Spring Boot Interview Questions

August 20, 2018

Contents

1. What are the features of Spring Boot applications?

Ans: 1. Spring boot performs many configurations automatically. So the development is faster.
2. Spring Boot includes support for embedded Tomcat, Jetty, and Undertow servers with default port 8080.
3. Using spring boot we can externalize our configurations so that we can work with the same application code in different environments. We can use properties files, YAML files, environment variables and command-line arguments to externalize configuration.
4. Spring Boot uses Commons Logging for all internal logging, but we can also implement our Logging. By default Logback is used.
5. Spring Boot provides auto-configuration for Redis, MongoDB, Neo4j, Elasticsearch, Solr and Cassandra NoSQL technologies.
6. Spring boot auto configures the necessary infrastructure to send and receive messages using JMS.
7. Spring boot provides `@EnableAutoConfiguration` that allows spring boot to configure spring application based on JAR dependencies that we have added.
8. Spring provides `@SpringBootApplication` annotation that is the combination of `@Configuration`, `@EnableAutoConfiguration` and `@ComponentScan` annotations.
9. Spring boot uses `SpringApplication.run()` inside java main method to bootstrap the application.
10. Spring Boot provides a `@SpringBootTest` annotation to test spring boot application.

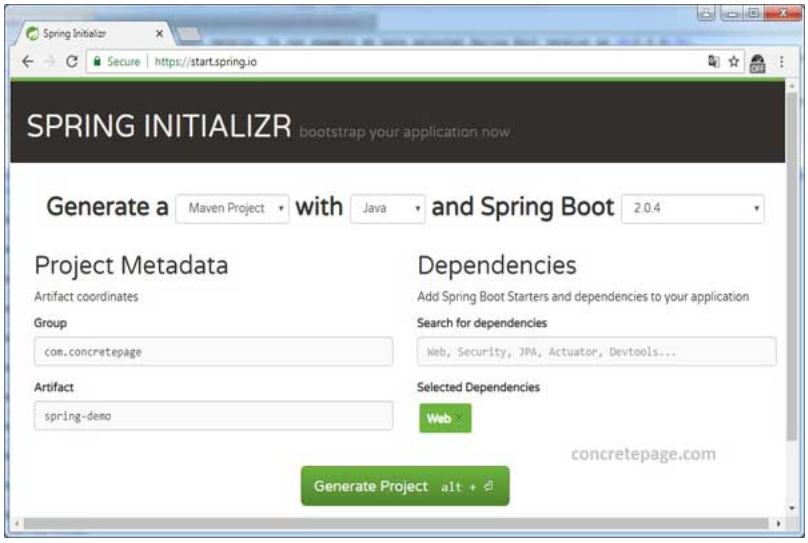
2. How to Create Spring Boot Project using Spring Initializer?

Ans: Go to the spring initializer URL **https://start.spring.io** and here we will select following details.
a. First select Maven project or Gradle project. For the example we have selected maven project.
b. Select the language such as Java, Kotlin or Groovy. I have selected Java.
c. Now select the spring boot version. In our example we have selected Spring Boot version as **2.0.4**.
d. We need to specify artifact coordinates i.e. group and artifact name. In our example I have specified project metadata as following.

Group: com.concretepage
Artifact: spring-demo

e. Now select dependencies required by project. If we want to create web project then enter **web** keyword and we will get drop-down for web and select it. This will provide all required JAR dependencies to develop web project.
f. Now click on **Generate Project** button. A project will get started to download.

Find the print screen.



3. What are the Spring Boot Starters?

Ans: There are many Spring Boot starters for different purposes to start a Spring Boot application. Find some of them.
spring-boot-starter-parent: It is a special starter that provides useful Maven defaults. It is used in parent section in the POM.
spring-boot-starter-web: Starter for building web and RESTful application with MVC.
spring-boot-starter-security: Starter for using Spring Security.
spring-boot-starter-web-services: Starter for using Spring web services.
spring-boot-starter-thymeleaf: Starter for Spring MVC using Thymeleaf views.
spring-boot-starter-freemarker: Starter for Spring MVC using FreeMarker views.

