Secure you REST APIs and Web Applications in easy steps

# The Funtamentals

## What is Security

Pics=============

## The key security components

Pic A

Authentication filter

Its get user credential and check in Security context if user is logged in or not

If not Its send to authentication manager

Authentication Managaer get user credential and send to authentication provider

Authentication provider ihas login business logic

It uses userDetailsService and password encoder to login the user

## Spring Security

Pic

By adding spring sercurity repositaory it provide default basic auth

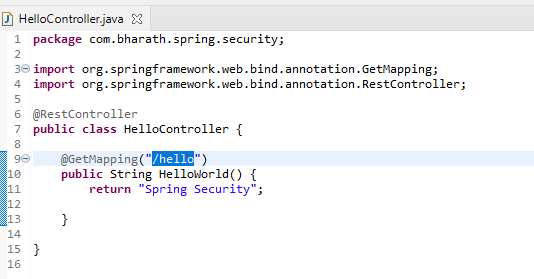
To override you can create spring configuration

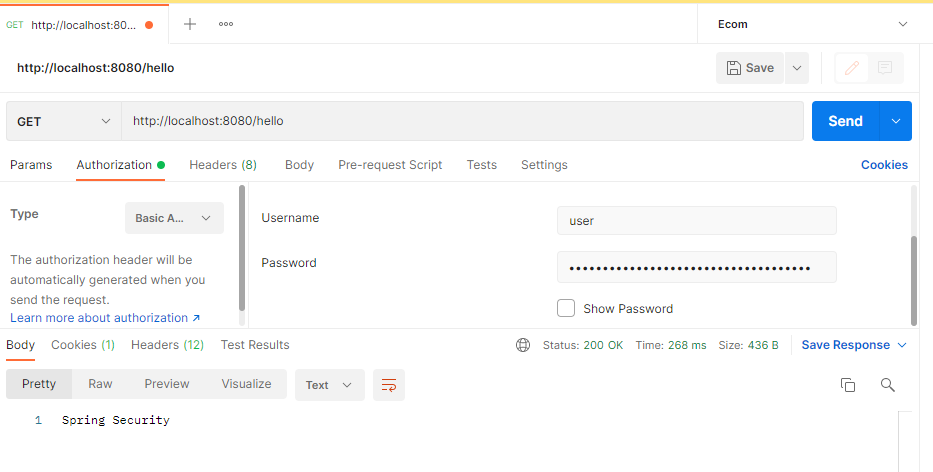
We can create own userdetailservice

We can create own authentication provider

## Spring Security in action

Cretate project with spring security dependency

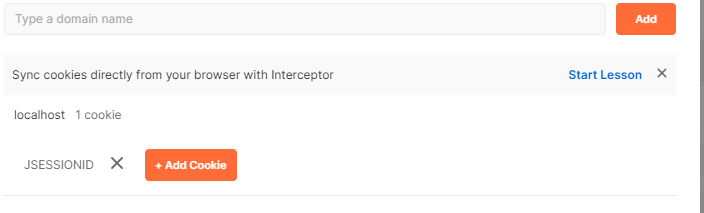




## Resending Basic Auth Details

When we login first time auth is successful it generates the jsessionid cookie. When we send another req this jsession id will send together.

First login Authenticationfilter check jsession id with security context. Jsessionid is key store user details in system context.



## Create Custom Security Configuration

Here we create own configuration for authentication manager and provider and auth type

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**public** **class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {

//own authendication manager for login process

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic();

http.authorizeRequests().anyRequest().authenticated();

}

}

## Use custom UserDetailsService

Authentication Manager has Authentication provider which use userDetailService has username and password. Authentication provider uses passwordEncoder to encode password.

**public** **class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

BCryptPasswordEncoder passwordEnoder = **new** BCryptPasswordEncoder();

InMemoryUserDetailsManager userDetailService = **new** InMemoryUserDetailsManager();

UserDetails user = User.*withUsername*("tom").password(passwordEnoder.encode("cruise")).authorities("read").build(); //Creating the user

userDetailService.createUser(user);//adding created use to userDetail Service

auth.userDetailsService(userDetailService).passwordEncoder(passwordEnoder);//providing our own userDetailService and passwordEncoder to authendicaion manager

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic();

http.authorizeRequests().anyRequest().authenticated();

}

}

## Tweak the PasswordEncoder

We create passwordEncoder bean so that authenticationmanagerbuilder get this password encoder bean

**public** **class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** PasswordEncoder passwordEnoder;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

// BCryptPasswordEncoder passwordEnoder = new BCryptPasswordEncoder();

InMemoryUserDetailsManager userDetailService = **new** InMemoryUserDetailsManager();

UserDetails user = User.*withUsername*("tom").password(passwordEnoder.encode("cruise")).authorities("read").build();

userDetailService.createUser(user);

//auth.userDetailsService(userDetailService).passwordEncoder(passwordEnoder);

auth.userDetailsService(userDetailService); //If we define BCryptPasswordEncoder bean then authenticationmanagerBuilder gets that passwordEncoder bean

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic();

http.authorizeRequests().anyRequest().authenticated();

}

@Bean

**public** BCryptPasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

}

## Create Custom AuthenticationProvider

We can create our own authentication by creating authentication provider.

Authenticate() this method has login process

Support() method tells which type of authentication its supports.

**import** java.util.Arrays;

**import** org.springframework.security.authentication.AuthenticationProvider;

**import** org.springframework.security.authentication.BadCredentialsException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.AuthenticationException;

@Component

**public** **class** MyAuthenticationProvider **implements** AuthenticationProvider {

@Override

**public** Authentication authenticate(Authentication authentication) **throws** AuthenticationException {

String userNAme = authentication.getName();

String password = authentication.getCredentials().toString();

**if**("tom".equals(userNAme)&&"cruise".equals(password)) {

**return** **new** UsernamePasswordAuthenticationToken(userNAme, password,Arrays.*asList*());

}

**throw** **new** BadCredentialsException("UserNAme password mismatch");

}

@Override

**public** **boolean** supports(Class<?> authentication) {

**return** authentication.equals(UsernamePasswordAuthenticationToken.**class**);

}

}

## Test and Configure

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

**public** **class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** PasswordEncoder passwordEnoder;

@Autowired

**private** MyAuthenticationProvider authenticationProvider;

// @Override

// protected void configure(AuthenticationManagerBuilder auth) throws Exception {

//// BCryptPasswordEncoder passwordEnoder = new BCryptPasswordEncoder();

// InMemoryUserDetailsManager userDetailService = new InMemoryUserDetailsManager();

// UserDetails user = User.withUsername("tom").password(passwordEnoder.encode("cruise")).authorities("read").build();

//

// userDetailService.createUser(user);

//

// //auth.userDetailsService(userDetailService).passwordEncoder(passwordEnoder);

