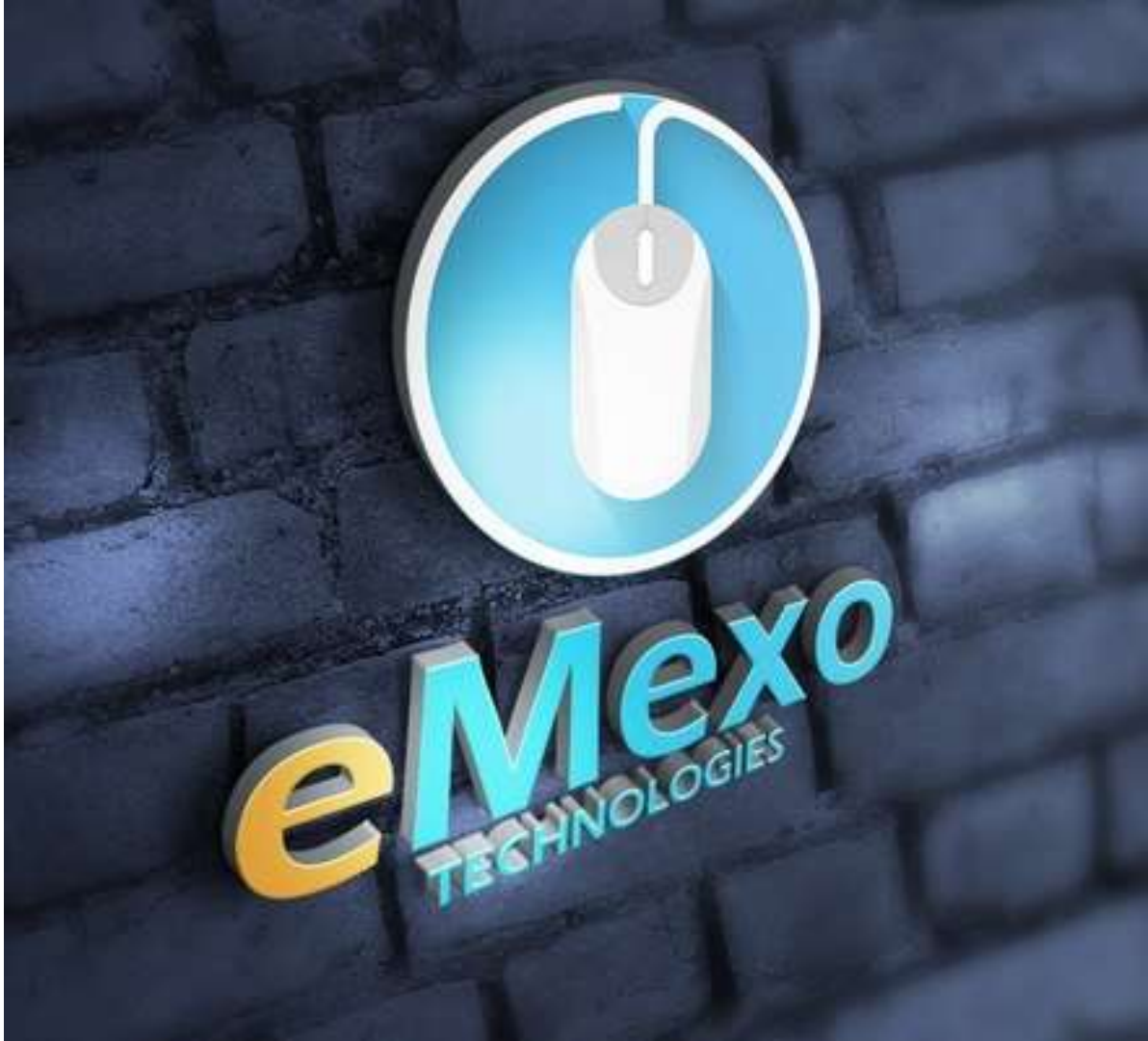
Welcome



Spring Framework

ph no: 9513216462 | email : info@emexotechnologies.com | website: https://www.emexotechnologies.com

Spring Security

What is Spring Security?

