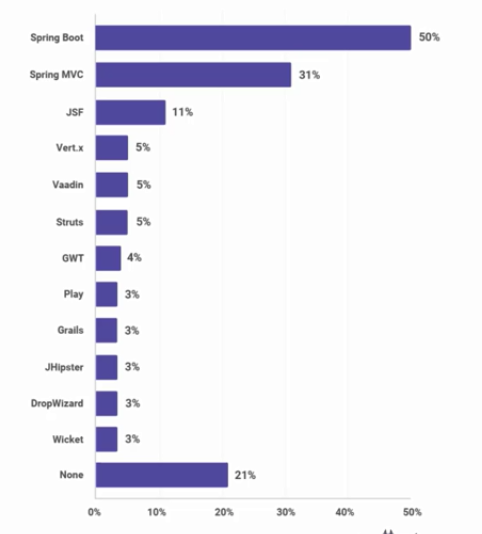
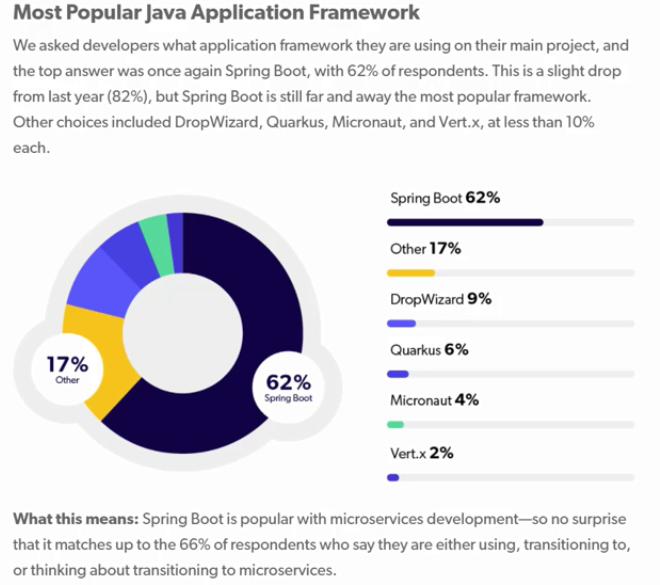
**SPRING BOOT NOTES BY VIJAY SIR**

**What is Spring?**

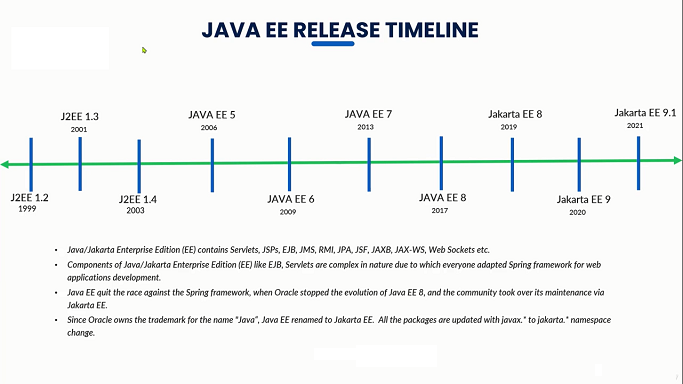
* **Spring is a very popular framework for building Java applications.**
* **Provides a large number of helper classes and annotations.**
* **The Spring Framework is a well-established, strong, and highly adaptable framework designed for creating web applications using Java.**
* **Spring makes programming Java quicker, easier, and safer. Focusing on speed, simplicity, and productivity has made it the world’s most popular Java framework.**
* **It doesn’t matter that you are intend to build secure, reactive, cloud-based microservices for the web or complex streaming data flows for the enterprise, spring has the tool to help.**
* **Spring is an alternative to EJB’s and struts which were used in the 2000’s offering more simplicity, more features, as well as offering third party integrations.**
* **Spring framework is an open-source.**
* **Spring website – official 🡺** [**www.spring.io**](http://www.spring.io)
* **Lightweight development with Java POJOs(Plain-Old-Java-Objects)**
* **Dependency injection to promote loose coupling(EJBs are heavy weight)**
* **Minimize boilerplate Java code.**

**Note: Java EE now called as Jakarta EE because of Oracle Renaming**

****

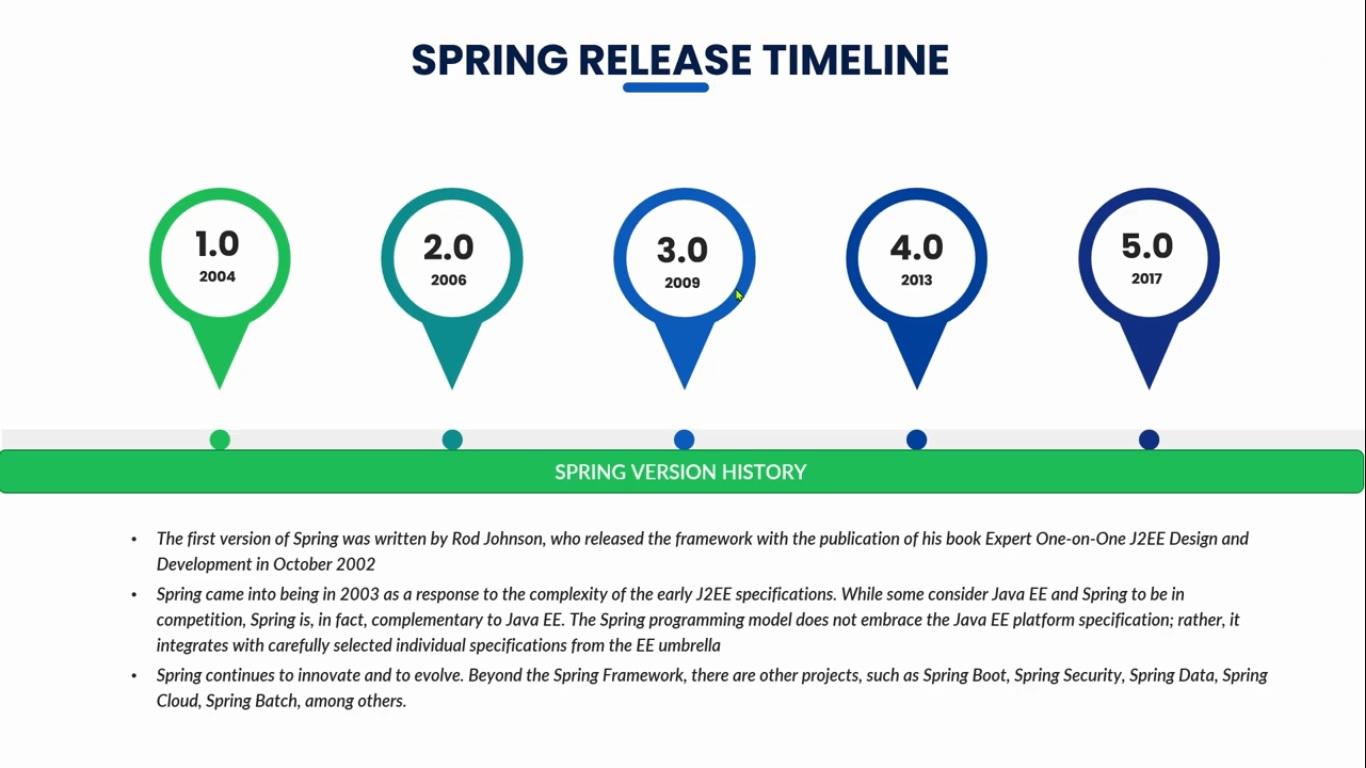


1. **Java EE Release Timeline**

****

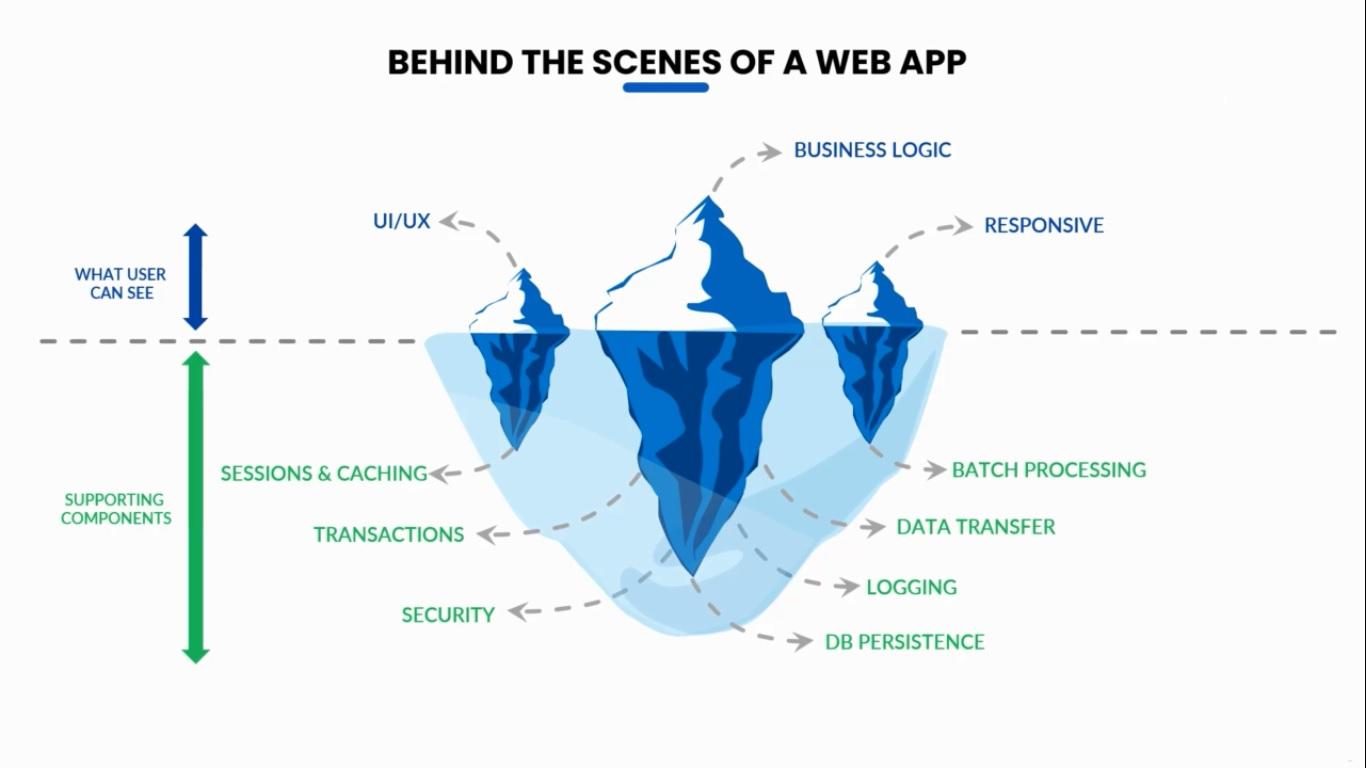
* **Spring framework developed by the person called Rod Johnson**
* **Roderick "Rod" Johnson is an Australian computer specialist who created the Spring Framework and co-founded SpringSource**
* **He released the framework with the publication of his book expert one-on-one J2EE design and development in October 2002.**

1. **Spring Release Timeline**

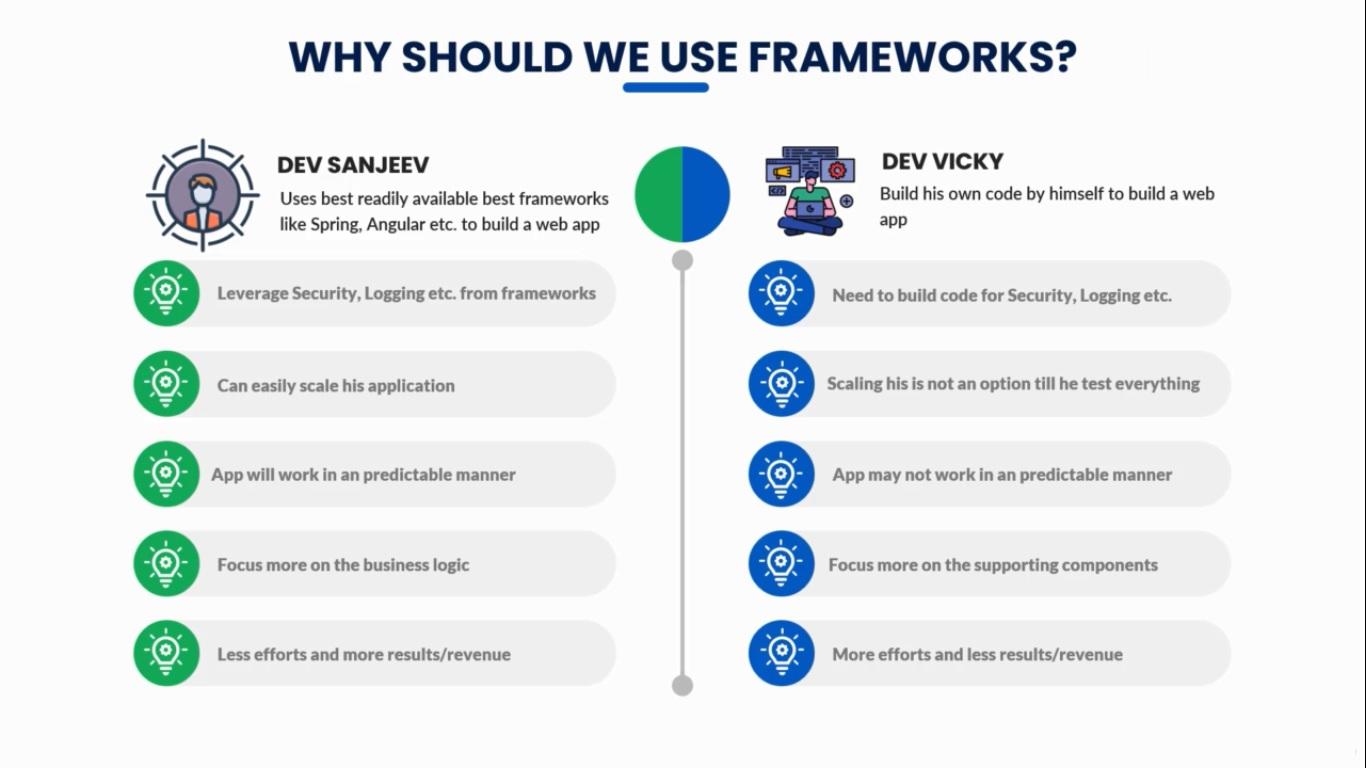
****

**Note:** Spring Framework 6.0 has been released on 16 November 2022 and came with a Java 17+ baseline and a move to Jakarta EE 9+ (in the jakarta namespace), with a focus on the recently released Jakarta EE 10 APIs such as Servlet 6.0 and JPA 3.1.

