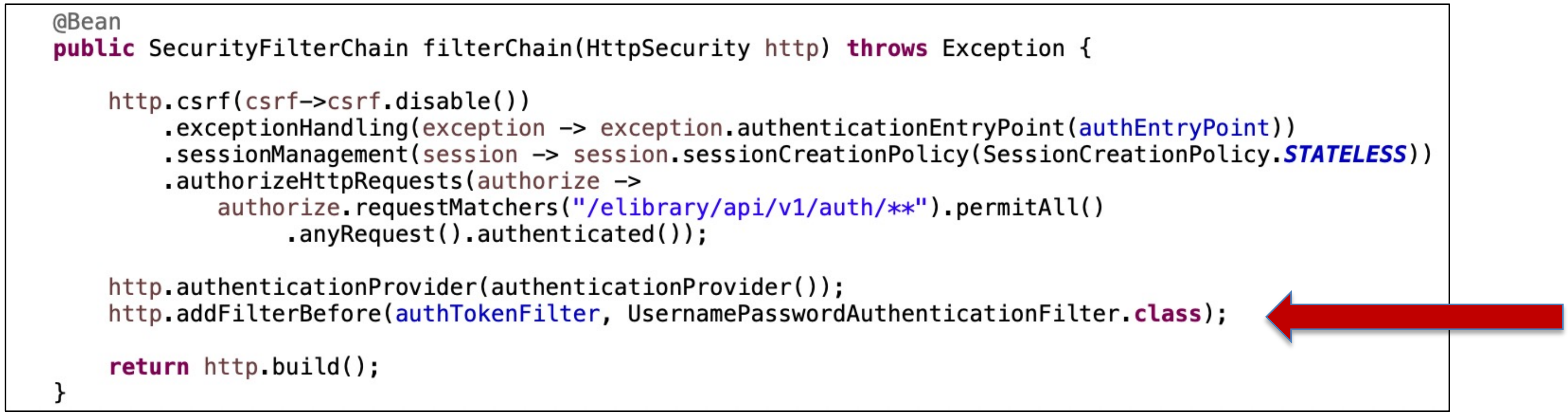
- HttpSecurity
  - Authentication
  - Authorization
  - Request Matcher
  - Exception Handling
  - Adding Filters
  - …



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


    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .formLogin(Customizer.withDefaults()) // Use default form login
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/", "/css/**").permitAll() // Allow public pages
                .requestMatchers("/addauthor/**", "/addbook/**", "/updatebook/**", "/deletebook/**").hasRole("ADMIN")
                .anyRequest().authenticated() // All other requests require authentication
            );

        return http.build();
    }
```

Spring Security Architecture

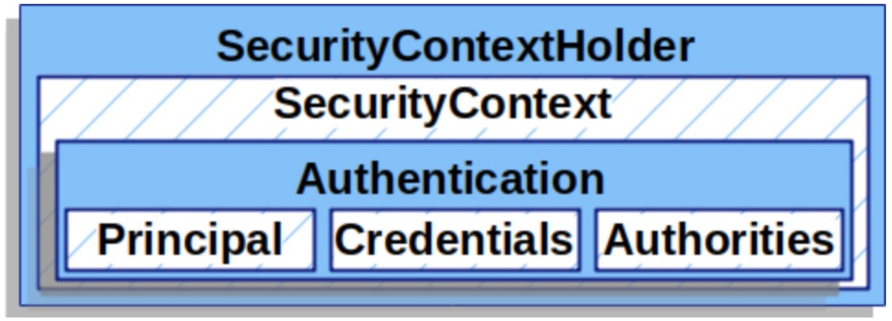
- Adding customer filters
- HttpSecurity comes with three methods for adding filters:
  - #addFilterBefore(Filter, Class<?>) adds your filter before another filter
  - #addFilterAfter(Filter, Class<?>) adds your filter after another filter
  - #addFilterAt(Filter, Class<?>) replaces another filter with your filter

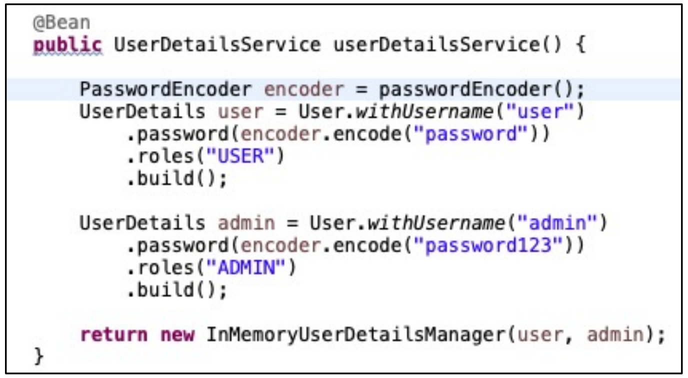