- Spring Security is an important feature that often goes well with the Spring Web MVC framework.
- Spring Security is a framework that enables a programmer to impose security restrictions to Spring-framework-based Web applications through JEE components.
- In short, it is a library that can be used, extended to customize as per the programmer's needs. Because it is a member of the same Spring family, it goes smoothly hand in hand with the Spring Web MVC.
- Its primary area of operation is to handle authentication and authorization at the Web request level as well as the method invocation level.

What is Spring Security? (Contd)

- The greatest advantage of this framework is that it is powerful yet highly customizable in its implementation.
- Although it follows Spring's convention over configuration, programmers can choose between default provisions or customizing it according to their needs.

Why should we use spring ?

- Web applications are vulnerable to security threats because they are exposed to the open world of the Internet.
- Access to certain Web pages, files, or other classified resources must be restricted to authorized personnel only.
- Of course, there are several layers of security that are often applied, such as firewall, proxy server, JVM security, and so forth.
- But, to control access, there must be some security restriction at the application level as well.
- Therefore, Spring Security, a part of the Spring Framework, is only an advice or provision to apply a level of security at the Java Application stratum.

Spring Security Features?

- Spring Security operates on two major areas
- Authentication and Authorization

What is Authentication?

- Authentication means that, while accessing certain restricted resources, the user actually is the right person to do so.
- There are two processes to make sure that the user is authentic: identification and verification.
- For example, a user is authenticated through their username and password, which is typically stored in a database; LDAP (Lightweight Directory Access Protocol, a lightweight protocol for accessing directory services); or CAS (Central Authentication Service, a single sign-on protocol for the Web).
- Spring Security also has required an interface to encode the password to make it more secure.

What is Authorization?

- Authorization determines the extent of a user's right to access restricted resources. It ensures that a user is allowed to access only those parts of the resource that one has been authorized to use.
- For example, an ADMIN user has unlimited access to application properties and can change or manipulate them—for good or for worse
- A normal USER or a GUEST user, on the other hand, has more controlled access and does not enjoy the same rights as the ADMIN user. This is called user role authorization.
- In any Web application, this is done through URL-based security. Spring provides filters to ensure the role of securing an application.

What is Authorization? (Contd..)

- But, URL-based security is not a very clever mechanism and often can be misused. Malicious users can manipulate the URL and gain access to a method that actually is meant for an administrative user.
- Because, in a URL-based system, restricted method access invocations are sent through hyperlinks, it quite easy to re-create the same method invocation from the URL and send it to the server.
- The server may naively execute the restricted operations without verifying the role of the user who invoked the request. Therefore, to tackle this problem, Spring offers method-level security.
- This simply means that only certain authorized users can invoke restricted methods and simply re-creating the URL and sending it to the server will not execute them.

Spring Security Annotations

- The annotation @EnableWebSecurity enables Web security; otherwise, it remains disabled by default.
- Now, to configure the security, we can either implements the interface called WebSecurityConfigurer or extend the more convenient class called WebSecurityConfigurerAdapter.
- The advantage of extending the adapter class is that we can configure Web security by overriding only those parts that we are interested in; others can remain their default form.

WebSecurityConfigurerAdapter

- There are three variations of the configure method that we can override to configure and secure the application:
- void configure(AuthenticationManagerBuilder auth)

  To configure user details services
- void configure(HttpSecurity http)

  To configure how requests are secured by interceptors
- void configure(WebSecurity web)

  To configure Spring Security's filter chain

The default filter chain is fine for most needs. So, we may configure the other two in the following manner.

configure(AuthenticationManagerBuilder auth)

```
public class SecurityConfiguration extends
WebSecurityConfigurerAdapter {
@Override
protected void configure(AuthenticationManagerBuilder auth)
throws Exception {
      auth.inMemoryAuthentication()
      .withUser("user").password("password").roles("USER").and(
      ) .withUser("admin").password("password").roles("USER",
      "ADMIN");
      }
       // ...
}
```

configure(HttpSecurity http)

```
public class SecurityConfiguration extends
WebSecurityConfigurerAdapter {

@Override
protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests() .antRequest().authenticated()
      .and().formLogin().and().httpBasic();
 }

// ... }
```