Find a sample `pom.xml` to start a Spring Boot web application.
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.concretepage</groupId>
        <artifactId>spring-boot-app</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>
        <name>spring-boot-app</name>
        <description>Spring Boot Application</description>
        <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.0.4.RELEASE</version>
            <relativePath/>
        </parent>
        <properties>
            <java.version>9</java.version>
        </properties>
        <dependencies>
          <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
          </dependency>
          <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
          </dependency>
        </dependencies>
        <build>
          <plugins>
             <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
             </plugin>
          </plugins>
        </build>
</project>
```

4. What is the role of @EnableAutoConfiguration annotation?

Ans: `EnableAutoConfiguration` enables auto-configurations that we need in our applications. Auto-configurations are usually based on our classpath and beans we have defined. If `tomcat-embedded.jar` is available in our classpath then in web application, embedded tomcat will be started on application startup. It is recommended that we should place `@EnableAutoConfiguration` in a root package so that all sub-packages and classes can be searched. We create a Java main class annotated with `EnableAutoConfiguration` as application starter.
MyApplication.java

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableAutoConfiguration
@ComponentScan
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

5. What is the role of @SpringBootApplication annotation?

Ans: `@SpringBootApplication` annotation is the combination of `@Configuration`, `@EnableAutoConfiguration` and `@ComponentScan` annotations. Our application starter main class can be created by annotating `@SpringBootApplication`.

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

@SpringBootApplication
public class MyApplication {
        public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

6. What are frequently used commands to create and run Spring Boot Applications?

Ans: Here we are consolidating some commands used to create and run spring boot applications.

1. For Maven

mvn dependency:tree: Prints tree of JAR dependencies.
mvn clean eclipse:eclipse: Creates `.classpath`
mvn clean package: Creates JAR/WAR for the application.
mvn spring-boot:run: Starts tomcat to run application in exploded form.

2. For Gradle

gradle dependencies: Prints list of direct and transitive dependencies.
gradle clean eclipse: Creates `.classpath`
gradle clean build: Creates JAR/WAR for the application.
gradle bootRun: Starts embedded tomcat server to run application in exploded form for web application.

3. For Java

a. Run Executable JAR.

```
java -jar <JAR-NAME>
```

b. Run a packaged application with remote debugging support enabled.

```
java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=8000,suspend=n \ -jar <JAR-NAME>
```

7. How to create executable JAR using Maven?

Ans: Find the steps to create executable JAR using Maven.

Step 1: Make sure your `pom.xml` should contain.

a.

```
<packaging>jar</packaging>
```

b.

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

Step 2: Run command: **mvn clean package**

Step 3: For the configuration

```
<artifactId>spring-boot-demo</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

We will get JAR within **target** directory as following.

spring-boot-demo-0.0.1-SNAPSHOT.jar : This is executable JAR

spring-boot-demo-0.0.1-SNAPSHOT.jar.original : This is original JAR and not executable.

Step 4: Run executable JAR as following.

```
java -jar target/spring-boot-demo-0.0.1-SNAPSHOT.jar
```

8. How to create executable JAR using Gradle?

Ans: spring-boot-gradle-plugin is available in gradle. We need not to configure it separately. Like Maven, we can create executable JAR using Gradle. **spring-boot-gradle-plugin** provides the following command functionality.

1. **gradle clean build** : Create executable and original JAR.
2. **gradle bootRun** : Starts the embedded tomcat server to run application in exploded form for web application.

9. How to use Spring Boot Developer Tools?

Ans: Spring provides `spring-boot-devtools` for developer tools. This is helpful in application development mode. One of the features of developer tool is automatic restart of the server. To configure developer tools using Maven we need to add `spring-boot-devtools` dependency.

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
     <optional>true</optional>
</dependency>
```

When using developer tools we should know the following points.

1. When we create a JAR or WAR as fully packaged application and run it then developer tools are automatically disabled.
2. When we run the application using **java -jar** or special classloader, then it is considered a "production application" and developer tools will be automatically disabled.
3. It is best practice to set **<optional>true</optional>** that will avoid developer tools to apply transitively on other module.
4. When developer tools has been configured, the project in exploded form are started using **restart** classloader and fully packaged application are started using **base** classloader by spring boot.

10. How to use LiveReload in Browser with Spring Boot Dev Tools?

Ans: In Spring MVC project, there is involvement of a web server and browser. In development mode whenever we change any file, generally we need to restart the server and refresh the browser to get updated data. Spring Boot developer tools have automatized these two tasks. To refresh browser automatically we need to install LiveReload in our browser. Follow the below steps to get ready with LiveReload.

1. Go to the LiveReload extension link and install it.
2. Make sure that `spring-boot-devtools` is there in `pom.xml`.
2. Spring Boot developer tool will start a LiveReload server.
3. LiveReload can be enabled and disabled. To refresh the page using LiveReload, it must be enabled.
4. When we run **mvn spring-boot:run** and update our code, application will auto-restart and page will be refreshed.

11. How to use CommandLineRunner?

Ans: To use `CommandLineRunner`, we create a class and implement it and override its `run()` method and annotate this class with Spring stereotype such as `@Component`. When Spring Boot application starts, just before finishing startup, `CommandLineRunner` is executed. We can pass command line arguments to `CommandLineRunner`. It is used to start any scheduler or log any message before application starts. Find the sample code to implement `CommandLineRunner`.

MyCommandLineRunner.java

