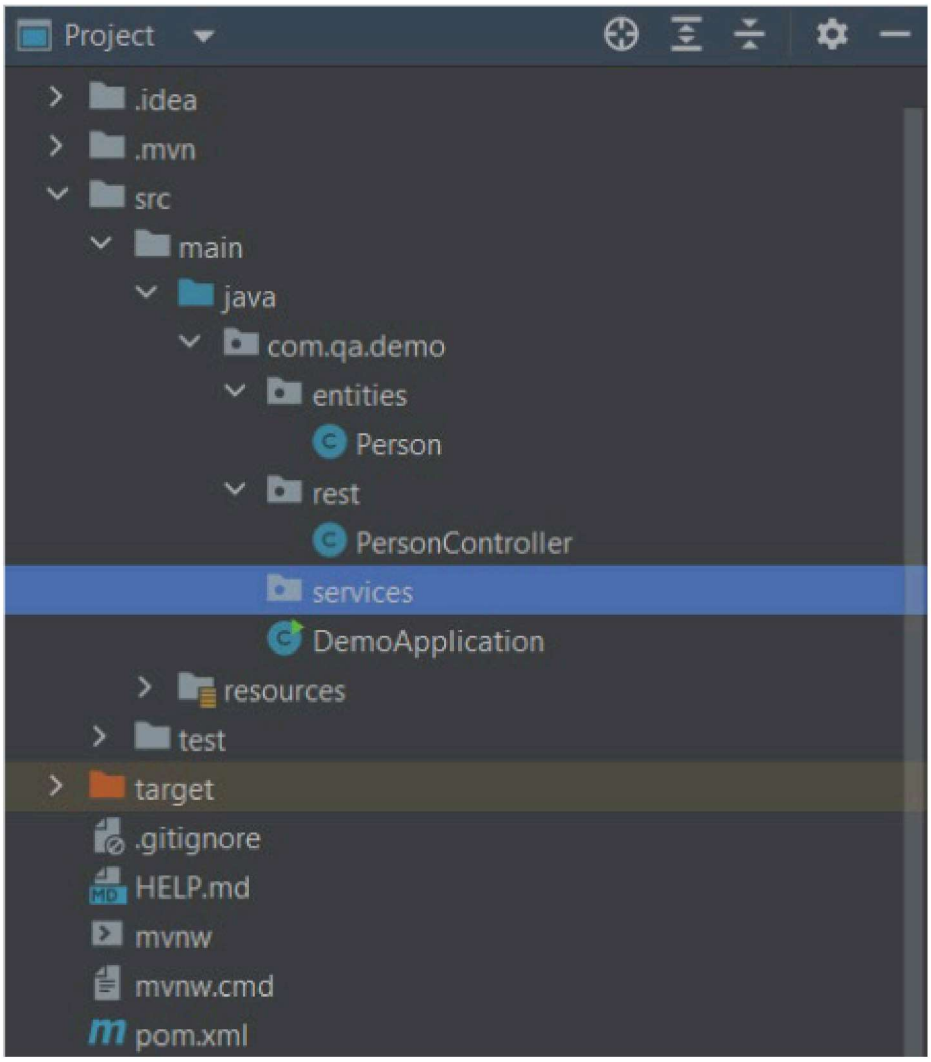
Lab 2 – Setting up the Service

The purpose of this task is to move our business logic from the controller into its own Service class.