```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.csrf(csrf->csrf.disable())
        .exceptionHandling(exception -> exception.authenticationEntryPoint(authEntryPoint))
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(authorize ->
            authorize.requestMatchers("/elibrary/api/v1/auth/**").permitAll()
                .anyRequest().authenticated());

    http.authenticationProvider(authenticationProvider());
    http.addFilterBefore(authTokenFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

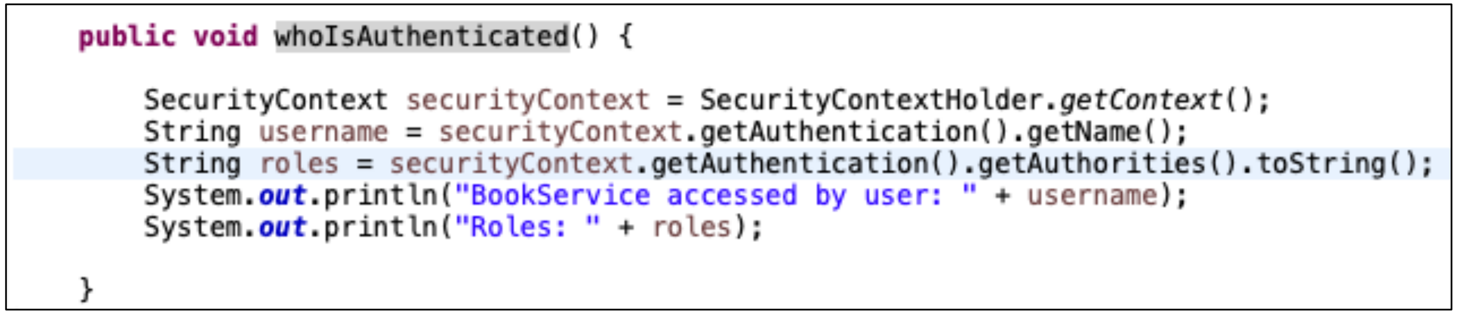
Spring Security Architecture

- UserDetailsService
- SecurityContextHolder

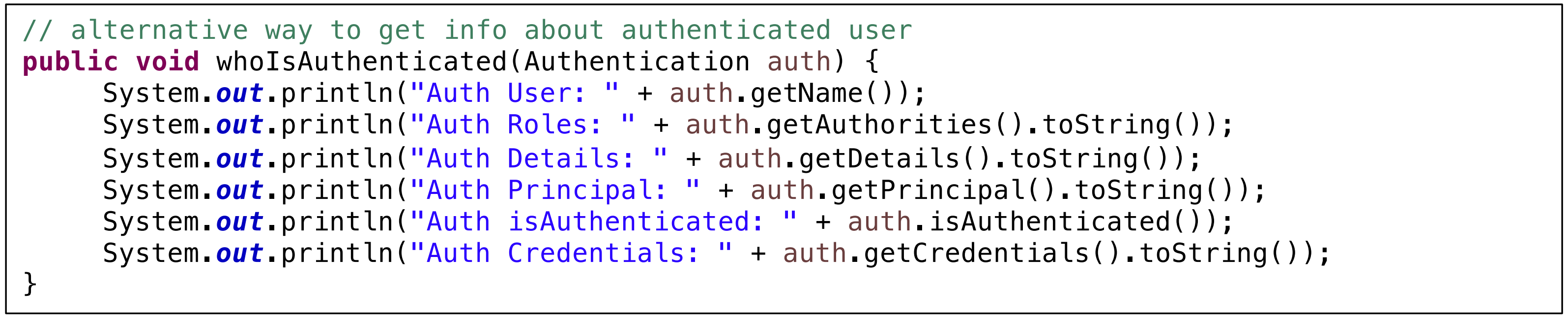




```
@Bean
public UserDetailsService userDetailsService() {

    PasswordEncoder encoder = passwordEncoder();
    UserDetails user = User.withUsername("user")
        .password(encoder.encode("password"))
        .roles("USER")
        .build();

    UserDetails admin = User.withUsername("admin")
        .password(encoder.encode("password123"))
        .roles("ADMIN")
        .build();