```
import java.util.Arrays;
import java.util.stream.Collectors;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
@Component
public class MyCommandLineRunner implements CommandLineRunner {
    private static final Logger logger = LoggerFactory.getLogger(MyCommandLineRunner.class);
    public void run(String... args) {
        String strArgs = Arrays.stream(args).collect(Collectors.joining("|"));
        logger.info("Application started with arguments:" + strArgs);
    }
}
```

12. How to use custom banner in Spring Boot application?

Ans: We use custom banner as following.

Text Banner: For text banner just create a file named as `banner.txt` with desired text and keep it at the location `src\main\resources`.

Image Banner: For image banner just create a file named as `banner.gif` and keep it at the location `src\main\resources`. Other file extensions such as jpg, png can also be used. Console should support to display image.

In the `application.properties` we can configure following banner properties.

banner.charset: It configures banner encoding. Default is UTF-8

banner.location: It is banner file location. Default is classpath:banner.txt

banner.image.location: It configures banner image file location. Default is classpath:banner.gif. File can also be jpg, png.

banner.image.width: It configures width of the banner image in `char`. Default is 76.

banner.image.height: It configures height of the banner image in `char`. Default is based on image height.

banner.image.margin: It is left hand image margin in `char`. Default is 2.

banner.image.invert: It configures if images should be inverted for dark terminal themes. Default is false.

13. How to use XML configuration in Spring Boot?

Ans: To load XML configuration, we use `@ImportResource` with `@SpringBootApplication` annotation.

MyApplication.java

```
@SpringBootApplication
@ImportResource("classpath:app-config.xml")
```

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

14. How to change default server port?

Ans: Default server port in Spring Boot application can be changed in many ways.

1. Using `application.properties`

```
server.port = 8585
```

Using `application.yml`

```
server:
  port: 8585
```

2. Using Java command with `--server.port`

```
java -jar my-app.jar  --server.port=8585
```

3. Programmatically with `SERVER.PORT` key.

MyApplication.java

```
@SpringBootApplication
public class MyApplication {
  public static void main(String[] args) {
     SpringApplication application = new SpringApplication(MyApplication.class);
     Map<String, Object> map = new HashMap<>();
     map.put("SERVER.PORT", "8585");
     application.setDefaultProperties(map);
     application.run(args);
  }
}
```

15. How to change context path in Spring Boot application?

Ans: We can change context path by configuring property `server.servlet.context-path` for Spring Boot 2.x and `server.contextPath` for Spring 1.x in property file as well as in command line as arguments with **java** command.

Find the configuration for Spring Boot 2.x.

Using property file.

```
server.servlet.context-path = /spring-boot-app
server.port = 8585
```

Using **java** command.

```
java -jar my-app.jar --server.servlet.context-path=/spring-boot-app --server.port=8585
```

Change context path programmatically.

MyApplication.java

```
import java.util.HashMap;
import java.util.Map;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class MyApplication {
        public static void main(String[] args) {
                SpringApplication application = new SpringApplication(MyApplication.class);
                Map<String, Object> map = new HashMap<>();
                map.put("server.servlet.context-path", "/spring-boot-app");
                map.put("server.port", "8585");
                application.setDefaultProperties(map);
                application.run(args);
```

```
        }
}
```

16. How to configure HikariCP in Spring Boot Application?

Ans: In Spring Boot 2.x, the JAR dependencies for HikariCP are resolved by default when using `spring-boot-starter-jdbc` or `spring-boot-starter-data-jpa`. But if we are using Spring Boot 1.x, we need to include HikariCP dependency.

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
</dependency>
```

