Applying Security On Spring Boot Rest Application using spring boot 3.x

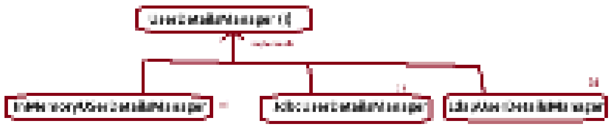
note:: In spring boot 3.x , the WebSecurityConfigurerAdapter is initially deprecated and then removed
So we need to develop the SecurityConfig class in spring boot 3.x having lots of @Bean methods
by getting few objects related to Security through AutoConfiguration to the parameters of of
@Bean methods

In spring boot 3.x security we have got lots of AuthenticationProvider classes as the predefined classes
to use different ones as the Authentication Info Providers .. we can make them as spring beans in
SecurityConfig class using @Bean methods and also using Build Design Pattern that internally uses
method Chaining concept

and store
InMemoryUserDetailsManager ---> To create username, password and roles as InMemory Details

JdbcUserDetailsManager ---> To create and store username, password and roles in the InMemory or external DB s/w

and etc...



In SecurityConfig class , we need to take one @Bean method returning SecurityFilterChain obj
as the spring bean having configurations related Authentication and Authorization

SecurityFilterChain (I)
implements
DefaultSecurityChain(C)

Defines a filter chain which is capable of being matched against an HttpServletRequest. in order to decide whether it applies to that request.

Example App [spring boot 3.x Security on Spring boot Rest App using InMemoryUserDetailsManager]

step1) create spring starter project adding the following starters
a) spring web b) spring security c) dev tools

step2) Add the following entries in application.properties

```
application.properties
-----------------------
spring.application.name=BootRestSecurityProj01-InMemoryDB-3.x

# Embedded Tomcat server port
server.port=1010

# specify the context path of the application
server.servlet.context-path=/BootSecurityApp01
```

step3) Create RestController class(es) having rest operation methods mapped with different urls

```
package com.nt.controller;

import java.util.Map;
import java.util.Random;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
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("/bank") //global path
public class BankOperationsController {
                                                    ResponseEntity object is the object that contains
                                                    output or response content and response status code
    @GetMapping("/welcome")
    public ResponseEntity<String> showHome() {
        return new ResponseEntity<String>("Welcome to home Page", HttpStatus.OK);
    }

    @GetMapping("/offers")
    public ResponseEntity<String> showOffers() {
        return new ResponseEntity<String>("offers page", HttpStatus.OK);
    }

    @GetMapping("/loan_approve")
    @PreAuthorize("hasAuthority('MANAGER')") //specifying authorization
    public ResponseEntity<String> approveLoan() {
        int amount=new Random().nextInt(200000);
        return new ResponseEntity<String>("Loan approved , the amount of amount::"+amount, HttpStatus.OK);
    }

    @GetMapping("/balance")
    @PreAuthorize("hasAnyAuthority('MANAGER','CUSTOMER')") //specifying the authorization
    public ResponseEntity<String> checkBalance() {
        int amount=new Random().nextInt(200000);
        return new ResponseEntity<String>("balance is ::"+amount, HttpStatus.OK);
    }

}
```

step4) Develop the SecurityConfig class as shown below having @Bean methods

```
//SecurityConfig.java
package com.nt.config;

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

Applying Security On Spring Boot Rest Application

using spring boot 3.x

===========

note:: In spring boot 3.x, the WebSecurityConfigureAdaptor is initially deprecated and then removed So we need to develop the SecurityConfig class in spring boot 3.x having lots of @Bean methods by getting few objects related to Security through AutoConfiguration to the parmeters of of @Bean methods

In spring boot 3.x security we have got lots of

AuthenticationProvider classes as the pre-defined classes

to use different envs as the Authentication Info Providers .. we can make them as spring beans in SecurityConfig class using @Bean methods and also using build Design Pattern that internally uses method Chaining concept

and store

InMemoryUserDetailsManager ---> To create username,password and roles as InMemory Details

JdbcUserDetailsManager ---> to create and store username, password and roles in the InMemory or external DB s/w and etc...

UserDetailsManager (1)

InMemoryUserDetails Manager (c)

implements

(c)

(c)

JdbcUserDetails Manager LdapUserDetailsManager

In SecurityConfig class, we need to take one @Bean method returing SecurityFilterChain obj

as the spring bean having configurations related Authentication and Authorizations

to

SEcurityFilterChain (1)

↑

implements

DefaultSEcurityChain(c)

Defines a filter chain which is capable of being matched against an HttpServletRequest. in order to decide whether it applies to that request.

Example App

(spring boot 3.x Security on Spring boot Rest App using InMemoryUserDetails Manager)

step1) create spring starter project adding the following starters a) spring web b) spring security c) dev tools

step2) Add the following entries in application.properties

application.properties

step3)

spring.application.name=BootSecurityProj03-InMemoryDB-3.x

# Embedded tomcat server port server.port=4041

# specify the context path of the application server.servlet.context-path=/FirstSecurityApp1

Create RestController classe(s) having rest operation methods mapped with different urls

