Java Backend Development Live-85

lecture-18

Agenda

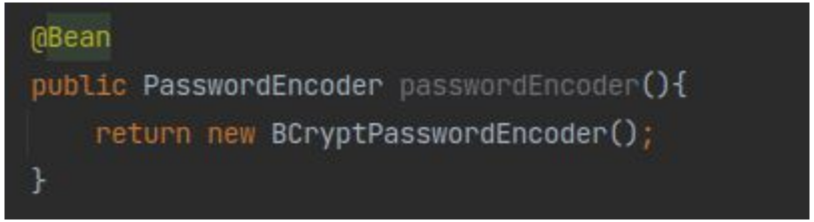
- Spring Security (cont.)
  - User Store
  - HttpSecurity Config
  - Authorization
- Calling Secure API from Postman
- Calling Secure API from curl
- CSRF (Cross-Site Request Forgery) Demo
- Distributed Session Management
- Minor-project-2
  - Spring-Security in Minor-project
  - Redis in Minor-project

PasswordEncoder

We should never save password as plain text.

PasswordEncoder bean in required to encode the passwords.eg. BCryptPasswordEncoder, NoOpPasswordEncoder

This encoding is one way.



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

Read from SecurityContext

SecurityContextHolder is used to fetch details of logged in user.

SecurityContextHolder.getContext().getAuthentication().getPrincipal() returns instance of UserDetail class.

In controller we can use @AuthenticationPrincipal to get UserDetails of authenticated user.

UserDetailsService

Spring Security application can interact with any User store via UserDetailsService interface.

We can provide implementation of UserDetailsService based on our user-store.

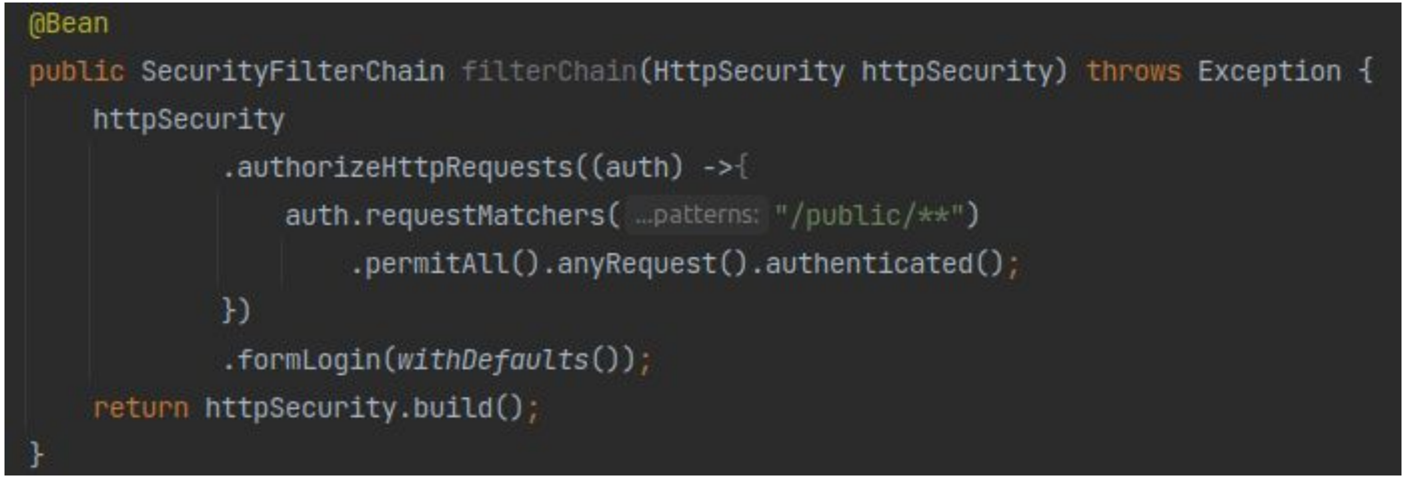
User-store can be in-memory(JVM), Redis, MySQL, any other Application exposed via APIs.

UserDetailsService has only one method definition. UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

Public API or URLs

We can define a bean of **SpringSecurityFilter** with code to expose public API.