// auth.userDetailsService(userDetailService); //If we define BCryptPasswordEncoder bean then authenticationmanagerBuilder gets that passwordEncoder bean

// }

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.authenticationProvider(authenticationProvider);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic();

http.authorizeRequests().anyRequest().authenticated();

}

}

## Use form based login

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin();

}

## Few more methods

@Override

protected void configure(HttpSecurity http) throws Exception {

//http.httpBasic();

http.formLogin();

//http.authorizeRequests().anyRequest().authenticated();

//http.authorizeRequests().antMatchers("/hello").authenticated().anyRequest().permitAll();

http.authorizeRequests().antMatchers("/hello").authenticated().anyRequest().denyAll();

}

}

## Create Custom Filter

We create our custom filter by implementing Filter from javaxservlet

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

public class MySecurityFilter implements Filter {

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

System.out.println("Before Filter");

chain.doFilter(request, response);

System.out.println("After Filter");

}

}

Now you need to apply filter , after basicAuthenticationFilter our Filter will run

@Configuration

**public** **class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** PasswordEncoder passwordEnoder;

@Autowired

**private** MyAuthenticationProvider authenticationProvider;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.authenticationProvider(authenticationProvider);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

//http.httpBasic();

http.httpBasic();

//http.authorizeRequests().anyRequest().authenticated();

//http.authorizeRequests().antMatchers("/hello").authenticated().anyRequest().permitAll();

http.authorizeRequests().antMatchers("/hello").authenticated();

http.addFilterBefore(**new** MySecurityFilter(), BasicAuthenticationFilter.**class**);

}

}

## Other Filter Classes

Extends OncePerRequest/GenericFilterBean

**Publicc** **class** MySecurityFilter **extends** OncePerRequestFilter {

……….

………….

}

## Other PasswordEncoders

@Test

**void** contextLoads() {

System.***out***.println(**new** BCryptPasswordEncoder().encode("password"));

System.***out***.println(**new** Pbkdf2PasswordEncoder().encode("password"));

}

## DelegatingPasswordEncoder

Dynamically pass key to delegate to encode password

Map<String, PasswordEncoder> encoders= **new** HashMap<>();

encoders.put("Bcrypt", **new** BCryptPasswordEncoder());

encoders.put("Scrypt", **new** SCryptPasswordEncoder());

encoders.put("pbkd", **new** Pbkdf2PasswordEncoder());

String encodedPassword = **new** DelegatingPasswordEncoder("Bcrypt", encoders).encode("password");

System.***out***.println(encodedPassword);

# Creating Microservices

In this section we are going to create product service and coupon service/ client create product from coupon service we apply coupons

## Create Database tables

use mydb;

create table product(

id int AUTO\_INCREMENT PRIMARY KEY,

name varchar(20),

description varchar(100),

price decimal(8,3)

);

create table coupon(

id int AUTO\_INCREMENT PRIMARY KEY,

code varchar(20) UNIQUE,

discount decimal(8,3),

exp\_date varchar(100)

);

COUPONSERVICE:

MODEL:

**package** com.bharath.springcloud.model;

**import** java.math.BigDecimal;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

@Entity

**public** **class** Coupon {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **long** id;

**private** String code;

**private** BigDecimal discount;

**private** String expDate;

**public** Coupon() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Coupon(**long** id, String code, BigDecimal discount, String expDate) {

**super**();

**this**.id = id;

**this**.code = code;

**this**.discount = discount;

**this**.expDate = expDate;

}

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getCode() {

**return** code;

}

**public** **void** setCode(String code) {

**this**.code = code;

}

**public** BigDecimal getDiscount() {

**return** discount;

}

**public** **void** setDiscount(BigDecimal discount) {

**this**.discount = discount;

}

**public** String getExpDate() {

**return** expDate;

}

**public** **void** setExpDate(String expDate) {

**this**.expDate = expDate;

}

@Override

**public** String toString() {

**return** "Coupon [id=" + id + ", code=" + code + ", discount=" + discount + ", expDate=" + expDate + "]";

}

}

REPO:

**package** com.bharath.springcloud.repo;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** com.bharath.springcloud.model.Coupon;

@Repository

**public** **interface** CouponRepo **extends** JpaRepository<Coupon, Long> {

Coupon findByCode(String code);

}

CONTROLLER:

**package** com.bharath.springcloud.controllers;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.bharath.springcloud.model.Coupon;

**import** com.bharath.springcloud.repo.CouponRepo;

@RestController

@RequestMapping("/couponapi")

**public** **class** CouponRestController {

@Autowired

**private** CouponRepo repo;

@PostMapping("/coupons")

**public** Coupon create(@RequestBody Coupon coupon) {

System.***out***.println(coupon);

**return** repo.save(coupon);

}

@GetMapping("/coupons/{code}")

**public** Coupon getCoupon(@PathVariable("code") String couponCode) {

System.***out***.println(couponCode);

**return** repo.findByCode(couponCode);

}

}

APPLICATION.PROPERTIES

spring.application.name=CouponService

spring.datasource.url=jdbc:mysql://localhost:3306/mydb

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto = update

#console

spring.jpa.show-sql = true

spring.jpa.properties.hibernate.format\_sql=true

PRODUCTSERVICE

Dto:

**package** com.bharath.springcloud.dto;

**import** java.math.BigDecimal;

**public** **class** Coupon {

**private** **long** id;

**private** String code;

**private** BigDecimal discount;

**private** String expDate;

**public** Coupon() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Coupon(**long** id, String code, BigDecimal discount, String expDate) {

**super**();

**this**.id = id;

**this**.code = code;

**this**.discount = discount;

**this**.expDate = expDate;

}

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getCode() {

**return** code;

}

**public** **void** setCode(String code) {

**this**.code = code;

}

**public** BigDecimal getDiscount() {

**return** discount;

}

**public** **void** setDiscount(BigDecimal discount) {

**this**.discount = discount;

}

**public** String getExpDate() {

**return** expDate;

}

**public** **void** setExpDate(String expDate) {

**this**.expDate = expDate;

}

@Override

**public** String toString() {

**return** "Coupon [id=" + id + ", code=" + code + ", discount=" + discount + ", expDate=" + expDate + "]";

}

}

MODEL:

**package** com.bharath.springcloud.model;

**import** java.math.BigDecimal;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Transient;

@Entity

**public** **class** Product {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **long** id;

**private** String name;

**private** String description;

**private** BigDecimal price;

@Transient

**private** String couponCode;

**public** Product() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Product(**long** id, String name, String description, BigDecimal price) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.description = description;

