Creating a Service

We'll cover the following

Defining a service

In a simple example such as the one we're looking at, we may skip the service and directly let the controller talk to the repository. However, introducing a service will help us see how Kotlin can be used to create a service. And if we decide to expand the example, to add more behavior, then that can readily go into the service.

The service will sit in between the controller and the database and take care of making all the calls necessary to manipulate the persistent data. The service needs to talk to the repository; but no worries, Spring can auto-wire that dependency in a blink. We need a method to get all the tasks—one to save a new task, and one to delete a task with a given `id`.

Kotlin vs. Java

Given all that, the Java version is sure to be more verbose than the Kotlin version. If we were to write in Java, we'd end up with something like this:

```
//Java code only for comparison purpose
package com.agiledeveloper.todo;

import java.util.Optional;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
class TaskService {
  private final TaskRepository repository;

  public TaskService(TaskRepository repository) {
    this.repository = repository;
  }
```

```
  Iterable<Task> getAll() {
    return repository.findAll();
  }


  Task save(Task task) {
    return repository.save(task);
  }

  boolean delete(Long id) {
    boolean found = repository.existsById(id);

    if (found) {
      repository.deleteById(id);
    }

    return found;
  }
}
```

Using Kotlin we can shave off some noise. Let's start by creating the file `todo/src/main/kotlin/com/agiledeveloper/todo/TaskService.kt` and update the code to:

```
package com.agiledeveloper.todo

import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional

@Service
@Transactional
class TaskService(val repository: TaskRepository) {
  fun getAll() = repository.findAll()

  fun save(task: Task) = repository.save(task)

  fun delete(id: Long):  Boolean {
    val found = repository.existsById(id)

    if (found) {
      repository.deleteById(id)
    }

    return found
  }
}
```

TaskService.kt

The `TaskService` class is annotated with `@Service` and `@Transactional`. In the class, the `getAll()` method returns the result of the synthesized method `findAll()` of the `TaskRepository`. The `save()` method takes an instance of the entity `Task` and sends it off to the `TaskRepository`'s `save()` method so it's inserted into the database. Finally, the `delete()` method looks up the given `id`. If an object with that `id` exists, then it's removed from the database, using the `TaskRepository`'s

that `id` exists, then it's removed from the database, using the `TaskRepository`'s `deleteById()` method, and `true` is returned as the result of this call. If an object with that `id` wasn't found, then `false is` returned.

As a last step, we need to integrate the service with the controller, which we'll do in the next lesson.