    return new InMemoryUserDetailsManager(user, admin);
}
```



```
public void whoIsAuthenticated() {

    SecurityContext securityContext = SecurityContextHolder.getContext();
    String username = securityContext.getAuthentication().getName();
    String roles = securityContext.getAuthentication().getAuthorities().toString();
    System.out.println("BookService accessed by user: " + username);
    System.out.println("Roles: " + roles);

}
```



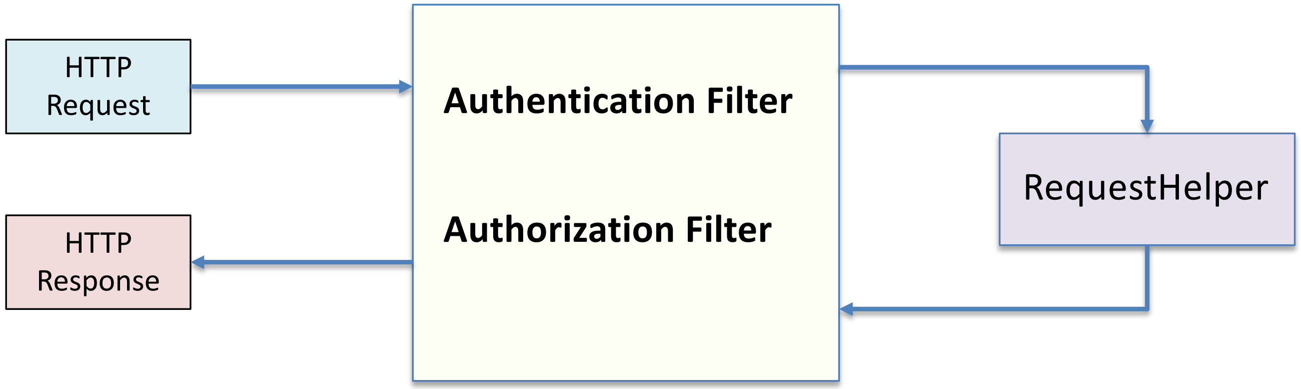
```
// alternative way to get info about authenticated user
public void whoIsAuthenticated(Authentication auth) {
    System.out.println("Auth User: " + auth.getName());
    System.out.println("Auth Roles: " + auth.getAuthorities().toString());
    System.out.println("Auth Details: " + auth.getDetails().toString());
    System.out.println("Auth Principal: " + auth.getPrincipal().toString());
    System.out.println("Auth isAuthenticated: " + auth.isAuthenticated());
    System.out.println("Auth Credentials: " + auth.getCredentials().toString());
}
```

