Adding Spring Security to CRM REST API

Introduction

In this lecture, you will learn how to integrate our CRM REST API with Spring Security.

GOALS

- Secure REST APIs
- Restrict REST APIs based on roles

TECHNICAL APPROACH

- Use Spring all Java configuration (no xml)
- Use Maven for project dependency management

Overview of Steps

1. Add new Maven Dependencies for Spring Security
2. Enable Spring Security Filters
3. Secure REST endpoints

See below for details.

1. Maven POM file

1. Add new Maven Dependencies for Spring Security

Open the file: pom.xml

Add new entries for Spring Security

```
<properties>
        ...
        <springsecurity.version>5.0.5.RELEASE</springsecurity.version>
        ...
</properties>

...

        <!-- Spring Security -->
        <!-- spring-security-web and spring-security-config -->

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>
```

2. Enable Spring Security Filters

1. In the package **com.luv2code.springdemo.config**,create a new class named: **SecurityWebApplicationInitializer**

Update the file with the code at this link: https://bit.ly/2kRGiaD

This file enables the Spring Security filters for the web application.

3. Secure REST endpoints

a. In the package **com.luv2code.springdemo.config**, create a new class named: **DemoSecurityConfig**

Update the file with the code at this link: https://bit.ly/2sE26L2

This configuration class adds three sample users. Currently we're using in-memory authentication to keep things simple. But you could easily use database storage with encrypted passwords. We covered db storage in previous videos.

4. Test the App with Postman

It is best to test with Postman. Because Postman will prompt for new credentials for each REST request.

1. Run your REST application.

2. In Postman access the REST endpoint: GET /api/customers

You will initial get a 401 error: Unauthorized

3. To resolve this, in Postman, click the "Authorization" section of the request.

4. In the "Type" drop-down list, select "Basic Auth"

5. Enter user id: john, password: test123

6. Using Postman send the request again

If the user is authenticated, then they'll get the results of REST request. If not, they'll see a 401 error

5. Secure REST APIs by Role

In this example, we'll secure all REST endpoints under "/api/customers" and adds following security authorizations

EMPLOYEE role can perform following

1. Get a list of all customers. GET /api/customers
2. Get a single customer. GET /api/customers/{customerId}

MANAGER role can perform following

1. Add a new customer. POST /api/customers
2. Update an existing customer. PUT /api/customers

ADMIN role can perform following

1. Delete a customer. DELETE /api/customers/{customerId}

1. Open the file DemoSecurityConfig.java

Update the file with the code at this link: https://bit.ly/2Lto8HA

Make note of the security rules for the various endpoints

```
.antMatchers(HttpMethod.GET, "/api/customers").hasRole("EMPLOYEE")
.antMatchers(HttpMethod.GET, "/api/customers/**").hasRole("EMPLOYEE")
.antMatchers(HttpMethod.POST, "/api/customers").hasAnyRole("MANAGER", "ADMIN")
.antMatchers(HttpMethod.POST, "/api/customers/**").hasAnyRole("MANAGER", "ADMIN")
.antMatchers(HttpMethod.PUT, "/api/customers").hasAnyRole("MANAGER", "ADMIN")
.antMatchers(HttpMethod.PUT, "/api/customers/**").hasAnyRole("MANAGER", "ADMIN")
.antMatchers(HttpMethod.DELETE, "/api/customers/**").hasRole("ADMIN")
```

This only allows access to the given endpoints based on the role. The use of "**" makes sure to secure endpoints if user enters additional information at the end of the URL.

6. CustomerRestController – Full CRUD Code

To fully test the different options, you'll need code in CustomerRestController that supports POST, PUT and DELETE.

Update your CustomerRestController file with the code at this link:
https://bit.ly/2Loj1YU

Make note of the mappings for @PostMapping, @PutMapping and @DeleteMapping.

7. Test the App with Postman

Walk through the different security scenarios

1. Confirm that users with EMPLOYEE role can access
    1. Get a list of all customers. GET /api/customers
    2. Get a single customer. GET /api/customers/{customerId}

    Use credentials: john / test123

2. Confirm that users with MANAGER role can access
    1. Add a new customer. POST /api/customers
    2. Update an existing customer. PUT /api/customers

To Add a new customer

POST http://localhost:8080/spring-crm-rest/api/customers

In Postman, be sure to select options: Body > raw > JSON (application/json)

Put the following JSON in request message body.

```
{
    "firstName": "Doug",
    "lastName": "Pederson",
    "email": "doug@luv2code.com"
}
```

Use credentials: mary / test123

To Update an existing customer

PUT http://localhost:8080/spring-crm-rest/api/customers

In Postman, be sure to select options: Body > raw > JSON (application/json)

Put the following JSON in request message body.

```
{
    "id": 1,
    "firstName": "Doug",
    "lastName": “PhillyPhilly”,
    "email": "doug@luv2code.com"
}
```

3. Confirm that users with ADMIN role can access

    1. Delete a customer. DELETE /api/customers/{customerId}

    To Delete a customer (id=2)

    DELETE http://localhost:8080/spring-crm-rest/api/customers/2

    Use credentials: susan / test123

8. Download Full Source Code

The full source code for this example is available here: https://bit.ly/2JeIFCN

That's it!