Spring Method security

- In order to enable Spring Method level Security, we need to annotate a @Configuration class with @EnableGlobalMethodSecurity

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
 public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
 @Autowired
 public void configureGlobalSecurity(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("bill").password("abc123").roles("USER");
    auth.inMemoryAuthentication().withUser("admin").password("root123").roles("ADMIN");
    auth.inMemoryAuthentication().withUser("dba").password("root123").roles("ADMIN","DBA");   }

@Override
protected void configure(HttpSecurity http) throws Exception {
     http.authorizeRequests()
    .antMatchers("/", "/home").access("hasRole('USER') or hasRole('ADMIN') or hasRole('DBA')")
    .and().formLogin().loginPage("/login").usernameParameter("ssoId").passwordParameter("password")
    .and().exceptionHandling().accessDeniedPage("/Access_Denied");
}         }
```

Spring Method security (Contd..)

- @EnableGlobalMethodSecurity enables Spring Security global method security
- Note that @EnableGlobalMethodSecurity can take several arguments, some are shown below:
- prePostEnabled :

  Determines if Spring Security's pre post annotations [@PreAuthorize,@PostAuthorize,..] should be enabled.
- secureEnabled :

  Determines if Spring Security's secured annotation [@Secured] should be enabled.

@Secured

- @Secured annotation is used to define a list of security configuration attributes for business methods.
- You can specify the security requirements[roles/permission etc] on a method using @Secured, and than only the user with those roles/permissions can invoke that method.
- If anyone tries to invoke a method and does not possess the required roles/permissions, an AccessDenied exception will be thrown.
- @Secured is coming from previous versions of Spring.
- It has a limitation that it does not support Spring EL expressions.
- Consider following example:

@Secured (Contd..)

- **Consider following example:**

```
public interface UserService {

    List<User> findAllUsers();

    @Secured("ROLE_ADMIN")
     void updateUser(User user);

    @Secured({ "ROLE_DBA", "ROLE_ADMIN" })
     void deleteUser();

