Spring Boot

**To create a project With (Spring Tool Suite) sts.**

All spring project needs to be generated from spring website “start.spring.io” (Because that is where all the dependencies for building java web apps are kept)

So Open STS and click new and choose “spring starter project”

Give your project a name and click next.

Then choose these 2 dependencies for your project.

1. Spring Web
2. Spring Data JPA

And click next or finish to wait for the project to be generated from the spring website together with all dependencies you selected

Remember, you need internet connection to download all the above dependencies.

**Simple project structure**

In the Main class that the project will come with by default. MYwebapplicatio.class (it can be anything base on what you specify)

// The @springBootApplication 🡺 this annotation tells java that this is a spring boot application (Because the file is just a java class and that’s the only ways java can determine spring app) It also tells java that, that’s the entry point of the app

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

// @SpringBootApplication is what tells java that this is a spring boot application and the entry point of the app

@SpringBootApplication

public class MywebApplication {

// Main method that will run when the app start

    public static void main(String[] args) {

        //the run() specify the calss to run

        SpringApplication.run(MywebApplication.class, args);

    }

}

Then create a new class in the same package (package is nothing but a folder in which a file is in) App.configuration.java To be a controller.

//@RestController mark this class as a rest api that will return a data instead of a webpage

// @RequestMapping(“url\_path”) 🡺 set a get request url path for a class

package com.example.demo;

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

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

//@RestController mark this class as a rest api that will return a data instead of a webpage

@RestController

public class Appconfiguration {

    // Set a url a get path for this class, is a get method

    @RequestMapping("/hello")

// the method to be executed when the above url is triggered

    public String hello() {

        //return some data when the above localhost:808/hello is navigated

        return "helloo world";

    }

}

Then in your project folder navigate to “src/main/resources/application.properties” This file contains the properties of your app

Then set the port number for your application server

server.port = 8081

Then run your application and navigate to localhost:8081/hello

If you don't set a port number, it will run on port 8080 by default.

**HOW TO CREATE AN OBJECT OF A CLASS WITH DEPENDENCY INJECTION**

Suppose I have Customers class like below.

I need to use the @Component annotation to tell this customers class that, hey allow other class to create object of you using dependency injection

package com.example.demo;

import org.springframework.stereotype.Component;

// @Component ==> Allow spring to use dependency injection to create object for this class

@Component

public class Customers {

    public void display() {

        System.out.println("customers object created successfully");

    }

}

Then inside main MywebApplication.java class

// The run() method returns an object of “ConfigurableApplicationContext” which has a method getBean() for creating an object of a class with dependency injection

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class MywebApplication {

    public static void main(String[] args) {

    // the run() method returns an object of ConfigurableApplicationContext object (hover on it to see)

    // use it to create dependency injection object of a class and add @Component to that class

    ConfigurableApplicationContext context  = SpringApplication.run(MywebApplication.class, args);

    //create an object of customers class with dependency injection

    Customers c = context.getBean(Customers.class);

    //call the display() method in customers

    c.display();

    }

}

**TO PASS QUERY STRING OR URL DATA FROM CONTROLLER TO JSP**

Create a new class HomeController.java add @controller to tell spring that, it is a web service controller and not just a plain java class.

Then use the HttpServletRequest to retrieve request object and set it to a session.

package com.example.demo;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import javax.websocket.Session;

import org.springframework.stereotype.Controller;

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

// Tell spring that this class is a web service or End-point that will return a data to the client

@Controller

public class HomeController {

    //set url

    @RequestMapping("/homepage")

    //HttpServletRequest ==> contains object of request

    public String hello(HttpServletRequest req) {

        // use req.getParameter("query\_string\_key") to retrieve a query string in url eg. http://localhost:8081/homepage?course=python

        String courseName = req.getParameter("course"); // get the value of “course” query string

        // Create a session object with req.getSession()

        HttpSession session = req.getSession();

        // set a session with key and value so that you can retrieve it in jsp files or views

        session.setAttribute("courseName", courseName);

// Call the below jsp homepage file(make sure to install Tomcat Jasper if not, it will not work)

        return "homepage.jsp";

    }

}

Then navigate to “src/main/webapp/homepage.jsp” (if is not available, just creat it manually)

// Then display the above session with ${session\_Key}

// Remember .jsp files are just views for your application.

