www.youtube.com/c/javaexpress
+91 8555929285
Java Express

Spring Boot Material 2022

Spring Introduction:

- It is very popular framework *for* building java applications
- It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.
- Spring is alternative framework for EJB Technology (For simplifying the problems of EJB they introduced Spring).
- Spring is light weight and opensource framework for building java applications.
    - Reason: In EJB components are heavy weight because it is forcibly to extend a class or interface for predefined API.
- It provides support to various frameworks such as Struts, Hibernate, EJB, JSF, etc.
- The main feature of Spring is Dependency Injection & IOC Container.
- To achieve loose coupling between different components by implementing dependency injection.
- Spring can be used for the development of particular layer of a real time application unlike struts [only for front end related] and hibernate [only for database related], but with spring we can develop all layers.
- Testing is very simple.
- Provides a large number of helper classes. makes things easier
- **If we use SF to develop project, we need to manage all the configurations required for that project**.
- Official Website is www.spring.io

Goals of Spring:

- Lightweight development with Java POJOS
- Dependency Injection to promote loose coupling
- Declarative programming with AOP (Aspect Oriented Programming)
    - Add our application services to given objects
- Minimize boilerplate Java Code

Spring Projects:

- Additional Spring modules built-on top of the Spring Framework
- Only use what you need
    - Spring Cloud, Spring Data
    - Spring Batch, Spring Security
    - Spring for Android, Spring Web Flow
    - Spring Web Services, Spring LDAP
- Spring | Projects

Types of Applications:

1. Standalone Application
2. Web Application

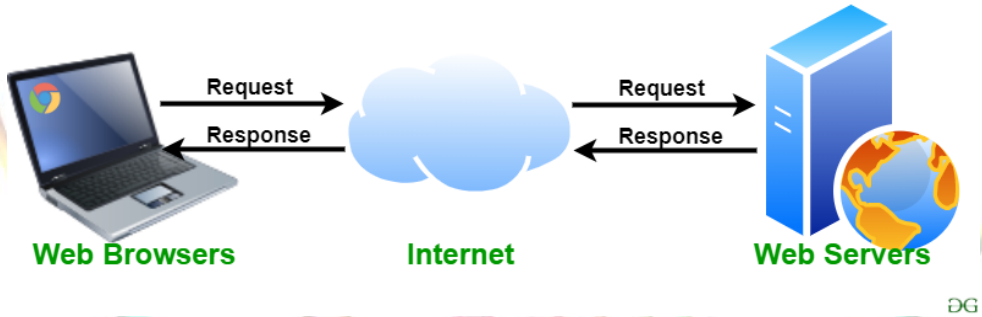
3. Distributed Application or Enterprise application

Standalone Application:

- The application which are installed in one computer and performs actions in the same computer are knows as standalone applications or Desktop Based Applications or Windows Applications.
- Note:
- According to programmer, Standalone applications means
    - No HTML input
    - No Server Environment
    - No Database Storage

Web Application:

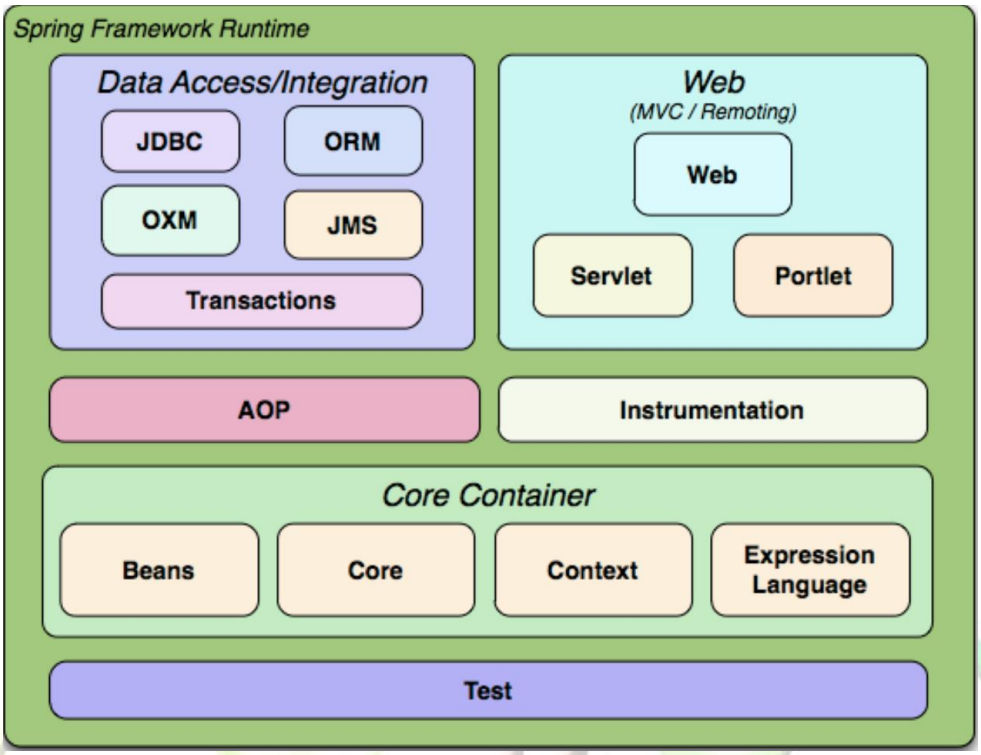
- A web application (or web app) is application software that runs on a web server
- Web applications are accessed by the user through a web browser with an active network connection.
- Web applications are meant for C 2 B (Customer to Business)
- End users will interact with web applications directly
- Web applications include online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email programs
    - Shopping Websites
    - Facebook
    - Gmail



Distributed Application:

- The application which is running in distributed environment and depending on the features like Security, Load Balancing & Clustering is known as Enterprise application or
- A distributed application is **a program that runs on more than one computer and communicates through a network**
- The application which is interacting with another application is called as Distributed application
- To develop distributed applications, we will use Webservices (spring web mvc module)
- Distributed applications are meant for B 2 B
    - Flipkart ----> Payment Integration
    - Zomato -----> Banking apps
    - MakeMyTrip ---> Airlines apps

Spring Modules:



Spring Modules:

- The **Core Container** consists of the Core, Beans, Context and Expression modules.
- **The Core and Beans modules** provide the most fundamental parts of the framework and provides the IoC and Dependency Injection features.
- The **Context** module build on the solid base provided by the Core and Beans modules, This module supports internationalization (I18N), EJB, JMS, Basic Remoting.
- **Spring DAO/Spring JDBC module** provides abstraction layer on plain JDBC to develop persistence logic.
- Spring also provides support to **ORM solutio**ns, and it provides integration with ORM tools for easy persistence of POJO objects in relational databases. such as Hibernate, JPA, OpenJPA, TopLink, iBATIS, and so on.
- **Spring AOP** module is given to apply aspects on Spring Applications
- This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.
- Web group comprises of **Web-MVC, Web-Sockets**. These modules provide support to create web application.

SpringBoot Environment Setup:

- JDK Software download from Oracle Website
    - Spring 5 requires Java 8 or higher
- Java Integrated Development Environment (IDE)
    - Eclipse
    - IntelliJ
    - STS (Spring Tool Suite)

Without SpringBoot the problems are:

- We need to hunt for all the compatible libraries for the specific Spring version and add them.
- Most of the times we have to configure the Data Source, JdbcTemplate, Transaction Manager, DispatcherServlet, HandlerMapping, ViewResolver etc beans in the same way.
- We should always deploy in external server.
- The problem with Spring component-scanning and auto wiring is hard to see how all of the components in an application are wired together.

Spring Boot:

- Spring Boot is an open-source java-based framework maintained by a company called Pivotal.
- Spring Boot is built on top of the Spring framework, with minimal or less configurations.
- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
- Spring Boot provides production-ready features (Actuators) such as metrics, health checks and externalized configuration.
- Spring Boot can be used to develop Spring based applications either by using JAVA or Kotlin or Groovy
- Spring Boot reduces application development time.
- Spring Boot will improve Productivity.
- SpringBoot provided auto-configuration mechanism to simplify programmers' life.