To configure Hikari specific connection pool settings, Spring Boot provides `spring.datasource.hikari.*` prefix. Find the sample configurations.

application.properties

```
#Spring Boot 2.0 includes HikariDataSource by default
#spring.datasource.type = com.zaxxer.hikari.HikariDataSource

spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=1200000
spring.datasource.hikari.auto-commit=true
```

17. How to configure Tomcat Connection Pool?

Ans: To use tomcat connection Pool in Spring Boot 2.0, we need to resolve `tomcat-jdbc` dependency.

```
<dependency>
     <groupId>org.apache.tomcat</groupId>
     <artifactId>tomcat-jdbc</artifactId>
</dependency>
```

`tomcat-jdbc` is used with either `spring-boot-starter-data-jpa` or `spring-boot-starter-jdbc`. In Spring Boot 1.x, we need not to resolve `tomcat-jdbc` and it is resolved by default by `spring-boot-starter-data-jpa` and `spring-boot-starter-jdbc`.

To configure Tomcat specific connection pool settings, Spring Boot provides `spring.datasource.tomcat.*` prefix. Find the sample configurations.

```
spring.datasource.type = org.apache.tomcat.jdbc.pool.DataSource

spring.datasource.tomcat.initial-size=15
spring.datasource.tomcat.max-wait=20000
spring.datasource.tomcat.max-active=50
spring.datasource.tomcat.max-idle=15
spring.datasource.tomcat.min-idle=8
spring.datasource.tomcat.default-auto-commit=true
spring.datasource.tomcat.test-on-borrow=false
```

18. How to configure Spring Boot properties?

Ans: Spring Boot loads `application.properties` and `application.yml` from classpath by default. If both files are present in the classpath then both are loaded and merged into environment. Using these files we configure our properties. Find sample examples.

a. Using property file **application.properties**

```
server.servlet.context-path = /spring-boot-app
server.port = 8585
```

b. Using yml file **application.yml**

```yaml
server:
  servlet:
    context-path: /spring-boot-app
  port: 8585
```

19. How to configure logging in Spring Boot application?

Ans: Spring Boot uses Logback, Log4J2 and java util logging. By default Spring Boot uses Logback for its logging. By default log is logged in console and can also be logged in files. The JAR dependency for Logback is resolved by **spring-boot-starter-logging**. When we use any spring boot starter then **spring-boot-starter-logging** is resolved by default. We need not to include it separately. If Logback JAR is available in classpath then Spring Boot will always choose Logback for logging. So to use other logging such as Log4J2, we need to exclude Logback JAR and add Log4J2 JAR in classpath. To use Logback logging we have to do nothing, just configure logging level in `application.properties` or `application.yml` and we are done. By default `ERROR`, `WARN` and `INFO` log level messages are logged in console. To change log level, use **logging.level** property. To get logs in file, we can configure **logging.file** or **logging.path** in property file. Log files will rotate when they reach 10 MB. Default logging configurations can be changed using following properties.

logging.level.* : It is used as prefix with package name to set log level.
logging.file: It configures a log file name to log message in file. We can also configure file name with absolute path.
logging.path: It only configures path for log file. Spring Boot creates a log file with name `spring.log`.
logging.pattern.console: It defines logging pattern in console.
logging.pattern.file: It defines logging pattern in file.
logging.pattern.level: It defines the format to render log level. Default is `%5p`.
logging.exception-conversion-word: It defines conversion word when logging exceptions.

20. How to configure logging using application.properties?

Ans: Find the sample logging configuration using `application.properties`.

```
logging.level.org.springframework.security= DEBUG
logging.level.org.hibernate= DEBUG

logging.path = concretepage/logs
logging.pattern.file= %d{yyyy-MMM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{15} - %msg%n
logging.pattern.console= %d{yyyy-MMM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{15} - %msg%n
```

21. How to configure logging using application.yml?

Ans: Find the sample logging configuration using `application.yml`.

```yaml
logging:
  level:
    org:
      springframework:
        security: DEBUG
    hibernate: DEBUG

  path: concretepage/logs
  pattern:
    file: '%d{yyyy-MMM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{15} - %msg%n'
    console: '%d{yyyy-MMM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{15} - %msg%n'
```