//By default, java does not recognize .jsp files as serverlet (java html files). You need to download the same version of tomcat Jesper dependency and add it to your pom.xml (check “maven dependency and look for the same version of embedded tomcat in your project and download the same tomcat jesper for it at mavenrepository.com)

// that will convert jsp files into servlet

Inside homepage.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

// display the courseName session set above

<p> The course Name is ${courseName } </p>

</body>

</html>

**USE MODELANDVIEW CLASS INSTEAD OF USING SESSION AND HTTSEVELETREQUEST ABOVE**

Inside the controller class. You can use modelandview to set the jsp view for a specific url and also the data to be transferred to the views.

package com.example.demo;

import org.springframework.stereotype.Controller;

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

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

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HomeController {

    //Create an object of ModeAndview

    ModelAndView mv = new ModelAndView();

    //Set url

    @RequestMapping("/homepage")

    // the function must return a ModelAndview

// @RequestParam(“nickName”) means if (name=justice) or (nyName=justice) just grab it. Eg. http://localhost:8081/homepage?name=justice

// OR <http://localhost:8081/homepage?myName=Bennard> (You can remove it if you don’t any nickname)

    public ModelAndView hello(@RequestParam("name") String myName) {

        // set the views name or jsp file name

        mv.setViewName("homepage");

        // use addObject("key", value) to send a data to the view (It's like a session);

        mv.addObject("name", myName);

        // return modelAndview

        return mv;

    }

    @RequestMapping("/about")

    public ModelAndView about() {

// set the vies name or jsp file name

        mv.setViewName("about");

         // return modelAndview

        return mv;

    }

}

Then inside the jsp vies. Use ${data\_Key} to grab the name of the data that was sent with the above controller

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<p> How are you doing? I learnt you were having a bad day </p>

// Retrieve the name value that was sent from the controller

<p> The course Name is ${name} </p>

</body>

</html>

Note (I have skipped the about jsp view) You can navigate to http://localhost:8081/homepage?name=justice to see the result.

**TO CONFIGURE FOLDER AND EXTENSION FOR JSP FILES**

Inside application properties. Use:

spring.mvc.view.prefix=/ The Name of the folder in src/main/webapp / foderName/

/

spring.mvc.view.suffix=.jsp

// Use .prefix 🡺 to set folder

// use .suffix 🡺 to set extension

spring.mvc.view.prefix=/pages/  // set the folder for where views are

spring.mvc.view.suffix=.jsp   // set the extension of your files in the above folder

Then inside your controller class. Return the name of the files in the above “pages” folder without the extension

@Controller

public class HomeController {

    @RequestMapping("/homepage")

    public String hello() {

    // "homepage" is the name of the jsp file (homepage.jsp) without the extension

        return "homepage";

    }

    @RequestMapping("/about")

    public String about() {

    // "about" is the name of the jsp file (about.jsp) without the extension

        return "about";

    }

}

**REQUEST METHODS ANOTATIONS**

**All mapping are specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET/POST/PUT/DELETE)**

@GetMapping - shortcut for @RequestMapping(method = RequestMethod.GET)

@PostMapping - shortcut for @RequestMapping(method = RequestMethod.POST)

@PutMapping - shortcut for @RequestMapping(method = RequestMethod.PUT)

@DeleteMapping - shortcut for @RequestMapping(method =RequestMethod.DELETE)

@GetMapping – Handle HTTP Get Requests

@PostMapping – Handle HTTP POST Requests

@PutMapping – Handle HTTP Put Requests

@DeleteMapping – Handle HTTP Delete Requests

***Technically speaking, you can use the annotation at each other's place but each annotation is created for different purpose.***

* Like to handle the HTTP Get requests and get the data only use @GetMapping
* To perform add/update operation, use HTTP POST/PUT request i.e. @PostMapping or @PutMapping
* And to perform delete operation, use HTTP Delete request i.e. @DeleteMapping annotation.

