KLFSPRO Learn & Grow

# Spring Boot + spring Security

## Spring Securiry

- Spring security is application framework to make your application secures.
- It provides
- 1. Login and logout functionality
- 2. Allow/block access to urls to logged in users
- 3. Allow/block access to urls to logged in users and with certain role.

- Some of the vulnerabilitites are taken care automatically by spring security framework
- Example:
- Session fixassion
- Cross site request forgery

## What all we can do using spring security

**1. Authentication**

**For security in spring you may check username and password**

**2. Authorization**

**3. Intra App autherization using oauth**

**Eg: Autherization by using google or facebook account**

**For authentication**

**4. Using JWT token**

**5. Method level security is also possible**

# Core concept of spring security

**1. Authentication**

**When you go out, you need to prove that its you by providing id card**

**2. Autherization**

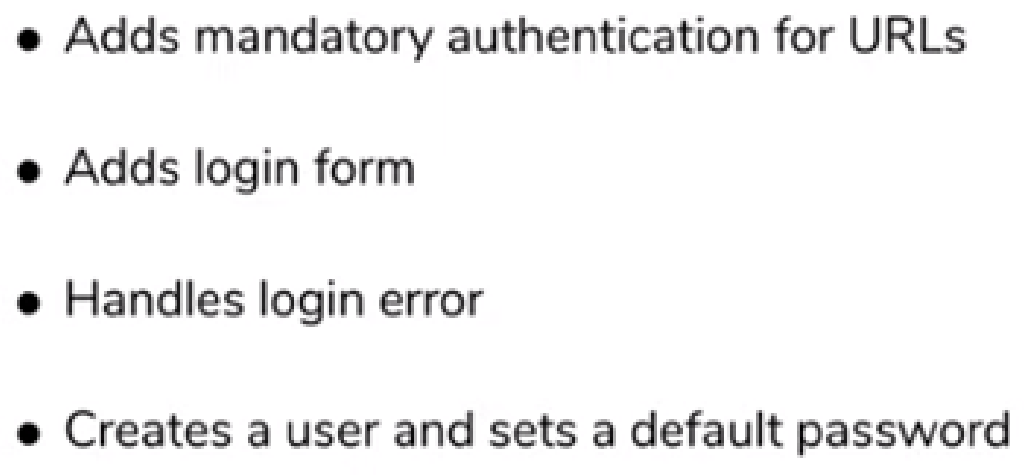
**3. Principal**

**a. It’s a person who logged in to the system**

**4. Roles**

**5. Granted authority**

## Steps for adding spring security in spring boot application



- Adds mandatory authentication for URLs
- Adds login form
- Handles login error
- Creates a user and sets a default password

# Entries in POM.xml

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
        <relativePath />
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>
```

# Entries in POM.xml

**Note : Security dependency in POM.xml will add basic security to your application**

```
<dependencies>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
       </dependency>
       <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-security -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
   <version>2.1.3.RELEASE</version>
</dependency>

       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
       </dependency>
    </dependencies>
```

# Application class

```
package com.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootStep1 {



            public static void main(String[] args) {
               SpringApplication.run(SpringBootStep1.class, args);
            }



}
```

```
package com.demo;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;
@RestController
public class TestGreet {


        @GetMapping(path="/greet")
        public String getEmployees()
        {
            return "<h1>Hello World</h1>";
        }
}
```

## Default security

- To find default username and password.

- By default username is user

- In console window you will be able to see password---------------- copy and paste it

- To modify default username and password

- Add following in application.properties

