Authentication and Authorization in Spring Boot 3.0 with Spring Security

Last Updated : 23 Jul, 2025

In Spring Security 5.7.0, the spring team deprecated the WebSecurityConfigurerAdapter, as they encourage users to move towards a component-based security configuration. Spring Boot 3.0 has come with many changes in Spring Security. So in this article, we will understand how to perform spring security authentication and authorization using Spring Boot 3.0.

Demo Project

Step 1: Create a New Spring Boot Project in Spring Initializr

To create a new Spring Boot project, please refer to How to Create a Spring Boot Project in Spring Initializr and Run it in IntelliJ IDEA. For this project choose the following things

- Project: Maven
- Language: Java
- Packaging: Jar
- Java: 17

Please choose the following dependencies while creating the project.

- Spring Web
- Spring Security

Below is the complete pom.xml file. Please cross-verify if you have missed some dependencies

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter Web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot Starter Security -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <!-- Spring Boot Starter Test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- Spring Security Test for Testing Security Configurations -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Spring Boot Maven Plugin -->
            <plugin>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Step 2: Create a UserController

Go to the src > main > java > controller and create a class UserController and put the below code. In this, we have created a simple REST API in our controller class.

```
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/auth")
public class UserController {

    @GetMapping("/welcome")
    public String welcome() {
        return "Welcome, this endpoint is not secure";
    }

    @GetMapping("/user/userProfile")
    @PreAuthorize("hasRole('USER')")  // Use hasRole for role-based access
control
    public String userProfile() {
        return "Welcome to User Profile";
    }

    @GetMapping("/admin/adminProfile")
    @PreAuthorize("hasRole('ADMIN')")  // Use hasRole for role-based access
control
    public String adminProfile() {
        return "Welcome to Admin Profile";
    }
}
```

Step 3: Create a SecurityConfig Class

Go to the **src > main > java > config** and create a class **SecurityConfig** and put the below code. This is the new changes brought in Spring Boot 3.0.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
```

```
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import static org.springframework.security.config.Customizer.withDefaults;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    // User Creation
    @Bean
    public UserDetailsService userDetailsService(PasswordEncoder encoder) {

        // InMemoryUserDetailsManager setup with two users
        UserDetails admin = User.withUsername("Amiya")
                .password(encoder.encode("123"))  // <-- Encode the password
                .roles("ADMIN", "USER")
                .build();

        UserDetails user = User.withUsername("Ejaz")
                .password(encoder.encode("123"))  // <-- Encode the password
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(admin, user);
    }

    // Configuring HttpSecurity
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf -> csrf.disable()) // Disable CSRF for simplicity
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/auth/welcome").permitAll() // Permit all
access to /auth/welcome
                .requestMatchers("/auth/user/**").authenticated() // Require
authentication for /auth/user/**
                .requestMatchers("/auth/admin/**").authenticated() // Require
authentication for /auth/admin/**
            )
            .formLogin(withDefaults()); // <-- Use withDefaults() for form-
based login
```

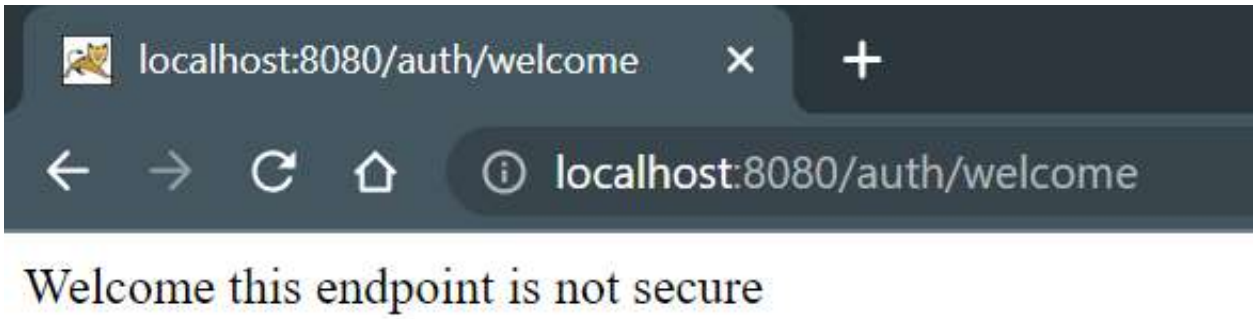
```
        return http.build();
    }

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

Test the Application

Now run your application and test it out. Hit the following URL

```
http://localhost:8080/auth/welcome
```

You can access this endpoint without any authentication as it is not secured.