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

**@column Annotation**

@column("columnname" Or properties) == Is used to set entity/db\_table column properties like, defaultValue, nullable and many more.

@entity

class Books(){

@id

@generatedValue

int id;

@column(nullable=false, updatable=false) //the bookName column can't be null and also can't be updated

String bookName;

}

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

@generatedValue() Anotation

@generatedValue(strategy=GenerationType.AUTO); ==> Set the id of a table in db table/entity as auto\_increment just like mysql primary key

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

**@service Anotation**

@service ==> marks a class as a service which will Connect to jpa to fetch data from db. A service is nothing but a java class that will inject it dependencies into other class like controllers. (Whenever you declare a class as a service, spring create a singleton instance object of it globally)

So you don't have to create object of service by doing: Books books = new Books(), "no but" Books books; and annotate it with @Autowired

// becuase you can't create a multiple object of a singleton class (So the object is created by default by spring, you just need to reference it in other class)

When you inject service into other class, make sure to append the @autowire anotation to it.

Because the object of every service is created by spring by default. And you just need to request it with the @autowire anotation

eg.

@service // mark this class as a service so that spring will creat singleton instance of it globally

class Myservice(){

}

// Then inside a controller in a different class, get the singleton object of the above service...

@Controller

class School{

 @autowire    // request the singleton object of above Myservice from spring

Public Myservice myService\_class;  // get the object withiout the new instance

}

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

**@autowired Anotation**

@autowired ==> Request an object of a singleton service, controller and component from spring-boot to the calling class.

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

@pathVariable Anotation

@pathVariable ==> Is used to grab a required parameter from the url specified with @requestMapping("/home/{parameterToGrab\_by\_path\_variable});

Remember the parameter url must be the same as the parameter in the method of the pathVariable

Remember ==> you can also specify a nickname for the @pathVariable in case you want to grab the required parameter using different name

// grab the required parameter id from url

@RequestMapping("/topics/{id})

// bellow (String id) = the id must always be the same name as the id above in "/topics/{id}

public String getTopic( @pathVariable String id ){

        return id;

      }

Also you can assign nickname to it like..

// @pathVariable("fool" ) String id ==> if the user pass "id" or fool, in the url "grab" it

@RequestMapping("/topics/{id})

public String getTopic( @pathVariable("fool" ) String id){

  return id;

}

Note= @pathVariable ==>Is for getting requried parameters. While @getParams is for getting query string values

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

@RequestBody Anotation

@RequestBody == use To grab all the name of the html input element the client is sending to the server. Like username, email and password.

Let say I have below form values from form input:

{"firstName" : "Justice", "lastName" : "Ankomah" }

//controller

@PostMapping("/register")

// Grab the above form values And assign it to the User object

public String getDescription(@RequestBody User user){

// getFirstName() method is getter method in below “User” class

 return user.getFirstName(); // return the user Firstname

}

//User class

 public class User{

 private String firstName;

 private String lastName;

 // + getters, setters

}

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

ResponseEntity<>

ResponseEntity<> == It represent the http response the server will send to the client including the hearders, body, statuscode. You can use it to configure what response to send to the client. (is just like res.send() in node.js)

If we want to use it, we have to return it from the method that uses it. Spring takes care of the rest.

ResponseEntity is a generic type which is this symbol <>. Consequently, we can use any type as the response body

@GetMapping("/hello")

ResponseEntity<String> hello() {

// use  HttpStatus.BAD\_REQUEST or HttpStatus.OK to set a status code

// beloow "Hello World!" represent the body of the response to send to the user.

// HttpStatus.OK is the statuscode of the response usually 200 =Ok or 400 =bad\_request

    return new ResponseEntity<>("Hello World!", HttpStatus.OK);

}

Additionally, you can use it to set HTTP headers:

@GetMapping("/customHeader")

ResponseEntity<String> customHeader() {

    HttpHeaders headers = new HttpHeaders();

    headers.add("Application-type", "text/json");

    return new ResponseEntity<>(  "Custom header set", headers, HttpStatus.OK);

}

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

**@entity class**

@entity ==> Tells jpa that, create a table in db with the same class name. And use the class properties to create columns for the same table. OR If the table already exist with the same name, it will map to it without recreating it and run sql-queries on that table.