```
Spring.security.user.name=myuser
Spring.security.user.password=mypass
```

## Configure spring security

- Spring has AuthenticationManager class -→ that has authenticate method
- This method returns whether successful authentication or throws exception if authentication fails

- To modify default configuration you cannot work with AuthenticationManager class directly
- Instead you have to use AuthenticationManagerBuilder class which is factory pattern for AutheticationManager

- **In AuthenticationManagerBuilder you need to specify type of Authentication**

**1. In Memory Authentication**

   a. Then set username, Password and role

   b. It may be one user or multiple user

**2. To do that we need a class which has configure method**

**3. And configure method -accepts AutheticationManagerBuilder as an argument.**

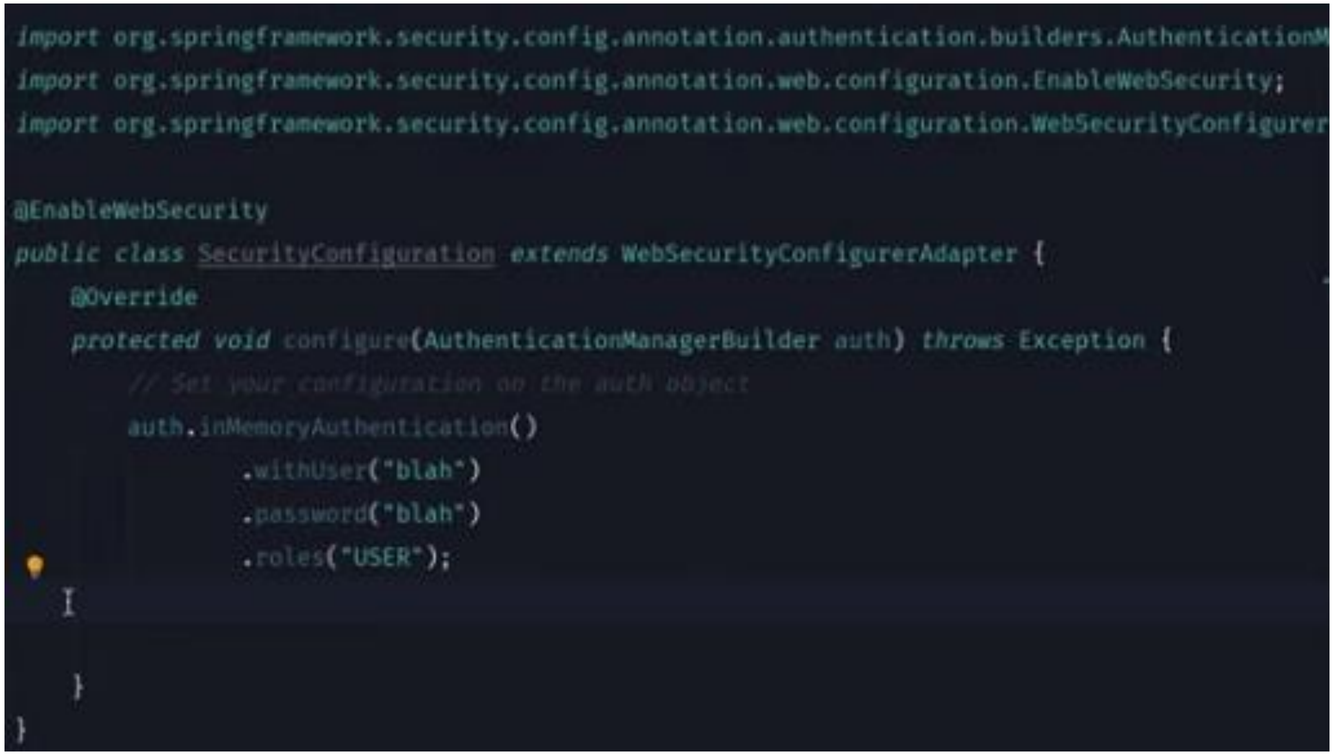
   a. Remove values from application.properties file

   b. Add a class MySpringSecurity extends WebSecurityConfigurerAdaptor

   c. Add configure method and annotate class with EnableSecurity as follows.

   d. This annotation tells, it is web Application security configuration

**4. Password cannot be kept as simple string it need to be in encoded form. So we need to add a Bean called passwordEncoder using @Bean.**

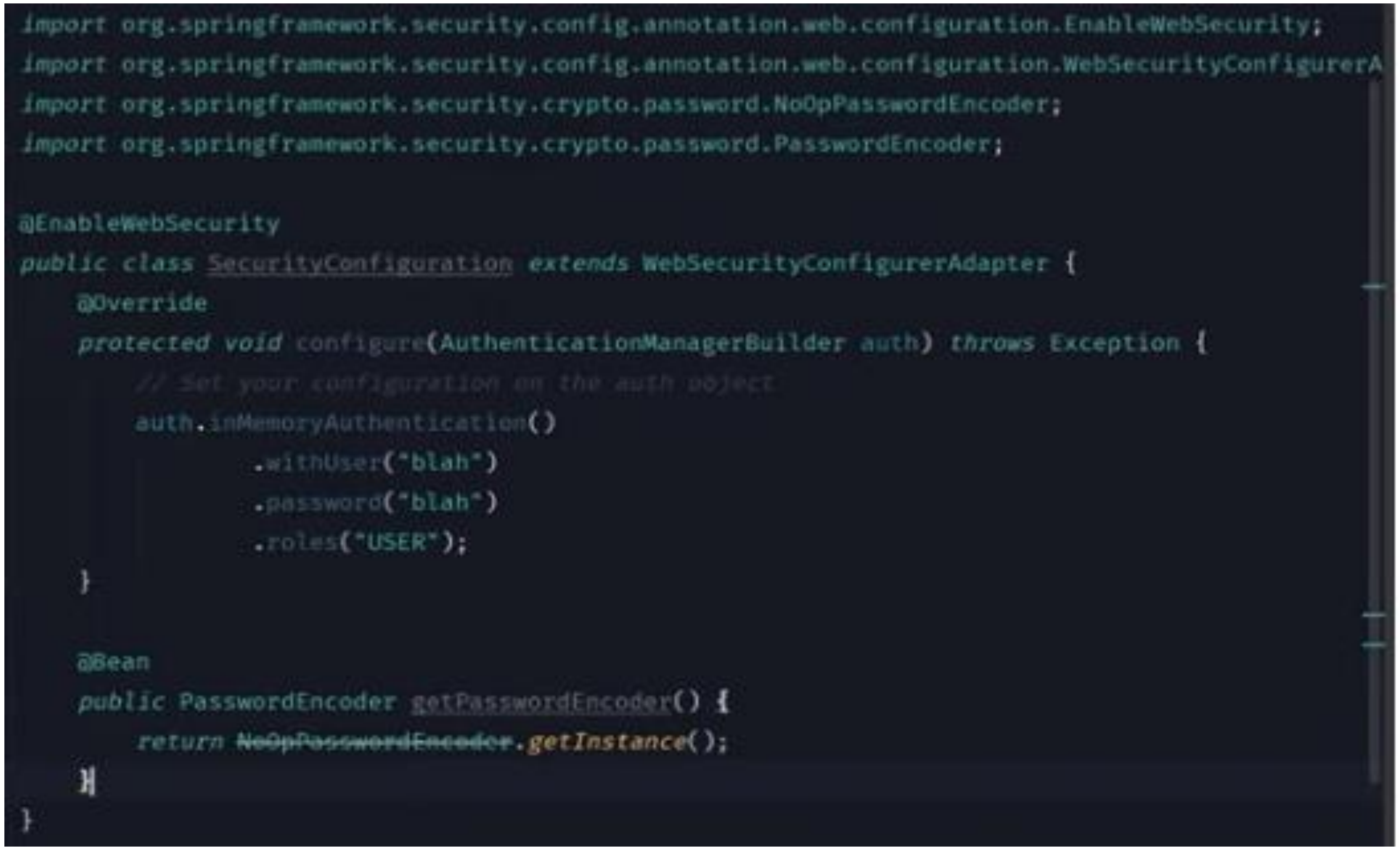