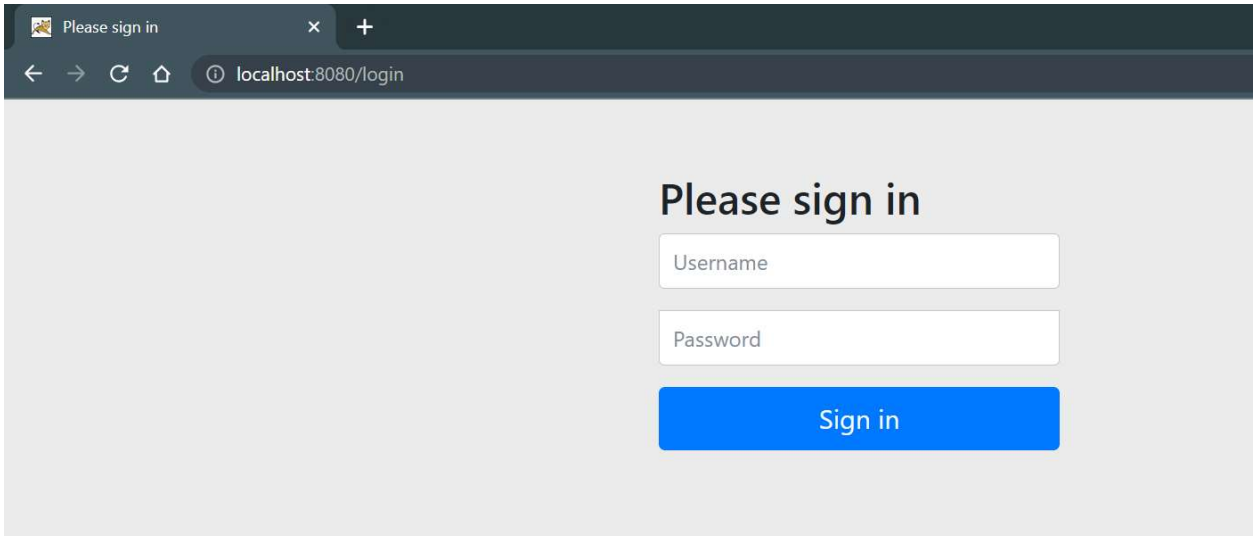
Now, hit the following URL:

```
http://localhost:8080/auth/user/userProfile
```

If Not Logged In: You will be redirected to the below URL:

```
http://localhost:8080/login
```

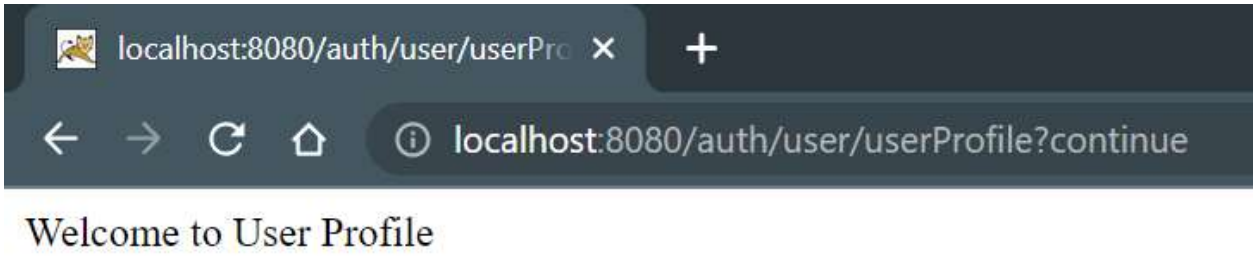
Output:



After putting the correct Username and Password you can access your endpoint. Put this Username and Password

- Username: Ejaz
- Password: 123

And you will get the output screen like this,



After logging in with the correct credentials, you will be able to access this endpoint if your role includes USER.

Similarly, you can hit and try other Users and play with it.