Sample Project Creation using STS : Live Demo

Advantages of SpringBoot:

- SpringBoot Starters: Transitive dependencies are available
  - Help easy dependency Management
  - To Simplify Maven Configuration Spring Boot providing actuators
    - Spring-boot-starter-parent
    - Spring-boot-starter-web – web applications or distributed applications
    - Spring-boot-starter-webflux – reactive programming
    - Spring-boot-starter-data-jpa – database jar hibernate
    - Spring-boot-starter-mail
    - Spring-boot-validation
    - Spring-boot-devtools
- Auto Configuration
  - Most of the commonly used built-in classes such as Data Source, JdbcTemplate, Transaction Manager, DispatcherServlet, HandlerMapping, ViewResolver etc using customizable properties.
  - So, we need to enable auto configuration by adding @EnableAutoConfiguration or @SpringBootApplication. It triggers auto-configuration functionalities for our application.
- Embedded Servlet Container (or) Embedded Servers
  - Tomcat, Jetty & Undertow – embedded web servers
  - SpringBoot providing embedded servers to run web application or distributed application.
  - By Default, tomcat uses embedded server in spring Boot Application

- Spring-boot-start-web automatically pulls spring-boot-starter-tomcat which starts tomcat as an embedded server. So, we don't have to deploy our application on any externally installed tomcat server.

- SpringBoot Actuators
  - SpringBoot Actuators are providing production ready features.
  - It is mainly used to get internals of running application like
    - Autoconfig details
    - Environment variables
    - Configuration properties
    - Memory usage
    - Garbage collection
    - Web request & data source usage.
  - Using Actuators, we can monitor our running application
    - States: UP or Down

Spring Boot Staters:

- Without SpringBoot, we need to configure all the dependencies required in our pom.xml file
- SpringBoot starters aggregate common groupings of dependencies into single starter dependency that can be added to a project's Maven or Gradle build

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- Parent Element is one of the interesting aspects in the pom.xml file

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.1</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

- The advantage of using the spring-boot-starter-parent POM file is that developers need not worry about find the right compatible version of different libraries such as Spring, Jersey, Junit, Hibernate, Jackson and so on.
- The jar versions are defined in <artifactId>Spring-boot-dependencies</artifactId>

Auto Configuration:

- Spring Boot is intelligent to understand what configuration is required for our project and it will provide.
- Spring Boot used Convention over configuration by scanning the dependent libraries available in the class path.

- For each spring-boot-starter-* dependency in the POM file, SpringBoot executes a default configuration classes.
- Soring Boot Provides 'spring-boot-autoconfigure' module(spring-boot-autoconfigure-<version>.jar) which contains many configuration classes to autoconfigure beans. The above jar file contains META-INF/spring.factories file which contains list of autoconfigure classes.