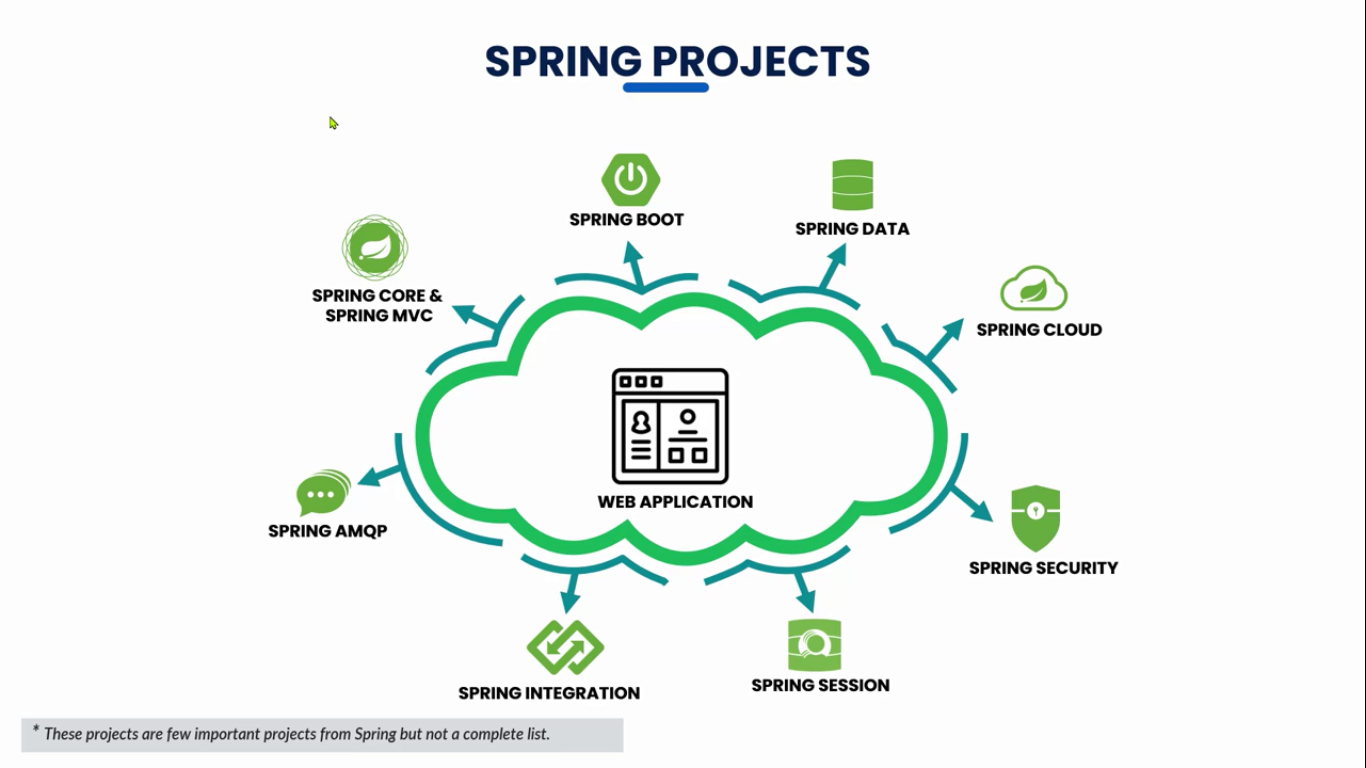
1. **Spring Release Timeline**

****

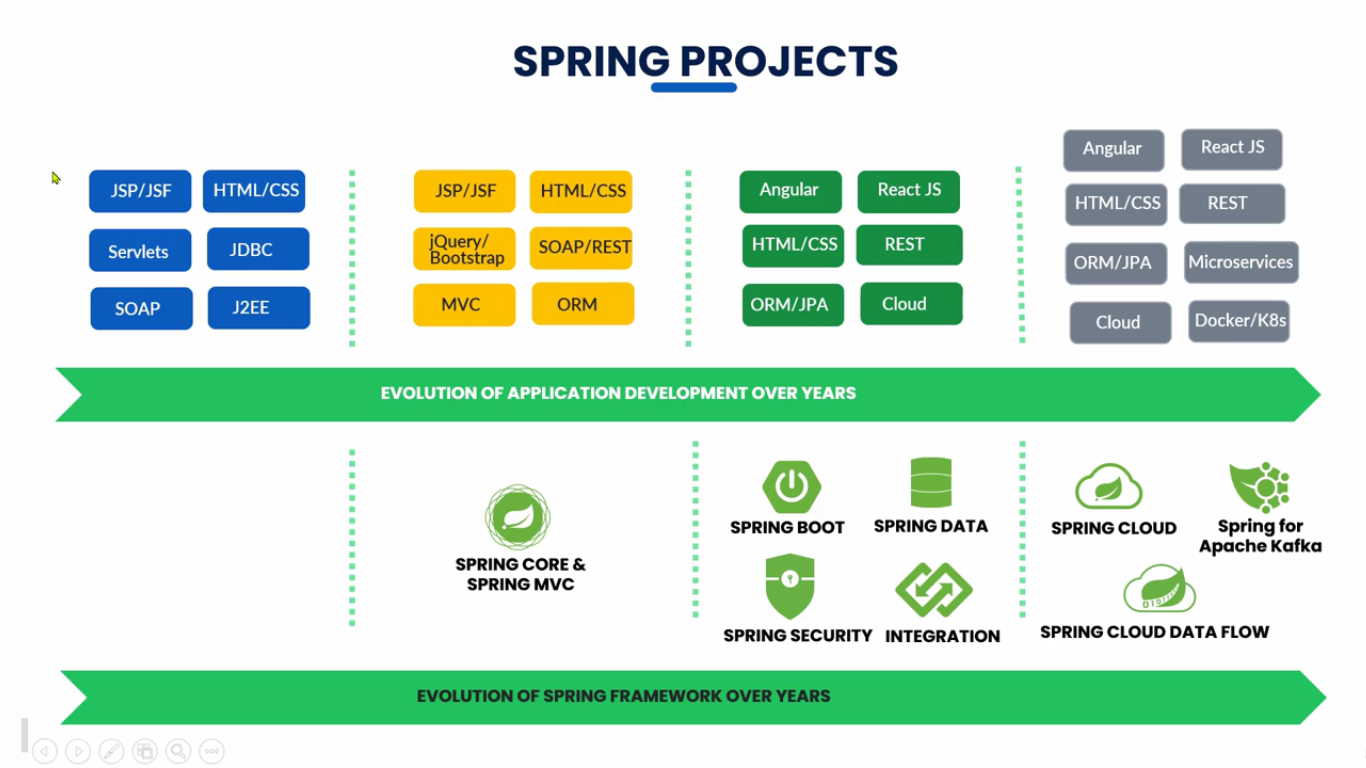
**5.Why The Frameworks?**

****

**6. Spring Projects**

****

**6. Evolution for Spring framework for web applications**

****

**The Problem:**

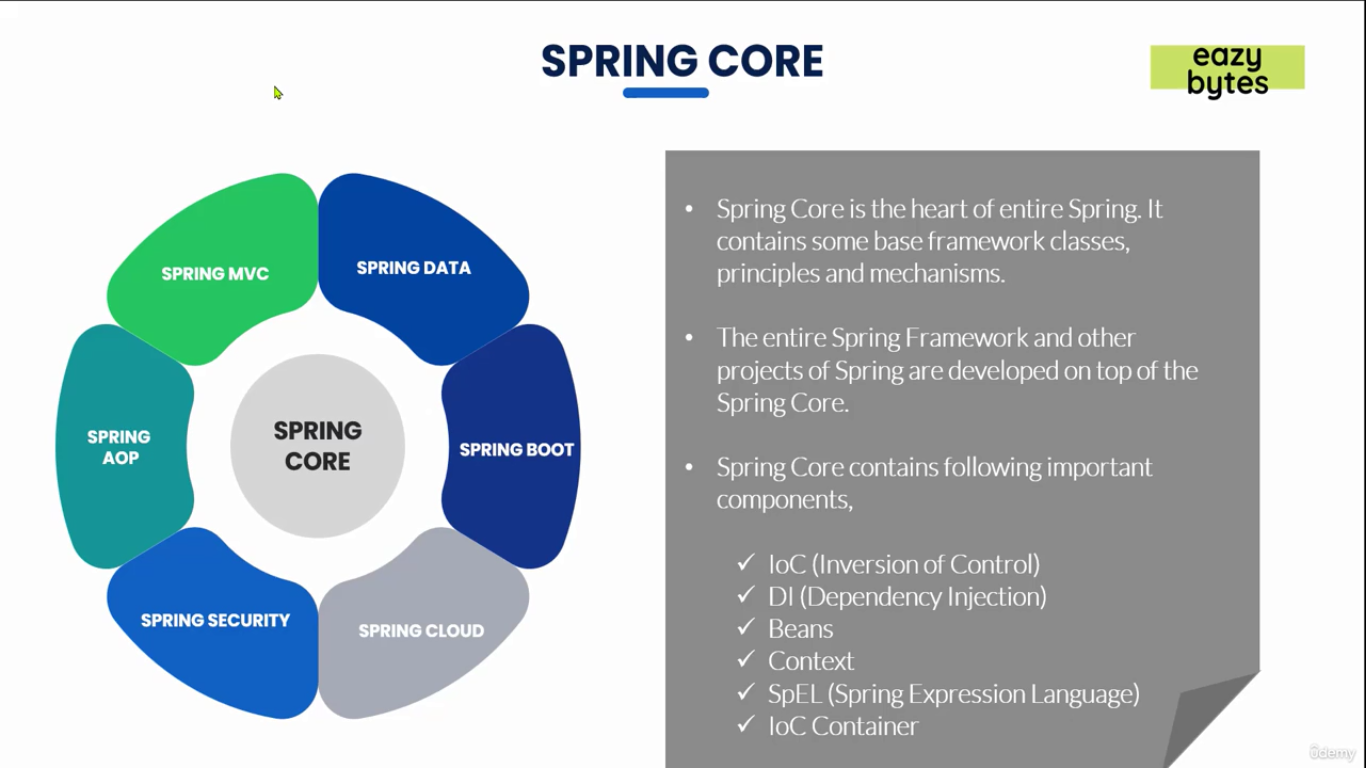
**Building a traditional spring application is really HARD!**

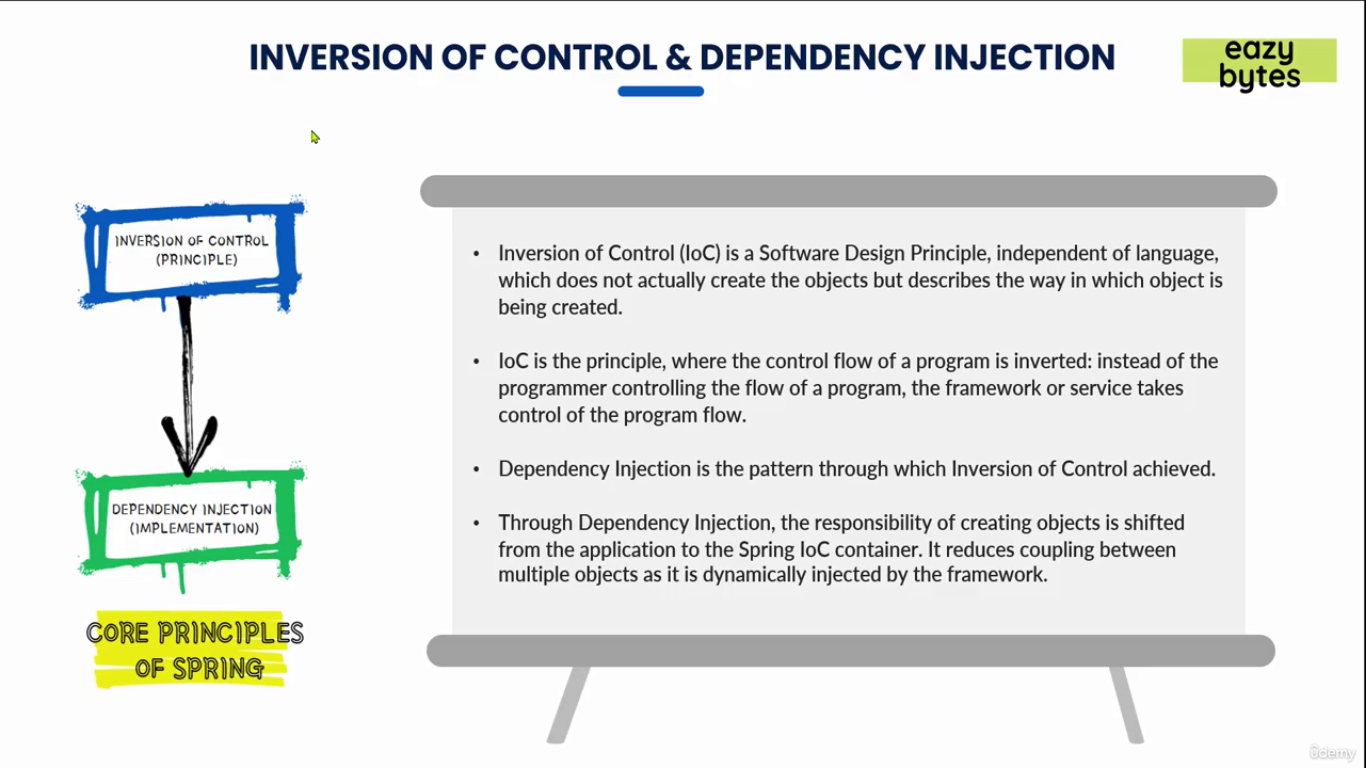
* **Which JAR dependencies do I need?**
* **How do I set up configuration(xml or Java)?**
* **How do I install server?(Tomcat, JBoss ect..)?**



**Spring Boot Solution**

* Make it easier to get started with spring development
* Minimize the amount of manual configuration(perform auto-configuration based on props files and JAR classpath)
* Helps to resolve dependency conflicts(Maven or Gradle)
* Provide an embedded HTTP server so you can get started quickly(Tomcat, Jetty, Undertow)



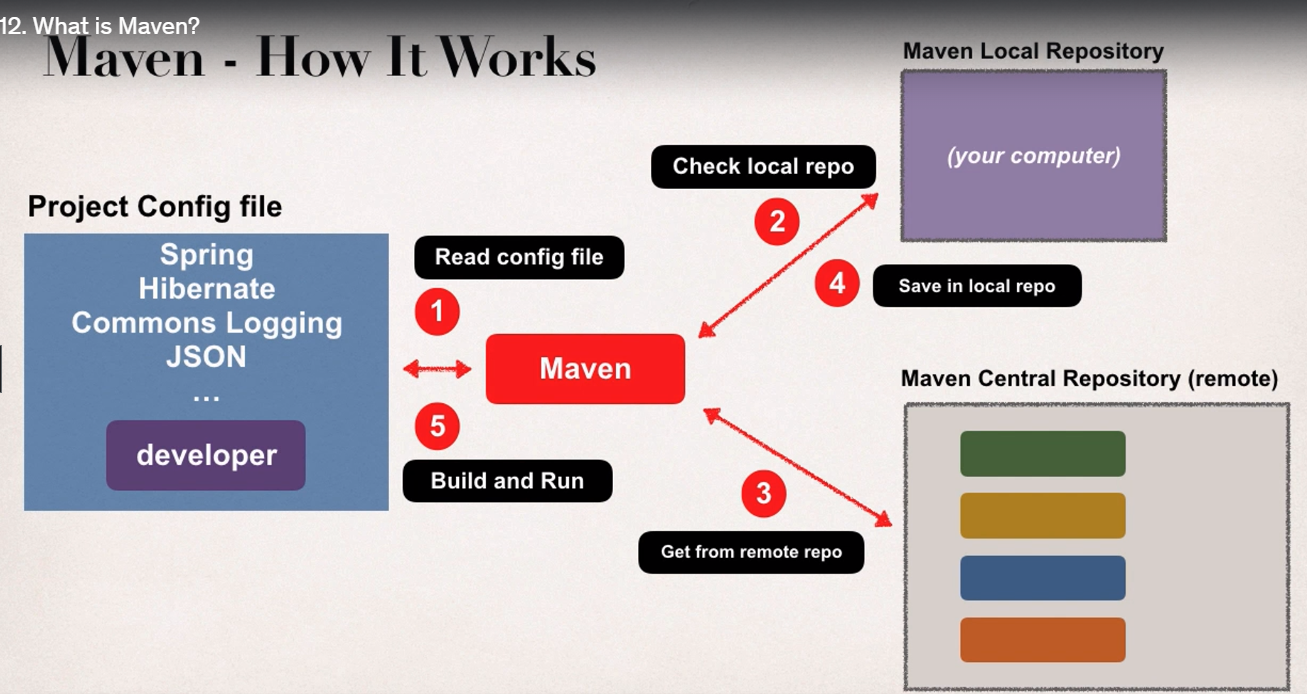


Note: before starting any development make sure that you install Java 17 and set JAVA\_HOME to JDK17/ and add path variable to path JDK17/bin folder

Note: Maven is a open source build tool.