22. How to use profiles in Spring Boot application?

Ans: Spring provides `@Profile` annotation using which we create profiles. `@Profile` is used with `@Configuration` and spring stereotypes such as `@Component`, `@Service` etc. Different profile is created for different environment. For example we can have environment such as production, development and testing. In development environment we can enable development profile and in production environment we can enable production profile and so on. A profile can be activated using property file **.properties/.yml**, command line and programmatically. We can also create a default profile that will work when there is no active profile. To add active profile in property file, we need to configure **spring.profiles.active** property. We can also configure profile using **spring.profiles.include** that will be

included for every active profile. When we add active profile using command line then the active profile added in property file is replaced with profile added by command line. We can add active and default profile programmatically by using `ConfigurableEnvironment`. In spring boot testing we add active profile by using `@ActiveProfiles` annotation. We can create property file using profile name using the convention `application-{profile}.properties`. The advantage of this approach is that we can configure properties profile specific. Suppose we have a development profile as `dev` then we will create property file as `application-dev.properties` and it can be activated in following ways.

1. Using `application.properties` file.

```
spring.profiles.active=dev
```

2. Using command line.

```
java -jar -Dspring.profiles.active=dev myapp.jar
```

3. Activate profile programmatically.

MyApplication.java

```
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication application = new SpringApplication(MyApplication.class);
        application.setAdditionalProfiles("dev");
        application.run(args);
    }
}
```

23. How to use Thymeleaf in Spring Boot application?

Ans: Thymeleaf is a server-side template engine that can process XML, HTML etc. To use Thymeleaf with Spring Boot application, we need `spring-boot-starter-thymeleaf` in Maven or Gradle file. If Spring Boot scans Thymeleaf library in classpath, it will automatically configures Thymeleaf.

Find the Maven dependency for Thymeleaf.

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

We can change the default Thymeleaf configurations configuring Thymeleaf properties in `application.properties`. We are listing some of them here.

spring.thymeleaf.mode: Template mode that will be applied on templates. Default is `HTML 5` .
spring.thymeleaf.prefix: This is the value that will prepend with view name to build the URL. Default value is `classpath:/templates/`.
spring.thymeleaf.suffix: This is the value that will be appended with view name to build the URL. Default value is `.html`.

24. How to configure datasource using application.properties file?

Ans: To connect with a database in our Spring Boot application, we need to include `spring-boot-starter-jdbc` or `spring-boot-starter-data-jpa` in our Maven or Gradle file. To configure datasource properties use prefix `spring.datasource.*` in `application.properties`.

application.properties

```
#spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/concretepage
spring.datasource.username=root
spring.datasource.password=cp

spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
```

```
spring.datasource.hikari.max-lifetime=1200000
spring.datasource.hikari.auto-commit=true
```

Spring Boot detects driver class automatically.

25. How to configure JPA properties using application.properties?

Ans: We configure JPA properties using prefix `spring.jpa.properties.*` in `application.properties` file. Find the sample JPA configurations with Hibernate.

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.id.new_generator_mappings=false
spring.jpa.properties.hibernate.format_sql=true
```

26. What is the Spring Boot starter to work with Database?

Ans: Use `spring-boot-starter-data-jpa` to work with Spring Data.

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

For JDBC we can also use `spring-boot-starter-jdbc` dependency.

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

27. How to print datasource used by our Spring Boot application?

Ans: We can autowire `DataSource` in our application to get its instance. Find the code snippet to print datasource using `CommandLineRunner`.

SpringBootAppStarter.java

```
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootAppStarter implements CommandLineRunner {
    @Autowired
    DataSource dataSource;

    public static void main(String[] args) throws Exception {
        SpringApplication.run(SpringBootAppStarter.class, args);
    }
    @Override
    public void run(String... args) throws Exception {
        System.out.println("DataSource = " + dataSource);
    }
}
```

28. How to configure Spring Security in our Spring Boot application?

Ans: To work with Spring Security we need to include `spring-boot-starter-security` in our Maven or Gradle file. If spring security is in the classpath then our Spring Boot web applications are automatically secured by default using basic authentication. A default username as 'user' and random password that will be displayed in console when server starts, can be used for login authentication. The password is printed in console as follows.

```
Using default security password: 7e9850aa-d985-471a-bae1-25d741d4da23
```

The above password is random and changes when server is restarted. By default spring uses in-memory authentication with single user named as 'user'. Find some configurations.

1. To enable Spring Security in Spring Boot application just use the following Spring Boot starter.

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

2. To change default password, Spring Boot provides `security.user.password` property that needs to be configured in `application.properties` as given below.