```
import org.springframework.security.config.annotation.authentication.builders.AuthenticationM
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer

@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // Set your configuration on the auth object
        auth.inMemoryAuthentication()
                .withUser("blah")
                .password("blah")
                .roles("USER");


    }
}
```

- **NoOpPasswordEncoder does nothing, it keeps the string as it is. But you may add some other encoder.**

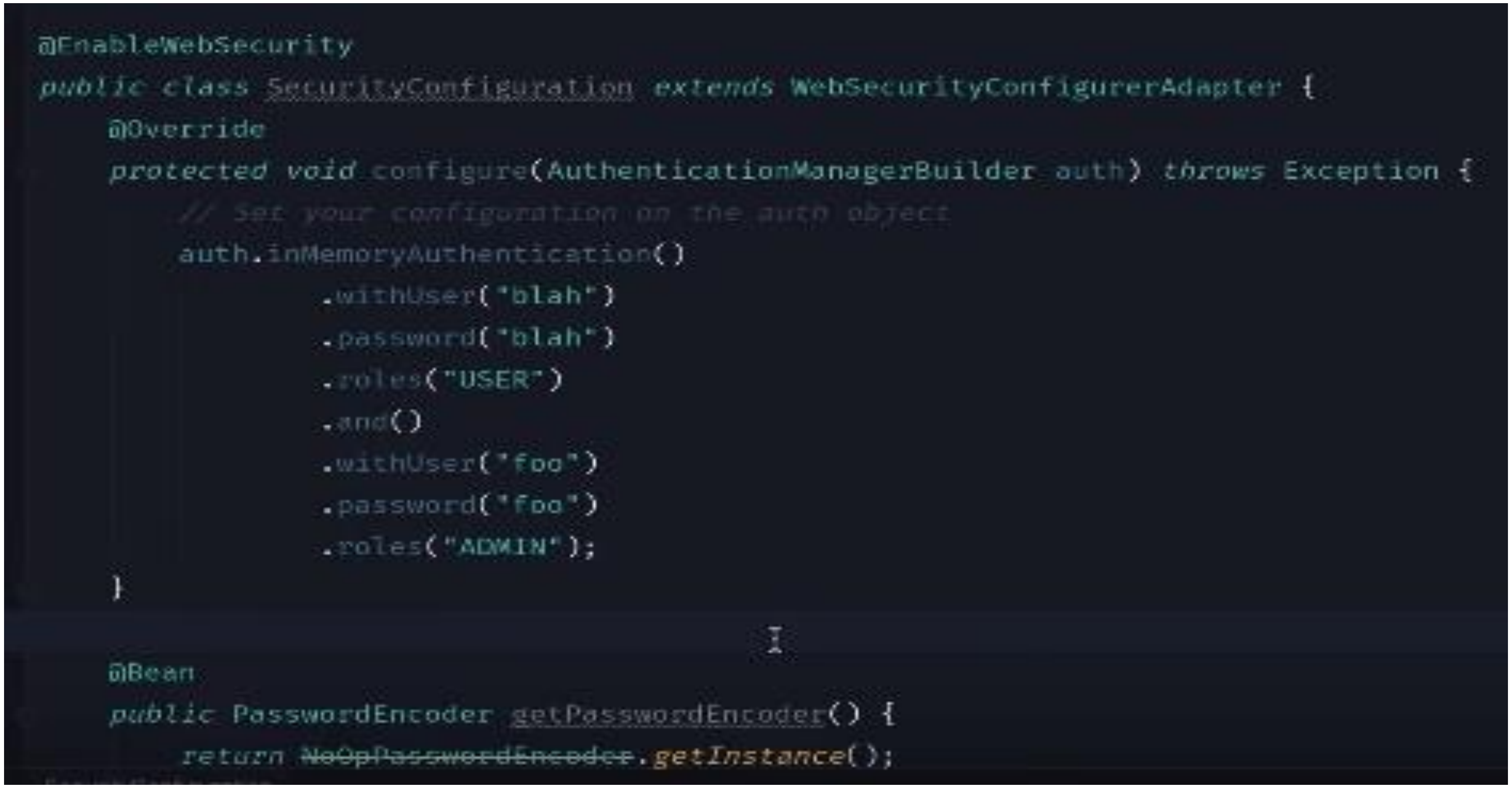