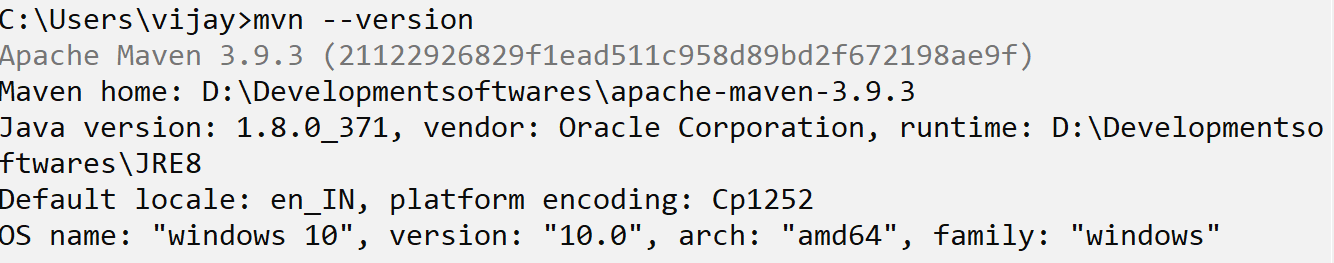
What is Maven?

* Maven is a open source build tool or Project Management Tool
* When you generate projects using spring initializr 🡺 start.spring.io it can generate a Maven project for us.
* Most popular use of Maven is for build management and dependencies.



Installing Maven:

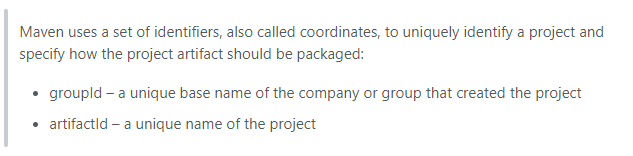
* Go to maven.apache.org
* Click on download
* Under Files select BinaryZipArchive - apache-maven-3.9.3-bin.zip
* Download and Extract the content and keep the folder inside DevelopmentSoftwares Folder
* Set the path of bin folder by Creating a new variable called MAVEN\_HOME 🡺 MyComputer 🡺 Properties 🡺 Select Environment Variables 🡺 Under user variables 🡺 create MAVEN\_HOME 🡺 as a value paste the bin folder location path.
* Set the path of bin folder location to already available path variable 🡺 MyComputer 🡺 Properties 🡺 Select Environment Variables 🡺 Under user variables 🡺 select path variable that is already available 🡺 as a value paste the bin folder location path.
* By default IDE’s may have already maven installed but it’s a good practice to install maven locally as well, so that you can execute the maven based commands locally by using command prompt as well.
* To find out the version of maven trigger

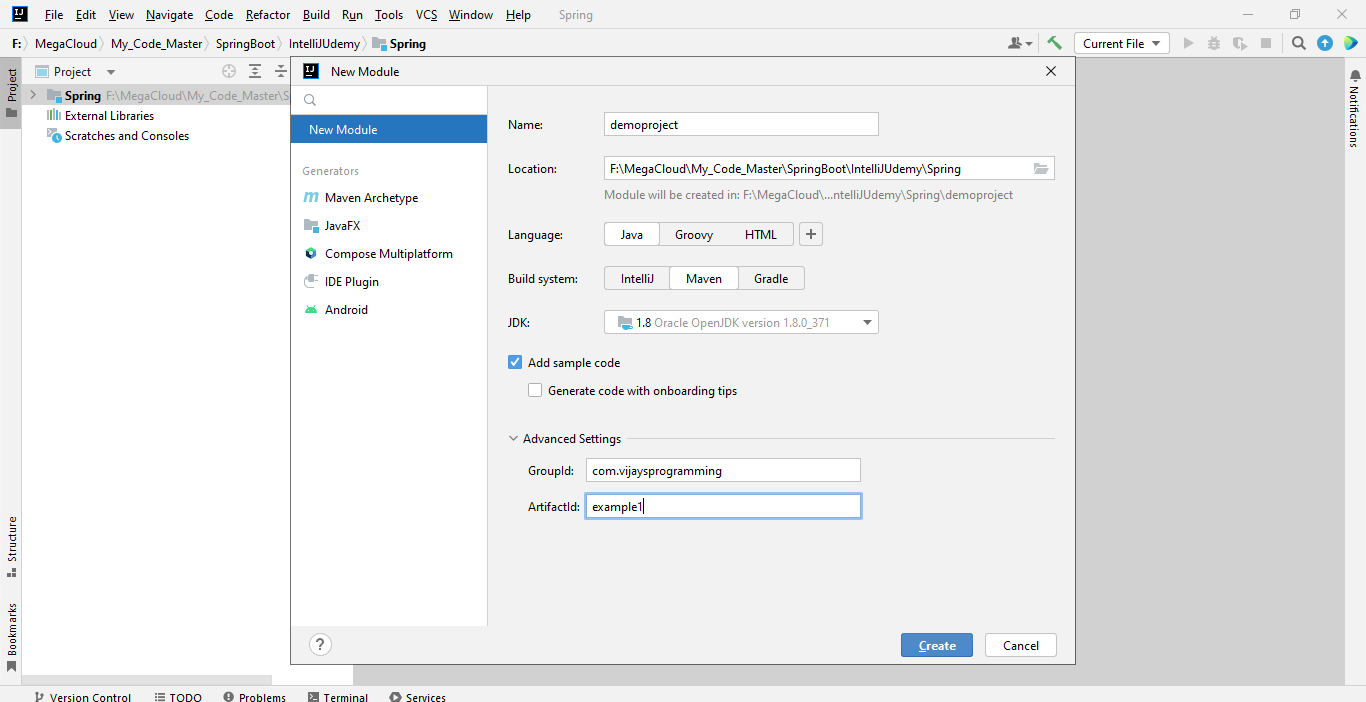


* Go to the intelliJ website and download community edition .exe or zip file

(.exe standalone installer and zip file for executable file)

* After installing intelliJ click on file 🡺 new 🡺 project 🡺 empty project 🡺 Give the name 🡺 Location 🡺 click on create
* Right click on the project 🡺 new 🡺 module 🡺 give the name as demoproject 🡺 language as Java and Build System as Maven 🡺 in Advanced settings create groupId ex: com.vijaysprogramming and create artifactID(by default the artifactID will be your module name) ex: example1
* artifactId is the name of the jar without version. If you created it then you can choose whatever name you want with lowercase letters and no strange symbols. If it's a third party jar you have to take the name of the jar as it's distributed. eg. maven, commons-math
* groupId will identify your project uniquely across all projects, so we need to enforce a naming schema. It has to follow the package name rules, what means that has to be at least as a domain name you control, and you can create as many subgroups as you want. Look at More information about package names. eg. org.apache.maven, org.apache.commons





**How enable word wrap in case of IntelliJ**

* press cntrl + shift + A
* search for soft wrap
* select the soft wrap these files option then the file extensions accordingly ex: \*.java; \*.xml; \*.properties;
* click on ok

**Quick Word on Maven**

**The steps to get the dependencies without using Maven**

* When building your Java Project, you may need additional JAR files, ex: Spring, Hibernate, Commons Logging, JSON etc…
* One approach is to download the JAR files from each project web site
* Manually add the JAR files to our build path / classpath

**The steps to get the dependencies using Maven**

* We can specify the projects that we are working with (dependencies) Spring, Hibernate etc…
* Maven will go out and download the JAR files for those projects for us and Maven will make those JAR files available during compile / run time.
* We can think Maven as our friendly helper.

**Spring Boot Initializr Demo**

* For Quickly creating the started Spring project go to <https://start.spring.io/>
* Select your dependencies.
* Create Maven/Gradle Project
* Import into IDE like Eclipse, IntelliJ, NetBeans etc..

**Development Process**

1. Configure out project at Spring Initializr website
2. Download the zip file
3. Unzip the file
4. Import the project into out IDE

**Note:** Whatever the name you give for name while creating the spring boot application that will be the class name.



It won’t take numbers in the beginning. If you are having the number simply class will be named as **Application**.

**Note:** To get the comments in application.properties file use #

**FIRST APPLICATION**

* Go to <https://start.spring.io/> under project select Maven
* Under Spring Boot select the latest version released. Avoid the “snapshot” versions since they are alpha/beta(testing stage)
* Under Project Metadata

Group - com.vijaysprogramming.demo

Artifact(project/application name) – myfirstapp

Name – myfirstapp

Description – Demo project for Spring Boot

Package name – com.vijaysprogramming.demo.myfirstapp

* Under packaging select Jar
* Under Java version select the Java version that is installed in your system for example 17
* Beside for the Dependencies click on add dependencies

(shortcut windows + B) select Spring Web and click on generate (shortcut windows + enter).

* Go to downloads and unzip the file.
* Copy the folder inside that zip file.
* Create a new folder in the Spring Boot folder name it as 1.dev-spring-boot and paste the folder that you have copied.
* Open the IntelliJ and select open and navigate to the folder where myfirstapp folder is there and select open.
* Click on check box and select trust the project.
* It will take some time to import the project.

Note: if you are getting Cannot resolve symbol 'String' then click on file 🡺 project structure 🡺 select the installed JDK path

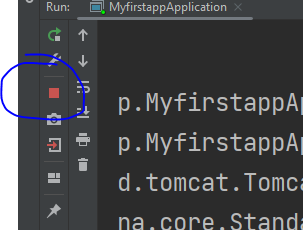
* Run it as a Java application.(cntr + shift + f10)
* Go to <http://localhost:8080/> and you should be getting this message



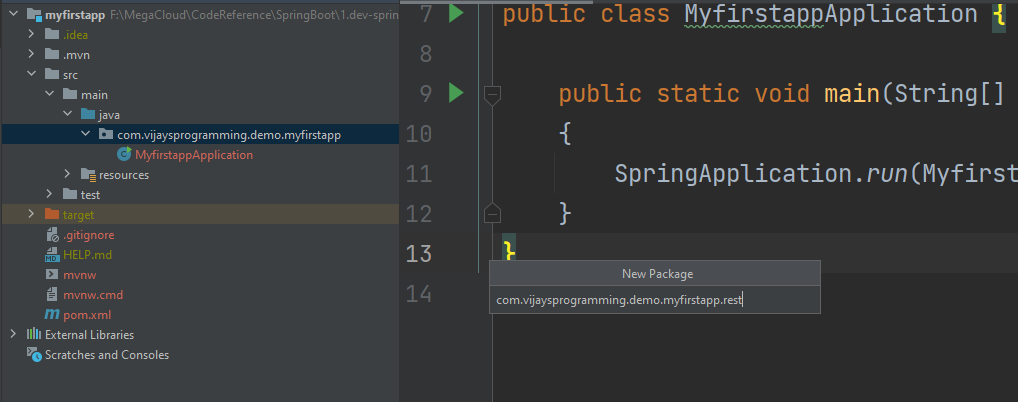
* Because we haven’t added any real code to our project yet no controllers, no view pages.

**Spring Boot – Create a REST Controller**

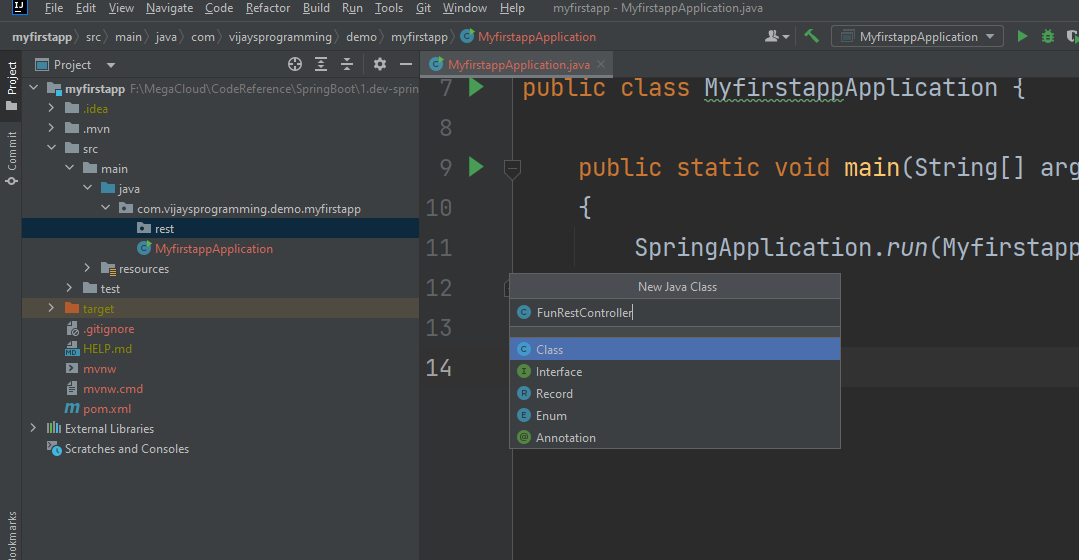
* Lets create a very simple REST Controller. Displaying Hello World on the web page.
* If the IDE is currently running then stop it by clicking on red button



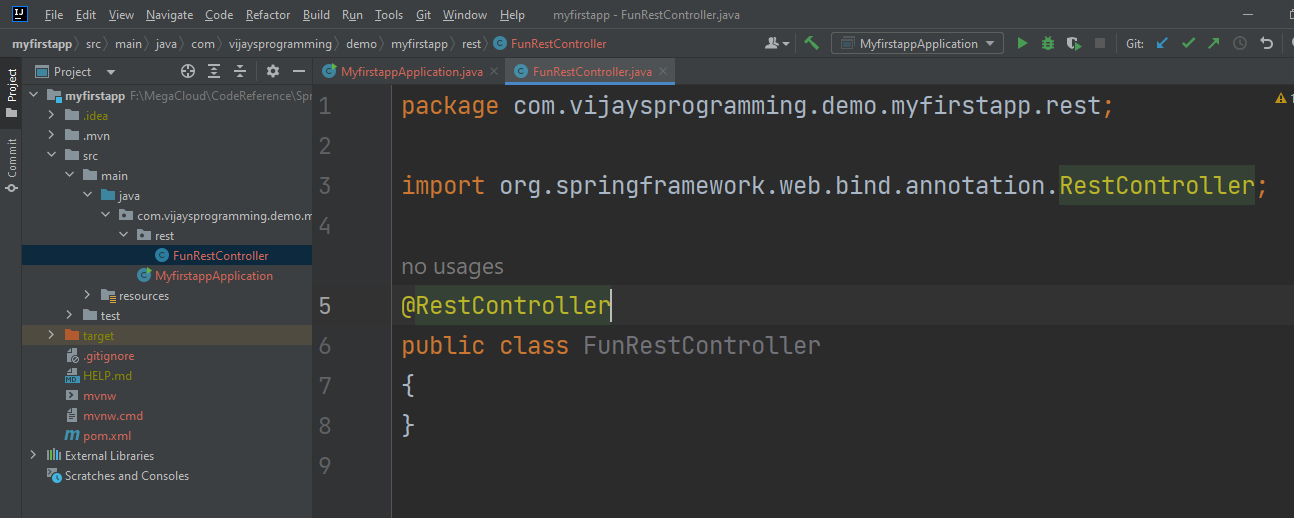
* Create a new package by right clicking on com.vijaysprogramming.demo.myfirstapp and select new package and type rest