```
security.user.password= concretepage
```

Now we can login application using **user/concretepage** credential. Other security properties can also be changed using the `security.*` prefix in `application.properties` as given below.

security.basic.enabled: It enables basic authentication. Default value is **true**.
security.basic.path: It configures paths to apply security. We need to provide comma separated paths.
security.enable-csrf: It enables CSRF. Default value is **false**.
security.require-ssl: It enables and disables SSL. Default value is **false**.
security.sessions: Default value is stateless. Values can be always, never, if_required, stateless.
security.user.name: It configures user name. Default user is **user**.
security.user.password: It configures password.
security.user.role: It configures role. Default role is **USER**.

3. If we have fine-tuned our logging configuration then to print default random password, we need to configure following property in `application.properties` with `INFO` level.

```
logging.level.org.springframework.boot.autoconfigure.security= INFO
```

4. By default static paths are not secured such as `/css/**`, `/js/**`, `/images/**`, `/webjars/**` and `**/favicon.ico`.
5. The features such as HSTS, XSS, CSRF, caching are provided by default in Spring Security.

Above properties can be switched on and off using `security.*` but if we want to use username and password in database then we need to use `UserDetailsService`. To control security related configuration, we can create a security configuration class that will extend `WebSecurityConfigurerAdapter` then override `configure()` method. This class will be annotated with `@Configuration` and `@EnableWebSecurity`. If we want to enable method level security, the configuration class will be annotated with `@EnableGlobalMethodSecurity`.

29. How to configure Jersey in Spring Boot application?

Ans: To configure Jersey in Spring Boot application, resolve `spring-boot-starter-jersey` dependency.

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

We can configure following Jersey properties in `application.properties` to change default Spring boot configurations for Jersey.
spring.jersey.application-path: Application path that acts as base URI. It overrides the value of `@ApplicationPath`.
spring.jersey.type: The value can be servlet or filter. Default value is servlet.
spring.jersey.filter.order: It defines the Jersey filter chain order. Default value is 0.
spring.jersey.init.*: Init parameters that will be passed to Jersey servlet or filter.
spring.jersey.servlet.load-on-startup: Load on startup priority for Jersey servlet. Default is -1.

30. How many ways can we run Spring Boot application?

Ans: We can run our Spring Boot application in following ways.
1. Using Maven Command: Go to the root folder of the project using command prompt and run the command.

```
mvn spring-boot:run
```

This will run Spring Boot application in exploded form and useful in development phase.

2. Using Eclipse: Using command prompt, go to the root folder of the project and run.

```
mvn clean eclipse:eclipse
```

and then refresh the project in eclipse. This will set classpath. Now run Main class by clicking **Run as -> Java Application**. This is useful in development phase.

3. Using Executable JAR: Using command prompt, go to the root folder of the project and run the command.

```
mvn clean package
```

We will get executable JAR such as **myapp.jar** in target folder. Run this JAR as

```
java -jar target/myapp.jar
```

31. How to work with Spring Boot REST application?

Ans: To work with Spring Boot RESTful web service, we need to provide `spring-boot-starter-web` Maven dependency as following.

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

The above Maven dependency collects the Jackson JSON library i.e. `jackson-databind` by default. Spring Boot REST gives JSON response by default because it detects `jackson-databind` in its classpath.
To support XML response in Spring Boot REST, we need to provide `jackson-dataformat-xml` library with `spring-boot-starter-web`. Find the Maven dependency.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.9.4</version>
</dependency>
```

Include the above dependency in `pom.xml` and we are all set to get XML response.

32. What is Spring Boot Starter for SOAP web service?

Ans: To create SOAP web service, we need `spring-boot-starter-web-services` as following.