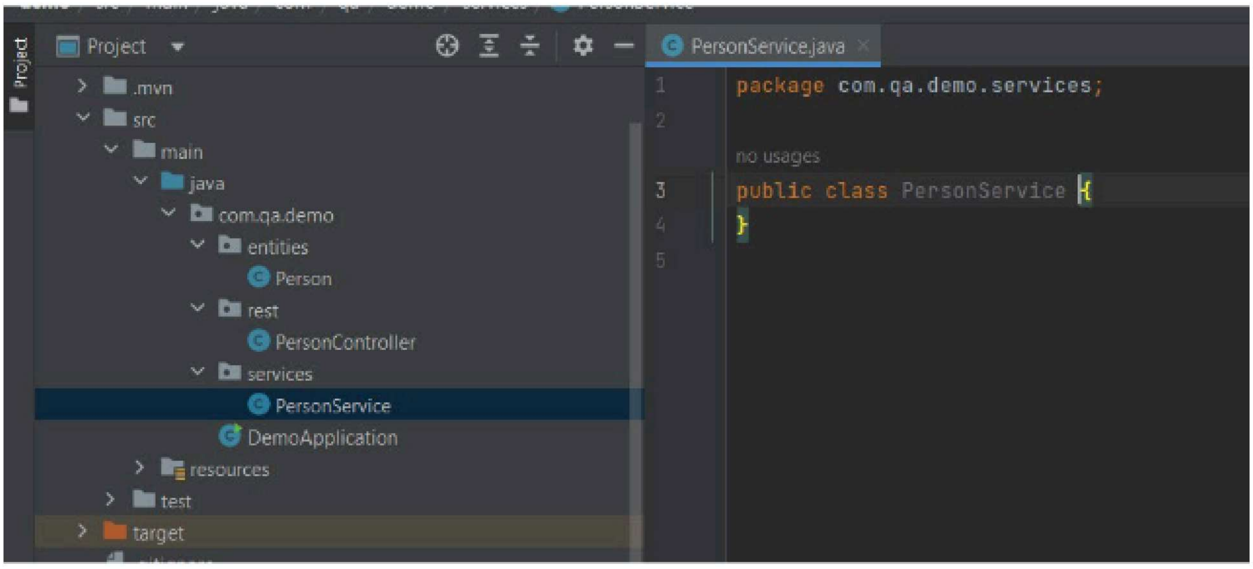
Task 1

1. Open the project you created in the previous lab.

2. Create a new package in **com.qa.demo** called **services.**

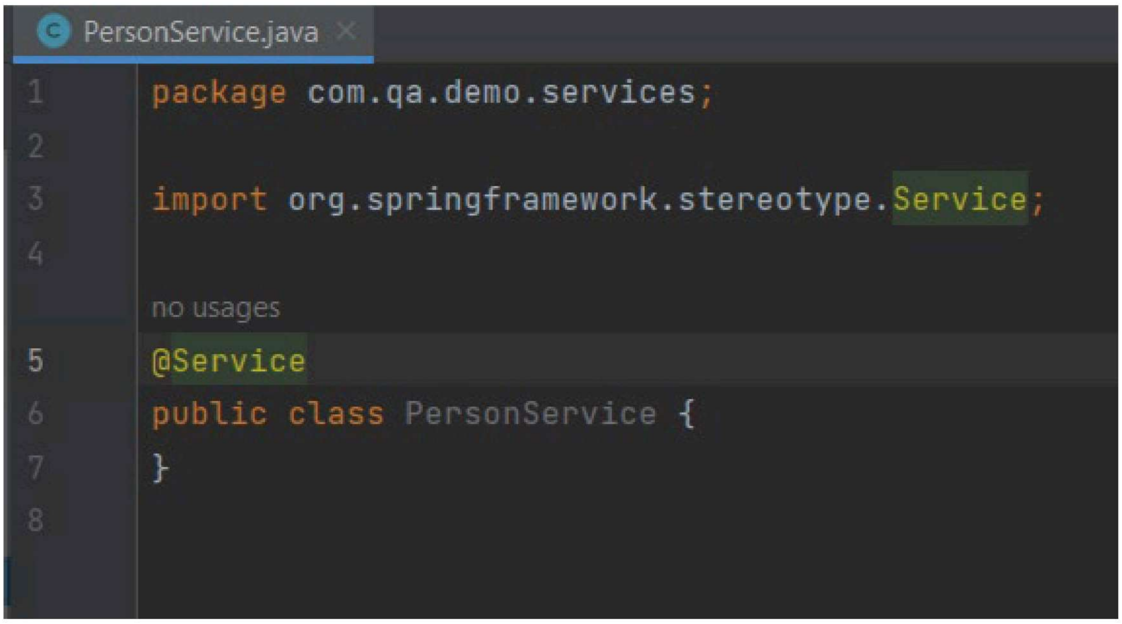


3. Inside the new package create a class **PersonService**.



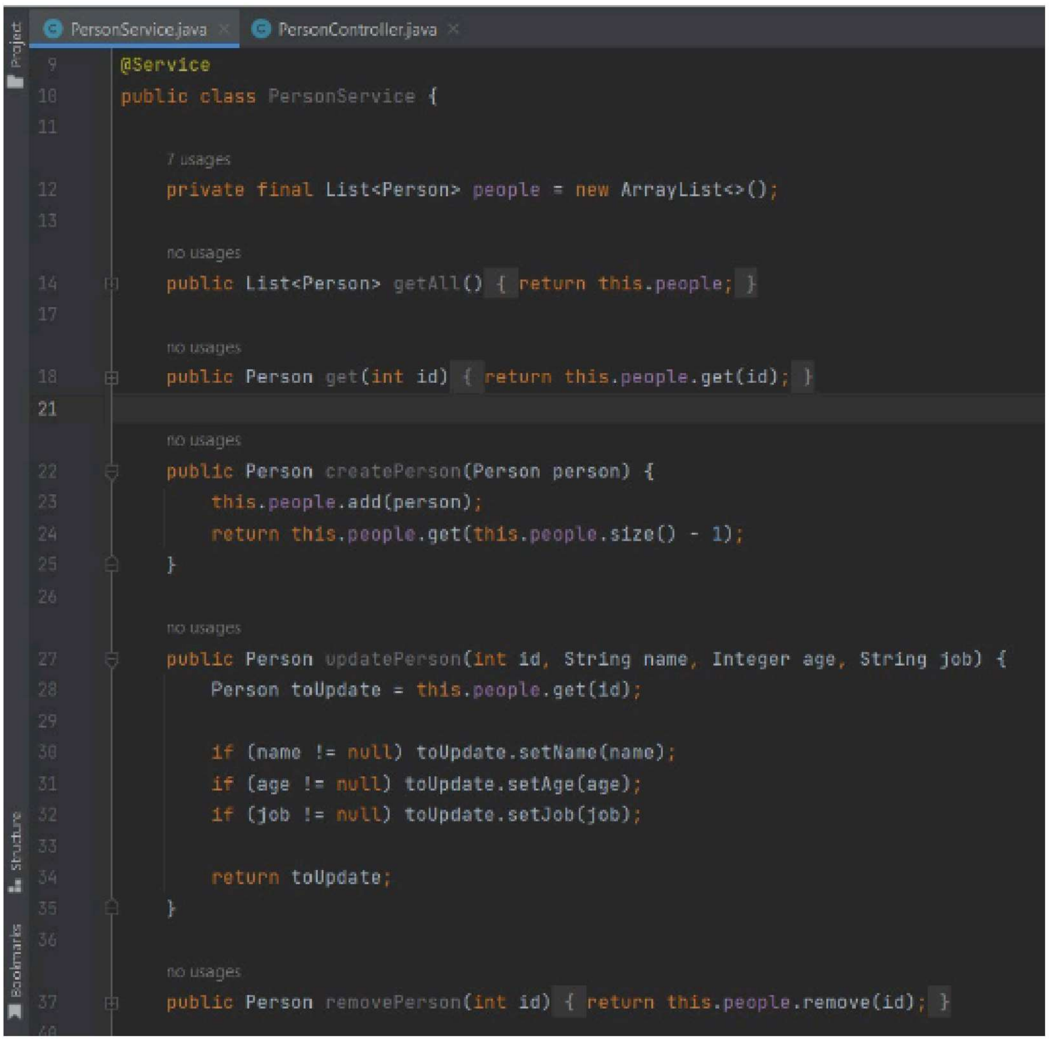
```java
package com.qa.demo.services;

no usages
public class PersonService {