**this**.price = price;

}

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** BigDecimal getPrice() {

**return** price;

}

**public** **void** setPrice(BigDecimal price) {

**this**.price = price;

}

**public** **void** setCouponCode(String couponCode) {

**this**.couponCode = couponCode;

}

@Override

**public** String toString() {

**return** "Product [id=" + id + ", name=" + name + ", description=" + description + ", price=" + price + "]";

}

**public** String getCouponCode() {

// **TODO** Auto-generated method stub

**return** couponCode;

}

}

REPO:

package com.bharath.springcloud.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.bharath.springcloud.model.Product;

@Repository

public interface ProductRepo extends JpaRepository<Product, Long>{

}

APPLICATION.PROPERTIES

spring.application.name=ProductService

server.port=9090

spring.datasource.url=jdbc:mysql://localhost:3306/mydb

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto = update

#console

spring.jpa.show-sql = true

spring.jpa.properties.hibernate.format\_sql=true

couponService.url=http://localhost:8080/couponapi/coupons

CONTROLLER:

**package** com.bharath.springcloud.controllers;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** org.springframework.web.client.RestTemplate;

**import** com.bharath.springcloud.dto.Coupon;

**import** com.bharath.springcloud.model.Product;

**import** com.bharath.springcloud.repo.ProductRepo;

@RestController

@RequestMapping("productapi")

**public** **class** ProductController {

@Autowired

**private** ProductRepo repo;

@Autowired

**private** RestTemplate restTemplate;

@Value("${couponService.url}")

**private** String couponUrl;

@PostMapping("products")

**public** Product create(@RequestBody Product product) {

System.***out***.println("====================CREATE ===========================");

Coupon coupon = restTemplate.getForObject(couponUrl+product.getCouponCode(), Coupon.**class**);

**if**(coupon!=**null**)

{product.setPrice(product.getPrice().subtract(coupon.getDiscount()));}

System.***out***.println("=============================================================");

System.***out***.println(couponUrl);

System.***out***.println(coupon);

System.***out***.println(product);

System.***out***.println("=============================================================");

**return** repo.save(product);

//return product;

}

}

# Secure Rest API

## Create Tables

drop database mydb;

create database mydb;

use mydb;

create table product(

id int AUTO\_INCREMENT PRIMARY KEY,

name varchar(20),

description varchar(100),

price decimal(8,3)

);

drop table coupon;

create table coupon(

id int AUTO\_INCREMENT PRIMARY KEY,

code varchar(20) UNIQUE,

discount decimal(8,3),

exp\_date varchar(100)

);

CREATE TABLE USER

(

ID INT NOT NULL AUTO\_INCREMENT,

FIRST\_NAME VARCHAR(20),

LAST\_NAME VARCHAR(20),

EMAIL VARCHAR(20),

PASSWORD VARCHAR(256),

PRIMARY KEY (ID),

UNIQUE KEY (EMAIL)

);

CREATE TABLE ROLE

(

ID INT NOT NULL AUTO\_INCREMENT,

NAME VARCHAR(20),

PRIMARY KEY (ID)

);

CREATE TABLE USER\_ROLE(

USER\_ID int,

ROLE\_ID int,

FOREIGN KEY (user\_id)

REFERENCES user(id),

FOREIGN KEY (role\_id)

REFERENCES role(id)

);

insert into user(first\_name,last\_name,email,password) values ('doug','bailey','doug@bailey.com','$2a$10$U2STWqktwFbvPPsfblVeIuy11vQ1S/0LYLeXQf1ZL0cMXc9HuTEA2');

insert into user(first\_name,last\_name,email,password) values ('john','ferguson','john@ferguson.com','$2a$10$YzcbPL.fnzbWndjEcRkDmO1E4vOvyVYP5kLsJvtZnR1f8nlXjvq/G');

insert into role values(1,'ROLE\_ADMIN');

insert into role values(2,'ROLE\_USER');

insert into user\_role values(1,1);

insert into user\_role values(2,2);

show tables;

select \* from user;

select \* from role;

select \* from user\_role;

select \* from coupon;

select \* from product;

## Create Entities

**package** com.bharath.springcloud.model;

**import** java.util.Set;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.ManyToMany;

**import** org.springframework.security.core.GrantedAuthority;

@Entity

**public** **class** Role **implements** GrantedAuthority {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

@ManyToMany(mappedBy = "roles")

**private** Set<User> users;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String getAuthority() {

**return** name;

}

}

**package** com.bharath.springcloud.model;

**import** java.util.Set;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.JoinTable;

**import** javax.persistence.ManyToMany;

@Entity

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

@ManyToMany(fetch = FetchType.***EAGER***)

@JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_id"), inverseJoinColumns = @JoinColumn(name = "role\_id"))

**private** Set<Role> roles;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Set<Role> getRoles() {

**return** roles;

}

**public** **void** setRoles(Set<Role> roles) {

**this**.roles = roles;

}

}

**package** com.bharath.springcloud.repo;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** com.bharath.springcloud.model.Role;

**public** **interface** RoleRepo **extends** JpaRepository<Role, Long> {

}

**package** com.bharath.springcloud.repo;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** com.bharath.springcloud.model.User;

**public** **interface** UserRepo **extends** JpaRepository<User, Long> {

User findByEmail(String email);

}

## Implement UserDetailsService

Here we are going to create our own userDatailService which fetch user detail from database and retuen UserDetail as Springsecurity expected

package com.bharath.springcloud;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import com.bharath.springcloud.model.User;

import com.bharath.springcloud.repo.UserRepo;

public class UserDetailServiceImpl implements UserDetailsService {

@Autowired

private UserRepo repo;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = repo.findByEmail(username);

if(user==null) {

throw new UsernameNotFoundException("User with email: "+username+" not present");

}

org.springframework.security.core.userdetails.User userDetail = new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), user.getRoles());

return userDetail;

}

}

## Create WebSecurityConfig

We create MySpringSecurityConfig configuration to configure our own userdetail service

**package** com.bharath.springcloud.security.config;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** com.bharath.springcloud.security.UserDetailServiceImpl;

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** PasswordEncoder passwordEncoder;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

//

auth.userDetailsService(userDetailService);

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

}

## Secure the URL’s and methods

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** PasswordEncoder passwordEncoder;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(userDetailService);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic();

http.authorizeRequests().mvcMatchers(HttpMethod.***GET***, "/couponapi/coupons/{code:^[A-Z]\*$}").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.***POST***, "/couponapi/coupons").hasRole("ADMIN").and().csrf().disable();

}

# Secure WebApp

## Create Index

Index.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<a href=*""*>Get Coupon</a>

<a href=*""*>Create Coupon</a>

</body>

</html>

=========================================================

couponController

@Controller

**public** **class** CouponController {

@GetMapping("/")