* Create a new class called FunRestController inside the newly created package



* On top of public class FunRestController keep the annotation called @RestController



* Add this much of code

package com.vijaysprogramming.demo.myfirstapp.rest;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class FunRestController  
{  
 @GetMapping("/")  
 public String sayHello()  
 {  
 return "Hello World";  
 }  
}

* Go to myfirstapp and run as Java application.
* Go to the localhost 8080 and refresh you should see Hello World as the output

**SECOND APPLICATION**

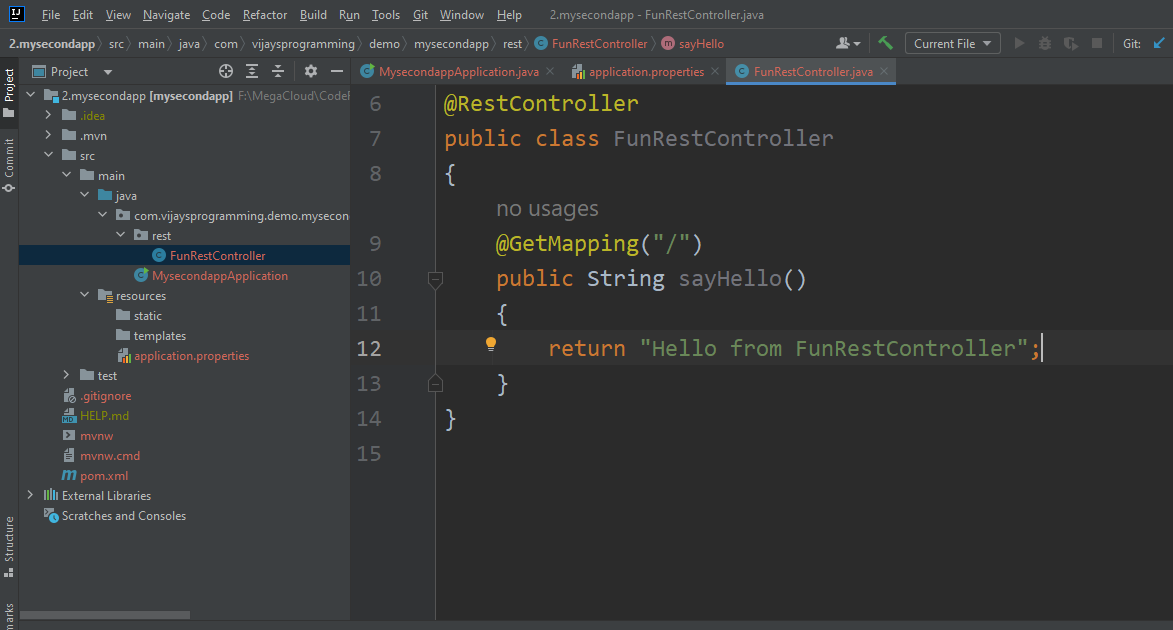
* Go to start.spring.io again and give the artifact name as mysecondapp and select dependency as spring web and click on generate.
* Extract the zip file and keep the mysecondapp folder inside the SpringBoot folder.
* Import that project inside the IDE
* Add one more print statement

package com.vijaysprogramming.demo.mysecondapp;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class MysecondappApplication  
{  
 public static void main(String[] args)  
 {  
 SpringApplication.*run*(MysecondappApplication.class, args);  
 System.*out*.println("My Second Spring Boot Application");  
 }  
}

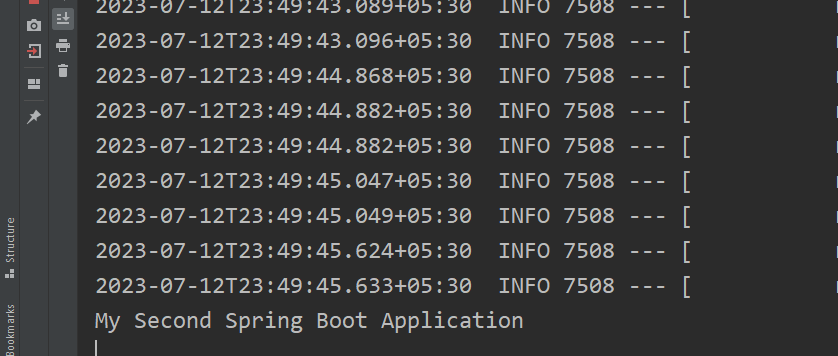
* Its recommended to change the port to other than 8080 because it might be occupied by some other application. To change the port number just expand resources folder and open application.properties and add this

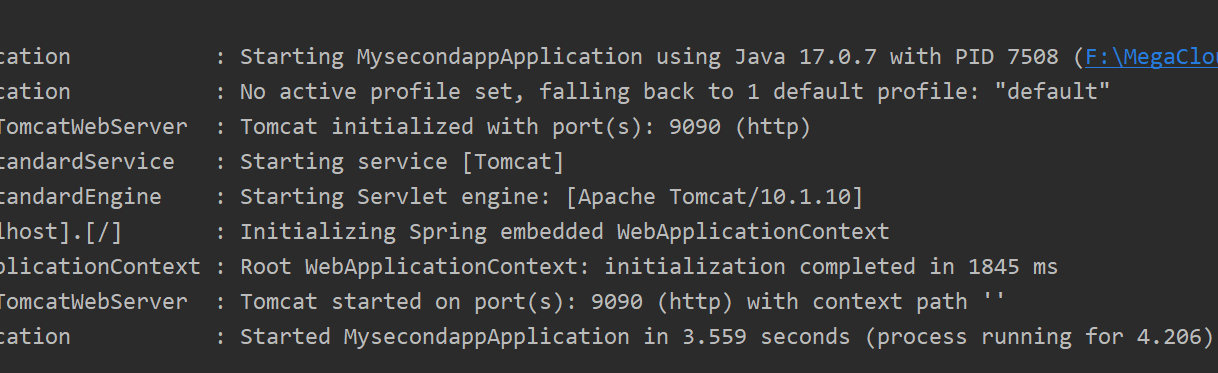
server.port=9090

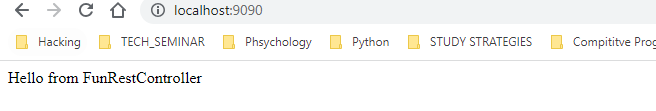
* Create another rest controller class like in the previous app and run the application and see the output in the console and also in the web browser by going to the address localhost:9090



* You should see this much of o/p







* Once the Spring Boot project is ready you can develop any number of controllers.

**THIRD APPLICATION**

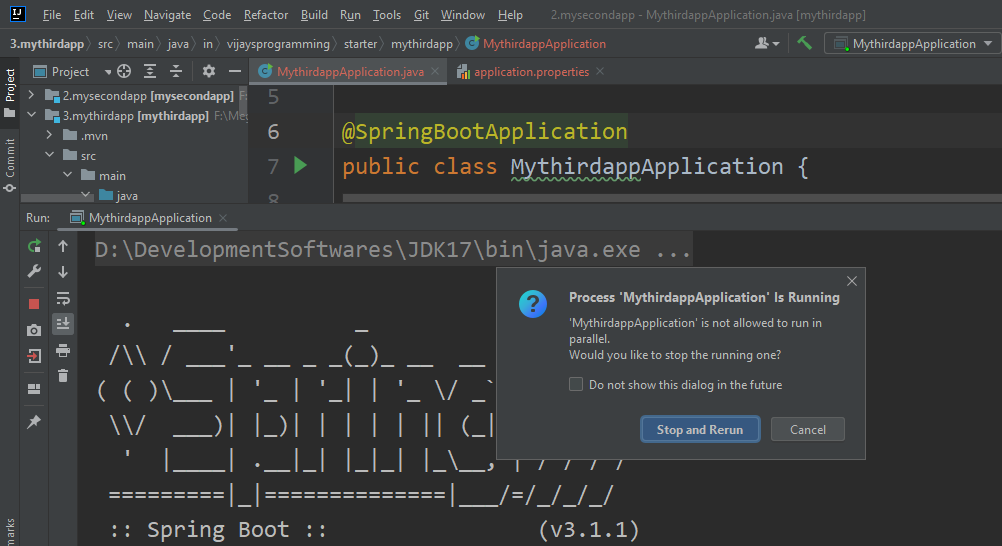
* You can choose same group name for the multiple projects. There is no issue.

Note: to import a new project to the same working directory just open file 🡺 project structure(cntrl + alt + shift + S) and select the project directory and import by selecting the second option(import project from external model) and select maven. Click on ok.

* Change the port number to 9090 and run the application.

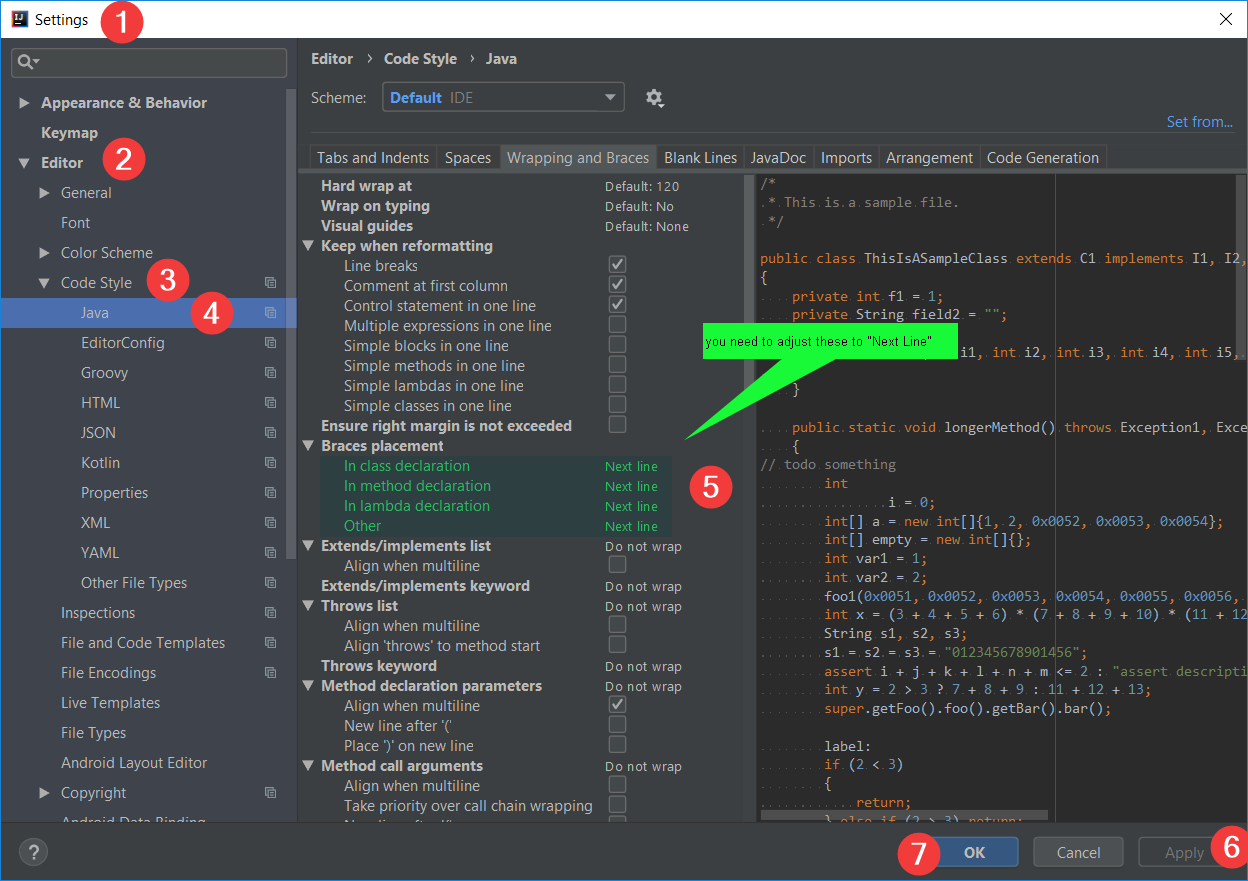
**Note:** if you are starting the application again without stopping the existing application then you will get an error.

**Note:** To avoid the error in IntelliJ you will be getting this notification if you try to run again while another application is already running. But in Eclipse you won’t get this.



* In eclipse you will be getting an error in the console called Web server failed to start. Port 9090 was already in use

**In IntelliJ to place the curly brace position to the next line**



Note: In IntelliJ IDEA, type sout and press the Enter or Tab button from your keyboard to generate System. out. println() automatically.

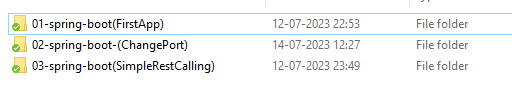
Note: if you are not setting the port number by default it will be considering port number 8080



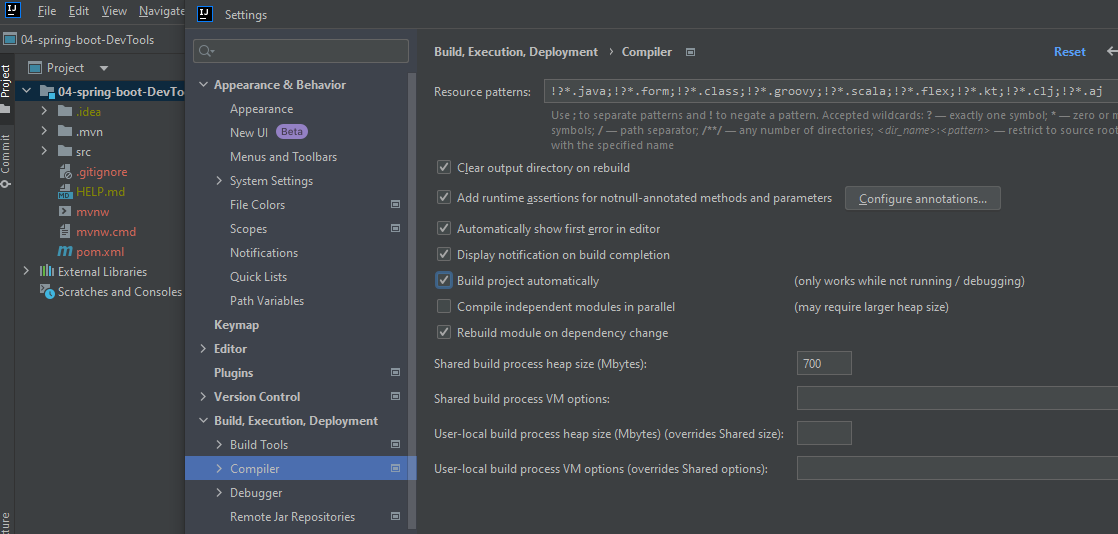
**Spring Boot Dev Tools**

* If you make some changes to your source code then you have to manually restart your application. To avoid this problem
* We have to use spring-boot-devtools
* After we configure spring-boot-devtools which will automatically restarts your application when code is updated
* In order to do this we need to add the dependency to our POM file.
* There is no need for writing additional code
* For IntelliJ we need some additional configurations. Because in IntelliJ community edition by default DevTools is supported.

