

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
<div>Spring Boot<ul style="list-style-type: none"><li>Introduction to Spring Boot</li><li>Multi-Tier Architecture</li><li>Beans</li><li>Bean Scopes</li><li>Bean Validation</li><li>Dependency Injection</li><li>Components</li><li>Configuration</li><li>Connecting to a Database</li><li>Entities</li><li>Postman</li><li>Controllers</li><li>Services</li><li>Repositories</li><li>Custom Queries</li><li>Data Transfer Objects</li><li>Lombok</li><li>Custom Exceptions</li><li>Swagger</li><li>Profiles</li><li>Pre-Populating Databases for Testing</li><li>Unit testing with Mockito</li></ul></div>

# Profiles

## Contents

- Overview
  - Property files
  - Setting the active profile
  - Making components profile-specific
- Tutorial
- Exercises

## Overview

Spring profiles provide two major functions: allowing for multiple project configurations through multiple property files, and allowing us to control which beans are created.

## Property files

Spring profiles are set up via property files; like those seen in the Configuration module.

Rather than using a single application.properties/yaml file, we can create files for specific profiles by following the application-<profile-name>.properties format.

The below shows example properties files for a development profile (application-dev.properties):

```
spring.h2.console.enabled=true
server.port=8081

spring.jpa.hibernate.ddl-auto=create-drop
```

And an example properties file for production (application-prod.properties):

```
server.port = 8181
spring.data.rest.base-path=/api

spring.jpa.hibernate.ddl-auto=validate

spring.datasource.url=jdbc:mysql://33.222.178.42:3306/ducks
spring.datasource.username=root
spring.datasource.password=notpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.datasource.initialization-mode=always
```

## Setting the active profile

The active profile can be set in application.properties using spring.profiles.active:

```
spring.profiles.active=dev
```

Alternatively, the active profile can be set using @ActiveProfiles in test classes:

<div><div></div><div>Testing</div></div>
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
@SpringBootTest
@ActiveProfiles("test")
public class ExampleTest {
    //
}
```

## Making components profile-specific

To set when a component will be created, we use the `@Profile` annotation.

Here, we've set our `AppConfig` file to be loaded in when accessed by both the `dev` and `prod` profiles:

```
@Configuration
@Profile({"dev", "prod"})
public class AppConfig {

    @Bean
    public ModelMapper mapper() {
        return new ModelMapper();
    }
}
```

`@Profile` can also be used to set when *not* to create a component.

Here, we've ensured that this configuration file is never used with the `test` profile:

```
@Configuration
@Profile("!test")
public class AppConfig {

    @Bean
    public ModelMapper mapper() {
        return new ModelMapper();
    }
}
```

## Tutorial

There is no tutorial for this module.

## Exercises

- Create `dev` and `prod` profiles for your project.
- Make the `dev` profile point at an *h2 database*.
- Make the `prod` profile connect to a MySQL database on GCP - if you're unsure of how to do this, a tutorial can be found in the [GCP MySQL Database module](#).)