**public** String index() {

**return** "index";

}

}

WEBSECURITYCONFIG

@Configuration

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** PasswordEncoder passwordEncoder;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(userDetailService);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.formLogin();

http.authorizeRequests().mvcMatchers(HttpMethod.***GET***, "/couponapi/coupons/{code:^[A-Z]\*$}","/","/index").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.***POST***, "/couponapi/coupons").hasRole("ADMIN").and().csrf().disable();

}

}

## Craete Coupon form

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form>

code: <input name=*"code"*>

discount: <input name=*"discount"*>

Expiry Date: <input name=*"expDate"*>

<input type=*"submit"*>

</form>

</body>

</html>

----------------------------------------------------------------------------------------------------------------------

**public** **class** CouponController {

@GetMapping("/")

**public** String index() {

**return** "index";

}

@GetMapping("/showCreateCoupon")

**public** String createCoupon() {

**return** "createCoupon";

}

}

----------------------------------------------------------------------------------------------------------------------

WEBSECURITYCONFIG.JAVA

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.formLogin();

http.authorizeRequests()

.mvcMatchers(HttpMethod.***GET***, "/couponapi/coupons/{code:^[A-Z]\*$}", "/", "/index", "/showCreateCoupon",

"/createCoupon")

.hasAnyRole("USER", "ADMIN").mvcMatchers(HttpMethod.***POST***, "/couponapi/coupons").hasRole("ADMIN").and()

.csrf().disable();

}

## Render Get Coupon

package com.bharath.springcloud.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.servlet.ModelAndView;

import com.bharath.springcloud.model.Coupon;

import com.bharath.springcloud.repo.CouponRepo;

@Controller

public class CouponController {

@Autowired

private CouponRepo repo;

@GetMapping("/")

public String index() {

return "index";

}

@GetMapping("/showCreateCoupon")

public String showCreateCoupon() {

return "createCoupon";

}

@PostMapping("/createCoupon")

public String createCoupon(Coupon coupon) {

repo.save(coupon);

return "couponCreated";

}

@GetMapping("/showGetCoupon")

public String showgetCoupon() {

return "getCoupon";

}

@PostMapping("/getCoupon")

public ModelAndView getCoupon(String couponCode) {

ModelAndView mv = new ModelAndView("couponDetails");

mv.addObject(repo.findByCode(couponCode));

return mv;

}

}

getCoupon.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form action=*"/getCoupon"* method=*"post"*>

code:<input name=*"couponCode"*>

<input type=*"submit"*>

</form>

</body>

</html>

Coupon Details.html

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h2>Coupon Detail</h2>

Code:<b th:text=*"${coupon.code}"*></b>

<br/>

Discount:<b th:text=*"${coupon.discount}"*></b><br/>

Expiry Date:<b th:text=*"${coupon.expDate}"*></b><br/>

</body>

</html>

package com.bharath.springcloud.security.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import com.bharath.springcloud.security.UserDetailServiceImpl;

@Configuration

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private PasswordEncoder passwordEncoder;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailService);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin();

http.authorizeRequests()

.mvcMatchers(HttpMethod.GET, "/couponapi/coupons/{code:^[A-Z]\*$}", "/", "/index",

"/showGetCoupon","/getCoupon","/couponDetails").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.POST,"/getCoupon").hasAnyRole("ADMIN","USER")

.mvcMatchers(HttpMethod.GET, "/showCreateCoupon","/createCoupon","/couponCreated").hasAnyRole("ADMIN")

.mvcMatchers(HttpMethod.POST, "/couponapi/coupons","/createCoupon").hasRole("ADMIN").and()

.csrf().disable();

}

}

## Implement Custom Login

Create Security Service

package com.bharath.springcloud.security;

public interface SecurityService {

boolean login(String username, String password);

}

SecutiyServiceImpl

@Service

public class SecurityServiceImpl implements SecurityService {

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private AuthenticationManager authenticationManager;

@Override

public boolean login(String username, String password) {

UserDetails userDetail = userDetailService.loadUserByUsername(username); //Gets userdetail from DB using our own userDetailsService

UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(userDetail,password,userDetail.getAuthorities()); //creating the token

authenticationManager.authenticate(token); //we pass token to Authentication manager to check credentials and authenticate token

boolean result = token.isAuthenticated();

if(result) {

SecurityContextHolder.getContext().setAuthentication(token); //If authenticated then we saving into security Context

}

return result;

}

}

UserDetailsSerciceImpl

@Service

public class UserDetailServiceImpl implements UserDetailsService {

@Autowired

private UserRepo repo;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

System.out.println("LOAD BY USER NAME: "+username);

User user = repo.findByEmail(username);

System.out.println("USER DAT: "+user.toString());

if(user==null) {

throw new UsernameNotFoundException("User with email: "+username+" not present");

}

org.springframework.security.core.userdetails.User userDetail = new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), user.getRoles());

return userDetail;

}

}

@Configuration

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private PasswordEncoder passwordEncoder;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailService);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin();

http.authorizeRequests()

.mvcMatchers(HttpMethod.GET, "/couponapi/coupons/{code:^[A-Z]\*$}", "/", "/index",

"/showGetCoupon","/getCoupon","/couponDetails").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.POST,"/getCoupon").hasAnyRole("ADMIN","USER")

.mvcMatchers(HttpMethod.GET, "/showCreateCoupon","/createCoupon","/couponCreated").hasAnyRole("ADMIN")

.mvcMatchers(HttpMethod.POST, "/couponapi/coupons","/createCoupon").hasRole("ADMIN").and()

.csrf().disable();

}

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

## Create the View and Controller

UserController

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import com.bharath.springcloud.security.SecurityService;

@Controller

public class UserController {

@Autowired

private SecurityService securityService;

@GetMapping("/")

public String index() {

return "login";

}

@PostMapping("/login")

public String loginAuthentication(String email,String password) {

boolean result = securityService.login(email, password);

if(result)

return "index";

return "login";

}

}

Login.html

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form action="login" method="post">

UserName:<input name="email" type="text"><br>

Password: <input type="password" name="password"><br>

<input type="submit">

</form>

</body>

</html>

SecurityServiceImpl

@Service

public class SecurityServiceImpl implements SecurityService {

@Autowired

private UserDetailServiceImpl userDetailsService;

@Autowired

private AuthenticationManager authenticationManger;

@Override

public boolean login(String username, String password) {

UserDetails userDetail = userDetailsService.loadUserByUsername(username); //Gets userdetail from DB using our own userDetailsService

UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(userDetail,password,userDetail.getAuthorities()); //creating the token

authenticationManger.authenticate(token); //we pass token to Authentication manager to check credentials and authenticate token

boolean result = token.isAuthenticated();

if(result) {

SecurityContextHolder.getContext().setAuthentication(token); //If authenticated then we saving into security Context

}

return result;

}

}

## lOGOUT

@GetMapping("/showReg")

**public** String showRegistrationPage() {

**return** "registerUser";

}

@PostMapping("/registerUser")

