As we created the Application till now but we it’s not secure so I’ll add spring security so it will work like shield for application , it means it will secure the application and then we can customize the security on our need basis server

Why Spring Security

App server

spring security

User

who you are, what you want

# It is application security framework

->Login and Logout functionality

->Allow/block access to URLs to logged in users

-> Allow/block access to URLs to logged in users and with certain roles

->flexible and customizable

-> handle vulnerabilities

->widely adopted so we can get lot of resources and support from internet and quick response

## What Spring Security can do

->User name/password authentication

->App level authorization

-> intra app authorization like oAuth

->Microservice security(Using tokens,JWT)

->Method level security

5 core concepts in spring security

### Authentication, Authorization, Principal, Granted Authority, Roles

now let’s understand each points one by one

App server

spring security

User

who you are, what you want

Here when spring security asks you question WHO YOU ARE and you provide the identity who you are then it’s authentication

when you go to any site like facebook and asks you this question and you provide username and password or any other information and you give then it’s called knowledge based authentication

## Knowledge based authentication

->password

->Pin Code

->Answer to a secret/personal question

->easy to implement and use

However it’s comes with disadvantages also

suppose someone steal your password or find you password then they can login using your account that’s the problem

other type of authentication like

possession based authentication

->Phone/text messages

->key cards and badges

->Access Token devices

## Multifactor authentication

it is combination of knowledge based+possession based authentication

->enter your password then verify your text messages

#### Authorization

### Can this user do what they trying to do-> Yes/No

->For authorization user first should be authenticated

#### Principal

->it means currently logged in user or account

->one user can have multiple accounts so principal will reflect to currently logged in account

#### How authorization happens

Here is a concepts comes granted authority so we can configure the authority for like manager, user, management department what they can do and spring security will take cares of the rest

## Roles

->Group of authorities

suppose if we have manager and management department have same access so instead of writing same permission for both we can give the role and assign the permission we can set the role Like ADMIN and USER

->Spring security will do all this automatically by adding the dependency of spring security

now we will add spring security dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Now automatically our application got secured now when we will access any url let’s say /users the it will automatically redirects us to the login page and opens the login form and default username will be user and password will be auto-generated and now we can customize our application security on our need basis

Now understand how spring security do all this

->the answer is something called Filters , it is basis servlet technology

A filter is an object used to intercept the HTTP requests and responses of your application. By using filter, we can perform two operations at two instances −

* Before sending the request to the controller
* Before sending a response to the client.

Server

Filter

Filter will authenticate the user, if it is verified then request will go forward otherwise request will be returned

#### Spring Security default behaviour

->Adds mandatory authentication for all the URLs

-> Adds login form

->Handles login error

->creates a user and set a default password

->by default username is user and password is auto-generated , spring security generates a new password each time you start the app

now if want use our custom username and password for example username scooby and password 123 then we go inside application.properties file and mention

spring. security.user.name=Scooby

spring.security.user.password=123

#### How to configure authentication in Spring Security

when we add spring security dependency then it creates default user and put your application behind form based authentication, we can obviously configure what will be the default user in the application.properties file but it’s not ideal because we want to have spring security authentication based on the multiple users present in the database or in some external states

now the way to configure authentication in spring security is by affecting what’s called the AuthenticationManager

-> AuthenticationManager manages the authentication in spring security application

AuthenticationManager has method called authenticate() that either return successful authentication or throws the exception when authentication fails

now how to affect the AuthenticationManager , the way to affect is not creating your own AuthenticationManager but instead it to configure what AuthenticationManager does by using Builder pattern, we don’t work with AuthenticationManager directly for most part but instead we can work with Builder class AuthenticationManagerBuilder , we will use AuthenticationManagerBuilder to configure what the authentication should actually do

Steps

1.Get hold of AuthenticationManagerBuilder

2.Set the configuration on it

so when we are dealing with AuthenticationManagerBuilder then first thing it will ask what type of authentication do you want and suppose we will say In memory authentication then AuthenticationManager will ask for username , password and roles of your In memory users are and then we ill give the information may be it could be one user or multiple user , now when we will configure the AuthenticationManagerBuilder with this properties then you can imagine and new AuthenticationManager is being created somehow which has the value that you want so we are not dealing with AuthenticationManager directly we are dealing with the AuthenticationManagerBuilder

How do I get hold of AuthenticationManagerBuilder

by using hooks that is already present in the spring security there is a class that has method called configure and takes AuthenticationManagerBuilder as the argument configure(AuthenticationManagerBuilder )

class (WebSecurityConfigurerAdapter)

override this method

configure(AuthenticationManagerBuilder)

extend this class

The reason for being this class so that it gives the developer an opportunity to extend the class override the configure method and do the configuration that you want , if you don’t extend this class and override the method then default configuration happens however and if we extend the class the spring security will do our configuration that we want

now we go to application.properties file and delete the username and password that we have mentioned

now suppose we create SecurityConfig configuration class inside config package and it will extends WebSecurityConfigurerAdapter class and it’ll have configure(AuthenticationManagerBuilder) method that we will override and because this is configuration class so we have to mention @Configuration on top of the class and other annotation for @EnableWebSecurity