```
package com.nt.controller;

import java.util.Map;

import java.util.Random;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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("/bank") //global path

public class BankOperationsController {

@GetMapping("/welcome")

public ResponseEntity<String> showHome() {

return new ResponseEntity<String>("welcome to home Page::", HttpStatus.OK);
```

ResponseEntity object is the object that contains output + response status code + response hearders

```
}

}

@GetMapping("/offers")

public ResponseEntity<String> showOffers() {

return new ResponseEntity<String>("offers page ",HttpStatus.OK);

}

@GetMapping("/loan_approve")

@PreAuthorize("hasAuthority('MANAGER')") //specifying authrorization

public ResponseEntity<String> approveLoan() {

}

int amount=new Random().nextInt(1200000);

return new ResponseEntity<String>("Loan Approved, the amount is::"+amount, HttpStatus.OK);

@GetMapping("/balance")

@PreAuthorize("hasAnyAuthority('MANAGER', 'CUSTOMER')") //specifying the authrorization public
ResponseEntity<String> showBalance() {

int amount=new Random().nextInt(200000);

return new ResponseEntity<String>("Balance is ::"+amount, HttpStatus.OK);

}
```

step4) Develop the SecurityConfig class as shown below having @Bean methods

```
//SecurityConfig.java
```

```
package com.nt.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
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.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password. PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.NoOpAuthenticationEntryPoint;
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {
@Bean
public SecurityFilterChain defaultSecurityFilterChain (HttpSecurity http) throws Exception {
http.authorizeHttpRequests((requests)->requests
.requestMatchers("/bank/welcome").permitAll()
.requestMatchers("/bank/balance","/bank/loan_approve","/bank/offers").authenticated())
.formLogin(Customizer.with Defaults());
return http.build(); //returns DefaultSecurityFilterChain obj which is the impl class obj of
SecurityFilterChain(1)
}
@Bean
public InMemoryUserDetails Manager userDetailsService() {
//*Approach 1 where we use with DefaultPassword Encoder()(deprecated) method while creating the user
details//*
/*UserDetails admin = User.with DefaultPasswordEncoder()
.username("admin")
.password("12345")
.authorities("admin")
.build();
UserDetails user = User.with DefaultPasswordEncoder()
.username("user")
```

.password("12345")

.authorities("read")

.build();

return new InMemoryUserDetails Manager(admin, user);*/

//*Approach 2 where we use BCryptPasswordEncoder Bean UserDetails user1 = User.withUsername("raja") .password(encoder().encode("rani"))

.authorities("CUSTOMER")

.build();

UserDetails user2= User.withUsername("mahesh")

.password(encoder().encode("hyd"))

.authorities("MANAGER","CUSTOMER")

.build();

return new InMemoryUserDetails Manager(user1, user2);

}

@Bean

public Password Encoder encoder() {

return new BCryptPasswordEncoder();

}

}

step6) Run the Application by Configuring tomcat 10.x server

localhost

X

while creating the user details//*

localhost:3131/BootSecurityPr

You are s

welcome to home Page::

localhost:3131/BootSecurity Proj03-In MemoryDB-3.x/bank/welcome

localhost:3131/BootSecurityProj03-In MemoryDB-3.x/bank/offers

Please sign in - localhost:3131/BootSecurityProj03-InMemoryDB-3.x/bank/offers

IT will be recirected to

localhost:3131/BootSecurity Proj03-In MemoryDB-3.x/login

Please sign in

raja

(rani)

Sign in

>

BootSecurityProj04-Spring DataJPA-3.x [boot] [devtools]

Deployment Descriptor: BootSecurity Proj04-Spring DataJPA-3.>

> AJAX-WS Web Services

✓ Java Resources

>

*

src/main/java

#com.nt

› BootSecurity Proj03 In MemoryDbApplication.java

ServletInitializer.java

>

com.nt.config