**public** String register(User user) {

user.setPassword(encoder.encode(user.getPassword()));

Role role = roleRepo.findById(2l).get();

user.addRole(role);

userRepo.save(user);

**return** "login";

}

http.authorizeRequests()

.mvcMatchers(HttpMethod.***GET***, "/couponapi/coupons/{code:^[A-Z]\*$}", "/index",

"/showGetCoupon","/getCoupon","/couponDetails").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.***POST***,"/getCoupon").hasAnyRole("ADMIN","USER")

.mvcMatchers(HttpMethod.***GET***, "/showCreateCoupon","/createCoupon","/couponCreated").hasAnyRole("ADMIN")

.mvcMatchers(HttpMethod.***POST***, "/couponapi/coupons","/createCoupon").hasRole("ADMIN")

.mvcMatchers("/", "/login", "/logout", "/showReg", "/registerUser").permitAll()

.and()

.csrf().disable()

.logout().logoutSuccessUrl("/");

}

# OAuth

## OAuth Intro

Pic========

## Grant types

Pic=====

## Google Oauth Palyground

# Oauth in Action

## Create WebSecurityConfig

package com.bharath.springcloud.security.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration

public class OAuth2SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Bean

public BCryptPasswordEncoder bCryptPasswordEncoder() {

return new BCryptPasswordEncoder();

}

}

## Create AuthorizationServer

package com.bharath.springcloud.security.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;

import org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;

import com.bharath.springcloud.security.UserDetailServiceImpl;

@Configuration

@EnableAuthorizationServer

public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

private static final String COUPONSERVICE = "couponservice";

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private BCryptPasswordEncoder encoder;

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory().withClient("couponclientapp").secret(encoder.encode("9999")) //registering client id and screet in inmemory token store

.authorizedGrantTypes("password", "refresh\_token").scopes("read", "write").resourceIds(COUPONSERVICE);

}

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.tokenStore(new InMemoryTokenStore()).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

.userDetailsService(userDetailService);

}

}

## Create Resource Server

package com.bharath.springcloud.security.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;

@Configuration

@EnableResourceServer

public class ResouceServerConfig extends ResourceServerConfigurerAdapter {

private static final String RESOURCE\_ID = "couponservice";

@Override

public void configure(ResourceServerSecurityConfigurer resources) throws Exception {

resources.resourceId(RESOURCE\_ID);

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().mvcMatchers(HttpMethod.GET, "/couponapi/coupons/{code:^[A-Z]\*$}")

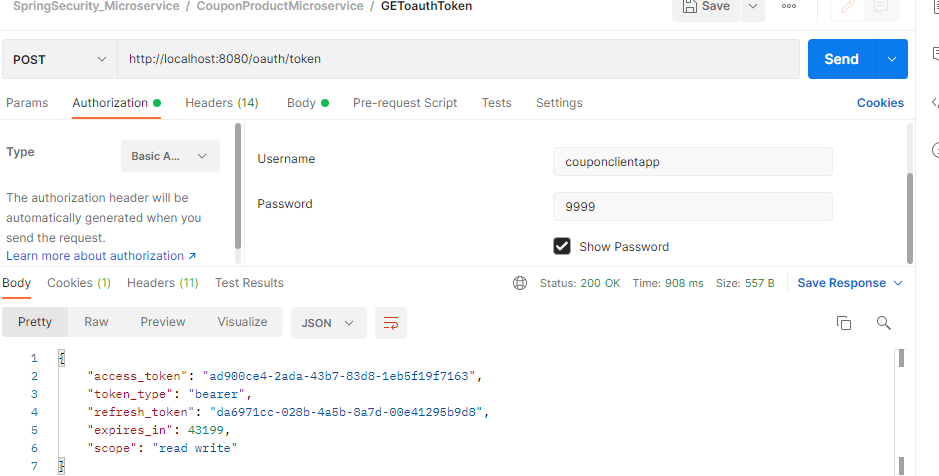
.hasAnyRole("USER", "ADMIN").mvcMatchers(HttpMethod.POST, "/couponapi/coupons").hasRole("ADMIN")

.anyRequest().denyAll().and().csrf().disable();

