@Autowired: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

@Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

@ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

@Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

@Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

@Controller: The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

@Service: It is also used at class level. It tells the Spring that class contains the **business logic**.

@Repository: It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database

Spring Boot Annotations

- **@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication**.
- **@SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.

Spring MVC and REST Annotations

- **@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. We use it with the class as well as the method.
- **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
- **@PostMapping:** It maps the **HTTP POST** requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
- **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
- **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes** a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
- **@PatchMapping:** It maps the **HTTP PATCH** requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
- **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
- **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

- **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
- **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
- **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.** For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
- **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody** annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
- **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

Spring Boot Starters or Dependencies

1. Spring Boot Starter Web

There are two important features of spring-boot-starter-web:

- It is compatible for web development
- Auto configuration

If we want to develop a web application, we need to add the following dependency in pom.xml file:

```xml
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>2.2.2.RELEASE</version>
</dependency>
```

Starter of Spring web uses Spring MVC, REST and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count. The spring-boot-starter-web transitively depends on the following:

- org.springframework.boot:spring-boot-starter
- org.springframework.boot:spring-boot-starter-tomcat
- org.springframework.boot:spring-boot-starter-validation
- com.fasterxml.jackson.core:jackson-databind
- org.springframework:spring-web
- org.springframework:spring-webmvc

2. Spring Boot DevTools

DevTools stands for **Developer Tool.** The aim of the module is to try and improve the development time while working with the Spring Boot application. Spring Boot DevTools pick up the changes and restart the application.

We can implement the DevTools in our project by adding the following dependency in the pom.xml file.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime<scope >
</dependency>
```

3. What is the H2 DATABASE?

H2 is an **embedded, open-source,** and **in-memory** database. It is a relational database management system written in Java. It is a **client/server** application. It is generally used in **unit testing**. It stores data in memory, not persist the data on disk.

Advantages

- Zero configuration
- It is easy to use.
- It is lightweight and fast.

- It provides simple Configuration to switch between a real database and in-memory database.
- It supports standard SQL and JDBC API.
- It provides a web console to maintain in the database.

Persistence vs. In-memory Database

The persistent database persists the data in physical memory. The data will be available even if the database server is bounced. Some popular persistence databases are **Oracle**, **MySQL**, **Postgres**, etc.

In the case of the **in-memory database,** data store in the **system memory**. It lost the data when the program is closed. It is helpful for **POC**s (Proof of Concepts), not for a production application. The widely used in-memory database is **H2.**

Spring Boot – Difference Between CrudRepository and JpaRepository

CrudRepository	JpaRepository
It is a base interface and extends Repository Interface.	It extends PagingAndSortingRepository that extends CrudRepository.
It contains methods for CRUD operations. For example save(), saveAll(), findById(), findAll(), etc.	It contains the full API of CrudRepository and PagingAndSortingRepository. For example, it contains flush(), saveAndFlush(), saveAllAndFlush(), deleteInBatch(), etc along with the methods that are available in CrudRepository.
It doesn't provide methods for implementing pagination and sorting	It provides all the methods for which are useful for implementing pagination.
It works as a marker interface.	It extends both CrudRepository and PagingAndSortingRepository.
To perform CRUD operations, define repository extending CrudRepository.	To perform CRUD as well as batch operations, define repository extends JpaRepository.
Syntax: public interface CrudRepository<T, ID> extends Repository<T, ID>	**Syntax:** public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>

22. Profiles

Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. Any @Component or @Configuration can be marked with @Profile to limit when it is loaded:

```
@Configuration
@Profile("production")
public class ProductionConfiguration {
    // ..
}
```

In the **normal Spring way, you can use a spring.profiles.active Environment property to specify which profiles are active**. You can specify the property in any of the usual ways, for example you could include it in your application.properties:

spring.profiles.active=dev,hsqldb

1.2 What are Profiles in Spring boot?

Profiles in simple term can be termed as the different environments which every application comprises of. For example – DEV, QA, or PROD. Each environment has its own specific requirement i.e. database connection, pipeline, and deployment configuration. In spring we maintain profiles with the help of properties and file files.

application. properties

```
## Spring boot application name
spring.application.name=Profiles
## Profiles are activated using this property
## Tells Spring which profiles to use
## E.g. - Here we set the profile as "qa"
#spring.profiles.active=dev
spring.profiles.active=qa
#spring.profiles.active=prod
```