Spring Annotations

- **@Configuration** - used to mark a class as a source of the bean definitions. Beans are the components of the system that you want to wire together. A method marked with the @Bean annotation is a bean producer. Spring will handle the life cycle of the beans for you, and it will use these methods to create the beans.
- **@ComponentScan** -use to make sure that Spring knows about your configuration classes and can initialize the beans correctly. It makes Spring scan the packages configured with it for the @Configuration classes.
- **@Import** - If you need even more precise control of the configuration classes, you can always use @import to load additional configuration. This one work even when you specify the beans in an XML file like it's 1999.
- **@Component** - Another way to declare a bean is to mark a class with a @Component annotation. Doing this turns the class into a Spring bean at the auto-scan time.
- **@Service -** Mark a specialization of a @Component. It tells Spring that it's safe to manage them with more freedom than regular components. Remember, services have no encapsulated state.
- **@Autowired -** To wire the application parts together, use the @Autowired on the fields, constructors, or methods in a component. Spring's dependency injection mechanism wires appropriate beans into the class members marked with @Autowired.
- **@Bean -** A method-level annotation to specify a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.
- **@Lookup -** tells Spring to return an instance of the method's return type when we invoke it.
- **@Primary -** gives higher preference to a bean when there are multiple beans of the same type.
- **@Required -** shows that the setter method must be configured to be dependency-injected with a value at configuration time. Use @Required on setter methods to mark dependencies populated through XML. Otherwise, a BeanInitializationException is thrown.
- **@Value -** used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.
- **@DependsOn -** makes Spring initialize other beans before the annotated one. Usually, this behavior is automatic, based on the explicit dependencies between beans. The @DependsOn annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.
- **@Lazy -** makes beans to initialize lazily. @Lazy annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.
- **@Scope -** used to define the scope of a @Component class or a @Bean definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

- **@Profile -** adds beans to the application only when that profile is active.

- **@SpringBootApplication**
- One of the most basic and helpful annotations, is @SpringBootApplication. It's syntactic sugar for combining other annotations that we'll look at in just a moment. @SpringBootApplication is @Configuration, @EnableAutoConfiguration and @ComponentScan annotations combined, configured with their default attributes.
- **@Configuration** and **@ComponentScan**
- The @Configuration and @ComponentScan annotations that we described above make Spring create and configure the beans and components of your application. It's a great way to decouple the actual business logic code from wiring the app together.
- **@EnableAutoConfiguration**
- Now the @EnableAutoConfiguration annotation is even better. It makes Spring guess the configuration based on the JAR files available on the classpath. It can figure out what libraries you use and preconfigure their components without you lifting a finger. It is how all the spring-boot-starter libraries work. Meaning it's a major lifesaver both when you're just starting to work with a library as well as when you know and trust the default config to be reasonable.
- **@Controller** - marks the class as a web controller, capable of handling the HTTP requests. Spring will look at the methods of the class marked with the @Controller annotation and establish the routing table to know which methods serve which endpoints.
- **@ResponseBody** - The @ResponseBody is a utility annotation that makes Spring bind a method's return value to the HTTP response body. When building a JSON endpoint, this is an amazing way to magically convert your objects into JSON for easier consumption.
- **@RestController** - Then there's the @RestController annotation, a convenience syntax for @Controller and @ResponseBody together. This means that all the action methods in the marked class will return the JSON response.
- **@RequestMapping** (method = RequestMethod.GET, value = "/path") - The @RequestMapping (method = RequestMethod.GET, value = "/path") annotation specifies a method in the controller that should be responsible for serving the HTTP request to the given path. Spring will work the implementation details of how it's done. You simply specify the path value on the annotation and Spring will route the requests into the correct action methods.
- **@RequestParam(value="name", defaultValue="World")** - Naturally, the methods handling the requests might take parameters. To help you with binding the HTTP parameters into the action method arguments, you can use the @RequestParam (value="name", defaultValue="World") annotation. Spring will parse the request parameters and put the appropriate ones into your method arguments.
- **@PathVariable("placeholderName")** - Another common way to provide information to the backend is to encode it in the URL. Then you can use the @PathVariable("placeholderName") annotation to bring the values from the URL to the method arguments.