```
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerA
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // Set your configuration on the auth object
        auth.inMemoryAuthentication()
                .withUser("blah")
                .password("blah")
                .roles("USER");
    }

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return ~~NoOpPasswordEncoder~~.getInstance();
    }
}
```

- **To add more users ------→ shown below we are using and() method which allows method chaining**



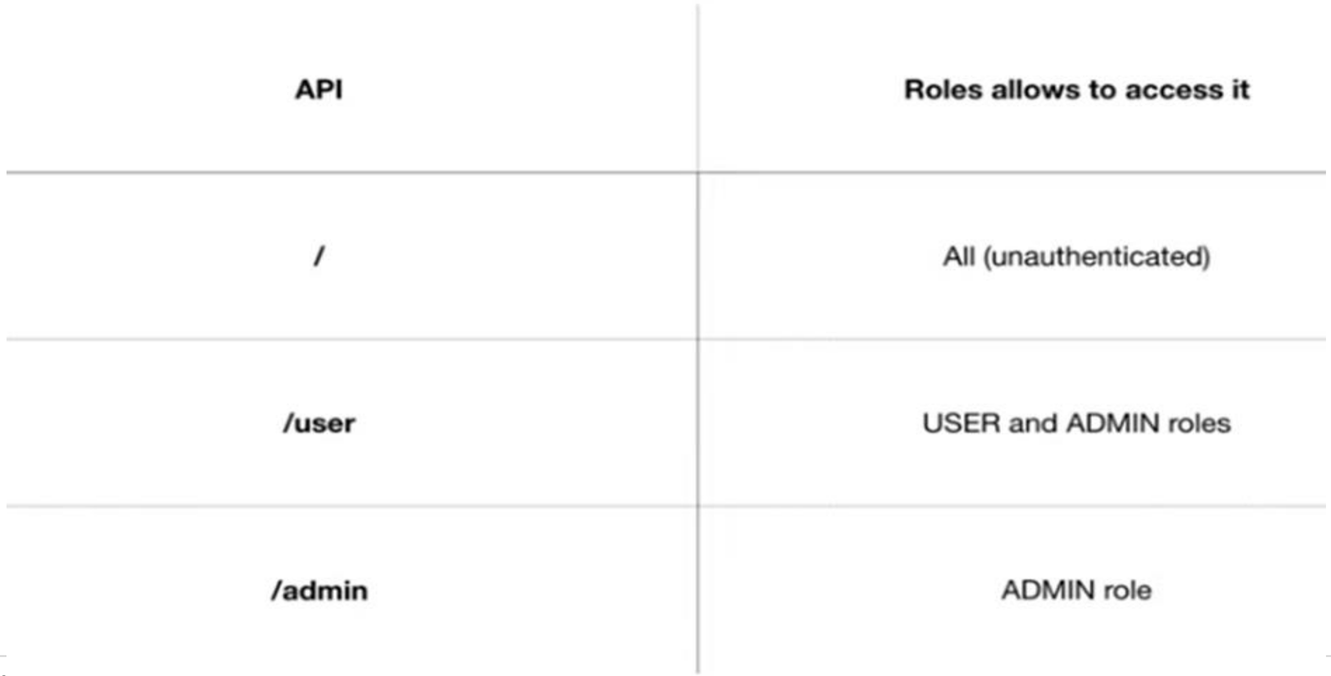
```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // Set your configuration on the auth object
        auth.inMemoryAuthentication()
                .withUser("blah")
                .password("blah")
                .roles("USER")
                .and()
                .withUser("foo")
                .password("foo")
                .roles("ADMIN");
    }