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.context.LifecycleAutoConfiguration,\
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
```

Spring Boot Annotations:

- This enables Spring component-scanning and Spring Boot auto-configuration. In Fact, @SpringBootApplication combines three other useful annotations
- The @SpringBootApplication is the combination of three annotations **@EnableAutoConfiguration**, **@ComponentScan**, and **@Configuration**.
- The @SpringBootApplication annotation is used on the application class while setting up a new Spring Boot project.
- The class that is annotated with this annotation must be placed in the base package.
- The main task of this annotation is to scan the projects; it scans its sub-packages only.
  - For example, if we annotate a class that is available in a sub-package then it will not scan the main package.
- **@SpringBootConfiguration**
  - It represents our class as Configuration class
  - is an alternative to the @Configuration annotation and main difference is that this annotation allows configuration to be automatically located (Especially useful for unit or integration tests)
  - To customize bean creation, we can write @Bean method in this class
- **@ComponentScan**
  - It is the built-in functionality in Spring Boot Application.
  - The is the process of identifying spring beans available in our application.
  - It Enables component-scanning so that the web controller classes and other components we write will be automatically discovered & registered as beans in the Spring Application Context. (Internally using <context: component-scan/)
  - It will scan from base package. After base package it will scan sub-packages of base package.
    - com.javaexpress - it will scan this package (Base Package)
    - com.javaexpress.beans - it will scan this package (Sub-package)
    - com.javaexpress.services – it will scan this package
    - in.javaexpress.dao - it will not participate scanning(different)
- **@EnableAutoConfiguration**

- The @EnableAutoConfiguration annotation is used to implicitly defines a base "search package". Usually, it is placed on the main application class. It will automatically configure the projects by adding beans based on classpath settings, beans, and other property settings.
- The use of this annotation is degraded in Spring Boot 1.2.0 release because there is a great alternative of the annotation **@SpringBootApplication** is available from.

What is Spring Bean?

- Any Normal Java class is initialized by Spring IOC Container is called Spring Bean.
- Spring IOC Manges the life cycle of spring bean cycle, bean scopes & injecting any required dependencies in the bean

Stereotype Annotations:

- **@Component**
  - This act as a more generic stereotype annotation to manage any component by spring whereas the other annotations are specialized for the specific use case.
  - The @Component annotation is used at the class level to mark a Java class as a bean. When we mark a class with @Component annotation, it will be found during the classpath.
  - The Spring framework will configure the annotated class in the application context as a Spring Bean.
- **@Service – Business Logic**
  - The @Service annotation is also used at the class level to mark a service implementation including business logic, calculations, call external APIs, etc. Generally, it holds the business logic of the application.
  - Business classes are annotated with @Service even though the current release of the spring doesn't us any impact of using it (other than exposing that class as spring bean)
- **@Repository – Database Logic**
  - This is used for exposing DAO class as spring bean.
  - The @Repository annotation is used on java classes which directly access the database. It acts as a marker for any class that is used as a repository or DAO (Data Access Object).
  - This annotation has a built-in translation feature means when the class found an exception, it will automatically handle with that; there is no need to add a try-catch block manually.

- **@Bean**
  - The @Bean annotation is an alternative of XML <bean> tag. If you ever have gone through the <bean> tag, you must be familiar with it's working.
  - It is a method level annotation, which tells the method to produce a bean to be managed by the Spring Container.
  - It also works with @Configuration annotation to create Spring beans. As the @Configuration holds methods to instantiate and configure dependencies.
  - The methods, which are annotated with @Bean annotation acts as bean ID and it creates and returns the actual bean.

```
@Bean
public BeanDemo beanDemo()
{
return new BeanDemo ();
}
```

- **@Controller**
  - When we mark a class with @Controller, then it will be exposed as Spring MVC Controller to handle form submissions.
  - The @Controller annotation is also used at the class-level; it is a specialized version of @Component.
  - It is used to annotate a class as a web request handler; most commonly used to serve web pages
  - It returns a string having a redirect route. Generally, it is used with @RequestMapping annotation.

Spring IOC Container:

- Inversion of Control
- Giving control to the container to get instance of object is called IOC.
- IOC container generating objects and giving to the programmer.
- IOC Container is a mechanism to achieve loose coupling between object dependencies.
- By default, container maintains Singleton object.
- Primary Functions
  - Create and Manage Objects (IOC)
  - Inject Object's dependencies (Dependency Injection)
- It will Perform some tasks like
  - To Create Objects
  - To Configure the objects
  - To Assemble the dependencies between objects
- Two Types of Containers
  - Bean Factory
  - ApplicationContext

Dependency Injection:

- Dependency Inversion Principle
- The client delegates to calls to another object the responsibility of providing its dependencies.
- Dependency Means Helper Objects
- Dependency Injections is a design pattern in order to remove dependency from the programming code.
- Way of injecting properties to an object is called Dependency Injection.

Injection Types:

- There are 3 types of injection with Spring
  - Constructor Injection
  - Setter Injection
  - Field Injection - autowired

Setter Injection:

- Inject dependencies by calling setter method on your class

Constructor Injection:

- Inject dependencies by calling parameterized constructor on your class

@Autowired:

- The @Autowired annotation is applied on fields, instance variables, setter methods, and constructors. It provides auto wiring to bean properties.
- When we apply the @autowire to a field, Spring contain will automatically assign the fields with assigned values.
- We can also use this annotation on private fields. Consider the below code:

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

public class Employee
{
        @Autowired
private Person p;
private int sal;

}
```