}
```

4. Annotate this class as a **@Service**.



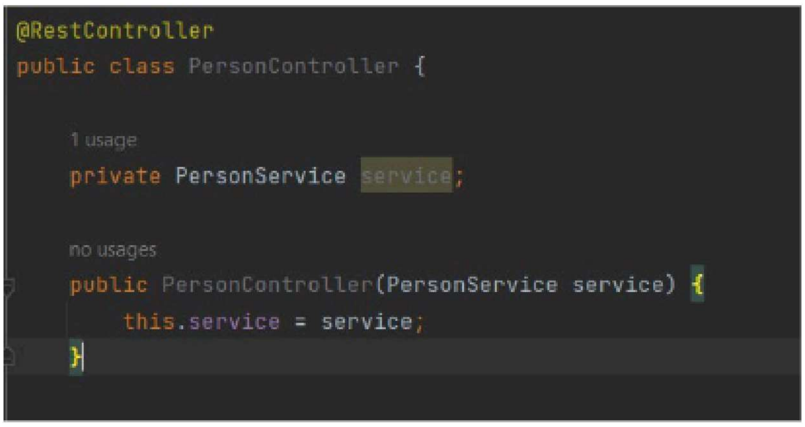
```java
package com.qa.demo.services;

import org.springframework.stereotype.Service;

no usages
@Service
public class PersonService {
}

```

5. Copy the **List** and all of the methods across from the **PersonController** into the **PersonService** *without any of the annotations.*



