Spring Boot Annotations Cheat Sheet

🔹 Core Annotations

@SpringBootApplication

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

@ComponentScan

```
@SpringBootApplication
@ComponentScan(basePackages = "com.example.services")
public class Application {
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
}
```

@Configuration , @Bean

```
@Configuration
public class AppConfig {
   @Bean
   public DataSource dataSource() {
      return new DataSource();
   }
}
```

🔹 REST API & MVC

@RestController, @RequestMapping

```
@RestController
@RequestMapping("/users")
public class UserController {
   @GetMapping("/")
   public List<User> getUsers() {
      return userService.getAllUsers();
   }
}
```

@GetMapping

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable("id") Long id) {
   return userService.getUserById(id);
}
```

@PostMapping

```
@PostMapping("/users")
public ResponseEntity<User> createUser(@RequestBody User user) {
   userService.saveUser(user);
   return ResponseEntity.ok(user);
}
```

@PathVariable

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
   return userService.findUserById(id);
}
```

@RequestBody

```
@PostMapping("/users")
public void addUser(@RequestBody User user) {
   userService.save(user);
}
```

@ResponseBody

```
@RequestMapping("/api")
@ResponseBody
public String getApiData() {
   return "API Response Data";
}
```

@CrossOrigin

```
@RestController
@CrossOrigin(origins = "http://example.com")
public class MyController {
   @GetMapping("/data")
   public List<Data> getData() {
      return dataService.getData();
   }
}
```

🔹 Dependency Injection

@Autowired, @Service

```
@Service
public class UserService {
   @Autowired
   private UserRepository userRepository;
}
```

@Repository

```
@Repository
public class UserRepository extends JpaRepository<User, Long> {}
```

@Component

```
@Component
public class MyComponent {
   public void execute() {
      // logic here
   }
}
```

@Qualifier

```
@Autowired
@Qualifier("mysqlDataSource")
private DataSource dataSource;
```

@Primary

```
@Primary
@Bean
public DataSource primaryDataSource() {
   return new MySQLDataSource();
}
```

🔹 Database & JPA

@Entity

```
@Entity
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String name;
}
```

@Table

```
@Entity
@Table(name = "users")
public class User {
}
```

@Id, @GeneratedValue

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

@Column

```
@Column(name = "user_name")
private String name;
```

@OneToOne

```
@OneToOne
private Address address;
```

@OneToMany

```
@OneToMany(mappedBy = "user")
private List<Order> orders;
```

@ManyToOne

```
@ManyToOne
private User user;
```

@ManyToMany

```
@ManyToMany
private Set<Role> roles;
```

@Transactional

```
@Transactional
public void saveUser(User user) {
   userRepository.save(user);
}
```

🔹 Security

@PreAuthorize

```
@PreAuthorize("hasRole('ADMIN')")
public void performAdminAction() {
  // action for admins
}
```

@Secured

```
@Secured("ROLE_USER")
public void performUserAction() {
  // action for users
}
```

@RolesAllowed

```
@RolesAllowed({"ROLE_USER", "ROLE_ADMIN"})
public void accessRestricted() {
  // restricted action
}
```

@EnableWebSecurity

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/admin/**").hasRole("ADMIN");
  }
}
```

@EnableMethodSecurity

```
@EnableMethodSecurity
public class SecurityConfig extends GlobalMethodSecurityConfiguration {
  // enable method-level security
}
```

@AuthenticationPrincipal

```
@GetMapping("/user-profile")
public String getUserProfile(@AuthenticationPrincipal User user) {
   return user.getUsername();
}
```

🔹 Caching & Performance

@EnableCaching

```
@Configuration
@EnableCaching
public class CacheConfig {
}
```

@Cacheable

```
@Cacheable("users")
public User getUserById(Long id) {
   return userRepository.findById(id).orElseThrow();
}
```

@CachePut

```
@CachePut(value = "users", key = "#user.id")
public User updateUser(User user) {
   return userRepository.save(user);
}
```

@CacheEvict

```
@CacheEvict(value = "users", allEntries = true)
public void clearUserCache() { }
```

🔹 Configuration & Properties

@Value

```
@Value("${app.name}")
private String appName;
```

@ConfigurationProperties

```
@ConfigurationProperties(prefix = "app")
public class AppProperties {
   private String name;
   private String version;
}
```

@Profile

```
@Profile("dev")
@Bean
public DataSource devDataSource() {
   return new DevDataSource();
}
```

🔹 Testing

@SpringBootTest

```
@SpringBootTest
public class MyServiceTest {
   @Autowired
   private MyService myService;

   @Test
   public void testService() {
      assertNotNull(myService);
   }
}
```

@WebMvcTest

```
@WebMvcTest(UserController.class)
public class UserControllerTest {
   @Autowired
   private MockMvc mockMvc;

   @Test
   public void testGetUsers() throws Exception {
      mockMvc.perform(get("/users"))
            .andExpect(status().isOk());
   }
}
```

@MockBean

```
@SpringBootTest
public class MyServiceTest {
   @MockBean
   private UserRepository userRepository;

   @Autowired
   private MyService myService;

   @Test
   public void testMockBean() {
      when(userRepository.findById(1L)).thenReturn(Optional.of(new User()));
      User user = myService.getUserById(1L);
      assertNotNull(user);
   }
}
```

@WithMockUser

```
@WithMockUser(username = "admin", roles = "ADMIN")
@Test
public void testAdminAccess() {
   // test logic for admin access
}
```