

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
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>

# Services

## Contents

- [Overview](#)
- [Tutorial](#)
- [Exercises](#)

## Overview

Spring **Service** components are used to store the main *business logic* of a Spring Boot application.

In Spring, services are annotated with `@Service`.  
`@Service` is functionally identical to `@Component`, the only difference is that `@Service` shows that you *intend* to use the class as a service.

## Tutorial

Carrying on from the example in the [Controllers Module](#) we will be moving our `people List`, and the relevant CRUD functionality, into a `PersonService`. This is a more appropriate place for it than the controller as business logic should be implemented in `@Service` classes.

Let's start with creating the `PersonService` in an appropriate package (I will be using `com.qa.demo.services`).

```
@Service
public class PersonService {

}
```

Now we can migrate the `people List` and the relevant methods into our service:

*The mappings and other annotations will stay on the controller's methods as the service is not exposed to external access*

<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

```
@Service
public class PersonService {

    private List<Person> people = new ArrayList<>();

    public Person addPerson(Person person) {
        // Add new Person
        this.people.add(person);
        // Return Last added Person from List
        return this.people.get(this.people.size() - 1);
    }

    public List<Person> getAllPeople() {
        // Return the whole List
        return this.people;
    }

    public Person updatePerson(int id, Person person) {
        // Remove existing Person with matching 'id'
        this.people.remove(id);
        // Add new Person in its place
        this.people.add(id, person);
        // Return updated Person from List
        return this.people.get(id);
    }

    public Person removePerson(int id) {
        // Remove Person and return it
        return this.people.remove(id);
    }

}
```

Now all that's left to do is simply update the **PersonController** to use the **PersonService** to handle the CRUD functionality.

Start with adding the **PersonService** dependency:

```
@RestController
public class PersonController {

    private PersonService service;

    public PersonController(PersonService service) {
        super();
        this.service = service;
    }

    // CRUD methods
}
```

Now we'll strip out the implementation of the CRUD functionality we created previously and replace it with calls to our new service:

```
@PostMapping("/create")
public Person addPerson(@RequestBody Person person) {
    return this.service.addPerson(person);
}

@GetMapping("/getAll")
public List<Person> getAllPeople() {
    return this.service.getAllPeople();
}

@PutMapping("/update")
public Person updatePerson(@PathParam("id") int id, @RequestBody Person
person) {
    return this.service.updatePerson(id, person);
}

@DeleteMapping("/delete/{id}")
public Person removePerson(@PathVariable int id) {
    return this.service.removePerson(id);
}
```

And that's the service layer!

Now try it out in Postman to make sure it all works!

## Exercises

---

Add an **AccountService** to your Account project and update your **AccountController** to work with it.