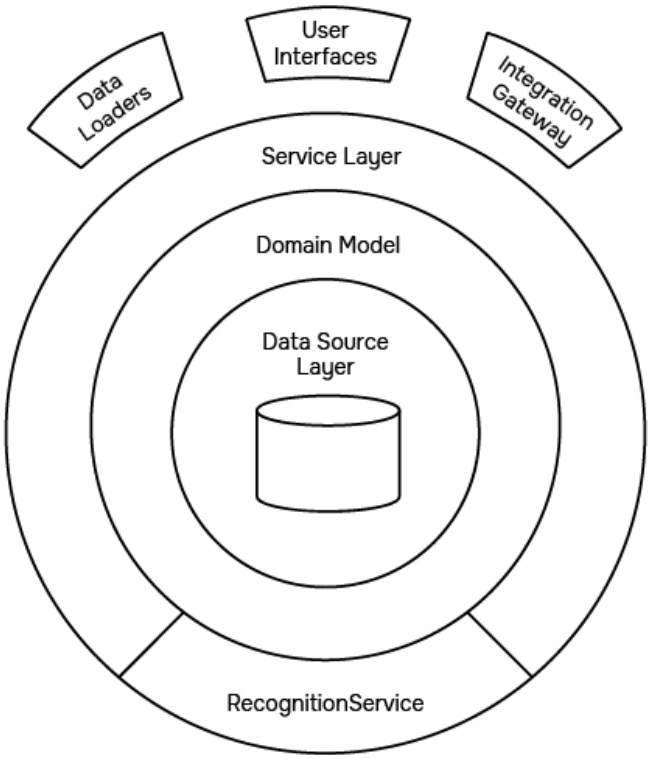
Слой сервисов

С ростом проекта, в контроллерах становится больше кода — он начинает повторяться. Одни и те же сценарии появляются в разных местах, что приводит к необходимости использовать код повторно. Для решения этой задачи в Spring Boot используется слой сервисов (Service Layer):



Понятие «слой» указывает, как строятся зависимости между разными уровнями приложения. Конкретно слой сервисов строится поверх моделей. Он объединяет в себе их взаимодействие в рамках каких-то бизнес-сценариев: от простых CRUD-операций до закрытия счета, перевода денег между счетами и других комплексных взаимодействий.

В Spring Boot сервисы представлены классами с аннотацией `@Service`. Каждый класс обычно отвечает за связанный набор бизнес-сценариев, где каждый сценарий — это метод. В простейшем случае сервисы создаются по сущностям — например, `PostService` или `UserService`. Но могут быть и более сложные варианты — например, `PaymentService`, который часто не привязан к конкретной сущности.

Вот так может выглядеть сервис для постов *src/main/io/hexlet/spring/service/PostService.java*:

```java
// src/main/io/hexlet/spring/service/PostService.java
package io.hexlet.blog.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import io.hexlet.blog.dto.PostCreateDTO;
import io.hexlet.blog.dto.PostDTO;
import io.hexlet.blog.dto.PostUpdateDTO;
import io.hexlet.blog.exception.ResourceNotFoundException;
import io.hexlet.blog.mapper.PostMapper;
import io.hexlet.blog.repository.PostRepository;
import io.hexlet.blog.util.UserUtils;

@Service
public class PostService {
    @Autowired
    private PostRepository repository;

    @Autowired
    private PostMapper postMapper;

    @Autowired
    private UserUtils userUtils;

    public List<PostDTO> getAll() {
        var posts = repository.findAll();
        var result = posts.stream()
                .map(postMapper::map)
                .toList();
        return result;
    }

    public PostDTO create(PostCreateDTO postData) {
        var post = postMapper.map(postData);
        post.setAuthor(userUtils.getCurrentUser());
        repository.save(post);
        var postDTO = postMapper.map(post);
        return postDTO;
    }
```

```
    public PostDTO findById(Long id) {
        var post = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found: " + id));
        var postDTO = postMapper.map(post);
        return postDTO;
    }

    public PostDTO update(PostUpdateDTO postData, Long id) {
        var post = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found"));
        postMapper.update(postData, post);
        repository.save(post);
        var postDTO = postMapper.map(post);
        return postDTO;
    }

    public void delete(Long id) {
        repository.deleteById(id);
    }
}
```

И его использование:

```
package io.hexlet.spring.controller.api;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import io.hexlet.blog.service.PostService;
import io.hexlet.blog.dto.PostDTO;

@RestController
@RequestMapping("/api")
public class PostsController {
```

```
    @Autowired
    private PostService postService;

    @GetMapping("/posts")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<List<PostDTO>> index() {
        var posts = postService.getAll();

        return ResponseEntity.ok()
                .header("X-Total-Count", String.valueOf(posts.size()))
                .body(posts);
    }
}
```

На примере этого сервиса видно, что сервисы отвечают за:

- Взаимодействием между сущностями и их отображением на базу
- Преобразованием из DTO и в DTO
- Работу с исключениями, их генерацию и обработку

Обратите внимание, что сервисы не отвечают за логику приложения (Application Logic). Она включает в себя все, что касается работы самого фреймворка с точки зрения HTTP — например, работа с HTTP-запросом или HTTP-ответом. Сервисы не должны читать и устанавливать заголовки, выполнять редиректы и тому подобное. Все это должно делаться снаружи. В идеальной ситуации сервисы могут быть использованы не только в контроллерах, но и в CLI- интерфейсе.

Когда имеет смысл вводить сервисы? Многие программисты предпочитают любую логику добавлять сразу в сервисы. Несмотря на это, важно помнить, что любой новый слой добавляет сложность, которая может быть больше, чем получаемая выгода.

Дополнительные материалы

1. Spring Validation in the Service Layer

Далее →