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return ~~NoOpPasswordEncoder~~.getInstance();
```

- Other type of security configuration is application/method level configuration

## Autherization

- Add 3 API and assign different roles access permission
- We will add different API with different Role



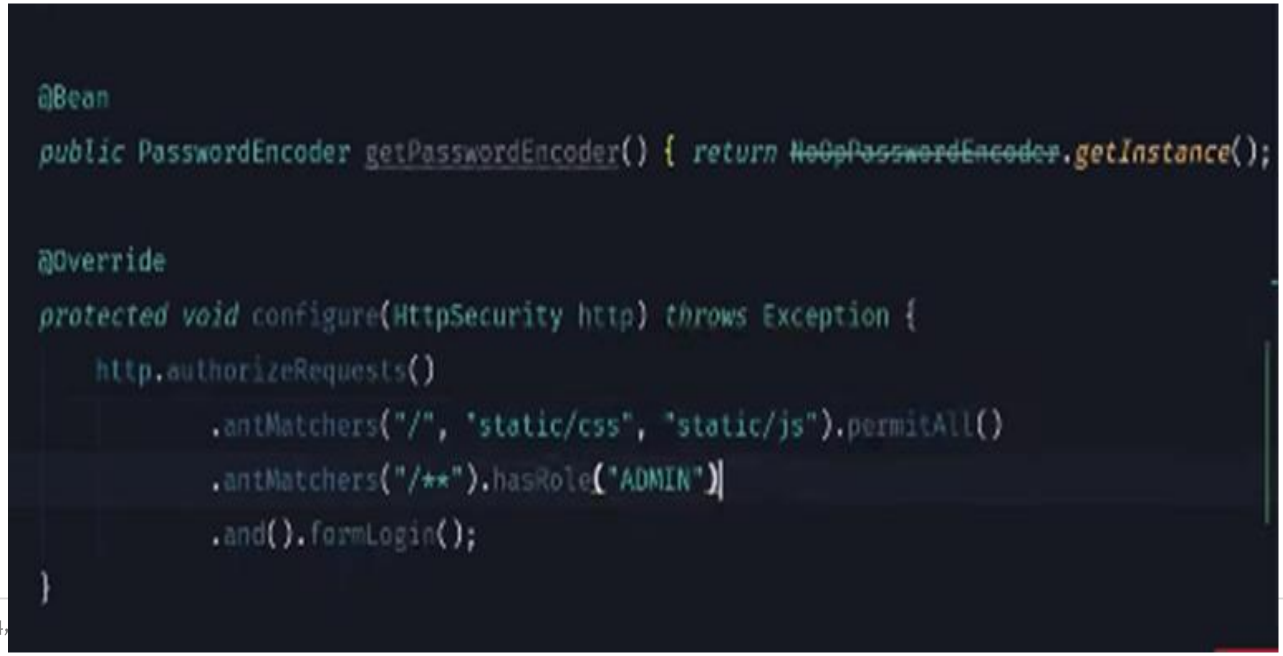
| API | Roles allows to access it |
|---|---|
| / | All (unauthenticated) |
| /user | USER and ADMIN roles |
| /admin | ADMIN role |

## HttpSecurity

- **To get access to this object for authorization we need to add a class that extends WebSecurityConfigurerAdaptor and add configure method that accepts HttpSecurity as a parameter**

- **These  methods are hooks provided by spring to configure default security**

- .permitAll method allows all users to access the url
- .antMatcher("/**").hasRole("user") --- represents all nested path in the current path. Starting from /
  This represents all urls are accessible to user whose role is USER
- To add multiple roles use hasAnyRole("USER","ADMIN")  instead of hasRole
- along with this you can also specify what type of login you need ----→ one of the choice is formLogin

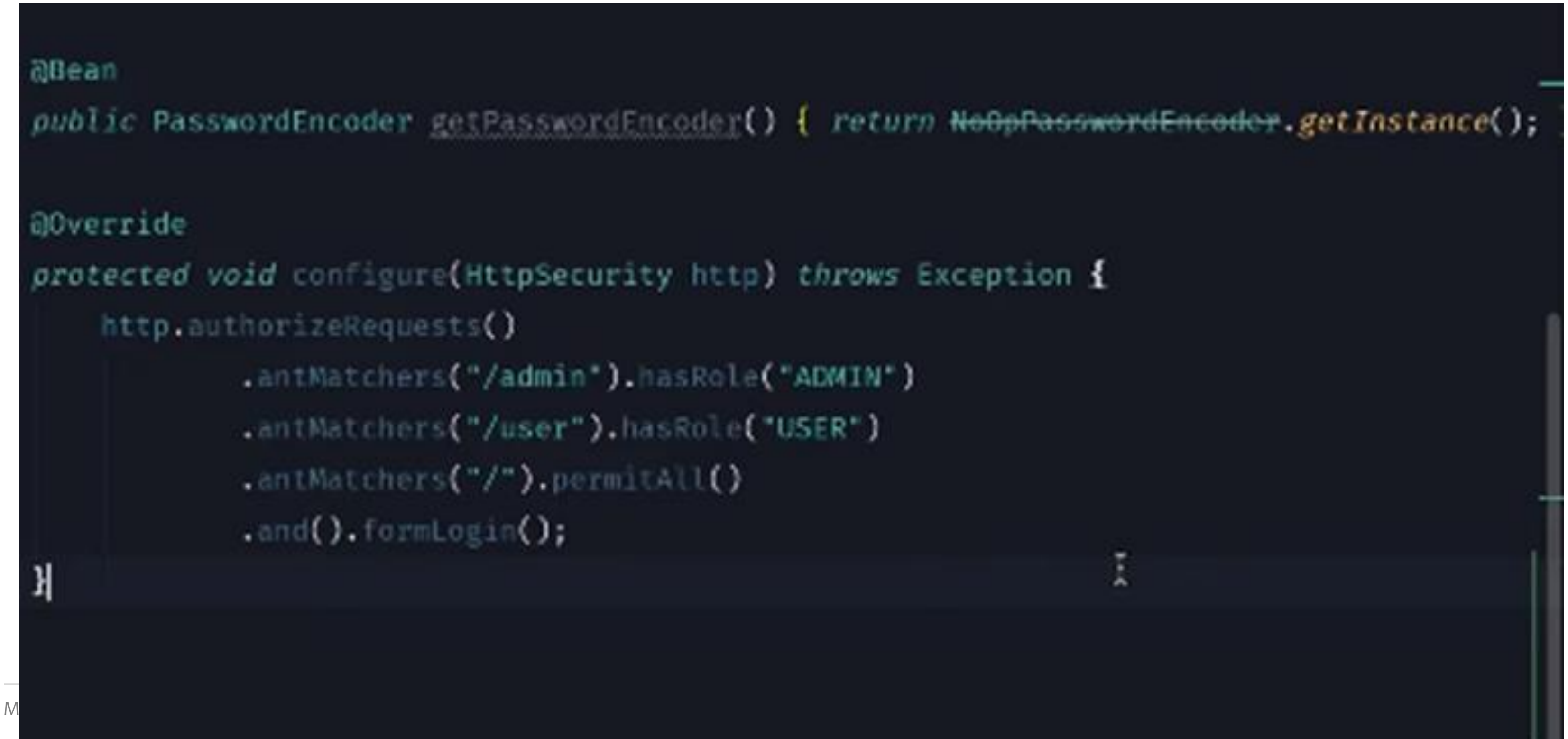