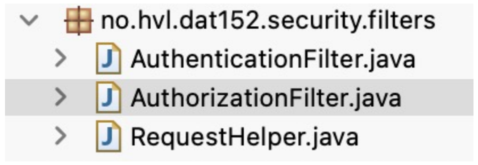
UserDetails

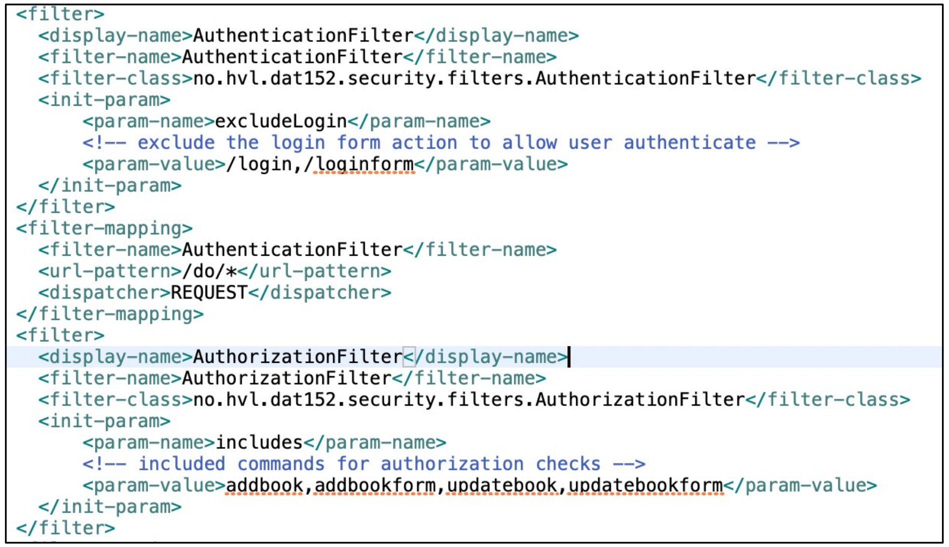
- Spring object for storing details/information of user

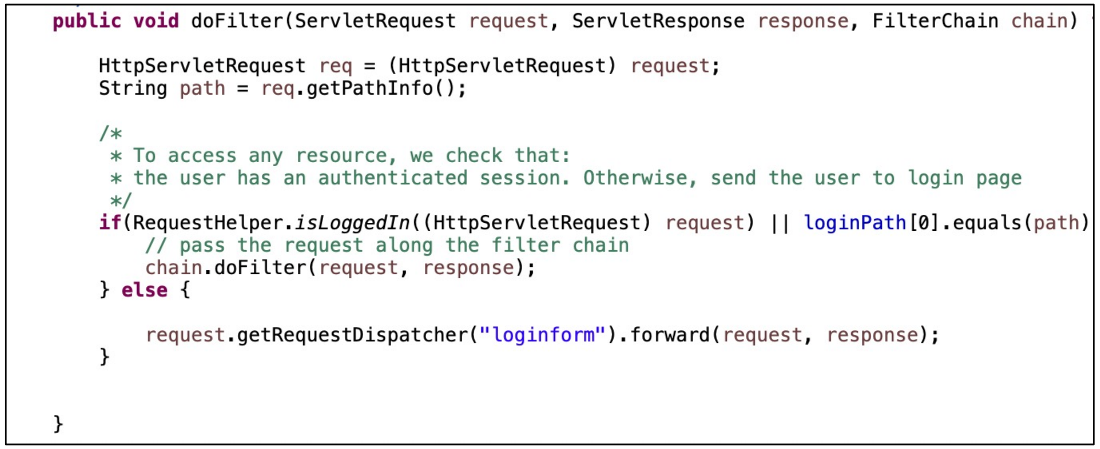
```
@Entity
@Table(name = "users")
public class User implements UserDetails {
    ...

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities()

    @Override
    public String getUsername()

    @Override
    public boolean isAccountNonExpired()

    @Override
    public boolean isAccountNonLocked()

    @Override
    public boolean isCredentialsNonExpired()

    @Override
    public boolean isEnabled()
}
```

