**@SpringBootApplication**

This annotation is used to mark the main class of a Spring Boot application. It encapsulates @SpringBootConfiguration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

**@EnableAutoConfiguration**

It automatically creates, and registers beans based on both the included jar files in the classpath and the beans defined by us.

**@ComponentScan**

@ComponentScan is an annotation used in the Spring Framework for auto-detecting and registering Spring-managed components (e.g. beans, controllers, services, repositories, etc.) within a specified package or set of packages.

**@Configuration**

The @Configuration annotation in Spring marks a class as a configuration class that provides bean definitions.

**Stereotype Annotations**

1. **Component**

without having to write any explicit code, Spring will:

Scan our application for classes annotated with @Component

Instantiate them and inject any specified dependencies into them.

1. Service
2. RestController/Controller
3. Repository

**Spring Core Annotations**

1. Configuration
2. Bean
3. Autowired
4. Qualifier
5. primary
6. Lazy
7. Value
8. PropertySource
9. ConfigurationProperties
10. Profile
11. Scope

**@Configuration:**

The @Configuration annotation in Spring marks a class as a configuration class that provides bean definitions.

**@Bean**

It indicates that a method produces a bean to be managed by the Spring Container. It is usually declared in Configuration class to create Spring bean definitions.

**@Qualifier**

The @Qualifier annotation is used to resolve the autowiring conflict, when there are multiple beans of same type.

**@Primary**

It indicates that a bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency.

**@Lazy**

It indicates that a bean should be lazily initialized, meaning it will only be created when it's first requested, rather than during the application context initialization.

**@Value**

@Value is a core annotation in Spring that is primarily used for assigning default values to variables and method parameters.

**@PropertySource**

It is used to provide properties file to Spring Environment.

**@ConfigurationProperties**

to map or bind the . properties or yml configuration values to Java objects. (with prefix)

**@Profile**

to resolve the challenge of controlling which parts of our application should be active under a particular set of conditions or environments.

**@Scope**

It indicates the lifecycle of an instance, such as singleton or prototype.

**REST API ANNOTATIONS**

1. RestController
2. Controller
3. RequestMapping
4. GetMapping
5. PostMapping
6. PutMapping
7. DeleteMapping
8. RequestBody
9. PathVariable
10. RequestParam
11. Controller Advice & ExceptionHandler

**@Controller**

@Controller is used to define a Spring MVC controller that processes HTTP requests and typically returns a view (like a JSP, Thymeleaf, or HTML page).

Methods in a @Controller class usually return the name of a view or a ModelAndView object. The returned view name is resolved by a view resolver to render the HTML page or template.

**@RestController**

@RestController is a specialized version of @Controller that is used for creating RESTful web services. It combines @Controller and @ResponseBody, meaning that the return values of methods are directly written to the HTTP response body rather than being resolved as a view name.

Methods in a @RestController class typically return objects, which are automatically serialized into JSON or XML (based on the content type requested) and sent in the HTTP response body.

**@RequestMapping**

This is an annotation used to map web requests to specific handler methods or classes. It can be applied at the class level or method level in a Spring controller.

**@ControllerAdvice**

It is used to define global exception handlers, model attribute handlers, and binding handlers that apply to all controllers or a subset of controllers in your application.

**@ExceptionHandler**

It is used to handle exceptions thrown by a specific controller method or across a set of controllers. It provides a way to define methods that should be called when a certain type of exception is thrown.

**SPRING DATA JPA**

1. Entity
2. Table
3. Column
4. Transactional