```
@Bean
public PasswordEncoder getPasswordEncoder() { return ~~NoOpPasswordEncoder~~.getInstance();

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers("/", "static/css", "static/js").permitAll()
            .antMatchers("/**").hasRole("ADMIN")
            .and().formLogin();
}
```

- If you try to login by user with different role it will show you error page

- While adding restrictions its better to add from most restrictive to least restrictive role
- ADMIN is most restrictive as only few users should have access to these pages.

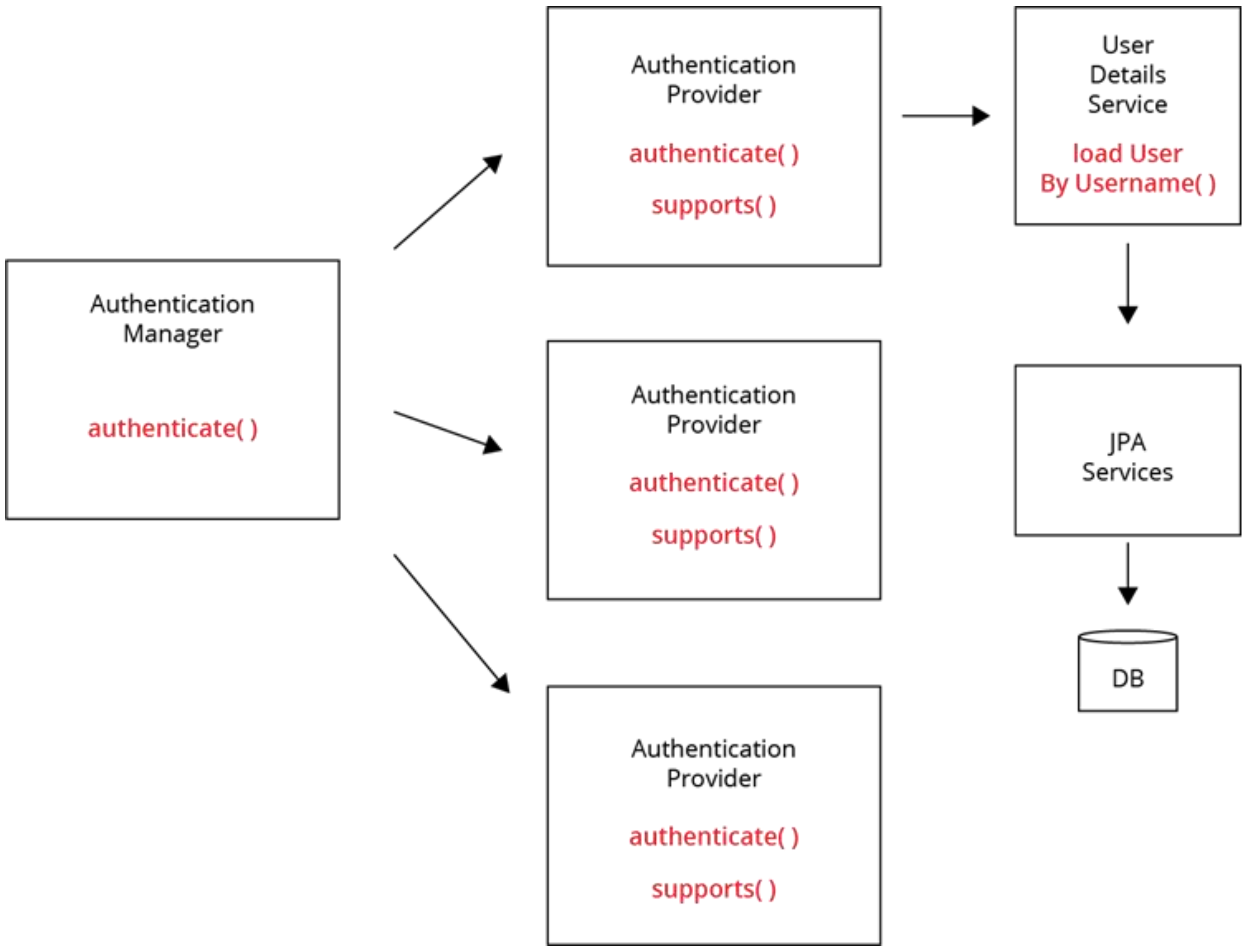


```
@Bean
public PasswordEncoder getPasswordEncoder() { return ~~NoOpPasswordEncoder~~.getInstance();

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

- **Spring default security also provides logout end point**
- **Localhost:9191/logout**

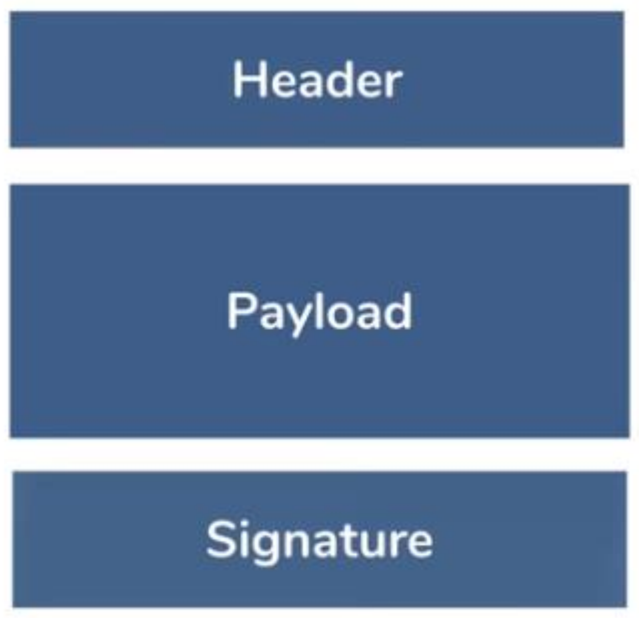
- **Will take you to logout page, once you click on logout button it will take you to login page.**


Authentication Manager
authenticate( )
Authentication Provider
authenticate( )
supports( )
Authentication Provider
authenticate( )
supports( )
Authentication Provider
authenticate( )
supports( )
User Details Service
load User By Username( )
JPA Services
DB

## JWT

## Structure of JWT

- **Three parts to JWT**



## Check using JWT.io

- You can check how jwt is changing if you change json

- Header represents algorithm used for encoding
- Header and payload are mostly in readable format
- Usually encoded in Base 64 format.

- JWT is only for authenticate user and identify user
- the footer is only be calculated and verified by server
- So if anyone changes the data then signature won't match and then server will say that you are not valid user.