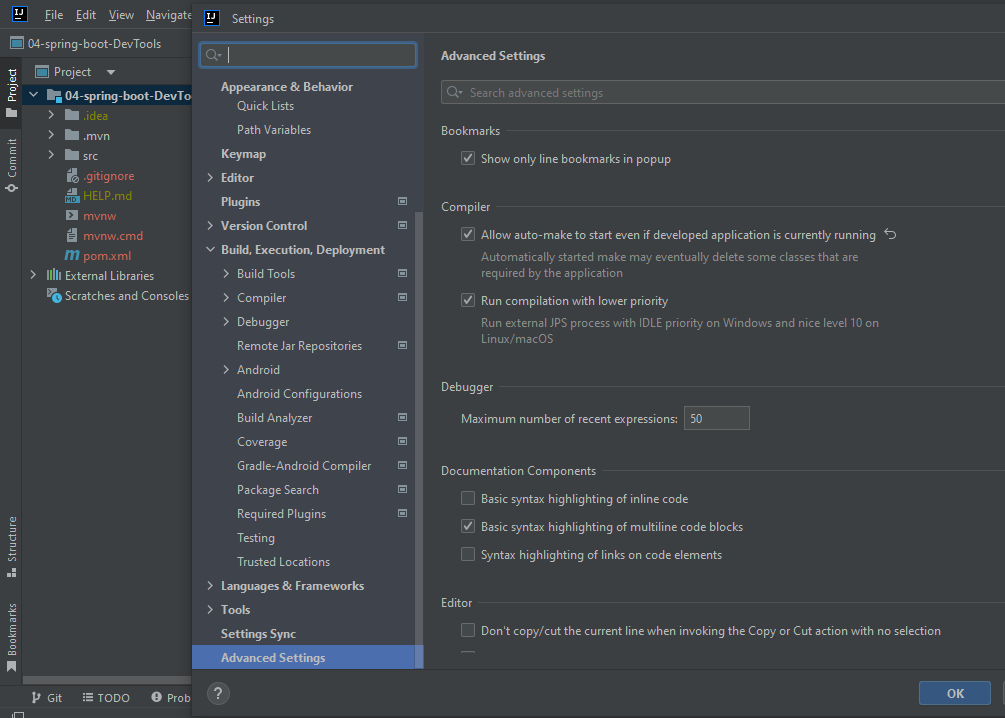
Note: make some changes in the folder names.



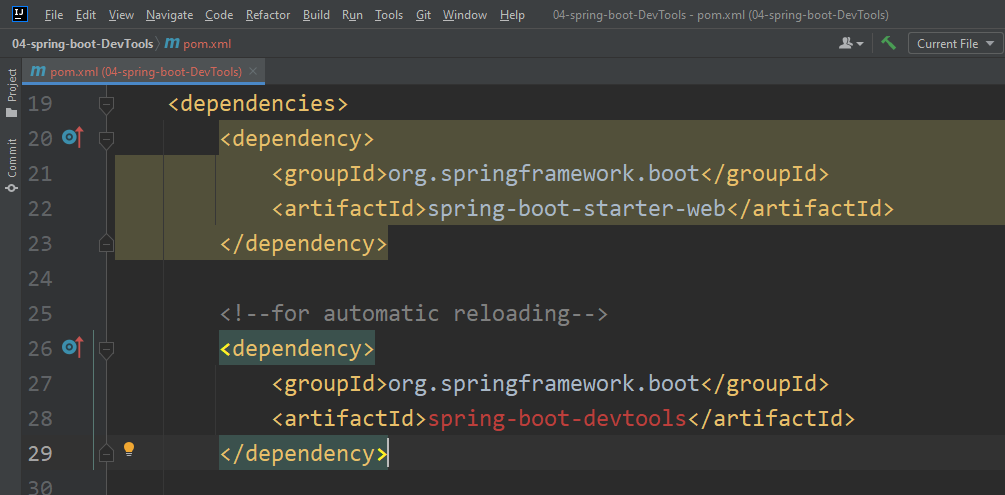
* Create another app 04-spring-boot-DevTools using start.spring.io
* Go to file 🡺 settings(cntrl + alt + S) 🡺 expand Build, Execution, Deployment 🡺 select compiler 🡺 select the checkbox build project automatically



* Go to Advanced Setting and Check Allow auto-make to start



* **Go to pom.xml file add spring DevTool dependency**
* **Before the closing of </dependencies> add this**



* **Be sure to reload the Maven changes by clicking on this icon**



* **Run the Application**
* **Create the rest controller like in the previous examples.**

**Create the package called rest, create the class named ApplicationRestController and create the method inside that called Authentication like this.**

package in.vijaysprogramming.springboot.springboot.DevTools.rest;

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

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

@RestController

public class ApplicationRestController

{

@GetMapping("/")

public String Authentication()

{

return "Authentication Succeded!";

}

}

* We can Expose any no. of end points for Ex:

**package in.vijaysprogramming.springboot.springboot.DevTools.rest;**

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

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

**@RestController**

**public class ApplicationRestController**

**{**

**@GetMapping("/")**

**public String Authentication()**

**{**

**return "Authentication Succeded!";**

**}**

**//new end point**

**@GetMapping("/Access")**

**public String APIAccess()**

**{**

**return "Access Granted!";**

**}**

**//new end point**

**@GetMapping("/Assembly")**

**public String Assemble()**

**{**

**return "Assembled!";**

**}**

**//new end point**

**@GetMapping("/Argument")**

**public String Argument()**

**{**

**return "Argument!";**

**}**

**}**

**For this scenario you will be getting error**

package in.vijaysprogramming.springboot.springbootSimpleRESTController.rest;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class MyRestController  
{  
 @GetMapping("/")  
 public String Authentication()  
 {  
 return "Authentication is Granted!!";  
 }  
  
 @GetMapping("/")  
 public String APIAccess()  
 {  
 return "APIAccess Granted!!";  
 }  
}

Cause:

Caused by: java.lang.IllegalStateException: Ambiguous mapping. Cannot map 'myRestController' method

in.vijaysprogramming.springboot.springbootSimpleRESTController.rest.MyRestController#APIAccess()

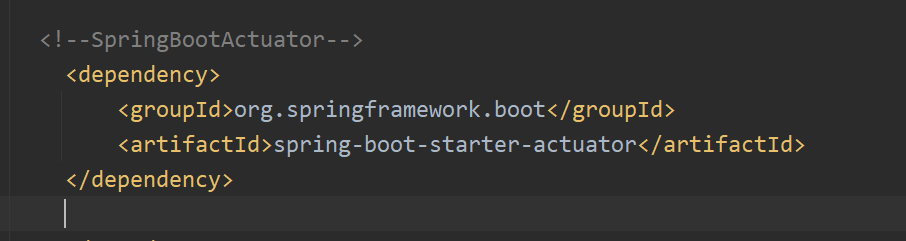
to {GET [/]}: There is already 'myRestController' bean method

**Spring Boot Actuator**

* Exposes endpoints to monitor and manage our application
* We can get DevOps functionality
* We Just need to add dependency to our POM file
* REST endpoints are automatically added to our application no need to write additional code. And we get new REST endpoints for free.
* We can check the application health
* We can access the application metrics

**Development**

* Create a project using spring starter name the project as 05-spring-boot-Actuator
* Edit pom.xml and add spring-boot-starter-actuator



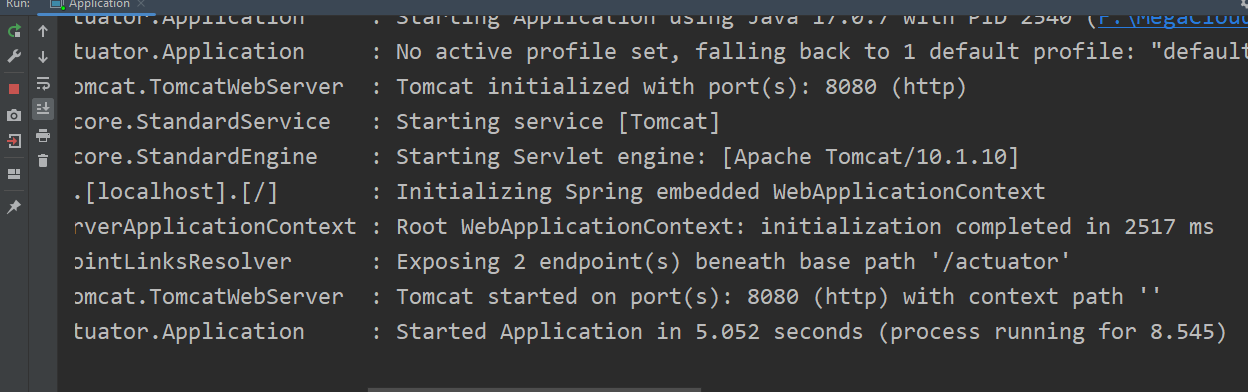
* By default only /health end point is exposed. /info provides more information about our application.
* Even though you are exposing info using comma separator it will be not exposed
* You need to provide additional instruction

management.info.env.enabled=true

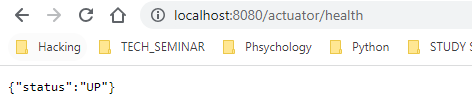
* To expose the end points add this to the application.properties file and add

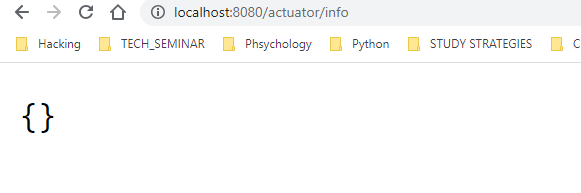
management.endpoints.web.exposure.include=health,info  
management.info.env.enabled=true

* Go ahead and execute the main application
* Take a look at the console for exposed endpoints



* Actuator endpoints are prefixed with /actuator
* In the browser you need to specify like this





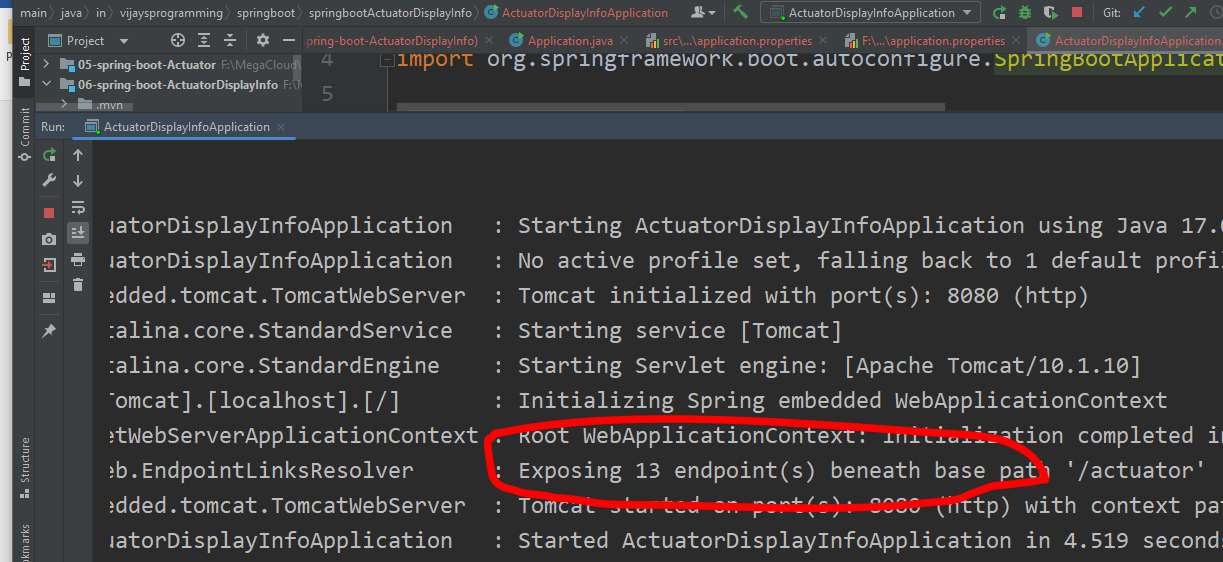
* By default the info actuator displays empty

**To Display some information about the application**

* Create a new app called 06-spring-boot-ActuatorDisplayInfo
* Add the actuator dependency
* To get the info about the application just enter this in the application.properties file

#use wildcard "\*" to expose all endpoints  
#can also expose individual endpoints with a comma-delimited list  
  
management.endpoints.web.exposure.include=\*  
management.info.env.enabled=true  
  
info.app.name=ActuatorExample  
info.app.description=TheWorkingsOfActuatorEndPoints  
info.app.version=1.0.0

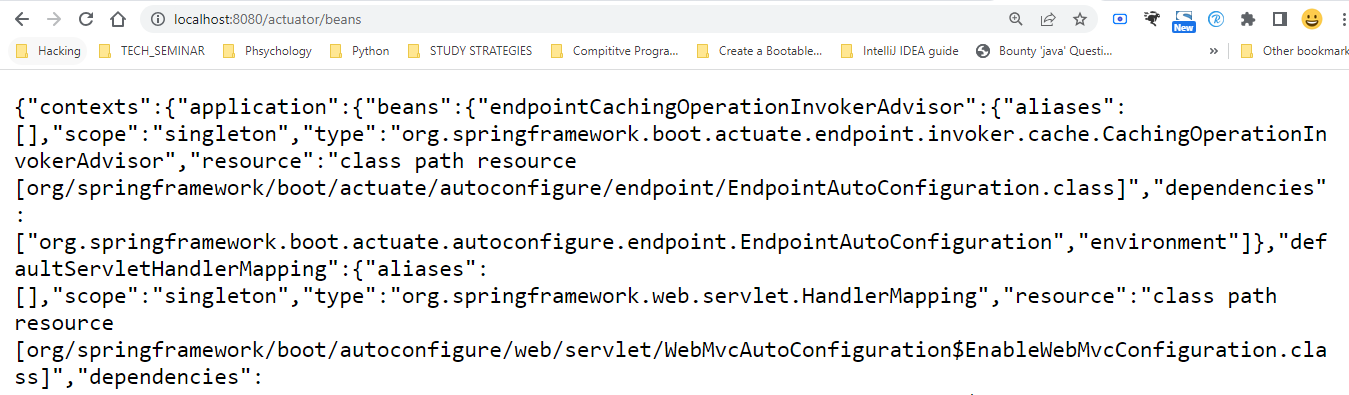
* Run the application
* If you look at the console you can see that more end points are exposed