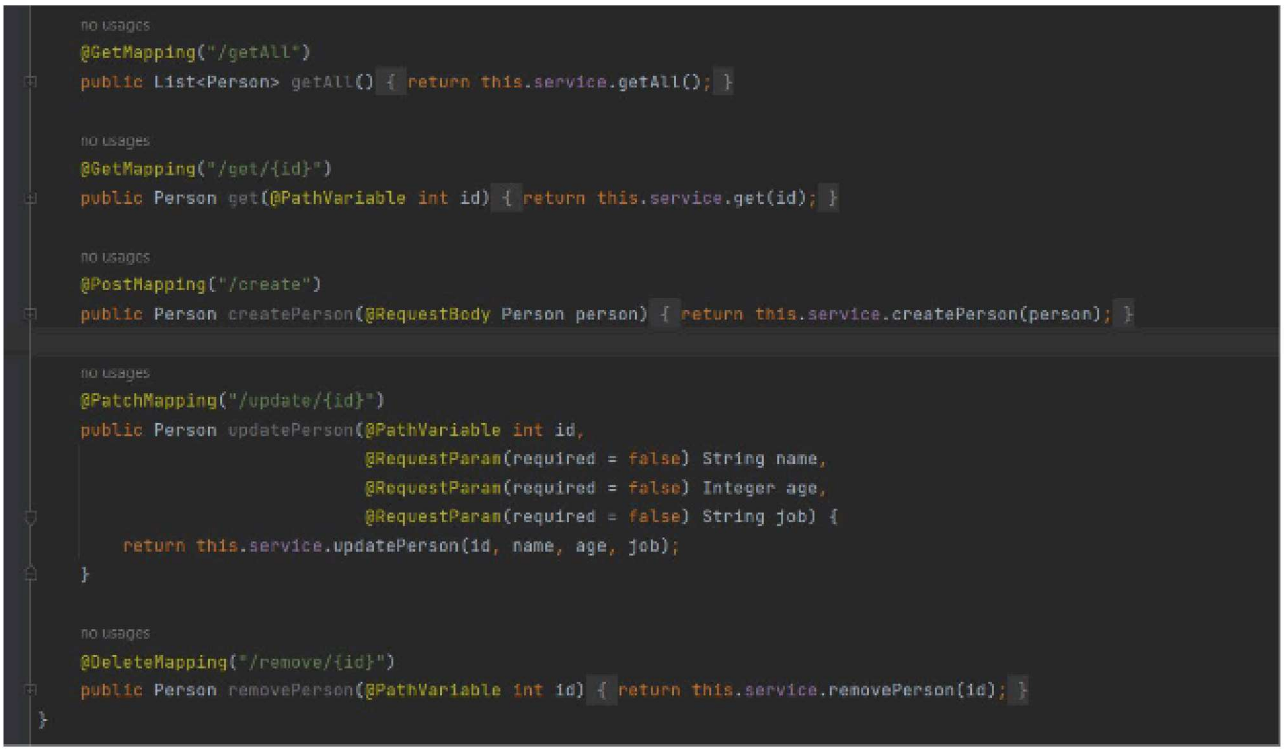
```
PersonService.java    PersonController.java
@Service
public class PersonService {

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

    no usages
    public List<Person> getAll() { return this.people; }

    no usages
    public Person get(int id) { return this.people.get(id); }

    no usages
    public Person createPerson(Person person) {
        this.people.add(person);
        return this.people.get(this.people.size() - 1);
    }

    no usages
    public Person updatePerson(int id, String name, Integer age, String job) {
        Person toUpdate = this.people.get(id);

        if (name != null) toUpdate.setName(name);
        if (age != null) toUpdate.setAge(age);
        if (job != null) toUpdate.setJob(job);

        return toUpdate;
    }

    no usages
    public Person removePerson(int id) { return this.people.remove(id); }
```

6. Now that all the logic exists in the service class, we can remove it from the controller. Start by injecting the **PersonService** into the **PersonController** using constructor injection and removing the **people** list.



```
@RestController
public class PersonController {

    1 usage
    private PersonService service;

    no usages
    public PersonController(PersonService service) {
        this.service = service;
    }
```

7. Replace the existing method bodies with calls to the service method of the same name.



```java
    no usages
    @GetMapping("/getAll")
    public List<Person> getAll() { return this.service.getAll(); }

    no usages
    @GetMapping("/get/{id}")
    public Person get(@PathVariable int id) { return this.service.get(id); }

    no usages
    @PostMapping("/create")
    public Person createPerson(@RequestBody Person person) { return this.service.createPerson(person); }

    no usages
    @PatchMapping("/update/{id}")
    public Person updatePerson(@PathVariable int id,
                               @RequestParam(required = false) String name,
                               @RequestParam(required = false) Integer age,
                               @RequestParam(required = false) String job) {
        return this.service.updatePerson(id, name, age, job);
    }

    no usages
    @DeleteMapping("/remove/{id}")
    public Person removePerson(@PathVariable int id) { return this.service.removePerson(id); }
}
```

8. Use the Swagger docs to check all of your endpoints still work the way you'd expect – if so, then we have successfully separated our business logic from the request handling.