> SecurityConfig.java

com.nt.controller

> BankOperationsController.java

src/main/resources

static

templates

application.properties

src/test/java

target/generated-sources/annotations

target/generated-test-sources/test-annotations

Libraries

>

>

>

Deployed Resources

>

src

>

tarnot

←

offers page

localhost:3131/BootSecurityProj03-In MemoryDB-3.x/bank/offers?continue

(uses the DB s/w as the Authentication Provider) Spring boot 3.x Security Example App using JdbcDetailsUserManager on spring Rest App

note:: JdbcDetailsUserManager takes the DataSource obj as dependent obj and uses the DB s/w that is represented by the DataSource obj as the Authentication Provider.. In that provider fixed schema (fixed table names, col names) should be used to maintain the usernames, passwords and roles

step1) take

copy of the Project

step2) Add the following additional staters

a) jdbc api b) MySQL driver (new)

c) security d) web (old starters)

step3)

add the following entries in application.properties file for data source configuration

#jdbc properites for mysql

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql:///ntspbms516db

step4)

spring.datasource.username=root

spring.datasource.password=root

create the following db tables having fixed table names and col names

use ntspbms516db;

CREATE TABLE users (

Parent db table

username VARCHAR(50) NOT NULL,

password VARCHAR(100) NOT NULL,

);

enabled TINYINT NOT NULL DEFAULT 1,

PRIMARY KEY (username)

executes these queries

CREATE TABLE authorities (

username VARCHAR(50) NOT NULL,

Child db table authority VARCHAR(50) NOT NULL,

FOREIGN KEY (username) REFERENCES users(username)

step5) insert records in to the above db tables representing the user names, passwords and roles

users (dB)

.parent

Result Grid

Filter Rows:

`Edit:`

Export/Import:

use ke password

mahesh

raja

NULL

NULL

$2a$10$/JADWLu6mFbf5F.uIzLDduBrLcr 7NJEVOM6a...

enabled

$2a$10$w/oJmsHS/Q63VazAk4wxbeIDOYWFHNbhs7...

1

1

NULL

encoded password using BcryptEncoder

authorities (child db table)

Result Grid 月

Filter Rows:

E

username

(FK)

authority

raja

CUSTOMER

mahesh

CUSTOMER

mahesh

MANAGER

step6)

remove

second @Bean method in SecurityConfig class (InMemoryUserDetails Manager) and

place @Bean method for making JdbcUserDetails Manager class obj as the spring bean assigning DataSource obj as the dependent obj by gathering it through AutoConfiguration

@Bean

public UserDetails Manager createJdbcUDM(DataSource ds) {

return new JdbcUserDetails Manager(ds); //For this we need db tables and their cols having fixed names

}

@Bean

public BCryptPassword Encoder encoder() { return new BCryptPassword Encoder();

}

step7) Run the Application

The above JdbcUSerDetails Manager class obj use this PasswordEnconder obj internally

BootSecurity Proj05-Spring Boot3.x-DB [boot] [devtools] Deployment Descriptor: BootSecurity Proj05-Spring Boot3.x-DB

JAX-WS Web Services

>

>

Java Resources

W

src/main/java

W

#com.nt

<

BootSecurityProj05DbApplication.java

ServletInitializer.java

com.nt.config

> SecurityConfig.java

com.nt.controller

> BankOperationsController.java

src/main/resources

static

templates

application.properties

> src/test/java

>

<

target/generated-sources/annotations target/generated-test-sources/test-annotations

Libraries

>

>

Deployed Resources

<

src

>

target

HELP.md

mvnw

mynw.cmd

=>In Spring boot 3.x, We need to configure @Bean method returning SecurityFilterChain object having Authentiation Configurations, we can place authroizations at method level of @Controller or @RestController class

=>In Spring boot 3.x, We need to configure @Bean method returning UserDetails Manager object having

Authentication Info provider cfgs (like Inmemory DB, JDBCAuthentication and etc...)