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web-services</artifactId>
</dependency>
<dependency>
     <groupId>wsdl4j</groupId>
     <artifactId>wsdl4j</artifactId>
</dependency>
```

For SOAP client we use following dependencies.

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

33. How to get JdbcTemplate in Spring Boot application?

Ans: Find the steps to get `JdbcTemplate` in Spring Boot application.

1. Resolve `spring-boot-starter-jdbc` dependency. This will auto-configure `JdbcTemplate`.

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

2. Configure datasource in `application.properties`.

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/concretepage
spring.datasource.username=root
spring.datasource.password=cp
```

3. Autowire `JdbcTemplate`.

```
@Repository
public class ArticleDAO {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    -----------------
}
```

34. How to use @EnableJpaRepositories in Spring Boot application?

Ans: Spring boot can automatically detect our repository if the package of that interface is the same or sub-package of the class annotated with `@SpringBootApplication` and if not then we need to use `@EnableJpaRepositories` annotation with `@SpringBootApplication`. Using `@EnableJpaRepositories` we will configure package name in which our repository classes reside. Suppose the package of our repository classes is `com.cp.repository`, we will use `@EnableJpaRepositories` as following.

```
@SpringBootApplication
@EnableJpaRepositories("com.cp.repository")
public class MyApplication {
------
}
```

If we want to configure specific classes then we need to use `basePackageClasses` attribute of the `@EnableJpaRepositories` annotation. Suppose we have a class `ArticleRepository` in the package `com.cp.repository`, then we can configure repository using `basePackageClasses` as following.

```
import com.cp.repository.ArticleRepository;
@SpringBootApplication
@EnableJpaRepositories(basePackageClasses= {ArticleRepository.class})
public class MyApplication {
------
}
```

35. How to get EntityManager in Spring Boot application?

Ans: Find the steps to get `EntityManager` in Spring Boot application.

1. Resolve following dependency.

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

2. Configure datasource in `application.properties`.

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/concretepage
```

```
spring.datasource.username=root
spring.datasource.password=cp
```

3. Use `@PersistenceContext` to get `EntityManager`.

```
@Repository
public class ArticleDAO {
   @PersistenceContext
   private EntityManager entityManager;
   ------

}
```

36. How to create Spring Boot MVC application?

Ans: Resolve `spring-boot-starter-web` dependency.

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

When Spring Boot scans Spring Web in classpath, it atomically configures Spring Web MVC. To change any configuration, Spring Boot provides properties to be configured in `application.properties`. Find some properties.
spring.mvc.async.request-timeout: Timeout in milliseconds for asynchronous request.
spring.mvc.date-format: Date format to use.
spring.mvc.favicon.enabled: It enables and disables favicon. Default is **true**.
spring.mvc.locale: Locale to use.
spring.mvc.media-types.*: Maps file extensions to media type for content negotiation.
spring.mvc.servlet.load-on-startup: It configures startup priority for Spring Web Services Servlet. Default value is **-1**.
spring.mvc.static-path-pattern: It configures path pattern for static resources.
spring.mvc.view.prefix: It configures prefix for Spring view such as JSP.
spring.mvc.view.suffix: It configures view suffix.

To take a complete control on Spring MVC configuration we can create a configuration class annotated with `@Configuration` and `@EnableWebMvc`. To override any settings we need to extend `WebMvcConfigurerAdapter` class. To create a view in Spring Boot MVC, we should prefer template engine and not JSP because for the JSP there are known limitations with embedded servlet container.

37. How to register Servlet in Spring Boot application?

Ans: We can register Servlet in many ways in Spring Boot application. Suppose we have `HelloCountryServlet`.
1. Using `ServletRegistrationBean`: Create a `ServletRegistrationBean` bean in JavaConfig and register Servlet. It works since Servlet 3.0.
WebConfig.java

```
@Configuration
public class WebConfig {
   @Bean
   public ServletRegistrationBean<HttpServlet> countryServlet() {
           ServletRegistrationBean<HttpServlet> servRegBean = new ServletRegistrationBean<>();
           servRegBean.setServlet(new HelloCountryServlet());
           servRegBean.addUrlMappings("/country/*");
           servRegBean.setLoadOnStartup(1);
           return servRegBean;
   }
}
```

2. Using `@ServletComponentScan`: `@ServletComponentScan` in Spring Boot will scan Servlets annotated with `@WebServlet` annotation. `@ServletComponentScan` is used with `@Configuration` or `@SpringBootApplication` annotations.
SpringBootAppStarter.java

```
@ServletComponentScan
@SpringBootApplication
public class SpringBootAppStarter {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootAppStarter.class, args);
    }
}
```

38. How to register Filter in Spring Boot application?

Ans: A Filter can be registered in Spring Boot in many ways. Suppose we have a `ABCFilter`.

1. Using `FilterRegistrationBean`: Create a `FilterRegistrationBean` bean in JavaConfig and register our Filter. It works since Servlet 3.0.