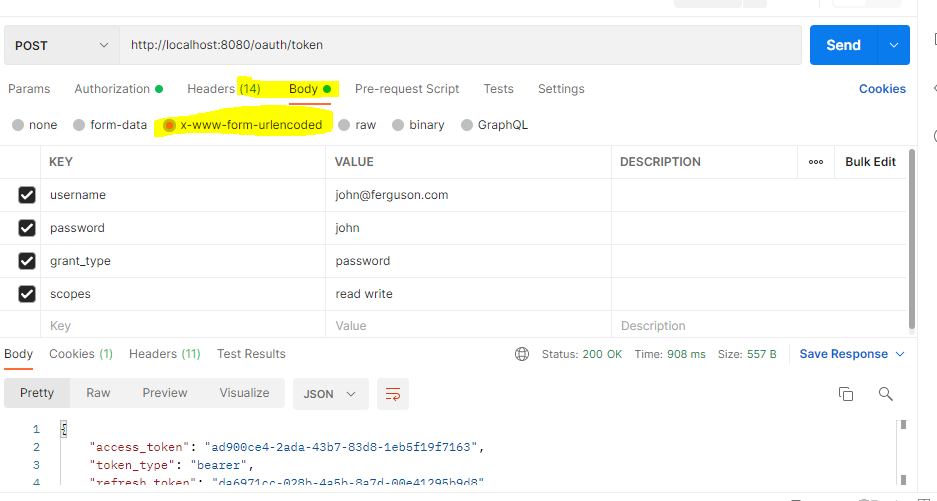
}

}

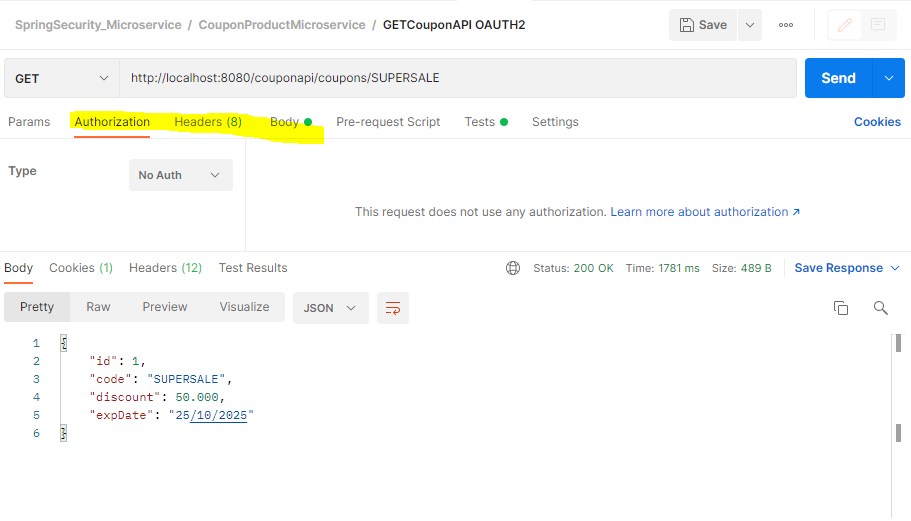
## TESTING

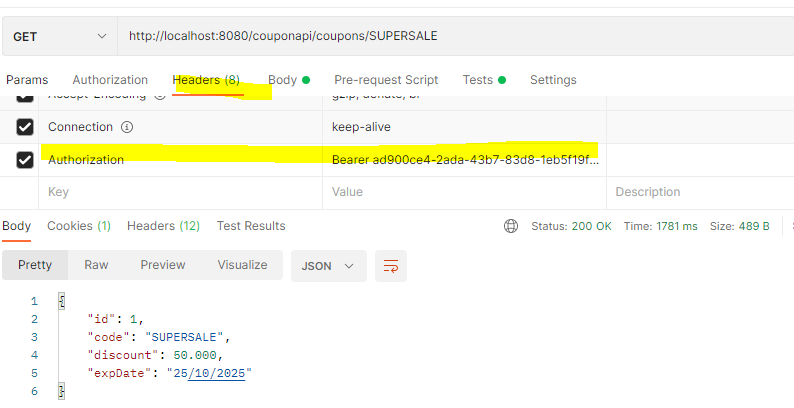


Node username and password should send in body



GET Coupon





## Use Db to store token

@Configuration

@EnableAuthorizationServer

public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

private static final String COUPONSERVICE = "couponservice";

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private BCryptPasswordEncoder encoder;

@Autowired

private DataSource dataSource;

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory().withClient("couponclientapp").secret(encoder.encode("9999")) //registering client id and screet in inmemory token store

.authorizedGrantTypes("password", "refresh\_token").scopes("read", "write").resourceIds(COUPONSERVICE);

}

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.tokenStore(new JdbcTokenStore(dataSource)).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

.userDetailsService(userDetailService);

}

}

use mydb;

create table oauth\_access\_token (

token\_id varchar(255) NOT NULL,

token blob,

authentication\_id varchar(255) DEFAULT NULL,

user\_name varchar(255) DEFAULT NULL,

client\_id varchar(255) DEFAULT NULL,

authentication blob,

refresh\_token varchar(255) DEFAULT NULL,

PRIMARY KEY (token\_id));

create table oauth\_refresh\_token (

token\_id varchar(255) NOT NULL,

token blob,

authentication blob,

PRIMARY KEY (token\_id));

select \* from oauth\_access\_token;

select \* from oauth\_refresh\_token;

# JWT

## The java keytool

Keytool is key and certificate management entity

We create keystore file which is password protedted in which we store private key with alias and password. When we access private we provide alias and password.

## Configure JWT

@Configuration

@EnableAuthorizationServer

public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

private static final String COUPONSERVICE = "couponservice";

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private BCryptPasswordEncoder encoder;

@Autowired

private DataSource dataSource;

@Value("${keyFile}")

private String keyFile;

@Value("${password}")

private char[] password;

@Value("${alias}")

private String alias;

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory().withClient("couponclientapp").secret(encoder.encode("9999")) //registering client id and screet in inmemory token store

.authorizedGrantTypes("password", "refresh\_token").scopes("read", "write").resourceIds(COUPONSERVICE);

}

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

// endpoints.tokenStore(new JdbcTokenStore(dataSource)).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

// .userDetailsService(userDetailService);

endpoints.tokenStore(tokenStore()).accessTokenConverter(accessTokenConverter()).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

.userDetailsService(userDetailService);

}

@Bean

public TokenStore tokenStore() {

return new JwtTokenStore(accessTokenConverter());

}

@Bean

public JwtAccessTokenConverter accessTokenConverter() {

JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter(); //use keys to generates signature

KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPathResource(keyFile), password); //read file from resources

KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias); //get key using alias name

jwtAccessTokenConverter.setKeyPair(keyPair); //set keypair

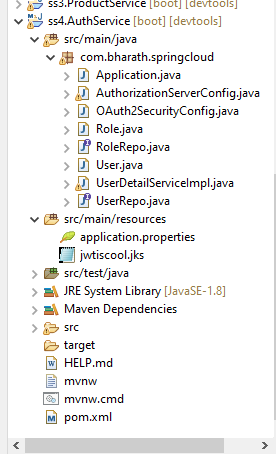
return jwtAccessTokenConverter;

}

}

## Create AuthorizationServer project

Create Auth project and copy following files from couponservice



## Configure ResourceServer to use public key

Remove AuthorizationService from CouponServiceProject.

In AuthServiceProject

**package** com.bharath.springcloud;

**import** java.security.KeyPair;

**import** javax.sql.DataSource;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

**import** org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;

**import** org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

**import** org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;

**import** org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;

**import** org.springframework.security.oauth2.provider.token.TokenStore;

**import** org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;

**import** org.springframework.security.oauth2.provider.token.store.JdbcTokenStore;

**import** org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;

**import** org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

**import** org.springframework.security.rsa.crypto.KeyStoreKeyFactory;

@Configuration

@EnableAuthorizationServer

**public** **class** AuthorizationServerConfig **extends** AuthorizationServerConfigurerAdapter {

**private** **static** **final** String ***COUPONSERVICE*** = "couponservice";

@Autowired

**private** AuthenticationManager authenticationManager;

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** BCryptPasswordEncoder encoder;

@Autowired

**private** DataSource dataSource;

@Value("${keyFile}")

**private** String keyFile;

@Value("${password}")

**private** **char**[] password;

@Value("${alias}")

**private** String alias;

@Override

**public** **void** configure(ClientDetailsServiceConfigurer clients) **throws** Exception {

clients.inMemory().withClient("couponclientapp").secret(encoder.encode("9999")) // registering client id and

// screet in inmemory token

// store

.authorizedGrantTypes("password", "refresh\_token").scopes("read", "write").resourceIds(***COUPONSERVICE***);

}

@Override

**public** **void** configure(AuthorizationServerEndpointsConfigurer endpoints) **throws** Exception {

// endpoints.tokenStore(new JdbcTokenStore(dataSource)).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

// .userDetailsService(userDetailService);

endpoints.tokenStore(tokenStore()).accessTokenConverter(accessTokenConverter())

.authenticationManager(authenticationManager) // we configuring our tokenstore,authentication amnager

// ans userdetail service

.userDetailsService(userDetailService);

}

@Override

**public** **void** configure(AuthorizationServerSecurityConfigurer security) **throws** Exception {

security.tokenKeyAccess("permitAll()");