@entity  // create a table with the name of this class and columns of it properties

class Books(){

@id  // this "int id" below I should be the primary key in the table

@generatedValue // auto\_increment this primary key "int id" in the db

int id;

String bookName;

}

// Above tells jpa to create a table in db called "Books" and clumns "id" and "bookName"

// the @id tells jpa that the "int id" should be the primary key of the table

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

**JPA**

JPA ==> contains all the logic or methods for performing crud operation on a db table. Is just like laravel eloquent, all it query methods will be converted to sql queries by spring when it get execute. (You can also create your own custom query too)

JpaRepository interface: Gives you all the methods for performing sql queries. It like eloquent in laravel. You don't need to write your own sql queries. Just use the methods it provides you with. Those methods will be converted into sql queries by spring jpa when you run your app.

But what you should know is that:

How will jpa know which table in your db to run sql-query on? Like: “Select \* from usersRegistration” ( How will jpa know “usersRegistration” table?)

Or which column you want to select values from? Like select \* from usersRegistration where [email =justy@gmail.com](mailto:email%20%20=justy@gmail.com) (how wil jpa know email column on userRegistration Table?)

So because of this, you need to tell jpa which table and columns you want to run queries on.

Because jpa methods are just like below sql statement:

select \* from ……

Select \* from ….. Where …. =….

So you are the one to fill all the above ….

Eg.

You have to create an interface that will implement jpa JpaRepository<Entity\_class\_name, The\_Table\_primary\_key\_DataType>

Above generic values in <> specify which Table class name and it primary key data-type (That’s the only 2 thing jpa need from you.)

You have to always annotate it with @Repository to tell spring that such interface is a repository

//Create a class to represent db table values

@Entity //create a class in db with name of the class and it properties as table columns

public class Topic {

    @Id

    @GeneratedValue

    private Integer id;

    private String topicName;

    private String topicPassword;

    // + getters setters goes here

    }

// create an interface to extends JpaRepository

Repository

Public interface TopicRepository extends JpaRepository<Topic, Integer> {

// Becuase this jpa interface is linked to the Topic table, all sql queries will be run on this table

}

// Then in your "Topic" service class, get the object of above interface and access the methods in it.

@Service

public class TopicService{

@autowired

TopicRepository obj\_TopicRepository;   // creat an object of above interface by Autowiring it

obj\_TopicRepository.findOne();   // findOne() is a method in JpaRepository for finding single user in db (Select \* from Users where name=justice)

}

**=====================================================================**

JPA AND ENTITY CLASS NOTE

You should note that, JpaRepository is linked to the entity class and it properties.

So it assumes that all the jpa-sql methods you run should relate to the Entity/db\_table property name;

Example I have below Entity class Users which is linked to below jpa Reository

@Entity

**public** **class** Users {

@Id

@GeneratedValue

**private** Integer id;

**private** String userName;

**private** String userPassword;

//getters and setters + toSTring method geos below

}

//My repostository interface

@Repository

**public** **interface** MyRepository **extends** JpaRepository<Users, Integer>{

//Note: JPA assumes that the method below “findByuserName()” is structured like this === findBy\_abovedbTablePropertyName();

//Which means it want you to add one of the entity property name to the findBy… (The … should be one of the above Users class property name)

// So the below sql will be == select \* from Users where userName = “someCustomeName”

findByuserName(String username);

So if you do like:

findByusername(String username); // this will be sql error because of wrong above Users class property name

Above will be an error because there is no property in the above entity with the name “username” it rather userName (case Sensitive)

The jpa findBy... means = Select \* from Users where userName = “something”;

}

**============================================**

**CORS ISSUE**

To fix cors issue in spring boot. Simple add @CrossOrigin to the top of your controller class and import it package and it will work.

Eg. Inside EmployeeController.java

@CrossOrigin   // add this to your controller class and import it package to solve all cors issue

@RestController

public class EmployeeController {

// controller method to add new users

@PostMapping("/add")

public ResponseEntity<Employee> addEmployee(@RequestBody Employee employee){

Employee newEmployee =  obj\_EmployeeService.addEmployee(employee);

return new ResponseEntity<>(newEmployee, HttpStatus.OK);

}

}

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

**SPRING BOOT SECURITY**

It's just like **a middleware in laravel and node.js** that will direct the user to the url he want to visit or terminate or redirect him to a different url.