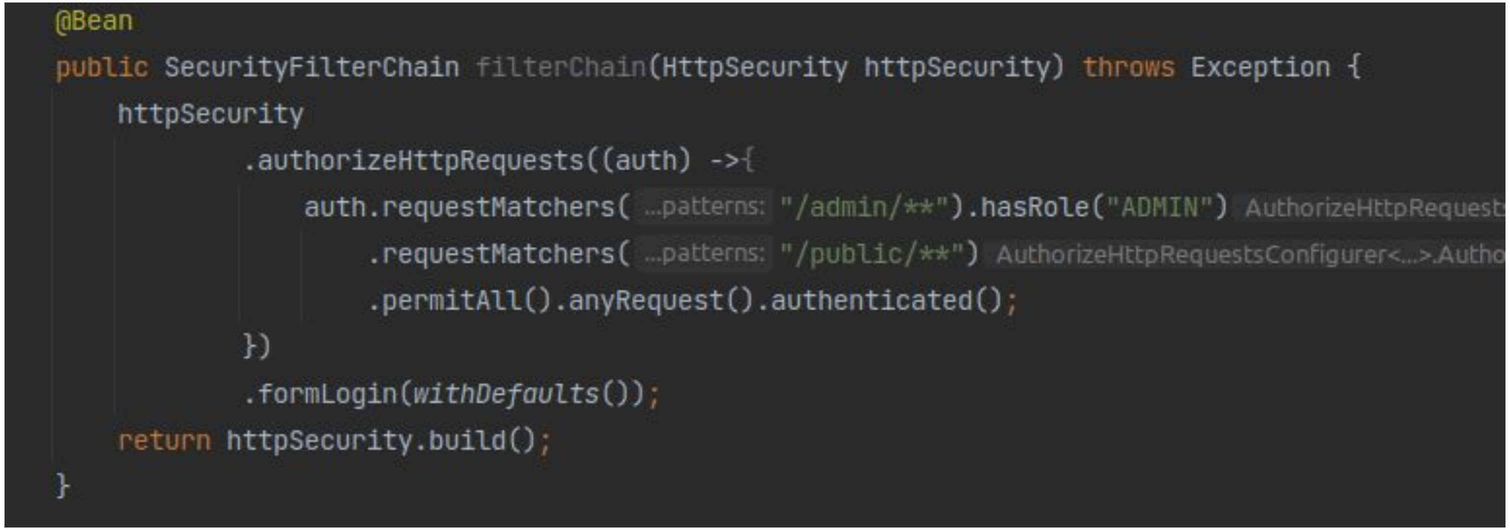
Permit all user to access public API. And all others API can only be accessed by authenticated users.



```
@Bean
public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
            .authorizeHttpRequests((auth) ->{
                auth.requestMatchers( ...patterns: "/public/**")
                    .permitAll().anyRequest().authenticated();
            })
            .formLogin(withDefaults());
    return httpSecurity.build();
}
```

Authorization

In **SpringSecurityFilter** bean we can define authorization rules as well.



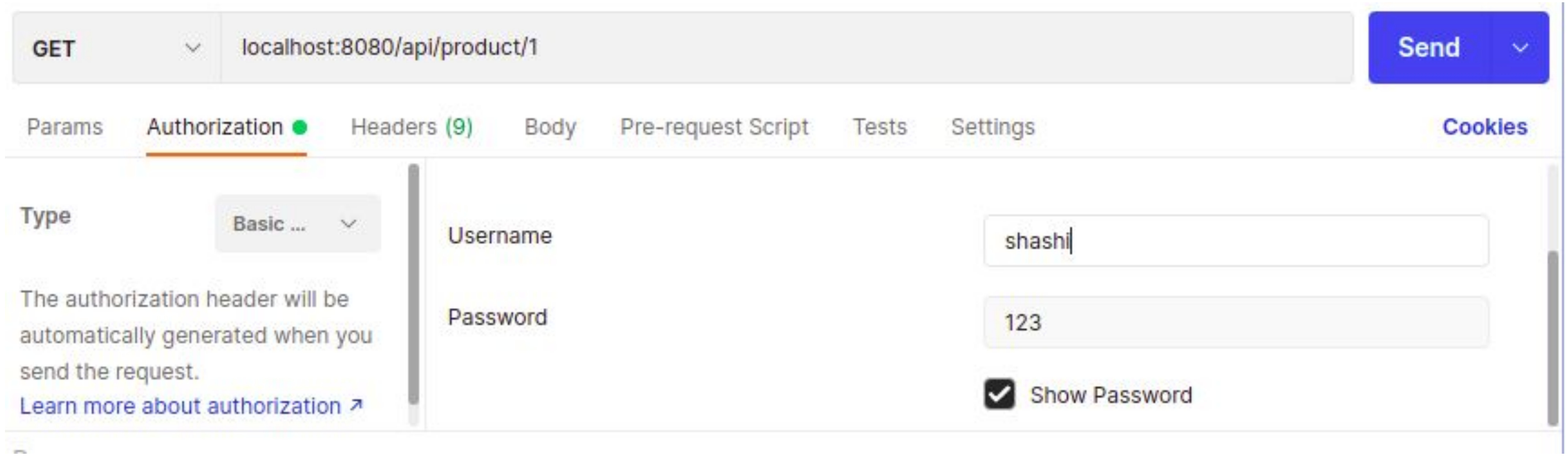
```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
            .authorizeHttpRequests((auth) ->{
                auth.requestMatchers( ...patterns: "/admin/**").hasRole("ADMIN") AuthorizeHttpRequest
                    .requestMatchers( ...patterns: "/public/**") AuthorizeHttpRequestsConfigurer<...>.Autho
                    .permitAll().anyRequest().authenticated();
            })
            .formLogin(withDefaults());
    return httpSecurity.build();
}
```