@Qualifier:

- The @Qualifier annotation is used along with @Autowired annotation.
- It provides more control over the dependency injection process.
- It can be specified on individual method parameters or constructors' arguments.
- It is useful for reducing the duplicity while creating more bean files of the same type and we want to use only one of them with a property.

```
@Component
public class B1 implements BeanInterface {
  //
}
@Component
public class B2 implements BeanInterface {
  //
}
```

- If B1 autowires the BeanInterface, it is hard for Spring to decide which one of the given implementations to inject.
- The solution to this problem is that we can use the @Qualifier annotation. Consider the below example:

```
@Component
public class B1 {
  @Autowired
  @Qualifier("beanB2")
  private BeanInterface dependency;
  ...
}
```

How IOC Container will start in SpringBoot Application?

- In SpringBoot Application, run method is responsible to start IOC Container.
- Run method is bootstrapping method in Spring Boot Application.
- Based on starter available in class path, it will start IOC Container.
    - Spring-boot-starter-webflux -> Reactive Type
    - Spring-boot-starter-web -> Web Servlet Type
    - Spring-boot-starter -> default type
- This run() method contains boot strapping logic to start spring boot application.
    - Start Stopwatch
    - Start Listeners
    - Prepares Application Arguments
    - Prepares Environment
    - Print Banner
    - Create IOC Container
    - Stop Watch
    - Print Application Startup Time
    - Call Runners
    - Return Context.

Runners in Spring Boot:

- Runners are used to execute the logic only one-time once application got started.
- If we need to execute some custom code just before boot application starting up? We can make that happen with a runner
- Spring Boot Provides 2 types of Runners (run)
    - Command Line Runner
    - Application Runner (New One)
- When we want to execute some piece of code exactly before the application startup completes then we can use these interfaces
- Real Time Use Cases:
    - Loading data to Cache
    - Configuring CRON Jobs
    - Spring Batch framework will trigger jobs using Runners only

Command Line Runner:

- The CommandLineRunner interface in Spring Boot lets you access all the raw command-line args

Disadvantage of Command Line Runner:

- You only get the arguments as a list of Strings.
- There is no argument parsing.
- You cannot inject these arguments into other components directly.

Application Runner:

- By default, Spring boot provides a bean of type *Application Arguments*. This bean contains all parsed and non-parsed command line arguments.
- we have used a wide range of methods from *ApplicationArguments* interface.

- args.getOptionNames() – Returns a list of parsed parameter keys.
- args.containsOption() – Checks and returns a Boolean based on the presence of given option.
- args.getOptionValues() – returns all matching values as an array for the given option(Yes, you can add an option more than once)
- args.getNonOptionArgs() – returns all orphan arguments that didn't follow the –option=value format.

Fat Jar:

- Below plug-in should be added in pom.xml file to create Fat Jar

<! —Build Configuration >

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

Logging Levels in SpringBoot

- In spring boot, it is very easy to log at a different level; also, spring boot provides us default logging. While using it, we do not require an external dependency because it is already included.
- Logging is used to keep track of all the activity of our application; by the use of this, we can identify the errors in our application at runtime if they occur. Also, we can check and track if the application working fine.
- These are very easy to use and handle because we do not require lots of configuration for this as the spring boot framework manages it.

Syntax:

logging. level. root = your logging level

Example: logging. level. root=DEBUG

Customize Logging Pattern in Console:

- If we want to customize logging pattern in spring boot application then we need to configure in application. properties file like below

```
#Logging Pattern for Console
```

- `logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n`
    - Date and Time — Millisecond precision.
    - Log Level — ERROR, WARN, INFO, DEBUG or TRACE.
    - Process ID.

- Thread name — Enclosed in square brackets (may be truncated for console output).
- Logger name — This is usually the source class name (often abbreviated).