@EnableWebSecurity this tells to the spring security that this is the web security configuration

and now inside configure method we will set our configuration on the auth object , first we will tell what’s the authentication we need and based on that we provide the input

so suppose I make inMemoryAuthentication so will write

auth. inMemoryAuthentication()

then we will pass the username, password and the role of the user that we need

now suppose we are configuring for one user with username Rohan, password as 123 and roles as USER

auth.inMemoryAuthentication().withUser("Rohan").password("123").roles("USER");

@Override  
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.inMemoryAuthentication().withUser("Rohan").password("123").roles("USER");

if you want add multiple user then

@Override  
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.inMemoryAuthentication().withUser("Sohan").password("123").roles("ADMIN")  
 .and().withUser("Nichol").password("1234").roles("ADMIN");

// or second wsy

auth.inMemoryAuthentication().withUser("Sohan").password("123").roles("ADMIN")  
 auth.inMemoryAuthentication().withUser("Rohan").password("1234").roles("NORMAL");  
}

now the last step is to do is when we are dealing with the username and password then we don’t want our password to be stored as plain text anywhere in our application so we have to encode our password before storing

so how to set a password encoder

Just expose an @Bean of type PasswordEncoder

@Bean  
public PasswordEncoder passwordEncoder(){  
 //return NoOpPasswordEncoder.getInstance();  
 return new BCryptPasswordEncoder(4);  
}

here if we use NoOpPasswordEncoder it means we are just keeping the simple plain text password , no encoding but if we don’t create this method then our application will throw an exception

now we will understand

#### How to configure Spring Security Authorization

Till now we have added the dependency and in-memory authentication configured

what you need now

Different APIs having different access requirements

|  |  |
| --- | --- |
| API | Roles allows to access it |
| / | All(unauthenticated) |
| /user | USER and ADMIN roles |
| /admin | ADMIN roles |

now for this authorization we will override configure(HttpSecurity http) just like we did for authentication is configure(AuthenticationManagerBuilder auth) and we will write our configuration

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http.authorizeHttpRequests()  
 .antMatchers("/admin").hasRole("ADMIN")  
 .antMatchers("/user").hasAnyRole("ADMIN","USER")  
 .antMatchers("/").permitAll()  
 .and()  
 .formLogin();  
}

here in antMatchers we can pass our url pattern and we are using form based authentication by mentioning formLogin()

#### How Spring Security Authentication works

interface

Authentication Provider

Authentication Manager

authenticate(Authentication auth)

(Token based)

Authentication Provider

(auth based)

Provider Manager

true

UserDetailsService

loadUseByUsername()

Authentication Provider

Filter

valid authenticated object

Authentication Provider

create Authentication object

(put authentication details)

set the authentication to securityContext

Spring Boot + Spring Security with JPA authentication and MySQL

UserDetailsService

loadUserByIsername()

Authentication Provider

authenticate()

supports()

Authentication Manager

authenticate()

Authentication Provider

authenticate()

supports()

JPA service

DB

Authentication Provider

authenticate()

supports()

we will autowired the UserDetailsService in configure class and pass inside configure(AuthenticationManagerBuilder auth)

@Autowired  
private UserDetailsService userDetailsService;  
@Override  
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService);  
}

now we will create the implementation of UserDetailsService, let’s say MyUserDetailsService and implements UserDetailsService and override below method

@Override  
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 return null;  
}

spring security will call this service every time when somebody is trying to authenticate , it will pass the username and it gonna trust whatever the service is returning(i.e UserDetails ) and will authenticate

now we will create the implementation of this UserDetails interface , let’s say MyUserDetails and override all the methods with hardcoded value except username

public class MyUserDetails implements UserDetails {  
 private String username;  
 public MyUserDetails(String username){  
 this.username=username;  
 }  
 public MyUserDetails(){  
   
 }  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return Arrays.*asList*(new SimpleGrantedAuthority("ROLE\_USER"));  
 }  
  
 @Override  
 public String getPassword() {  
 return "pass";  
 }  
  
 @Override  
 public String getUsername() {  
 return username;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

Now when we will run our application we can pass username whatever we want and password as pass, till now we have not used Jpa and MySQL

now we will setup jpa and create entity class and UserRepository interface and define method findByUsername(String username) inside UserRepository

and now in MyUserDetailsService class we will override UserRepository and call findByUsername method

@Service  
public class MyUserDetailsService implements UserDetailsService {  
  
 @Autowired  
 private UserRepository repository;  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 Optional<User> user=repository.findByUsername(username);  
if(!user.isPresent()){  
 throw new UsernameNotFoundException("user not found");  
}  
return new MyUserDetails(user.get());

}  
}

now in MyUserDetails class we will remove hardcoded value and get the value from database using jpa

public class MyUserDetails implements UserDetails {  
 private String username;  
 private String password;  
 private boolean active;  
 private List<GrantedAuthority> authorities;  
  
  
 public MyUserDetails(User user){  
 this.username=user.getUsername();  
 this.password=user.getPassword();  
 this.active=user.isActive();  
 this.authorities=Arrays.*stream*(user.getRoles().split(","))  
 .map(SimpleGrantedAuthority::new)  
 .collect(Collectors.*toList*());  
 }  
 public MyUserDetails(){  
  
 }  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return authorities;  
 }  
  
 @Override  
 public String getPassword() {  
 return password;  
 }  
  
 @Override  
 public String getUsername() {  
 return username;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return active;  
 }  
}

After this run the application and that’s it!

##### Spring Boot + Spring Security + JWT

objectives

* Create a new authentication API
* Examine every incoming request for valid JWT and authorize