}

@Bean

**public** TokenStore tokenStore() {

**return** **new** JwtTokenStore(accessTokenConverter());

}

@Bean

**public** JwtAccessTokenConverter accessTokenConverter() {

JwtAccessTokenConverter jwtAccessTokenConverter = **new** JwtAccessTokenConverter(); // use keys to generates

// signature

KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPathResource(keyFile), password); // read

// // file

// // from

// // resources

KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias); // get key using alias name

jwtAccessTokenConverter.setKeyPair(keyPair); // set keypair

**return** jwtAccessTokenConverter;

}

}

**In couponservice**

We use ResourceServiceConfig. We get public key to verify tojken

In application.properties

spring.application.name=CouponService

spring.datasource.url=jdbc:mysql://localhost:3306/mydb

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto = update

#console

spring.jpa.show-sql = true

spring.jpa.properties.hibernate.format\_sql=true

keyFile=jwtiscool.jks

alias=jwtiscool

password=jwtiscool

publicKey=ddd

package com.bharath.springcloud.security.config;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;

import org.springframework.security.oauth2.provider.token.TokenStore;

import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;

import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

@Configuration

@EnableResourceServer

public class ResouceServerConfig extends ResourceServerConfigurerAdapter {

private static final String RESOURCE\_ID = "couponservice";

@Value("${publicKey}")

private String publicKey;

@Override

public void configure(ResourceServerSecurityConfigurer resources) throws Exception {

resources.resourceId(RESOURCE\_ID).tokenStore(tokenStore());

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().mvcMatchers(HttpMethod.GET, "/couponapi/coupons/{code:^[A-Z]\*$}")

.hasAnyRole("USER", "ADMIN").mvcMatchers(HttpMethod.POST, "/couponapi/coupons").hasRole("ADMIN")

.and().csrf().disable();

}

@Bean

public TokenStore tokenStore() {

return new JwtTokenStore(accessTokenConverter());

}

@Bean

public JwtAccessTokenConverter accessTokenConverter() {

JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter(); //use keys to generates signature

//KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPathResource(keyFile), password); //read file from resources

//KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias); //get key using alias name

jwtAccessTokenConverter.setVerifierKey(publicKey); //get public key from application.properties

return jwtAccessTokenConverter;

}

}

## Use Public Key Dynamically

In AuthServiceProject

We give permission to get puplic key

@Configuration

@EnableAuthorizationServer

**public** **class** AuthorizationServerConfig **extends** AuthorizationServerConfigurerAdapter {

…….

@Override

**public** **void** configure(AuthorizationServerSecurityConfigurer security) **throws** Exception {

security.tokenKeyAccess("permitAll()");

}

}

**In CouponServiceProject**

**We remove** accessTokenConverter and tokenstore bean from ResourceServiceConfig

**application.properties**

security.oauth2.resource.jwt.key-uri=http://localhost:8081/oauth/token\_key

## Using Symmetric Keys

Symmetric key means we use same key in both authorization service and Auth Service to sign and verify Token

**In CouponService Project**

In resourceserverConfig

@Bean

**public** JwtAccessTokenConverter accessTokenConverter() {

JwtAccessTokenConverter jwtAccessTokenConverter = **new** JwtAccessTokenConverter(); //use keys to generates signature

//KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPathResource(keyFile), password); //read file from resources

//KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias); //get key using alias name

//jwtAccessTokenConverter.setVerifierKey(publicKey); //get public key from application.properties

jwtAccessTokenConverter.setSigningKey("testkey");

**return** jwtAccessTokenConverter;

}

In AuthServiceProject

@Configuration

@EnableAuthorizationServer

**public** **class** AuthorizationServerConfig **extends** AuthorizationServerConfigurerAdapter {

**private** **static** **final** String ***COUPONSERVICE*** = "couponservice";

@Autowired

**private** AuthenticationManager authenticationManager;

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** BCryptPasswordEncoder encoder;

@Autowired

**private** DataSource dataSource;

@Value("${keyFile}")

**private** String keyFile;

@Value("${password}")

**private** **char**[] password;

@Value("${alias}")

**private** String alias;

@Override

**public** **void** configure(ClientDetailsServiceConfigurer clients) **throws** Exception {

clients.inMemory().withClient("couponclientapp").secret(encoder.encode("9999")) // registering client id and

// screet in inmemory token

// store

.authorizedGrantTypes("password", "refresh\_token").scopes("read", "write").resourceIds(***COUPONSERVICE***);

}

@Override

**public** **void** configure(AuthorizationServerEndpointsConfigurer endpoints) **throws** Exception {

// endpoints.tokenStore(new JdbcTokenStore(dataSource)).authenticationManager(authenticationManager) //we configuring our tokenstore,authentication amnager ans userdetail service

// .userDetailsService(userDetailService);

endpoints.tokenStore(tokenStore()).accessTokenConverter(accessTokenConverter())

.authenticationManager(authenticationManager) // we configuring our tokenstore,authentication amnager

// ans userdetail service

.userDetailsService(userDetailService);

}

@Override

**public** **void** configure(AuthorizationServerSecurityConfigurer security) **throws** Exception {

security.tokenKeyAccess("permitAll()");

}

@Bean

**public** TokenStore tokenStore() {

**return** **new** JwtTokenStore(accessTokenConverter());

}

@Bean

**public** JwtAccessTokenConverter accessTokenConverter() {

JwtAccessTokenConverter jwtAccessTokenConverter = **new** JwtAccessTokenConverter(); // use keys to generates

// signature

// KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPathResource(keyFile), password); // read

// // file

// // from

// // resources

// KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias); // get key using alias name

// jwtAccessTokenConverter.setKeyPair(keyPair); // set keypair

jwtAccessTokenConverter.setSigningKey("testkey");

**return** jwtAccessTokenConverter;

}

# CSRF

## CSRF Intro

In previous we disabled CSRF, CSRF is like it add \_csrf in post request so that when we submit details server find\_xsrf and validate if \_csrf is wrong then post req fails

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form action=*"#"* th:action=*"@{/login}"* method=*"post"*>