The log messages

Change Default Banner in SpringBoot:

- To change the default banner text that we usually see during the startup time, we need to put our custom text content in file or custom banner either in the classpath or in a sperate location
- When we run spring boot applications, SpringBoot logo is printing on the console that is called as Spring Boot Banner
- Banner Printing is part of SpringApplication.run(..) method
- Spring Boot banner is having below 3 modes
  - OFF (Disable the SpringBoot Banner)
  - LOG (To print logo in file)
  - CONSOLE (Default Printing in console)
- We can customize banner text in SpringBoot by creating banner.txt file in src/main/resources folder.
- To generate banner use below link
  - Spring Boot banner.txt generator (datenkollektiv.de)
- First preference will be given to banner.txt file if banne.txt not available then default banner text will be printed in the console

```
spring.main.banner-mode=console
spring.banner.location=classpath:banner.txt
```

Spring Boot Actuators:

- The Problem with Auto discovery & Autoconfiguration is difficult to know which beans were configured and how these beans wired together.
- Actuators are used to provide production ready features of our application
- Using Actuators, we can monitor and manage our applications
- Actuator endpoints are available to get information about application
- The Spring Boot Actuators provide us details such as bean have been configured, bean dependencies, Autoconfig report (which contains both positive and negative matches), environment variables, health, configuration properties, memory usage, garbage collection, web requests and data source usage.
- The following starter dependency should be added in pom.xml

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

- Spring Boot Actuators providing following Details:
  - What beans have been configured in the Spring Application Context

- Information about the application
- Health of the application
- What are the configuration properties, Environment variables, System Variables, Command Line Arguments are available in our application?
- A trace of recent HTTP Requests handled by our application.
- Various metrics pertaining to memory usage, Garbage collection, web requests & Data source usage.

- To get complete endpoints list use below URL
  - http://localhost:8080/actuator

ID	Description
auditevents	The **auditevents** endpoint is used to expose the audit events information for the current application. It requires an `AuditEventRepository` bean.
beans	The **beans** endpoint is used to display a complete list of all the Spring beans in our application.
caches	The **caches** endpoint is used to exposes available caches.
conditions	The **conditions** endpoint is used to display the conditions that were evaluated on configuration and auto-configuration classes. Further, it displays the reasons why they did or did not match.
configprops	The **configprops** endpointis used to display a collated list of all `@ConfigurationProperties`.
env	The **env** endpoint is used to expose the properties from Spring's `ConfigurableEnvironment`.
flyway	The **flyway** endpoint is used to display any Flyway database migrations that have been applied. It requires one or more `Flyway` beans.
health	The **health** endpoint is used to display the application health information.
httptrace	The **httptrace** endpoint is used to display the HTTP trace information (by default, the last 100 HTTP request-response exchanges). It requires an `HttpTraceRepository` bean.
info	The **info** endpoint is used to display the arbitrary application info.
integrationgraph	The **integrationgraph** is used to display the Spring Integration graph. It requires a dependency on `spring-integration-core`.
loggers	The **loggers** endpoint is used to display and modify the configuration of loggers in the application.
liquibase	The **liquibase** endpoint is used to display any Liquibase database migrations that have been applied. It requires one or more `Liquibase` beans.
metrics	The **metrics** endpoint is used to display 'metrics' information for the current application.
mappings	The **mappings** endpoint is used to display a collated list of all `@RequestMapping` paths.
scheduledtasks	The **scheduledtasks** endpoint is used to display the scheduled tasks in our application.
sessions	The **sessions** endpoint is used to allow retrieval and deletion of user sessions from a Spring Session-backed session store. It requires a Servlet-based web application using Spring Session.
shutdown	The **shutdown** endpoint is used to let the application shutdown gracefully. It is disabled by default.
startup	The **startup** endpoint is used to display the startup steps data collected by the `ApplicationStartup`. It requires the `SpringApplication` to be configured with a `BufferingApplicationStartup`.

Note:

All Actuator REST end points display response in JSON Format. Hence add JSON Viewer through chrome extension (chrome browser -> Customize and Control Google Chrome -> More Tools -> Extensions -> Get More Extensions -> In Chrome web store -> search the store with Json viewer -> Add to Chrome)

```
#Root Logs
logging.level.root=trace
#Library Logs
logging.level.org.springframework.beans=debug
#Project Logs
logging.level.com.javaexpress=debug
```

www.youtube.com/c/javaexpress
+91 8555929285
Java Express