SpringBoot Anti Patterns

1. Overusing *@Autowired*

- **Issue:** Using @Autowired excessively for dependencies can lead to tight coupling and make testing difficult.

- **Better Practice:** Prefer constructor injection over field injection to enforce immutability and support better testability.

Palmurugan C
@palmuruganc

2. Large *@RestController* Classes

- **Issue:** Combining too many responsibilities (e.g., validation, business logic, response transformation) in a single controller violates the Single Responsibility Principle.
- **Better Practice:** Delegate responsibilities to services, and keep controllers focused on routing and delegation.

Palmurugan C
@palmuruganc

3. Hardcoding *Property* Values

- **Issue:** Hardcoding sensitive or environment-specific configurations in code makes the application inflexible and difficult to deploy in multiple environments.

- **Better Practice:** Use application.properties or application.yml, and externalize secrets with tools like Spring Cloud Config or environment variables.

Palmurugan C
@palmuruganc

4. Ignoring *Transaction* Management

- **Issue:** Failing to use transactions (@Transactional) where necessary can lead to inconsistent data in databases during complex operations.

- **Better Practice:** Use transactions appropriately in service methods to ensure atomicity of database operations.

Palmurugan C
@palmuruganc

5. Using *@ComponentScan* Inefficiently

- **Issue:** Scanning too many packages (e.g., the root package) can slow application startup and introduce unintended beans into the context.
- **Better Practice:** Specify the exact packages to scan to improve performance and avoid surprises.

Palmurugan C
@palmuruganc

6. Overloading the *application.yml* File

- **Issue:** Putting everything into a single application.properties file leads to clutter, making configurations difficult to manage.

- **Better Practice:** Use profiles (application-dev.yml, application-prod.yml) and structured configuration files.

Palmurugan C
@palmuruganc

SpringBoot Anti Patterns

7. Improper Use of *@SpringBootApplication*

- **Issue:** Placing the @SpringBootApplication annotation in a deeply nested package can prevent Spring Boot from properly scanning beans.

- **Better Practice:** Place the main class in the root package of your application.

Palmurugan C
@palmuruganc

8. Incorrect *Error Handling*

- **Issue:** Throwing generic exceptions or letting exceptions propagate without proper handling leads to poor user experience.

- **Better Practice:** Use @ControllerAdvice for centralized exception handling and provide meaningful error responses.

Palmurugan C
@palmuruganc

9. Fetching *Too Much Data* with JPA

- **Issue:** Using *@OneToMany* or *@ManyToMany* without considering lazy/eager fetching can lead to performance issues like N+1 queries.

- **Better Practice:** Use *fetch = FetchType.LAZY* and explicitly fetch related entities only when needed.

Palmurugan C
@palmuruganc

10. Misusing *Caching*

- **Issue:** Using caching (@Cacheable, etc.) without understanding cache invalidation strategies can lead to stale data issues.

- **Better Practice:** Design proper cache strategies with clear policies for eviction and refresh.

Palmurugan C
@palmuruganc

Connect with me

"Stay ahead with cutting-edge technical tips and best practices—connect with me on LinkedIn today!"

@palmuruganc

Palmurugan C

@palmuruganc