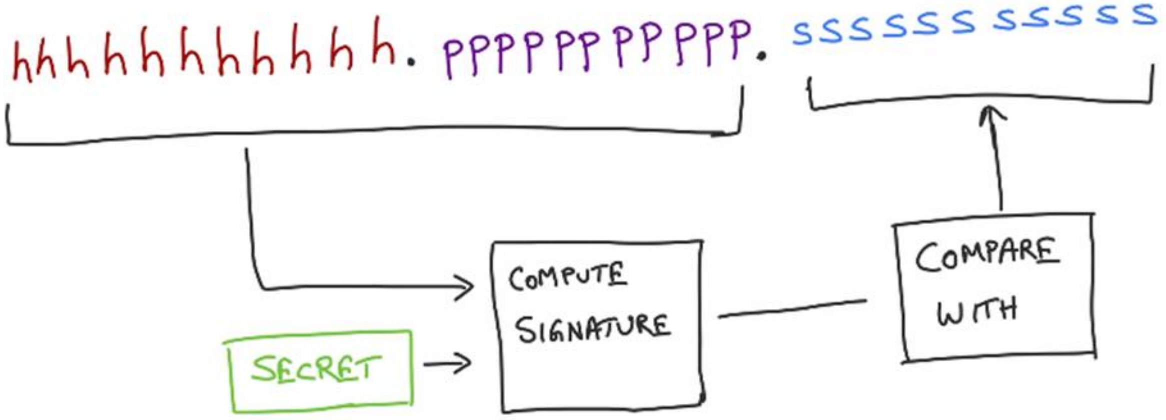
- It is not very secure.

## How to use

- **Server authenticates user and generates JWT token and sends it to user.**
- **JWT is not used for authentication but used for identification in subsequent communication**
- **So jwt is specific to autherization. JWT comes into picture when authentication is complete.**

- **Once user receives JWT token then client can store it in local storage or cookies. And need to send it in every communication.**

- **Oce you receive JWT token then it need to be added in autherization header in further communication**


hhhhhhhhhhh. PPPPPPPPPPP. SSSSSSSSSSSS
COMPUTE SIGNATURE
SECRET
COMPARE WITH

- **Not safe for passing confidential information**
- **someone  can hack it and use to  get my identity. Because server doesnot find, who send it .**
- **So we need to careful while passing.**
- **You may pass it through cookies or through localstorage**
- **Better to use it with https of oAuth which has its own secure way of transmitting data. So no one can steal it.**

## OAuth

## What is OAuth

- OAuth is simply a secure authorization protocol that deals with the authorization of third party application to access the user data without exposing their password.
- eg. (Login with fb, gPlus, twitter in many websites..) all work under this protocol.
- The Protocol becomes easier when you know the involved parties.
- Basically there are three parties involved: oAuth Provider, oAuth Client and Owner.
- oAuth Provider provides the auth token such as Facebook, twitter.
- oAuth Client are the applications which want access of the credentials on behalf of owner
- owner is the user which has account on oAuth providers such as facebook and twitter.

## What is OAuth2

- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean.
- It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account.
- OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

## OAuth2 Roles

- **OAuth2 provides 4 different roles.**
- **Resource Owner: User**
- **Client: Application**
- **Resource Server: API**
- **Authorization Server: API**

## OAuth2 Grant Types

- **Following are the 4 different grant types defined by OAuth2**
- **Authorization Code: used with server-side Applications**
- **Implicit: used with Mobile Apps or Web Applications (applications that run on the user's device)**
- **Resource Owner Password Credentials: used with trusted Applications, such as those owned by the service itself**
- **Client Credentials: used with Applications API access**

## OAuth2 Authorization Server Config

- **This class extends AuthorizationServerConfigurerAdapter and is responsible for generating tokens specific to a client.**
- **Suppose, if a user wants to login to mycompany.com via facebook then facebook auth server will be generating tokens for mycompany.com.**
- **In this case mycompany.com becomes the client which will be requesting for authorization code on behalf of user from facebook - the authorization server.**
- **Following is a similar implementation that facebook will be using.**

- **Here, we are using in-memory credentials with client_id as mycompany-client and CLIENT_SECRET as mycompany-secret.**

- **But you may use JDBC implementation too.**

- **@EnableAuthorizationServer: Enables an authorization server.AuthorizationServerEndpointsConfigurer**
- **defines the authorization and token endpoints and the token services**

## OAuth2 Resource Server Config

- **Resource in our context is the REST API which we have exposed for the crud operation.To access these resources, client must be authenticated.In real-time scenarios, whenever an user tries to access these resources, the user will be asked to provide his authenticity and once the user is authorized then he will be allowed to access these protected resources.**

- **@EnableResourceServer: Enables a resource server**

## Security Config

- **This class extends WebSecurityConfigurerAdapter and provides usual spring security configuration.**
- **We are using bcrypt encoder to encode our passwords.**
- **You can try this online Bcrypt Tool to encode and match bcrypt passwords.**
- **Following configuration basically bootstraps the authorization server and resource server.**

- **@EnableWebSecurity : Enables spring security web security support.**

- **@EnableGlobalMethodSecurity: Support to have method level access control such as @PreAuthorize @PostAuthorize**

- **Here, we are using inmemorytokenstore**
- **you may use JdbcTokenStore or JwtTokenStore.**