Important Annotations for Spring Boot REST API and MVC Here's a comprehensive list of **important annotations** for **Spring Boot REST API and MVC**, categorized for ease of learning and relevance for a **3-year experience level**. --- ### **1. Core Spring Annotations** | **Annotation** | **Purpose** | |---|---| | `@SpringBootApplication` | Combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` for a Spring Boot application. | | `@Component` | Marks a class as a Spring-managed component (used for general-purpose beans). | | `@Service` | Marks a class as a service layer bean (used for business logic). | | `@Repository` | Indicates that the class is a DAO (Data Access Object) and encapsulates database interactions. | | `@Controller` | Marks a class as a web controller in Spring MVC. | | `@RestController` | Combines `@Controller` and `@ResponseBody` for RESTful APIs. | | `@Configuration` | Indicates a class contains Spring bean definitions. | | `@Bean` | Defines a Spring-managed bean. | | `@Lazy` | Initializes the bean only when it's first accessed. | --- ### **2. Web Annotations** | **Annotation** | **Purpose** | |---|---| | `@RequestMapping` | Maps HTTP requests to handler methods or classes (supports all HTTP methods). | | `@GetMapping` | Maps `GET` requests to a method. | | `@PostMapping` | Maps `POST` requests to a method. | | `@PutMapping` | Maps `PUT` requests to a method. | | `@DeleteMapping` | Maps `DELETE` requests to a method. | | `@PatchMapping` | Maps `PATCH` requests to a method (for partial updates). | | `@RequestParam` | Binds query parameters from the URL to method arguments. | | `@PathVariable` | Binds values from the URL path to method arguments. | | `@RequestBody` | Binds the HTTP request body to a Java object (typically used in `POST` or `PUT` methods). | | `@ResponseBody` | Indicates that the return value of a method should be written directly to the HTTP response body. | | `@RequestHeader` | Binds HTTP request headers to method arguments. | | `@CookieValue` | Binds a cookie value to a method parameter. | | `@CrossOrigin` | Enables Cross-Origin Resource Sharing (CORS) for REST APIs. | | `@SessionAttributes` | Used to store model attributes in an HTTP session. | | `@ModelAttribute` | Binds a method parameter or method return value to a model attribute. | --- ### **3. Validation Annotations** | **Annotation** | **Purpose** | |---|---| | `@Valid` | Triggers validation on an object annotated with constraints. | | `@NotNull` | Ensures the field is not `null`. | | `@NotBlank` | Ensures the field is not `null` or empty. | | `@Size` | Specifies size constraints for collections, strings, or arrays. | | `@Email` | Validates that a string is a valid email address. | | `@Pattern` | Validates a string against a regular expression. | | `@Min` / `@Max` | Specifies the minimum or maximum value of a numeric field. | | `@Past` / `@Future` | Ensures a date is in the past or future. | --- ### **4. Exception Handling Annotations** | **Annotation** | **Purpose** | |---|---| | `@ExceptionHandler` | Defines a method to handle specific exceptions thrown in controllers. | | `@ControllerAdvice` | Handles global exceptions across multiple controllers. | | `@ResponseStatus` | Sets a custom HTTP status code for exceptions or methods. | --- ### **5. Persistence and Transaction Management Annotations** | **Annotation** | **Purpose** | |---|---| | `@Entity` | Marks a class as a JPA entity (table in the database). | | `@Table` | Specifies the database table associated with an entity. | | `@Id` | Specifies the primary key of an entity. | | `@GeneratedValue` | Specifies the primary key generation strategy (e.g., `AUTO`, `IDENTITY`). | | `@Column` | Specifies the column mapping for a field. | | `@OneToOne` | Defines a one-to-one relationship between two entities. | | `@OneToMany` | Defines a one-to-many relationship between two entities. | | `@ManyToOne` | Defines a many-to-one relationship between two entities. | | `@ManyToMany` | Defines a many-to-many relationship between two entities. | | `@Transactional` | Indicates that a method or class is transactional. | --- ### **6. Security

Annotations** | **Annotation** | **Purpose** | |--------------------------|---| | `@EnableWebSecurity` | Enables Spring Security in a configuration class. | | `@PreAuthorize` | Specifies security conditions for method execution (e.g., `@PreAuthorize("hasRole('ADMIN')")`). | | `@PostAuthorize` | Performs security checks after a method is executed. | | `@Secured` | Restricts method access to specific roles (e.g., `@Secured("ROLE_USER")`). | | `@RolesAllowed` | Specifies roles allowed to access a method or class. | | `@AuthenticationPrincipal` | Accesses the currently authenticated user. | --- ### **7. Testing Annotations** | **Annotation** | **Purpose** | |--------------------------|---| | `@SpringBootTest` | Loads the full application context for integration tests. | | `@WebMvcTest` | Loads the Spring MVC components for controller testing. | | `@MockBean` | Mocks a Spring bean for testing purposes. | | `@Test` | Marks a method as a test case. | | `@BeforeEach` / `@AfterEach` | Executes methods before or after each test case. | | `@RunWith` | Specifies the test runner (e.g., `SpringRunner` for JUnit 4). | --- ### **8. Utility Annotations** | **Annotation** | **Purpose** | |--------------------------|---| | `@PropertySource` | Loads properties from an external file into the Spring environment. | | `@Value` | Injects a value into a field (from properties or environment variables). | | `@Autowired` | Injects dependencies automatically by type. | | `@Qualifier` | Specifies which bean to inject when multiple candidates exist. | | `@EnableAutoConfiguration` | Lets Spring Boot configure your app automatically. | | `@ConditionalOnProperty` | Enables or disables beans based on configurations. | | `@Profile` | Activates beans based on the app’s running profile. | | `@Async` | Executes tasks asynchronously to boost performance. | --- ### **Example Code for Utility Annotations** #### Example: Using `@Async` ```java import org.springframework.scheduling.annotation.Async; import org.springframework.stereotype.Service; @Service public class NotificationService { @Async public void sendEmail(String email) { try { Thread.sleep(5000); // Simulate delay System.out.println("Email sent to: " + email); } catch (InterruptedException e) { e.printStackTrace(); } } } ``` #### Example: Using `@ConditionalOnProperty` ```java import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty; import org.springframework.stereotype.Component; @Component @ConditionalOnProperty(name = "feature.enabled", havingValue = "true") public class FeatureComponent { public FeatureComponent() { System.out.println("FeatureComponent is enabled!"); } } ``` #### Example: Using `@Value` ```java import org.springframework.beans.factory.annotation.Value; import org.springframework.stereotype.Component; @Component public class AppConfig { @Value("${app.name}") private String appName; public void printAppName() { System.out.println("Application Name: " + appName); } } ``` --- Would you like further examples or clarification?