}
```

@Secured (Contd..)

- In above example, updateUser method can be invoked by someone with ADMIN role while deleteUser can be invoked by anyone with DBA OR ADMIN role.
- If anyone tries to invoke a method and does not possess the required role, an AccessDenied exception will be thrown.
- What if you want to specify an ‘AND’ condition. I mean , you want deleteUser method to be invoked by a user who have both ADMIN & DBA role.
- This is not possible straight-away with @Secured annotation.
- This can however be done using Spring’s new @PreAuthorize/@PostAuthorize annotations which supports Spring EL, that means possibilities are unlimited.

@PreAuthorize / @PostAuthorize

- Spring's @PreAuthorize/@PostAuthorize annotations are preferred way for applying method-level security, and supports Spring Expression Language out of the box, and provide expression-based access control.
- @PreAuthorize is suitable for verifying authorization before entering into method. @PreAuthorize can take into account, the roles/permissions of logged-in User, argument passed to the method etc.
- @PostAuthorize , not often used though, checks for authorization after method have been executed, so it is suitable for verifying authorization on returned values. Spring EL provides returnObject object that can be accessed in expression language and reflects the actual object returned from method

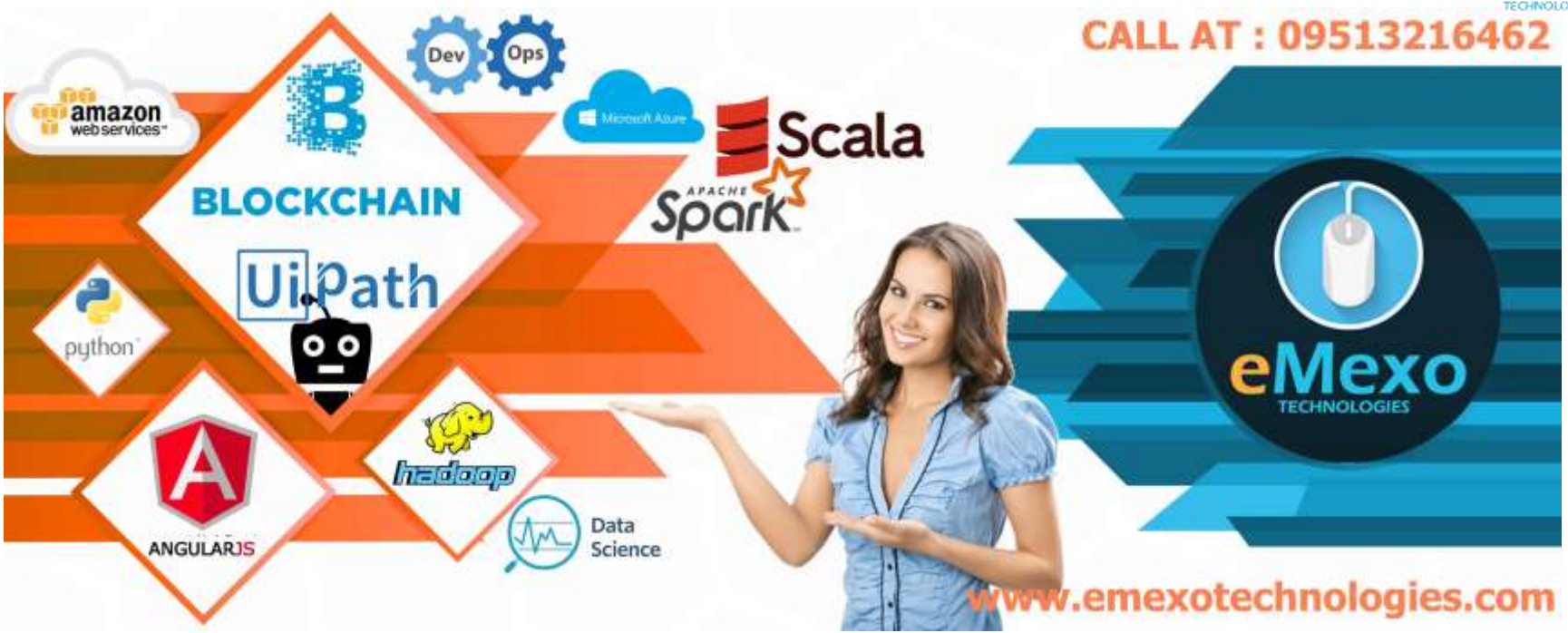
@PreAuthorize / @PostAuthorize (Contd..)

```
public interface UserService {

    List<User> findAllUsers();

    @PostAuthorize ("returnObject.type == authentication.name")
    User findById(int id);

    @PreAuthorize("hasRole('ADMIN')")
    void updateUser(User user);

    @PreAuthorize("hasRole('ADMIN') AND hasRole('DBA')")
    void deleteUser(int id);
}
```

@PreAuthorize / @PostAuthorize

- Since @PreAuthorize can use Spring Expression Language, any condition can easily be expressed using EL. Method deleteUser is now configured to be invoked by a user who have both ADMIN & DBA roles.

- Additionally, we have added a method findById() with @PostAuthorize annotation. With @PostAuthorize, the returned value from the method(User object) will be accessible with returnObject in Spring Expression Language, and individual properties of return user object can be used to apply some security rules. In this example we are making sure that a logged-in user can only get it's own User type object.


eMexo
TECHNOLOGIES
CALL AT : 09513216462
amazon web services
Dev
Ops
Microsoft Azure
Scala
APACHE Spark
BLOCKCHAIN
UiPath
python
ANGULARJS
hadoop
Data Science
eMexo
TECHNOLOGIES
www.emexotechnologies.com

THANK YOU!

Any questions?

You can find us at

- Email: info@emexotechnologies.com
- Call/WhatsApp: 9513216462