```
@Bean
public FilterRegistrationBean<ABCFilter> abcFilter() {
   FilterRegistrationBean<ABCFilter> filterRegBean = new FilterRegistrationBean<>();
   filterRegBean.setFilter(new ABCFilter());
   filterRegBean.addUrlPatterns("/app/*");
   filterRegBean.setOrder(Ordered.LOWEST_PRECEDENCE -1);
   return filterRegBean;
}
```

2. Using `@Component` and `@Order`: We can register a filter using `@Component` and set order using `@Order`.

ABCFilter.java

```
@Order(Ordered.LOWEST_PRECEDENCE -1)
@Component
public class ABCFilter implements Filter {
  ------
}
```

3. Using `@ServletComponentScan`: To register a filter in Spring Boot, we can use `@ServletComponentScan` and the filter should be annotated with `@WebFilter` annotation. We need to use `@ServletComponentScan` with `@Configuration` or `@SpringBootApplication` annotations. `@ServletComponentScan` in Spring Boot will scan servlets annotated with `@WebServlet`, filters annotated with `@WebFilter` and listeners annotated with `@WebListener` only when using an embedded web server.

39. How to register Listener in Spring Boot application?

Ans: A Listener in Spring Boot application can be registered in many ways. Suppose we have a `SessionCountListener`.

1. Using `ServletListenerRegistrationBean`: Create a `ServletListenerRegistrationBean` bean in JavaConfig and register Listener.

```
@Bean
public ServletListenerRegistrationBean<SessionCountListener> sessionCountListener() {
   ServletListenerRegistrationBean<SessionCountListener> listenerRegBean = new ServletListenerRegist
   listenerRegBean.setListener(new SessionCountListener());
   return listenerRegBean;
}
```

2. We can register servlet listeners in Spring Boot by annotating it with Spring `@Component`.

```
@Component
public class SessionCountListener implements HttpSessionListener {
   ------
}
```

3. We can register servlet listeners using `@ServletComponentScan` with `@Configuration` or `@SpringBootApplication` annotations. The servlet listeners annotated with `@WebListener` will be scanned by `@ServletComponentScan`.

40. How to use Redis Cache in Spring Boot application?

Ans: Find the steps to use Redis cache in Spring Boot application.

1. To get Redis connections, we can use Lettuce or Jedis client libraries. Spring Boot 2.0 starter `spring-boot-starter-data-redis` resolves Lettuce by default. To get pooled connection factory we need to provide `commons-pool2` on the classpath. To work with Lettuce we need following Maven dependencies.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
</dependency>
```

2. Redis properties are configured using `spring.redis.*` prefix in `application.properties` file.

```
spring.redis.host=localhost
spring.redis.port=6379
spring.redis.password=

spring.redis.lettuce.pool.max-active=7
spring.redis.lettuce.pool.max-idle=7
spring.redis.lettuce.pool.min-idle=2
spring.redis.lettuce.pool.max-wait=-1ms
spring.redis.lettuce.shutdown-timeout=200ms

spring.cache.redis.cache-null-values=false
spring.cache.redis.time-to-live=600000
spring.cache.redis.use-key-prefix=true
```

3. Enable cache using `@EnableCaching` annotation with `@Configuration` or `@SpringBootApplication` annotations.

```
@SpringBootApplication
@EnableCaching
public class SpringBootAppStarter {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootAppStarter.class, args);
    }
}
```

41. How to use Jedis client with Spring Boot 2.x Redis?

Ans: By default Spring Boot 2.0 starter `spring-boot-starter-data-redis` uses Lettuce. To use Jedis we need to exclude Lettuce dependency and include Jedis. Find the Maven dependencies to use Jedis.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
  <exclusions>
    <exclusion>
        <groupId>io.lettuce</groupId>
        <artifactId>lettuce-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
</dependency>
```

`jedis` dependency will automatically resolve `commons-pool2` on the classpath.

To configure Jedis pool we need to use `spring.redis.*` prefix with Jedis pool connection properties. Find the Jedis pool sample configurations.

application.properties

```
spring.redis.host=localhost
spring.redis.port=6379
spring.redis.password=

spring.redis.jedis.pool.max-active=7
spring.redis.jedis.pool.max-idle=7
spring.redis.jedis.pool.min-idle=2
spring.redis.jedis.pool.max-wait=-1ms
```

References

Spring Boot Reference Guide
Spring Boot Tutorials