Middleware is nothing but a function that sit in-front of all url requested to say something like “hey you cannot see the dashboard url unless you’re login. So login please.”

What normally happens is that, it will authenticate the user to see if he is allow to access such url or not.

Example, you can allow all users who has logged-in already to view their dashboard. Therefore if a user is not logged-in but want to view his dashboard, you can add spring security as a middleware to authenticate the user and redirect him to the login page to login before he can access the dashboard url.

**HOW TO ADD SPRING BOOT SECURITY DEPENDENCY TO YOUR PROJECT**

To add spring boot security to your project, you just have to add the “spring boot security” dependency to your project when creating it at starter.spring.io.

Or simple add this 2 below dependency to your pom.xml parent <dependency> tag.

<dependency> <groupId>org.springframework.boot</groupId ><artifactId>spring-boot-starter-security</artifactId> </dependency>

<dependency><groupId>org.springframework.security</groupId><artifactId>spring-security-test</artifactId><scope>test</scope></dependency>

**Remember**, immediately you add the above dependency to your project, spring boot will add a middleware to your project. It will create two urls, <http://localhost:8080/login> and <http://localhost:8080/logout> by default.

So whenever you run your application and navigate to any url, it will redirect you to the login page to login before you can access any url you want because of this default middleware.

So the job of this default middleware is to allow only users who has login to access any other urls.

But how will you know which username and password to login?

The default username by spring boot is: user

But you will get a different password generated on “spring tool suit ide console” each time you run your project like:

Using generated security password: 979a20a7-f632-42c5-b3c3-9a8004620946

Then you can use those two to login to access any url you want.

But you can set default username and password in your application.properties file with below properties:

spring.security.user.name=justice // To set a default spring security useName

spring.security.user.password=justicepassword //to set a defualt spring security password

spring.security.user.role=USER //set a role for the user

Then all you will have to do is to use the above details to login all the time. When spring security see this above details in your project application.properites file, it will not generate userName and password for you again.

To learn more about spring boot security, watch this video = <https://www.youtube.com/watch?v=Of4HFbsPKqk&list=PLEocw3gLFc8XRaRBZkhBEZ_R3tmvfkWZz>

**How To Override Spring Boot Default Authentication And Authorization**

Now you can override all the above default spring boot authentication and configuration to your choice.

There is a Class called “WebSecurityConfigurerAdapter” it contains all the methods you need to override for configuring spring-security.

Basically, you need to Override it:

“configure(AuthenticationManagerBuilder auth)” = You use the “auth” object to tell spring which users are allowed to access your app (In real world application this users will be stored in database)

“configure(HttpSecurity )” = Override this methods to tell spring which url the above authenticated users in the above method are allowed to access (This is because, you may have “admin” and “users” page so you need to tell spring which users are to access admin or users page )

Also you need to create a method annotated with @Bean that will return password Encoder. (You have to use this to encrypt their password)

You need to create a different class and extend this class to configure it.

Basically there are 3 things you need to do.

1. Override the configure authentication method to configure which users you want to allow them to access your app. Also you need to encrypt users password here
2. Override the configure authorization method to decide which which are allowed to access some url and whom are not
3. Create a Password encoder that will decrypt the users to see if their credentials in form input is equal to what is saved in db

So create a new class to extends webSecurityconfigurerAdapter and annotate this class as @EnableWebSecurity = This will mark it as the class for configuring spring-security in the project. Eg:

//Create a class that will extend WebSecurityConfigurerAdapter and overide it methods

@EnableWebSecurity    //MARK THIS AS A CLASS THAT WILL HANDLE WEB SECURITY AUTHENTICATION IN THE PROJECT

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    // Overide "configure(AuthenticationManagerBuilder auth)" method

    @Override

    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

        // ======== BEGEINING OF SPRING-SECURITY-AUTHENTICATION-CONFIGURATION ====

        // SET YOUR AUTHENTICATED USERS DETAILS WITH THE  "auth" OBJECT

        /\* ALL BELOW IS CALLED SPRING-SECURITY-AUTHENTICAION-CONFIGURATION.

