



Common Annotations in a Spring Boot App



@SpringBootApplication

This annotation marks the main class of a Spring Boot application, combining **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**. It serves as the entry point, enabling Spring Boot's autoconfiguration and component scanning for seamless application setup.

@SpringBootApplication

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

@Controller

Identifying a class as a Spring MVC controller, this annotation handles HTTP requests, responding with the appropriate view or data. It plays an important role in web request processing.

@Controller

```
public class ProductController {  
    @Autowired private ProductApiClient client;  
  
    @GetMapping("/products")  
    public String getProducts(Model model) {  
        var products = client.listAllProducts();  
        model.addAttribute("products", products);  
        return "products";  
    }  
}
```

@RestController

Similar to **@Controller**, this annotation is specifically designed for RESTful web services. Combining **@Controller** and **@ResponseBody**, it simplifies the development of RESTful APIs by automatically converting methods' return values to JSON or XML.

@RestController

```
@RequestMapping("/movies")
```

```
public class MoviesController {
```

```
    // ...
```

```
    @GetMapping("/{id}")
```

```
    public Movie getMovie(@PathVariable String id) {  
        return movieService.validateAndGetMovie(id);
```

```
    }
```

```
}
```

@RequestMapping

This annotation maps HTTP requests to handler methods within a controller. It specifies the URL patterns to handle and the methods to execute, providing a way to define the request-handling logic.

```
@RestController
```

```
@RequestMapping("/movies")
```

```
public class MoviesController {
```

```
    // ...
```

```
    @GetMapping("/{id}")
```

```
    public Movie getMovie(@PathVariable String id) {  
        return movieService.validateAndGetMovie(id);  
    }
```

```
}
```

@Autowired

Used for automatic dependency injection, it reduces the need for manual bean wiring. Whether applied to a constructor, field, or method, it signals that Spring should inject the required dependency.

@Controller

```
public class ProductController {
```

```
    @Autowired private ProductApiClient client;
```

```
    @GetMapping("/products")
```

```
    public String getProducts(Model model) {
```

```
        var products = client.listAllProducts();
```

```
        ...
```

```
    }
```

```
}
```

@Service

Marking a class as a service, this annotation is typically used for encapsulating business logic. It aids in component scanning and code organization, helping to structure an application with a dedicated service layer.

@Service

```
public class MovieServiceImpl implements MovieService {  
    @Autowired  
    private MovieRepository movieRepository;  
  
    @Override  
    public List<Movie> getMovies() {  
        return movieRepository.findAll();  
    }  
}
```

@Repository

Identified as a Spring Data repository, this annotation allows a class to interact with a data source, handling database operations and exceptions translation. It simplifies data access in a Spring application.

@Repository

```
public interface MovieRepository extends  
    CrudRepository<Movie, String> {  
}
```


@Configuration

Designating a class as a configuration class, this annotation defines application beans, replacing XML configuration with Java-based configuration. It plays an important role in configuring the Spring application context.

@Configuration

```
public class ObjectMapperConfig {
```

```
    @Bean
```

```
    public ObjectMapper objectMapper() {
```

```
        return new ObjectMapper()
```

```
            .configure(FAIL_ON_UNKNOWN_PROPERTIES, false);
```

```
    }
```

```
}
```

@Bean

This annotation is used to declare a method as a bean definition. This method produces a bean instance that Spring will manage within its container.

@Configuration

```
public class ObjectMapperConfig {
```

```
    @Bean
```

```
    public ObjectMapper objectMapper() {
```

```
        return new ObjectMapper()
```

```
            .configure(FAIL_ON_UNKNOWN_PROPERTIES, false);
```

```
    }
```

```
}
```

@Component

Serving as a generic stereotype annotation for any Spring-managed component, it marks a class as such, allowing it to be automatically detected and configured by Spring. It simplifies the process of declaring and managing Spring components.

@Component

```
public class NewsProducer {  
    public void send(NewsMessage message) {  
        // ...  
    }  
}
```

@Value

This annotation injects values from properties files or environment variables into fields, providing a clean and concise way to handle external configuration. It simplifies the retrieval of configuration values within the application.

@Configuration

```
public class MovieApiClientConfig {  
    @Value("${movie-api.url}")  
    private String movieApiUrl;  
    // ...  
}  
---
```

```
movie-api.url=http://localhost:9080
```

That's all

I hope you enjoy it!



Follow me on:

- LinkedIn: @ivanfranchin
- Twitter: @ivangfr
- GitHub: @ivangfr
- Medium: @ivangfr