UserName:<input name=*"email"* type=*"text"*><br>

Password: <input type=*"password"* name=*"password"*><br>

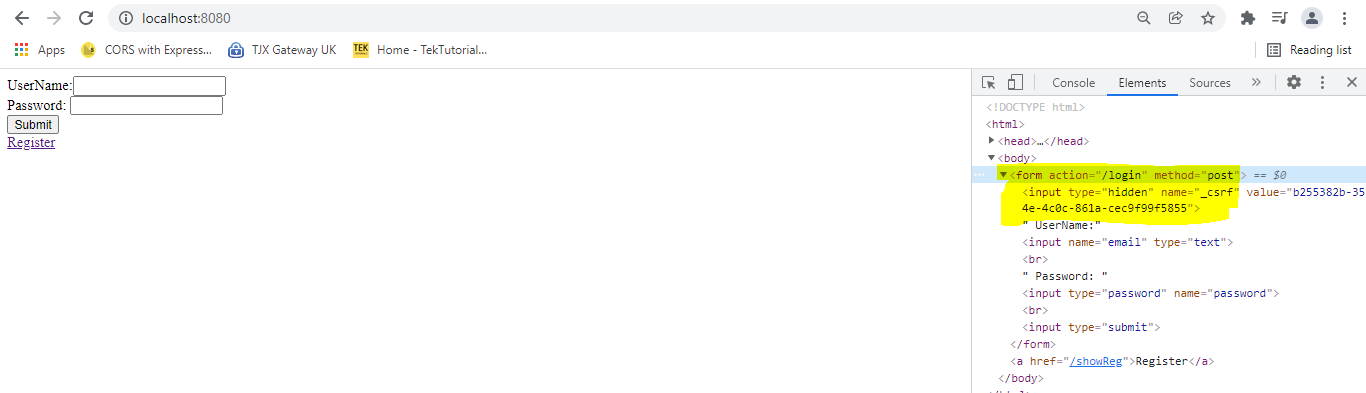
<input type=*"submit"*>

</form>

<a href=*"/showReg"*>Register</a>

</body>

</html>



## Use Custom Configuration

In Couponservice

@Configuration

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** UserDetailServiceImpl userDetailService;

@Autowired

**private** PasswordEncoder passwordEncoder;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(userDetailService);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.mvcMatchers(HttpMethod.***GET***, "/couponapi/coupons/{code:^[A-Z]\*$}", "/index",

"/showGetCoupon","/getCoupon","/couponDetails").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.***POST***,"/getCoupon").hasAnyRole("ADMIN","USER")

.mvcMatchers(HttpMethod.***GET***, "/showCreateCoupon","/createCoupon","/couponCreated").hasAnyRole("ADMIN")

.mvcMatchers(HttpMethod.***POST***, "/couponapi/coupons","/createCoupon").hasRole("ADMIN")

.mvcMatchers("/", "/login", "/logout", "/showReg", "/registerUser").permitAll()

.and()

//.csrf().disable()

.logout().logoutSuccessUrl("/");

http.csrf(csrfCustomiser -> {

csrfCustomiser.ignoringAntMatchers("/couponapi/coupons/\*\*");

//RequestMatcher requestMatcher = new RegexRequestMatcher("/couponapi/coupons/{code:^[A-Z]\*$}", "POST");

RequestMatcher requestMatcher = **new** MvcRequestMatcher(**new** HandlerMappingIntrospector(), "/getCoupon");

csrfCustomiser.ignoringRequestMatchers(requestMatcher);

});

}

# CORS

PIC

## CORS

@crossOrgin allows CORS policy

@RestController

@RequestMapping("/couponapi")

@CrossOrigin

**public** **class** CouponRestController {

**REACT FRONTEND**

import React from 'react';

import axios from "axios";

import {useEffect} from "react";

function App() {

useEffect(()=>{

axios.get('http://localhost:9091/couponapi/coupons/SUPERSALE').then((response) => {

document.write("Coupon Code: "+response.data.code+"<br/>")

document.write("Coupon Discount: "+response.data.discount+"<br/>")

document.write("Coupon Expiry Date: "+response.data.expDate+"<br/>")

});

},[])

return (

<div className="App">

</div>

);

}

export default App;

In webSecurityConfig

@Configuration

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailServiceImpl userDetailService;

@Autowired

private PasswordEncoder passwordEncoder;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailService);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.mvcMatchers(HttpMethod.GET, "/couponapi/coupons/{code:^[A-Z]\*$}", "/index",

"/showGetCoupon","/getCoupon","/couponDetails").hasAnyRole("USER", "ADMIN")

.mvcMatchers(HttpMethod.POST,"/getCoupon").hasAnyRole("ADMIN","USER")

.mvcMatchers(HttpMethod.GET, "/showCreateCoupon","/createCoupon","/couponCreated").hasAnyRole("ADMIN")

.mvcMatchers(HttpMethod.POST, "/couponapi/coupons","/createCoupon").hasRole("ADMIN")

.mvcMatchers("/", "/login", "/logout", "/showReg", "/registerUser").permitAll()

.and()

//.csrf().disable()

.logout().logoutSuccessUrl("/");

// http.csrf(csrfCustomiser -> {

// csrfCustomiser.ignoringAntMatchers("/couponapi/coupons/\*\*");

// //RequestMatcher requestMatcher = new RegexRequestMatcher("/couponapi/coupons/{code:^[A-Z]\*$}", "POST");

// RequestMatcher requestMatcher = new MvcRequestMatcher(new HandlerMappingIntrospector(), "/getCoupon");

// csrfCustomiser.ignoringRequestMatchers(requestMatcher);

// });

http.cors(corsCustomiser -> {

CorsConfigurationSource configurationSource= request ->{ //CorsConfigurationSource- search google its function Interface

CorsConfiguration corsConfiguration=new CorsConfiguration();

corsConfiguration.setAllowedOrigins(List.of("localhost:3000")); //Allowing list of orgins

corsConfiguration.setAllowedMethods(List.of("GET")); //Itt allows onlt Http Get Request

return corsConfiguration;

};

corsCustomiser.configurationSource(configurationSource);

});

}

# Method Level Security

## @PreAuthorize and @ postAuthorize

@Controller

**public** **class** CouponController {

@Autowired

**private** CouponRepo repo;

// @GetMapping("/")

// public String index() {

// return "index";

// }

@GetMapping("/showCreateCoupon")

@PreAuthorize("hasRole('ADMIN')")

**public** String showCreateCoupon() {

**return** "createCoupon";

}

@PostMapping("/createCoupon")

**public** String createCoupon(Coupon coupon) {

repo.save(coupon);

**return** "couponCreated";

}

@GetMapping("/showGetCoupon")

**public** String showgetCoupon() {

**return** "getCoupon";

}

@PostMapping("/getCoupon")

@PostAuthorize("returnObject.discount<60")

**public** ModelAndView getCoupon(String couponCode) {

ModelAndView mv = **new** ModelAndView("couponDetails");

mv.addObject(repo.findByCode(couponCode));

**return** mv;

}

## Other Anotation

@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = **true**,jsr250Enabled = **true**,securedEnabled = **true**)

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

}

------------------------------------------------------------------------------------------------------------

@GetMapping("/showCreateCoupon")

@PreAuthorize("hasRole('ADMIN')")

@Secured("ADMIN")

@RolesAllowed("ADMIN")

**public** String showCreateCoupon() {

**return** "createCoupon";

}