        Because you are creating the details of users to be allowed to login.

        \*/

        // create inMemoryAuthentication configuration (Meaning all the below users details will be saved in my spring boot project memory)

        // But you have to store it in mysql db in real project instead of inMemoryAuthentication()

        auth.inMemoryAuthentication()

        // Create users with details that can login to the application to access urls

        .withUser("admin")  //username

        //user password hashed with below getpasswordEncorder() method.

        // it has encode() interface method in it return type data. pass in the user password you want to hash

        .password(getpasswordEncorder().encode("admin"))

        .roles("admin")     //user role either admin or regular user or whatever you want

        .and()              // To add another user with below details

        .withUser("user")

        .password(getpasswordEncorder().encode("user"))

        .roles("user");

    }

    // create a bean of any method that will return PasswordEncoder to hash the users password With BCrypt

    @Bean

    public PasswordEncoder getpasswordEncorder() {

        return new BCryptPasswordEncoder();

    }

    // ======== END OF SPRING-SECURITY-AUTHENTICATION-CONFIGURATION ====

    // ======== BEGINNING OF SPRING-SECURITY-AUTHORIZATION-CONFIGURATION ====

    /\* authorization means which urls do you want the above authenticated users to access

     Becuase you my want student to access only students-urls and teachers to access only teachers-urls\*/

    @Override

    protected void  http configure(HttpSecurity ) throws Exception {

        //USE THE ABOVE http OBJECT TO AUTHORIZE URLS

        http.authorizeRequests()

        .antMatchers("/").permitAll()             // allow all users to access "/" url (Thats the home page http://localhost:8080)

        .antMatchers("/admin").hasRole("admin")   // allow only userls with role "admin" to access http://localhost:8080/admin url

        .antMatchers("/user").hasAnyRole("user","admin")  // allow users with role "user" or "admin" to access http://localhost:8080/

        .and().formLogin();    //Direct user to login-page http://localhost:8080/login if he is not authorized to access a url.

        // if you have a custom login-page you can pass it into the above .and().formLogin("/justicelogin");

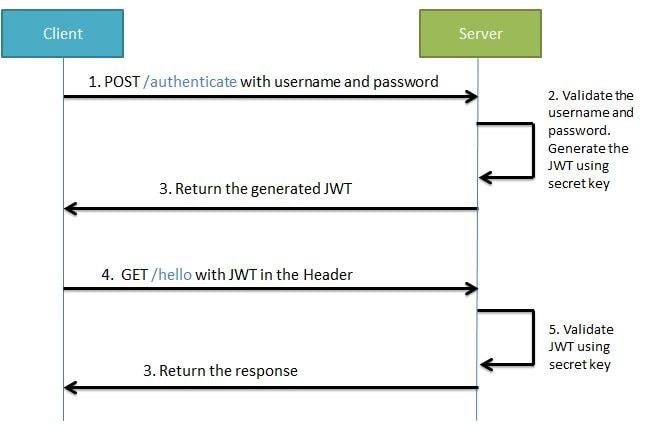
    }

    // ======== BEGINNING OF SPRING-SECURITY-AUTHORIZATION-CONFIGURATION ====

}

Jwt Request Flow between the Client and the server

1. The client will send his username and password in a post method to the server
2. The server will authenticate his details to see if is correct in what is in saved in dB
3. If the client details are wrong, the sever will send and error response. If the user details are correct, the sevser will generate a JWT token and send it to the user.
4. Now, if the client want to access any url, he has to send the JWT token in the request headers
5. The server will validate the client JWT token with the secret key it was generated with
6. The server will authorize him to the url it want to access if his JWT token is valid. If his JWT token is not valid or is expired, the server will return an error to the client



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

**Java Spring BOOT TIPS**

**To see all methods of extended class you can implement** => Press CTRL+3, then type "override" in the search box

OR

Right click on STS ide and choose source=>overridden/implement methods (To see all the methods of the extended class)