Suraj Kumar

Java & Spring Boot

Spring Boot @Profile

Swipe for more

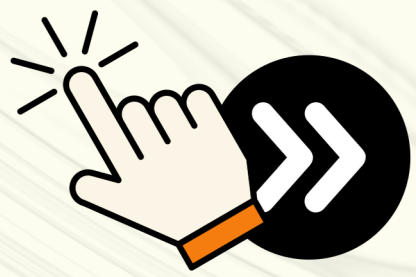
@Profile in Spring Boot

@Profile is a Spring Boot annotation that enables environment-specific bean configurations

Why Use It?

- Manage different configurations for dev, test, prod environments.
- Avoid hardcoding properties in the main application.
- Improve flexibility and maintainability.

Swipe for more



The Problem Without @Profile

What happens when we don’t use @Profile?

- **Hardcoded configurations** – Difficult to switch between environments.
- **Manual code changes** – Risk of deploying incorrect settings.
- **Poor maintainability** – More effort required for configuration management.

Swipe for more

Using @Profile in Spring Boot

Content:

- Define environment-specific beans using @Profile.
- Easily switch environments without modifying code.
- Maintain clean and modular configurations

Swipe for more

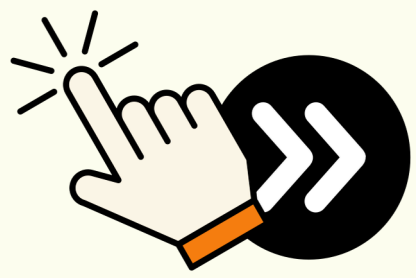
Create a Service Interface

```
public interface MessageService {
    String getMessage();
}
```

Explanation:

- Defines a common interface that different profile implementations will use.

Swipe for more



Define Beans for Each Profile

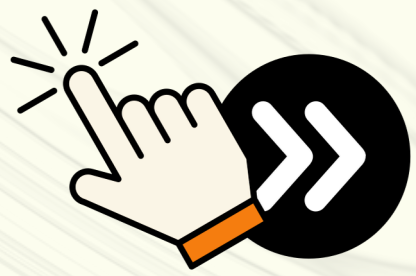
Development Profile (@Profile("dev"))

```
@Component
@Profile("dev")
public class DevMessageService implements MessageService {
    @Override
    public String getMessage() {
        return "Running in Development Mode!";
    }
}
```

Production Profile (@Profile("prod"))

```
@Component
@Profile("prod")
public class ProdMessageService implements MessageService {
    @Override
    public String getMessage() {
        return "Running in Production Mode!";
    }
}
```

Swipe for more



Injecting the Profile-Specific Bean

```
@RestController
public class ProfileController {
    private final MessageService messageService;

    @Autowired
    public ProfileController(MessageService messageService) {
        this.messageService = messageService;
    }

    @GetMapping("/profile")
    public String getProfileMessage() {
        return messageService.getMessage();
    }
}
```

Explanation:

- Spring Boot automatically injects the correct MessageService based on the active profile.

Swipe for more

Configuring Profiles in application.yml

```yaml
spring:
  profiles:
    active: dev

---
spring:
  config:
    activate:
      on-profile: dev
message: "Welcome to Development Mode"
---
spring:
  config:
    activate:
      on-profile: prod
message: "Welcome to Production Mode"
```

Explanation:

- spring.profiles.active=dev sets the default active profile.
- The application will load profile-specific configurations automatically.

Swipe for more

Switching Profiles at Runtime

Using Command Line Arguments:

- mvn spring-boot:run -Dspring-boot.run.profiles=prod

Setting in Environment Variables:

- export SPRING_PROFILES_ACTIVE=prod

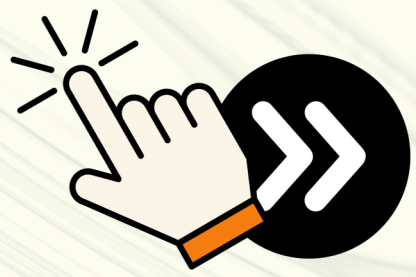
Passing as JVM Argument:

- java -jar app.jar --spring.profiles.active=prod

Explanation:

- Easily switch between profiles without modifying code.

Swipe for more



Summary & Best Practices

- Always use @Profile for managing multiple environments.
- Keep configuration clean using application.yml.
- Use @Profile for database connections, API endpoints, and feature toggles.
- Avoid manually changing environment settings in code.

Swipe for more

Suraj Kumar

If you find this helpful, please like and share it with your friends