All APIs starting with “/admin/” can only be access by admin.

Calling Secure API from Postman

Add username and password in Basic Auth from authorization tab. This will add Authorization header in http request.

Make sure HttpBasic is enabled at backend.



localhost:8080/api/product/1

Type: Basic ...

The authorization header will be automatically generated when you send the request.

Username: shashi

Password: 123

Calling Secure API from curl

User following curl syntax:

curl --user "shashi:123" http://localhost:8080/user/hello

We can also use Authorization header.

```
curl --location --request GET 'localhost:8080/admin/user/2'  --header
'Authorization: Basic c2hhc2hpQGdtYWlsLmNvbToxMjM0NTY='
```

CSRF (Cross-Site Request Forgery)

Cross-Site Request Forgery (CSRF) is an attack that forces an **end user to execute unwanted actions** on a web application in which they’re currently authenticated. With a little help of social engineering (such as **sending a link via email or chat**), an attacker may trick the users of a web application into executing actions of the attacker’s choice.

For example, this might be to change the email address on their account, to change their password, or to make a funds transfer.

CSRF Demo

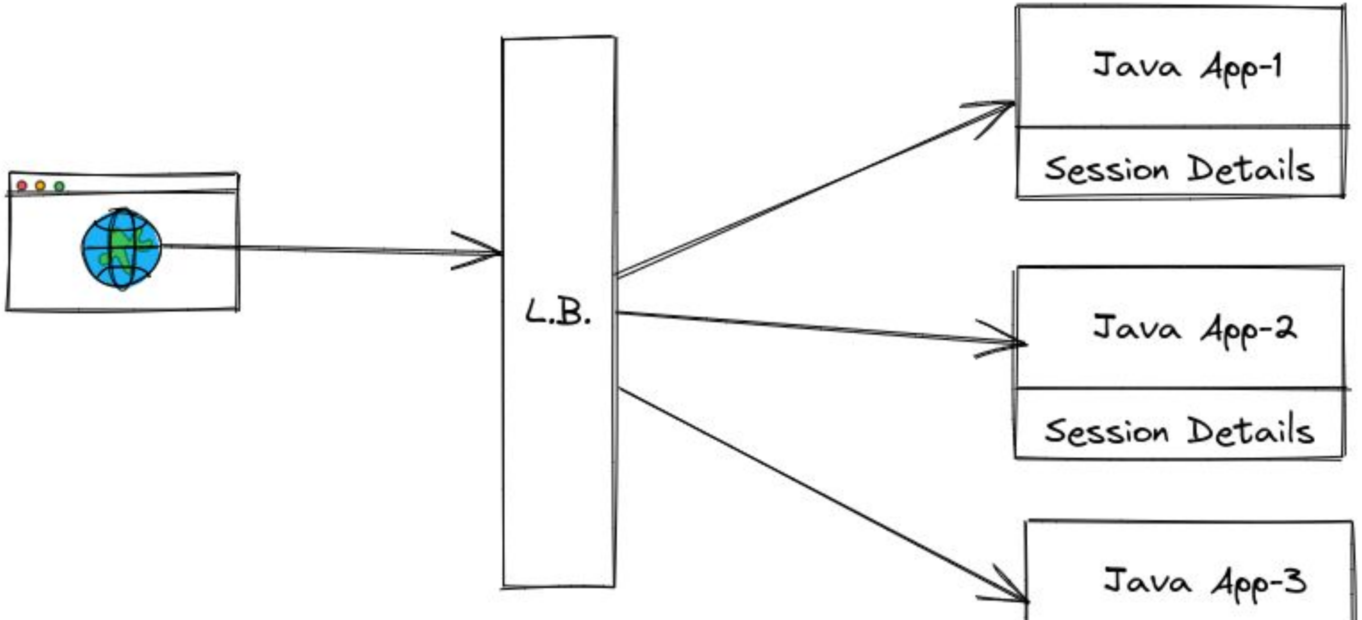
Exploit GET API: Send email to user with links of GET APIs that updates data in backend. E.g. changePassword API.