Custom FrontController Security Architecture




```
no.hvl.dat152.security.filters
  AuthenticationFilter.java
  AuthorizationFilter.java
  RequestHelper.java
```


```xml
<filter>
  <display-name>AuthenticationFilter</display-name>
  <filter-name>AuthenticationFilter</filter-name>
  <filter-class>no.hvl.dat152.security.filters.AuthenticationFilter</filter-class>
  <init-param>
      <param-name>excludeLogin</param-name>
      <!-- exclude the login form action to allow user authenticate -->
      <param-value>/login,/loginform</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AuthenticationFilter</filter-name>
  <url-pattern>/do/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter>
  <display-name>AuthorizationFilter</display-name>
  <filter-name>AuthorizationFilter</filter-name>
  <filter-class>no.hvl.dat152.security.filters.AuthorizationFilter</filter-class>
  <init-param>
      <param-name>includes</param-name>
      <!-- included commands for authorization checks -->
      <param-value>addbook,addbookform,updatebook,updatebookform</param-value>
  </init-param>
</filter>
```


```java
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

    HttpServletRequest req = (HttpServletRequest) request;
    String path = req.getPathInfo();

    /*
     * To access any resource, we check that:
     * the user has an authenticated session. Otherwise, send the user to login page
     */
    if(RequestHelper.isLoggedIn((HttpServletRequest) request) || loginPath[0].equals(path)
        // pass the request along the filter chain
        chain.doFilter(request, response);
    } else {

        request.getRequestDispatcher("loginform").forward(request, response);
    }


}
```

Authentication Methods in Spring

- BasicAuthenticationFilter
- UsernamePasswordAuthenticationFilter
- BearerTokenAuthenticationFilter
- OAuth2LoginAuthenticationFilter

BasicAuthenticationFilter

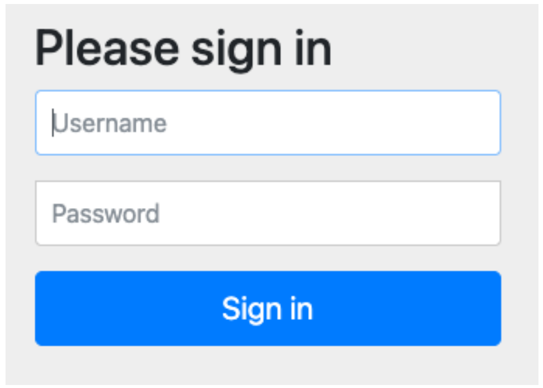
- Handles basic authentication using Base64-encoded username and password in the ‘Authorization’ header
- Performs Base64(username:password)
- Stateless authentication and browser stores the credential and sends it in each request
- Insecure as base64 encoding can be easily decoded, no built-in expiration and limited authorization
- Not meant to be used as form authentication method

```
GET /api/orders/1
Authorization: Basic dXNlcjpwYXNzd29yZA==
```

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
http
    .httpBasic(Customizer.withDefaults()) // Use default basic authentication
```

UsernamePasswordAuthFilter

- Used for Form-based authentication
- Extracts username and password
- Delegates to AuthenticationManager to perform authentication
- Saves authenticated user in SecurityContext
- Default login page accessible at ***/login***
- Default logout page accessible at ***/logout***