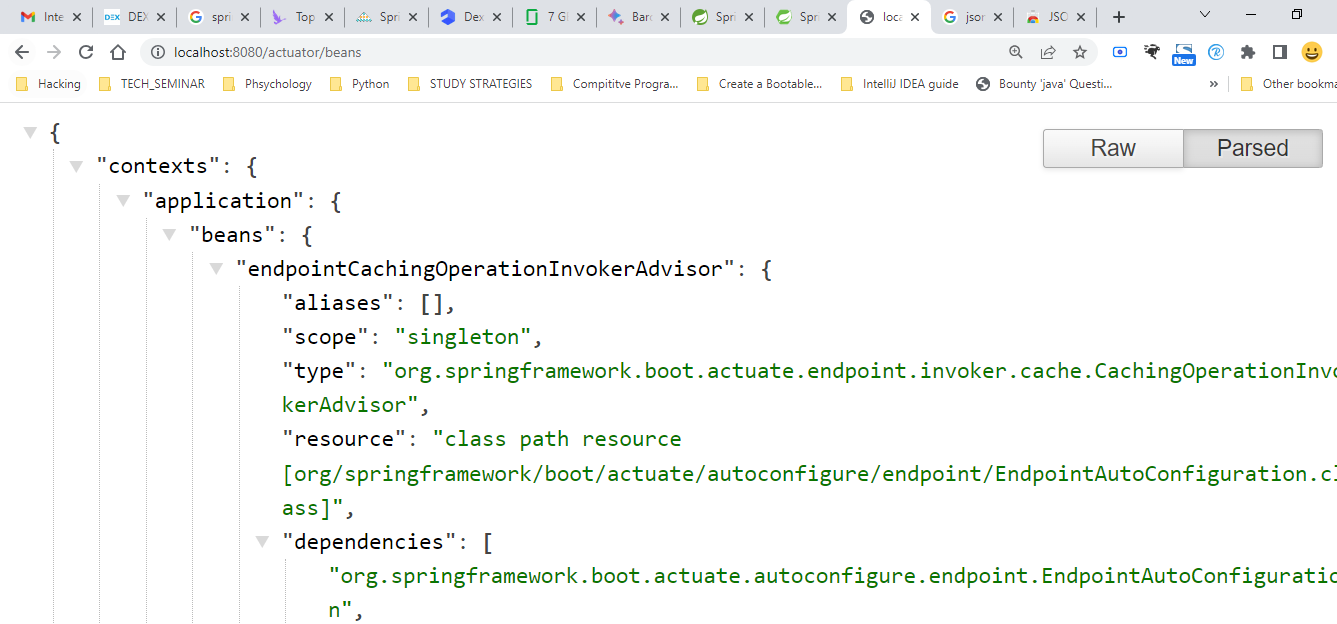
* Go to browser and call like this



* For displaying all the exposed endpoints we need to use /beans
* /beans represents List of all spring beans registered with our application. Spring boot internal beans and also our custom beans for our app

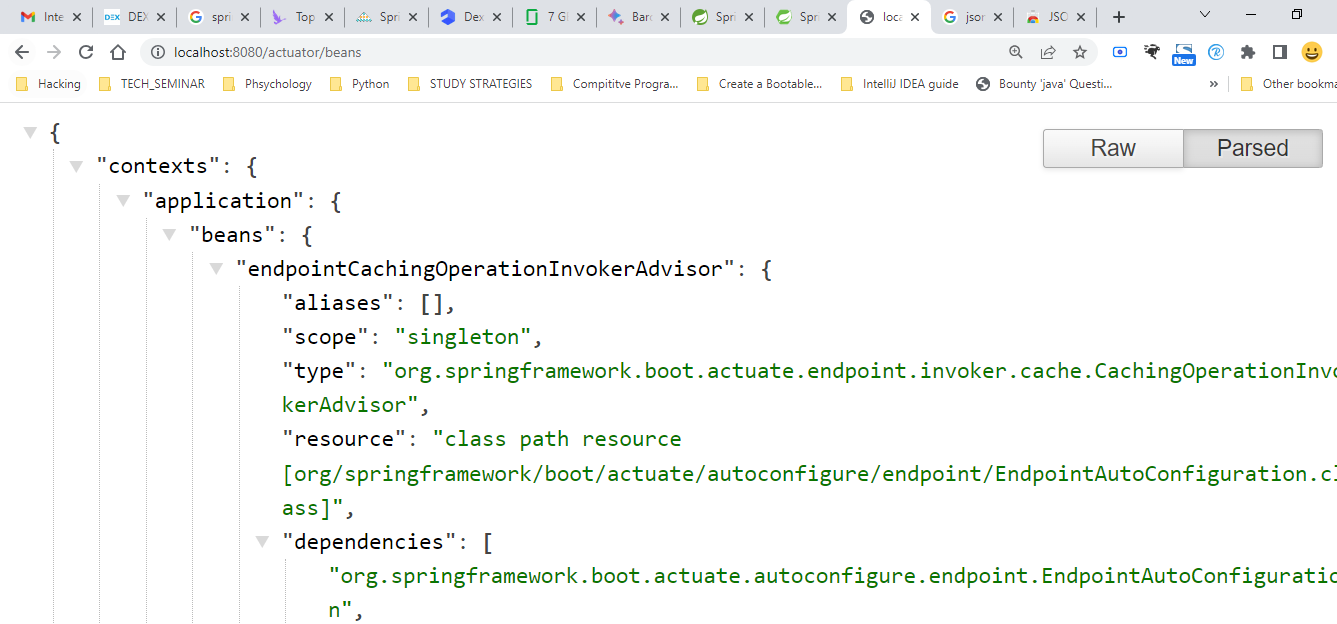


* To display all this exposed end point info content in the browser download and install the extension from <https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa>
* After installing the extension you will be getting the output like this

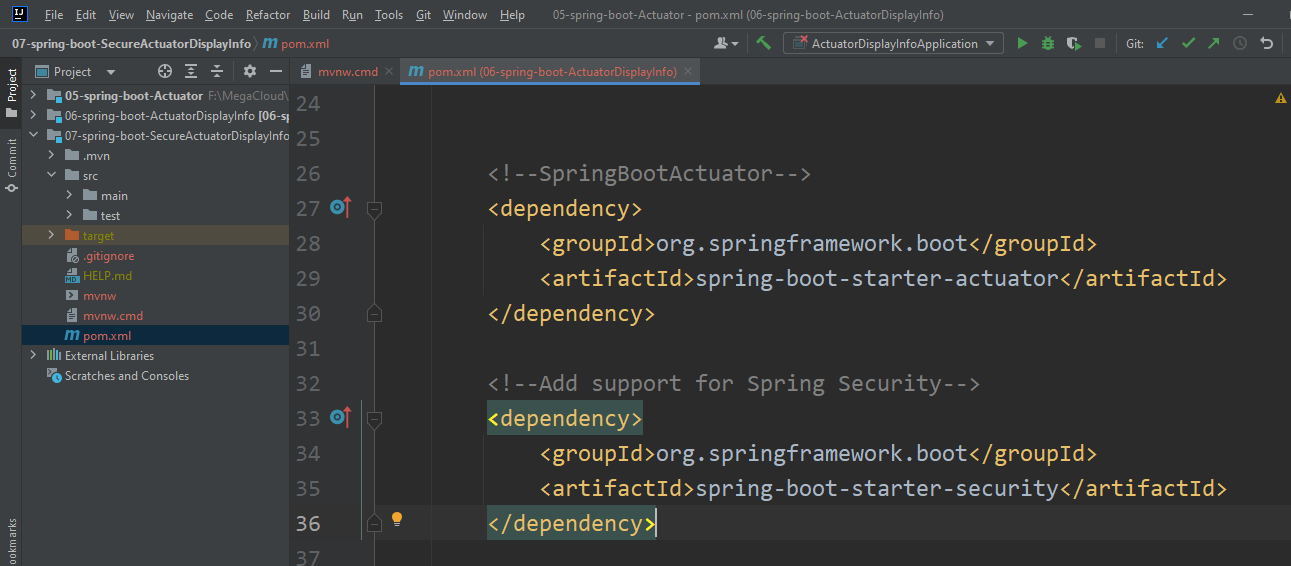


**Achieving security of the end point actuators**

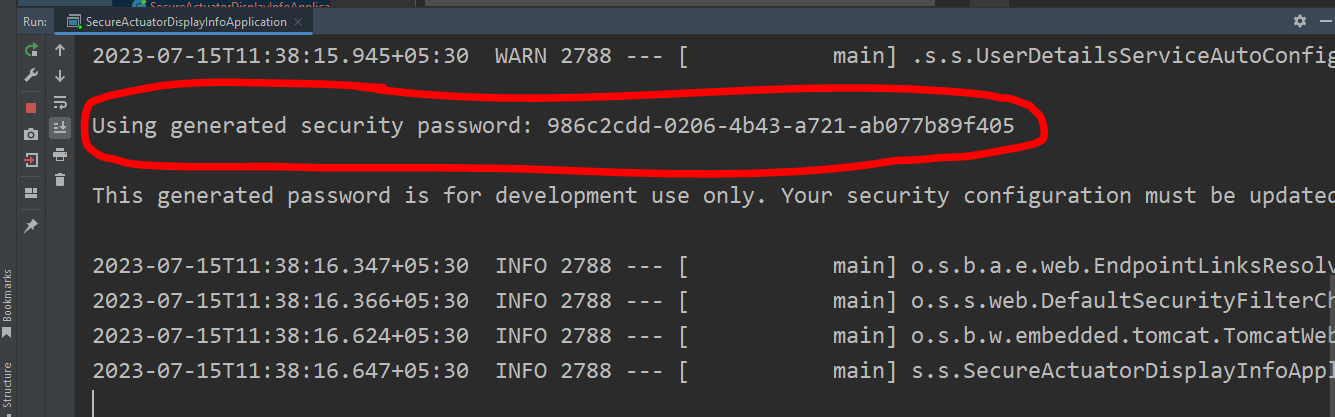
* We should not expose all of this information on the web because of the security issue.



* We need to make endpoints secured by adding security.
* Copy the previous project and rename it as 07-spring-boot-SecureActuatorDisplayInfo
* Edit pom.xml and add spring-boot-starter-security

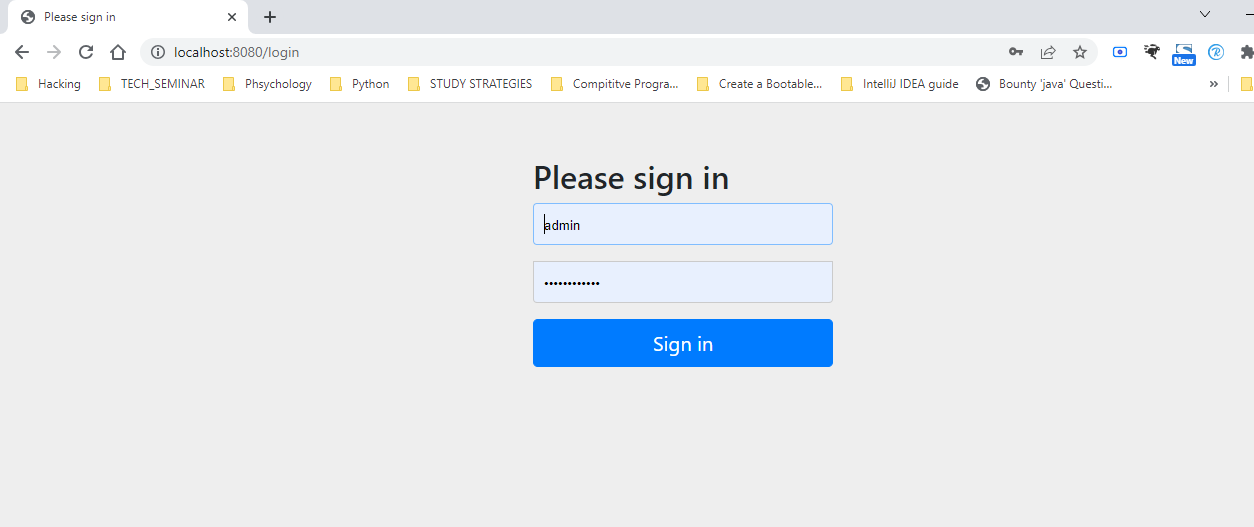


* It will be automatically secure REST end points
* Save the maven changes by clicking on maven icon
* Run the program

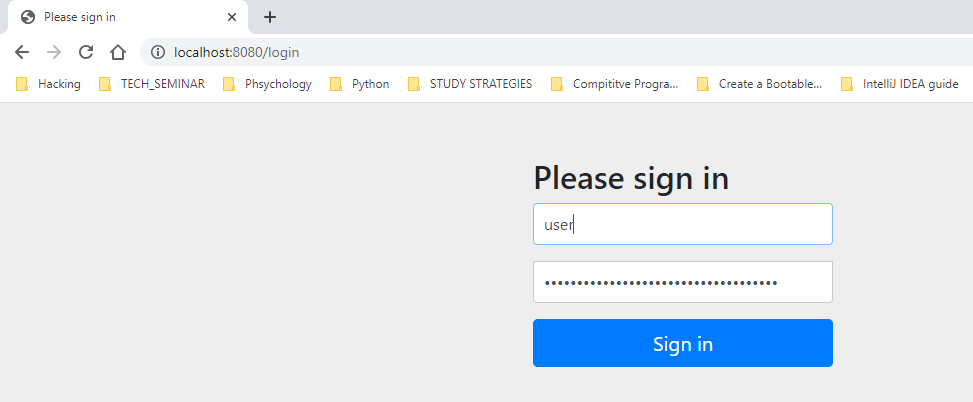


* Go to localhost:8080/actuator/mappings

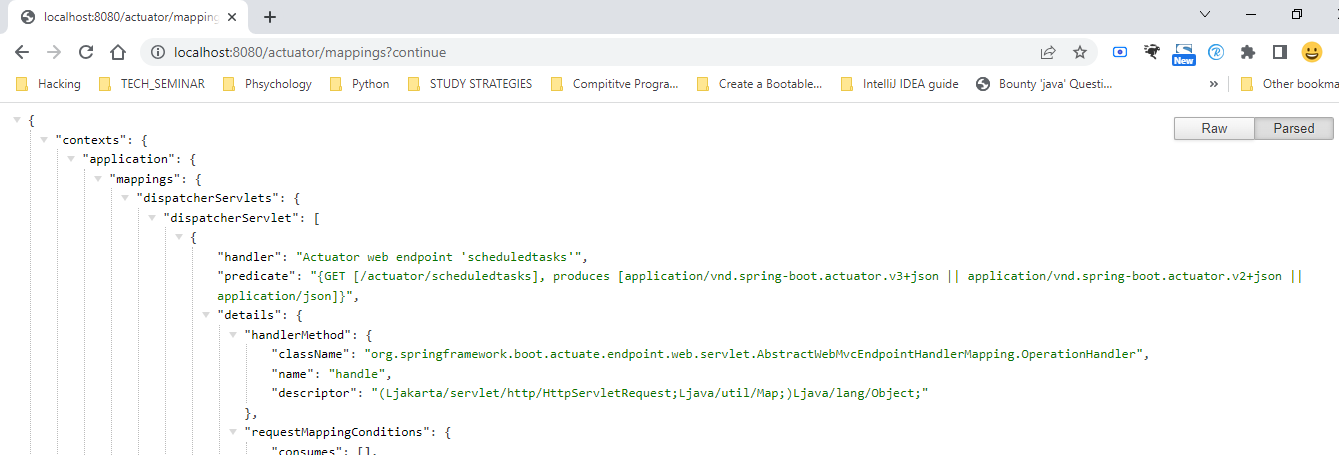
You will get like this



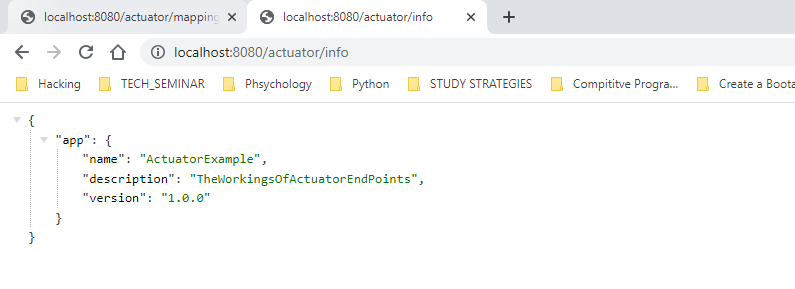
* Default user name is ‘user’ and the password is generated password on the console ‘986c2cdd-0206-4b43-a721-ab077b89f405’