Exploit POST API: Create a form with fixed values and submit the form on page load via javascript.

To protect MVC applications, **Spring adds a CSRF token to each generated view.** This token must be submitted to the server on every HTTP request that modifies state (PATCH, POST, PUT, and DELETE – not GET). This protects our application against CSRF attacks since an attacker can't get this token from their own page.

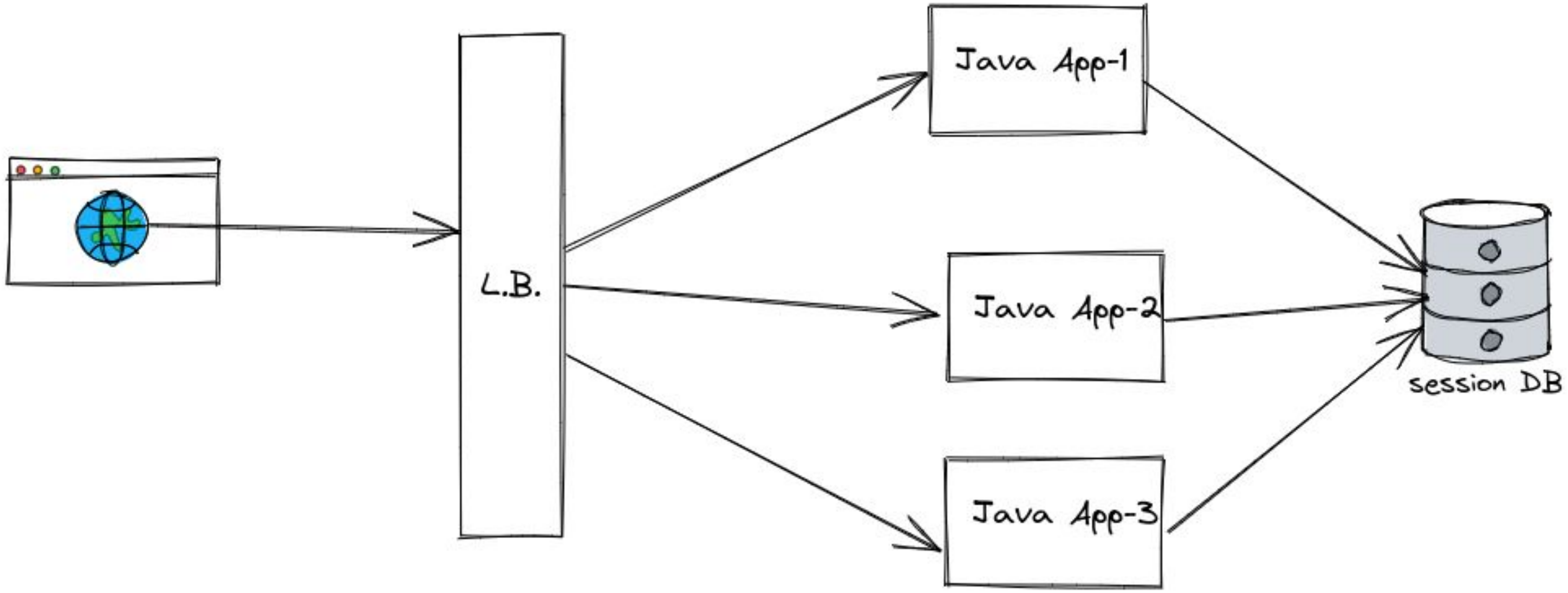
Distributed Session Management

In case of memory based session storage Java Application will store session details in JVM only. If we **run multiple instances of application behind load balancer** there might me a case where user's request route to **different server** then application will ask user to **login again**.



Distributed Session Management(Solutions)

- configure the load balancer to be **sticky**
- configure the servers to use persistent sessions in central database/redis.
- configure the servers in a cluster, and to distribute/replicate the sessions on all the nodes of the cluster



Distributed Session with Redis

Dependencies:

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
		<groupId>org.springframework.session</groupId>
		<artifactId>spring-session-data-redis</artifactId>
</dependency>
```

Properties:

```
spring.redis.host=localhost
spring.redis.port=6379
spring.redis.password=
spring.session.store-type=redis
```

Minor-project-2

Add required dependencies spring security and redis

Changes in Entities to support MySQL based UserStore.

Defining authorization based on API url patterns and Roles of user.

Create username field or we can user email as username.

Define roles or authorities.

Redis in Minor-project

Identify APIs and Flows where the cache can be used.