```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
http
    .formLogin(Customizer.withDefaults()) // Use default form login
```

BearerTokenAuthenticationFilter

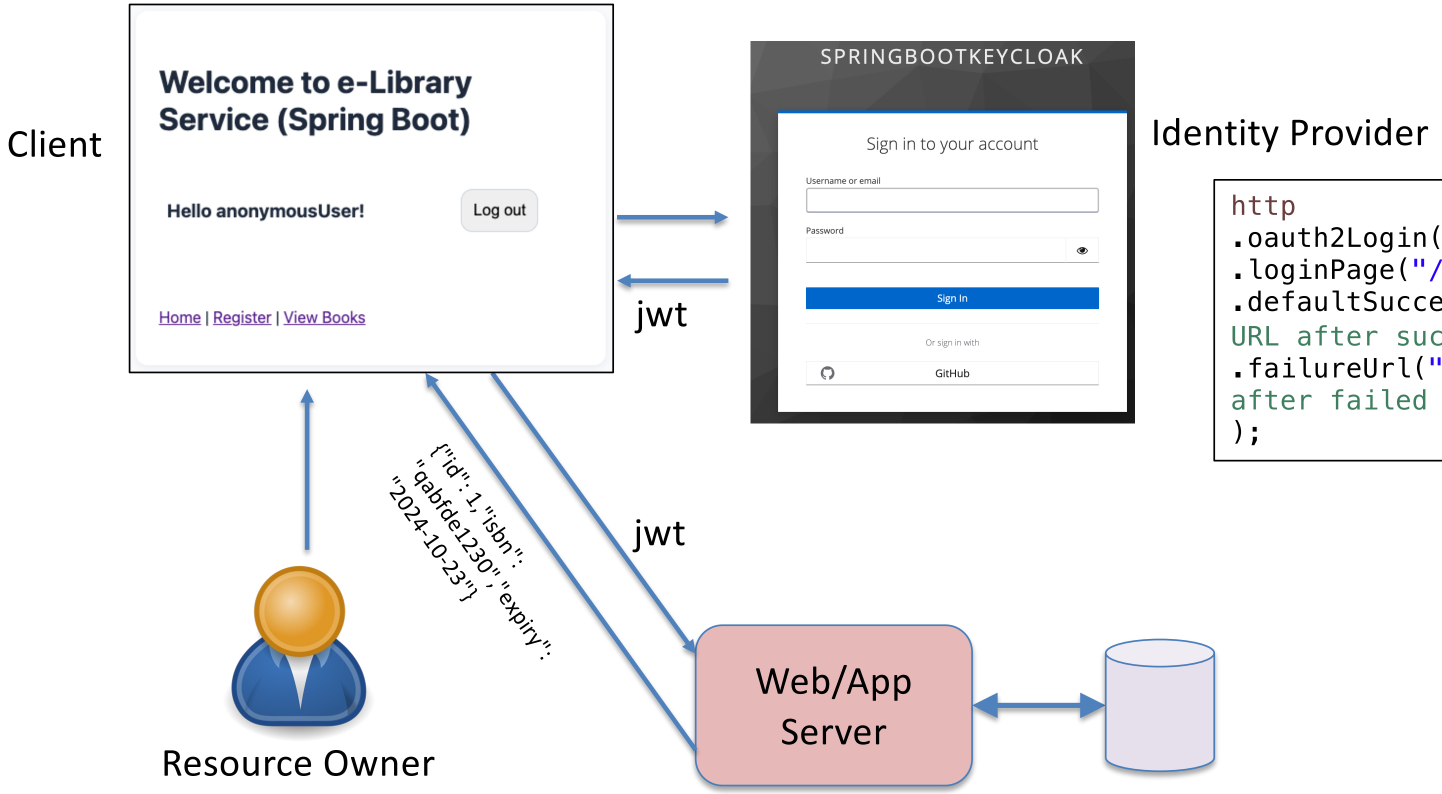
- A client sends an authentication token using the ‘Authorization’ header in the request to access a resource
- The filter extract the Bearer Token (JWT) from the Authorization header
- Validates the token (Authentication)
- Optionally, stores the user in SecurityContextHolder object

```
GET /api/orders/1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMj...
```

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(authorize -> authorize
        .anyRequest().authenticated() // All other requests require authentication
        )
        .oauth2ResourceServer(oauth2 -> oauth2
            .jwt(Customizer.withDefaults()) // Enable JWT authentication
        )
```

OAuth2LoginAuthenticationFilter

- OAuth2 Authorization flow allows a client to negotiate for an openid/access token from a server-side (secure)



```
http
.oauth2Login(oauth2 -> oauth2
.loginPage("/login") // Custom login page
.defaultSuccessUrl("/oauth2success") // Redirect
URL after successful login
.failureUrl("/login?error=true") // Redirect URL
after failed login
);
```