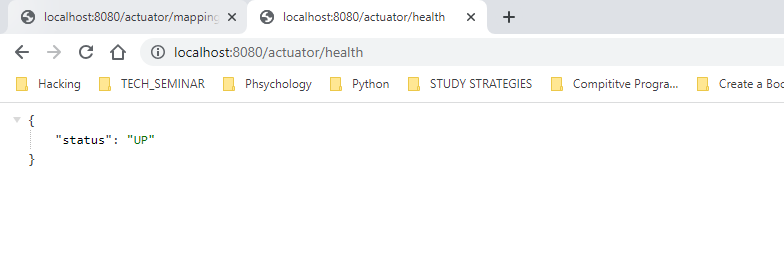


* Then you will get the info all the end points



* But still if you go to localhost:8080/actuator/info or localhost:8080/actuator/health still you can see





Because by default its available

* To disable those endpoints of /health and /info
* Go to xml file and add this

#use wildcard "\*" to expose all endpoints  
#can also expose individual endpoints with a comma-delimited list  
  
management.endpoints.web.exposure.include=\*  
management.info.env.enabled=true  
  
# Exclude individual endpoints with a comma-delimited list  
management.endpoints.web.exposure.exclude=health,info  
  
info.app.name=ActuatorExample  
info.app.description=TheWorkingsOfActuatorEndPoints  
info.app.version=1.0.0

Note: if you stop and rerun the application again you will be getting login page if you try to access any end points like /health or /info

Note: each time when you run your application using security a separate password will be generated each time

* Verify:You will get whitelabel error if you try accessing /health and /info because we have disabled both

**RUN SPRING BOOT APPLICATION FROM COMMAND LINE**

* There is no need for IDE while running the spring boot application
* Since we using spring boot, the server is embedded in our JAR file and there is no need to have separate server installed/running because spring boot apps are self-contained.
* Two options for running the app

1. Use java –jar ex: java –jar myapp.jar

java 🡺 starts our app

-jar 🡺 also start embedded server(TomCat)

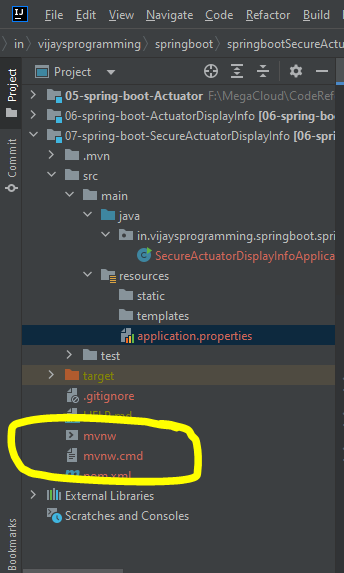
myapp.jar 🡺 name of our JAR file

1. Use spring boot maven plugin mvnw spring-boot:run

* mvnw allows us to run a Maven project
* No need to have Maven installed or present on your path
* If correct version of Maven is NOT found on our computer

Then Automatically downloads correct version and runs Maven

* Two files are provided
  + mvnw.cmd for MS windows (mvnw clean compile test)
  + mvnw.sh for Linux/Mac (./mvnw clean compile test)



Note: if you already installed Maven then you can ignore/delete the mvnw files just use Maven normally like **mvn clean compile test**

Note: in the pom.xml spring boot is already generated this dependency

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

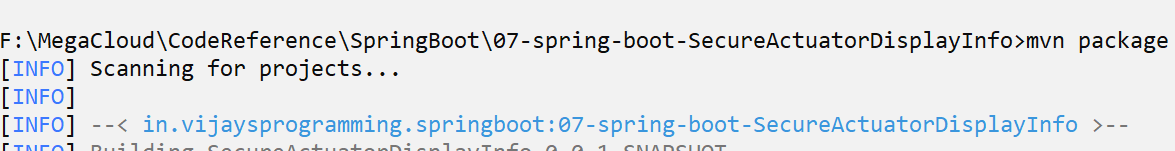
This will be used for packaging executable jar or war archive and we can also run the app easily

Ex:

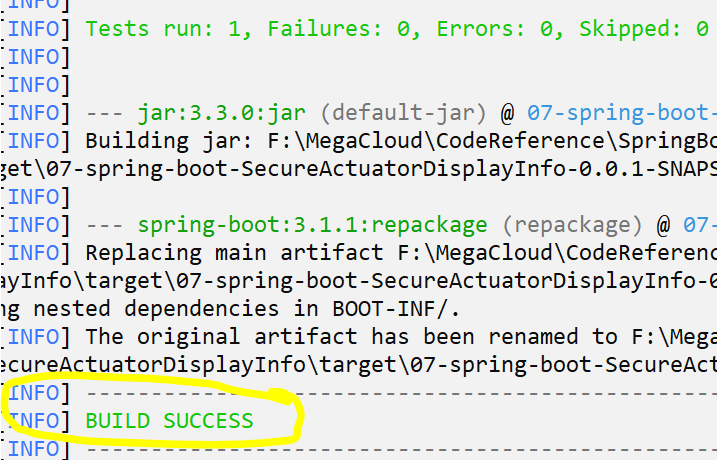
mvn package

mvn spring-boot:run

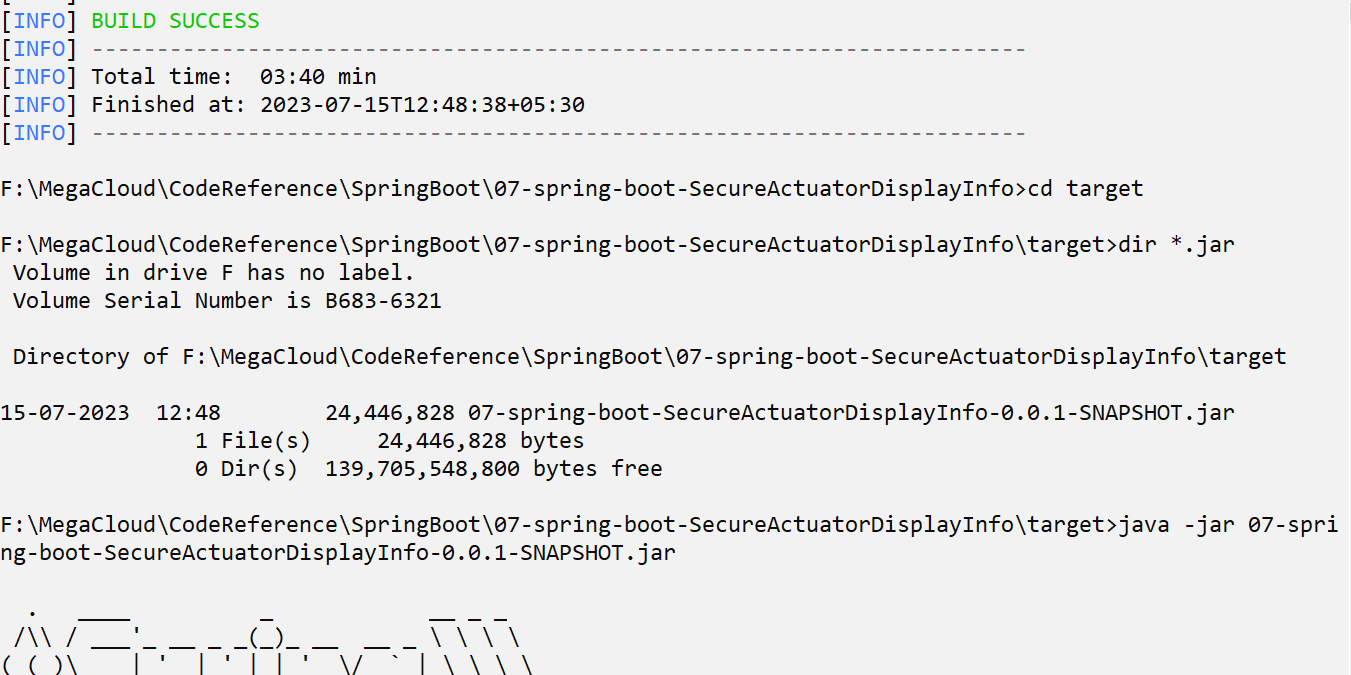
* To run the project go to the root of your project



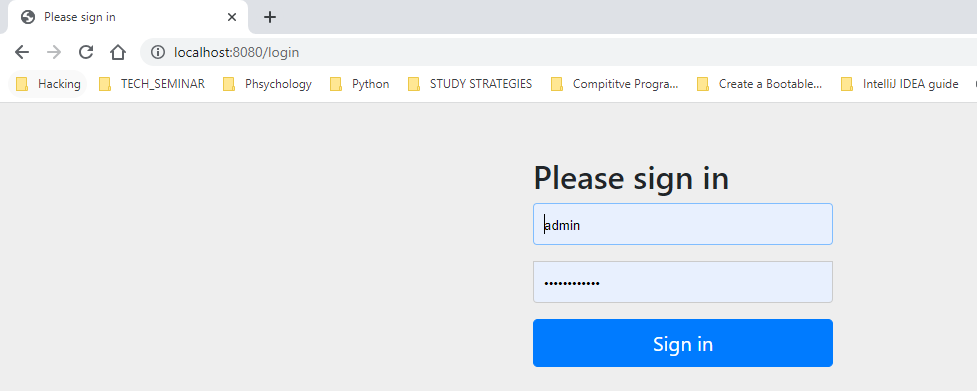
* You should get



* After this run the following commands



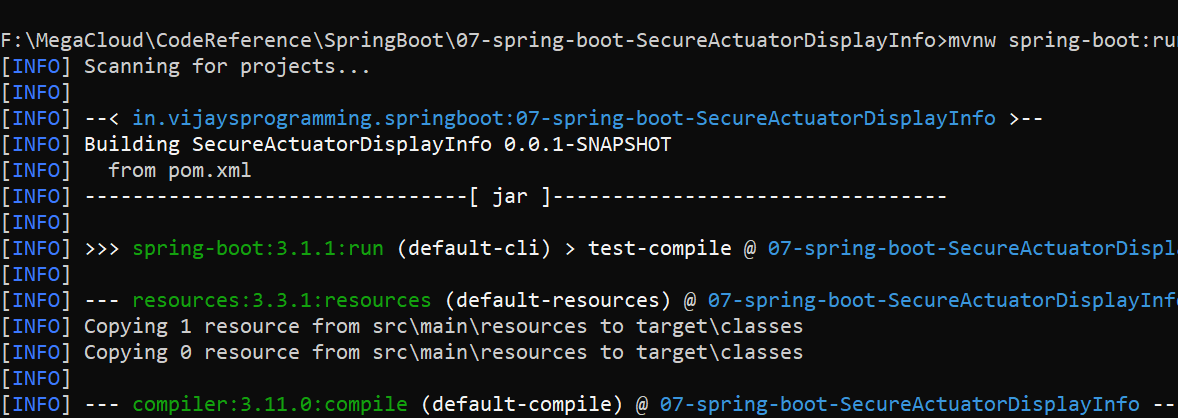
* Go to localhost:8080 for ensuring app is running



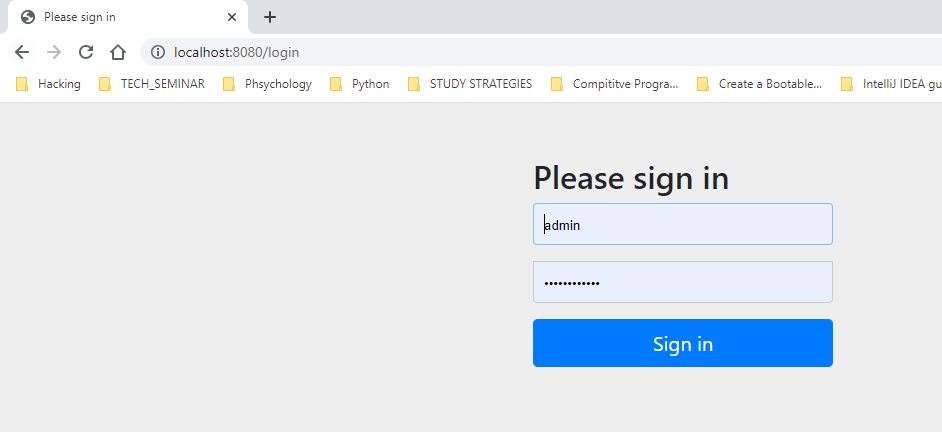
* To stop the application cntrl + c
* Running app using spring boot Maven plugin

Note: you should be having JAVA\_HOME set for this

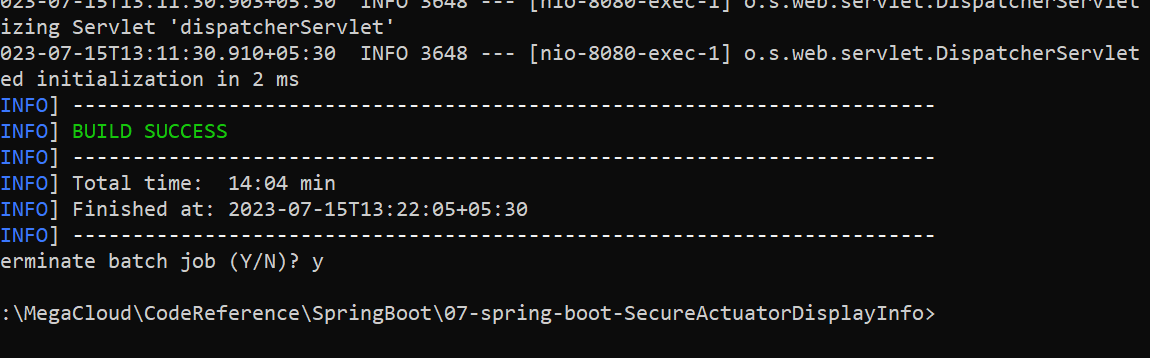
* Go to the root of the project so trigger cd ..
* And trigger mvnw spring-boot:run



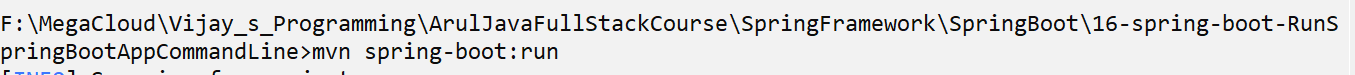
* Verify the project is running



* Cntrl + c



Note: you can also use



**SPRING BOOT CUSTOM APPLICATION PROPERTIES**

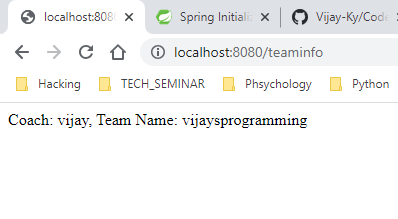
* Our application should be configurable. No hard-coding of values
* We should be able to read app configuration from a properties file.
* By default spring boot reads information from a standard properties file located at: src/main/resources/application.properties
* We can define any custom properties in this file
* Our spring boot app can access properties using @Value
* No additional coding or configuration required.