Authorization in Spring Web Framework

- User permission – based on RBAC
- Uses the concepts of Roles and Authorities
- Approaches to configure authorization rules
  - In the **<u>SecurityFilterChain</u>** Bean
  - At the **<u>Method</u>** levels
  - Programmatically, by using the **Authentication** object from SecurityContextHolder

Authorization configuration

- SecurityFilterChain: permission/access levels can be configured
  - Granularity levels:
    - endpoints
    - type of HttpMethod

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(authorize -> authorize
        .requestMatchers("/", "/css/**").permitAll() // Allow public pages
        .requestMatchers(HttpMethod.GET, "/addbooks/**", "/deletebook/**").hasRole("ADMIN")
        .requestMatchers.authenticated() // All other requests require authentication
```

authorizeHttpRequests(): Configures authorization for HTTP requests.
requestMatchers(): Specifies URL patterns.
permitAll(): Allows access to everyone.
hasRole("ROLE_NAME"): Requires a specific role.
hasAnyRole("ROLE1", "ROLE2"): Requires any of the specified roles.
authenticated(): Requires any authenticated user.

Authorization configuration

- At the Method level: permission can be applied using the annotations:
  - @PreAuthorize
  - @PostAuthorize

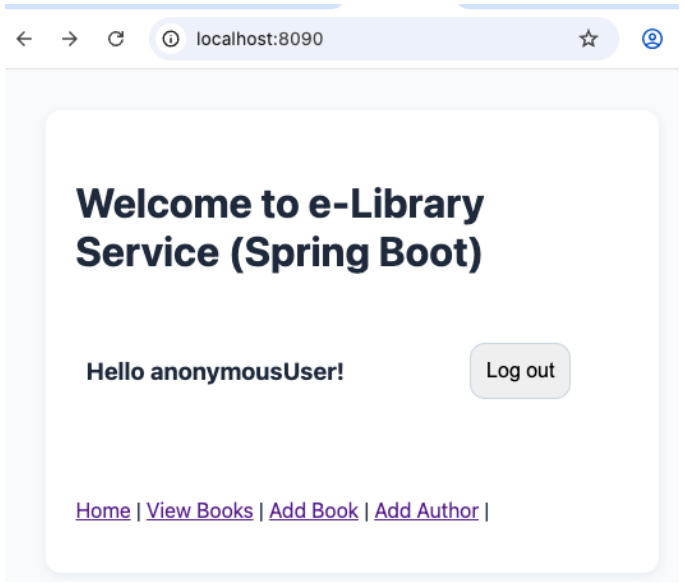
```
@PreAuthorize("hasAuthority('ADMIN')")
@PostMapping("/updatebook")
public String updateBook(@RequestParam String isbn,
        @RequestParam String title,
        @RequestParam String authorid,
        @RequestParam Long id,
        Model model) throws BookNotFoundException, UpdateBookFailedException {

    Book book = bookService.updateBook(isbn, title, authorid, id);
    model.addAttribute("book", book);
    return "viewbook";
}
```

Authorization configuration

- Using **Authentication** object: You can obtain the permission granted an authenticated user and use it to make authorization decisions.

```
@GetMapping("/updatebook")
public String updateBook(@RequestParam Long id, Model model, Authentication auth) throws
BookNotFoundException {

    // only ADMIN can update books
    if (auth.getAuthorities().stream().anyMatch(a -> a.getAuthority().equals("ADMIN"))) {
    return "updatebook";
    } else {
    return "error";
}
```

Lab - Spring Web MVC exercise - B


localhost:8090

Welcome to e-Library Service (Spring Boot)

Hello anonymousUser!

Home | View Books | Add Book | Add Author |

- Presentation Layer (View)
  - login.html (custom login form)
- Controller
  - To handle GET request for login form display
- Security Configuration
  - Securityfilter:
    - formlogin
    - Logout
- Authorization
  - Display menu on view based on user roles
    - Index.html