**DevelopmentProcess**

* Create a new app called 08-spring-boot-CustomApplicationProperties
* Edit application.properties file and this content

*#define our own custom properties  
#we can give any custom property names we wish*coach.name=vijay  
team.name=vijaysprogramming

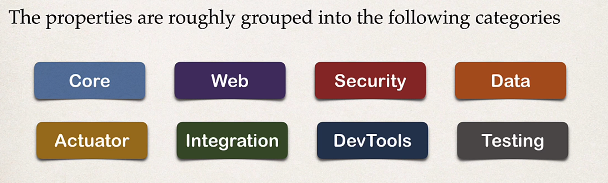
* Inject properties into spring boot application using @Value
* Create REST controller
* we are injecting values of application property to this private members.
* Expose new end point for the ‘teaminfo’ with method called getTeamInfo() FromLineNo 18 to 22
* Run the application and go to local host



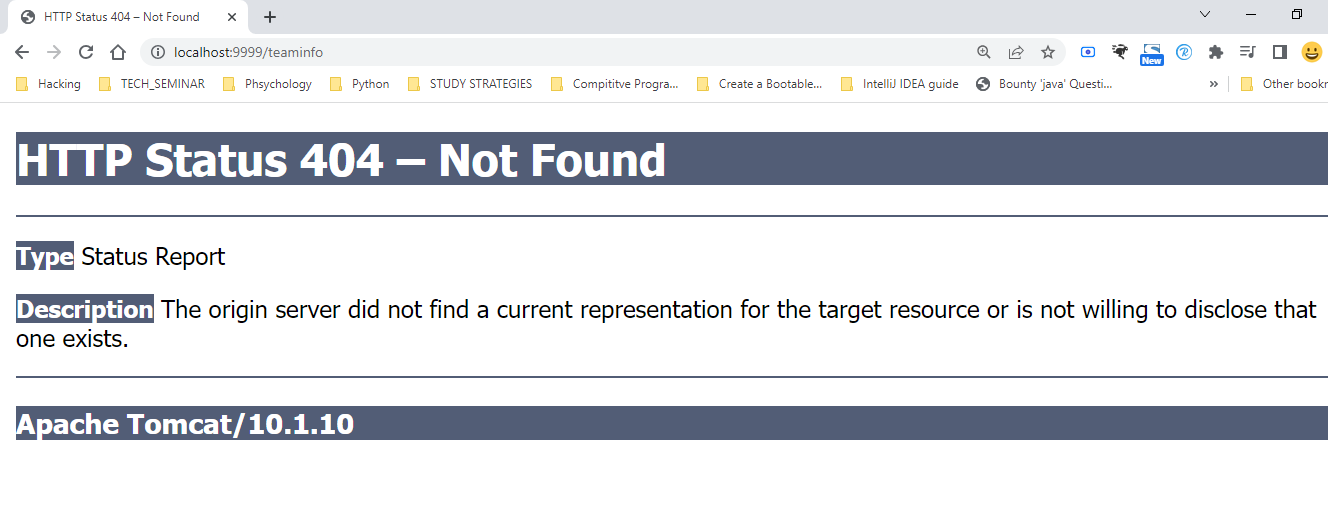
**Configuring the spring boot server**

**Spring Boot Properties**

* Spring boot can be configured in the application.properties file
* Some of the properties that we can set is Server port, context path, actuator, security etc..
* Spring boot has over 1000+ properties.



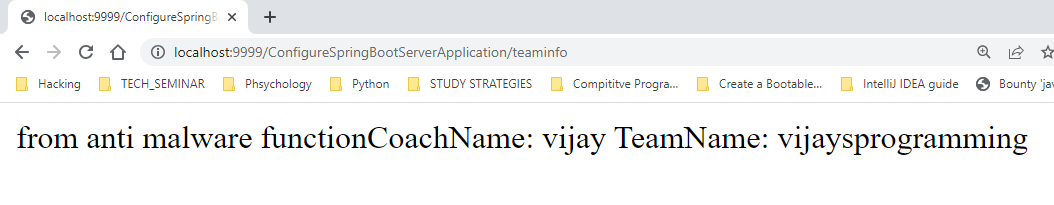
* Refere [ThisLink](https://github.com/Vijay-Ky/VijaySirJavaNotesAndExamples/blob/main/SpringBootExamples/09-spring-boot-ConfigureSpringBootServer/src/main/resources/application.properties) LineNo1. Indicates that we can able to change the port number. Default is 8080
* LineNo27 indicates that setting the context path of the application. i,e we are making All requests should be prefixed with /ConfigureSpringBootServerApplication
* If you go to localhost:9999/teaminfo you will get this error

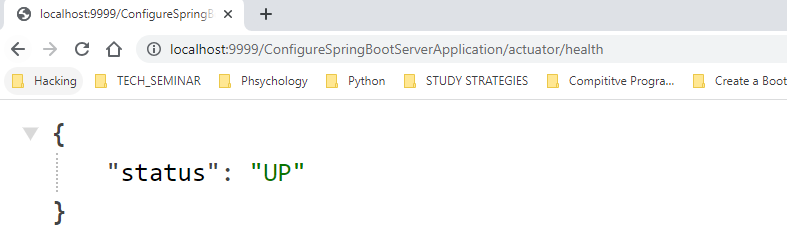


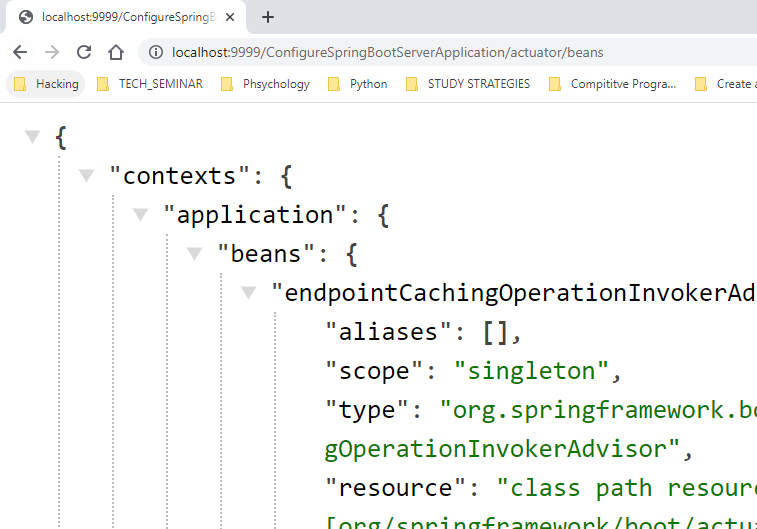
* You can fix this by prefixing the context path

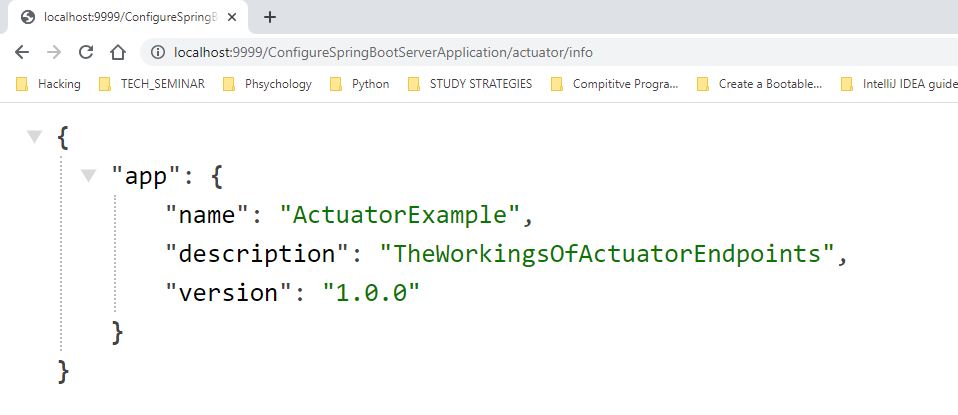
localhost:9999/ConfigureSpringBootServerApplication/teaminfo

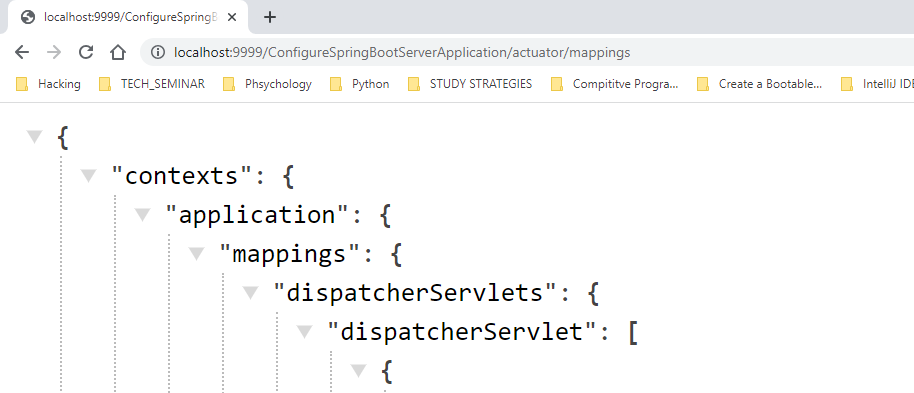
* You need to access all the resources by prefixing context path











⇒ if you are setting the server.port to 0000 then by default tomcat will be starting at random port number which is available.

**Spring boot inversion of control(IoC)**

* The approach of outsourcing the construction and management of objects.

**Spring boot dependency injection(DI)**

* Spring container “injects” objects into other objects or “dependencies”
* The dependency injection is a type of design pattern. That removes the dependency of the programs.
* DI is a process whereby objects define their dependencies.
* DI is the ability of an object to supply dependencies of another object.
* In spring DI is achieved through the Inversion of Control container. The container manages the creation and lifecycle of objects and injects the required dependencies into the object when it is created.
* The client delegates to another object the responsibility of providing its dependencies.

**@Component annotation**

* @Component marks the class as a Spring Bean
* A Spring Bean is just a regular Java class that is managed by Spring
* @Component also makes the bean available for dependency injection.

**@Autowired Annotation**

⇒ is optional if the RestController class contains only one constructor.

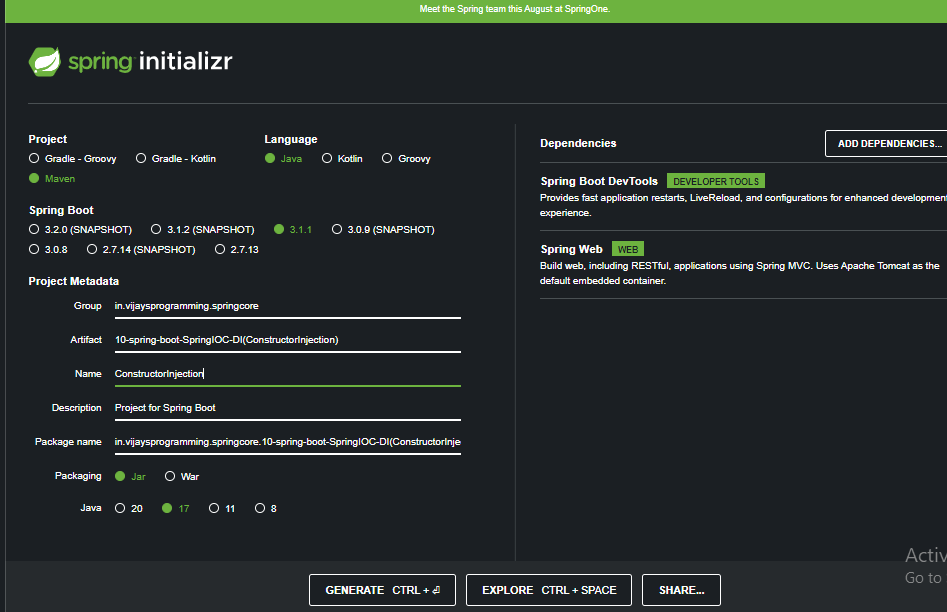
⇒ @Autowired annotation is used for automatic dependency injection

⇒ @Autowired annotation is used to inject the bean automatically.

⇒ @Autowired helps us inject a bean into the target bean.

**Development**

* Go to start.spring.io add the dependency Spring Boot DevTools and Spring web and generate the project



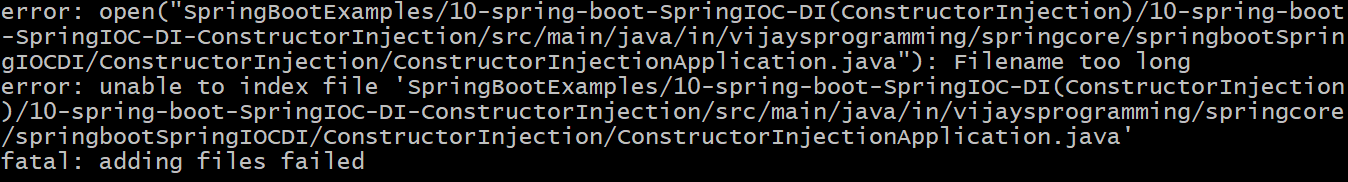
**Note:** if you are getting path too long error while extracting the zip folder then use [7zip](https://www.7-zip.org/download.html) for extracting. And if you are getting path too long again while copying to the development folder then copy the zip file to the development folder and extract there itself.

**Note:** even you if you keep *management.endpoints.web.exposure.include=health,info  
management.info.env.enabled=true*

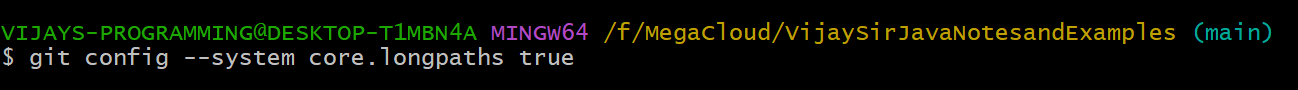
Info will be still not showing the values you need to enable it by using \* wild card character

* Create a new interface called Trainer

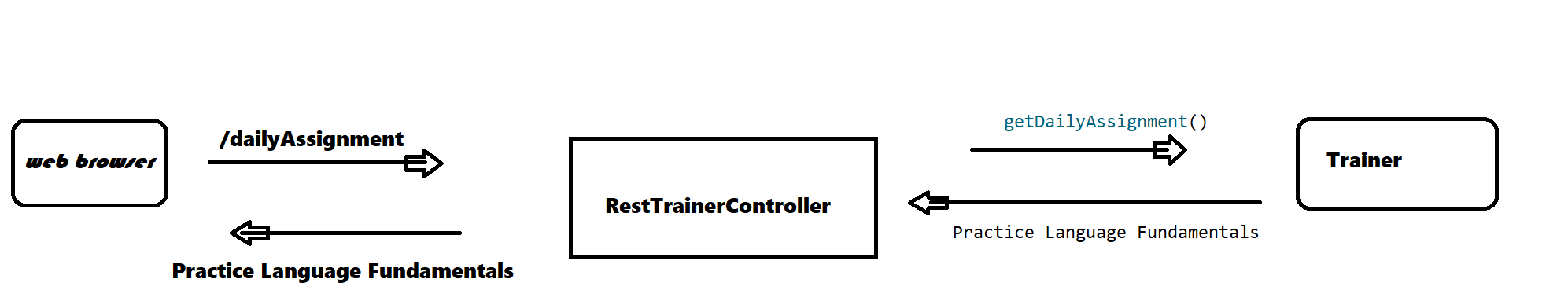
Note: if you are getting this error while adding in git bash



trigger this command



* create a new Trainer interface.
* create a new JavaTrainer class.
* JavaTrainer implements Trainer and implement method
* On top of the class give @Component that indicates that this class as a spring bean
* Refer [ThisLink](https://github.com/Vijay-Ky/VijaySirJavaNotesAndExamples/blob/main/SpringBootExamples/10-spring-boot-SpringIOC-DI(ConstructorInjection)/10-spring-boot-SpringIOC-DI-ConstructorInjection/src/main/java/in/vijaysprogramming/springcore/springbootSpringIOCDI/ConstructorInjection/RestTrainerController/RestTrainerController.java) create the class.
* LineNo.12 @Autowired annotation tells Spring to inject a dependency.
* LineNo.12 to 16 constructor and we are injecting
* LineNo.19 to 23 Write the mapping and method.
* Run The program



Note: if you are getting “no usage” that’s because spring framework is dynamic. IDE may not be able to determine if a given class/method is used at runtime