CRUD на Spring Boot

- Создание сущности
- Создание репозитория
- Создание DTO
- Создание маппера
- Создание теста
- Реализация контроллера

Типовая задача в веб-разработке — это создание API для CRUD сущности.

CRUD (*Create/Read/Update/Delete*) — это набор типовых операций, который обычно выполняется над сущностью. В этом уроке мы соберем все изученное и создадим эталонный CRUD на примере поста в блог.

Общий план создания CRUD выглядит так:

1. Создаем сущность
2. Создаем репозиторий
3. Создаем DTO
4. Создаем маппер
5. Пишем тест
6. Реализуем контроллер

Создание сущности

У поста в блоге есть четыре основных свойства:

- Название
- Слаг
- Текст
- Автор

Кроме того, еще можно добавить дату создания и последнего обновления:

```
package io.hexlet.spring.model;
```

```java
import static jakarta.persistence.GenerationType.IDENTITY;

import java.time.LocalDate;

import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Entity
@Getter
@Setter
@EntityListeners(AuditingEntityListener.class)
@ToString(includeFieldNames = true, onlyExplicitlyIncluded = true)
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Table(name = "posts")
// Позже мы обсудим BaseEntity подробнее
public class Post implements BaseEntity {
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @ToString.Include
    @EqualsAndHashCode.Include
    private Long id;

    @ManyToOne
    @NotNull
    private User author;
```

```
    @Column(unique = true)
    @ToString.Include
    @NotNull
    private String slug;

    @NotBlank
    @ToString.Include
    private String name;

    @NotBlank
    @ToString.Include
    @Column(columnDefinition = "TEXT")
    private String body;

    @LastModifiedDate
    private LocalDate updatedAt;

    @CreatedDate
    private LocalDate createdAt;
}
```

Создание репозитория

Использование слага в URL-адресе подразумевает, что мы сможем делать выборку сущности по слагу:

```
postRepository.findBySlug(/* slug */);
```

Сразу добавим этот метод в репозиторий:

```
package io.hexlet.spring.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import io.hexlet.spring.model.Post;

@Repository
```

```java
public interface PostRepository extends JpaRepository<Post, Long> {
    Optional<Post> findBySlug(String slug);
}
```

Создание DTO

В целом нам понадобятся три разных DTO — для создания, обновления и просмотра поста:

```java
// PostCreateDTO
package io.hexlet.spring.dto;

import jakarta.validation.constraints.NotNull;
import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class PostCreateDTO {
    @NotNull
    private Long authorId;

    @NotNull
    private String slug;

    @NotNull
    private String name;

    @NotNull
    private String body;
}
```

```java
// PostDTO
package io.hexlet.spring.dto;

import java.time.LocalDate;

import lombok.Getter;
import lombok.Setter;
```

```java
@Setter
@Getter
public class PostDTO {
    private Long id;
    private Long authorId;
    private String slug;
    private String name;
    private String body;
    private LocalDate createdAt;
}

// PostUpdateDTO
package io.hexlet.spring.dto;

import org.openapitools.jackson.nullable.JsonNullable;

import jakarta.validation.constraints.NotNull;
import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class PostUpdateDTO {
    @NotNull
    private JsonNullable<Long> authorId;

    @NotNull
    private JsonNullable<String> slug;

    @NotNull
    private JsonNullable<String> name;

    @NotNull
    private JsonNullable<String> body;
}
```

Создание маппера

Для CRUD нам нужны три операции:

- Конвертация DTO для создания в пост

- Конвертация поста в DTO для просмотра
- Обновление поста на основе DTO для обновления

```
package io.hexlet.spring.mapper;

import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.MappingConstants;
import org.mapstruct.MappingTarget;
import org.mapstruct.NullValuePropertyMappingStrategy;
import org.mapstruct.ReportingPolicy;

import io.hexlet.spring.dto.PostCreateDTO;
import io.hexlet.spring.dto.PostDTO;
import io.hexlet.spring.dto.PostUpdateDTO;
import io.hexlet.spring.model.Post;

@Mapper(
    uses = { JsonNullableMapper.class, ReferenceMapper.class },
    nullValuePropertyMappingStrategy = NullValuePropertyMappingStrategy.IGNORE,
    componentModel = MappingConstants.ComponentModel.SPRING,
    unmappedTargetPolicy = ReportingPolicy.IGNORE
)
public abstract class PostMapper {
    @Mapping(target = "author", source = "authorId")
    public abstract Post map(PostCreateDTO dto);

    @Mapping(source = "author.id", target = "authorId")
    public abstract PostDTO map(Post model);

    @Mapping(target = "author", source = "authorId")
    public abstract void update(PostUpdateDTO dto, @MappingTarget Post model);
}
```

Самое интересное в этом коде — это конвертация `authorId` из DTO в свойство `author` внутри поста. Чтобы выполнить эту операцию, нужно сделать запрос в базу данных и извлечь объект автора. По умолчанию MapStruct такого не умеет, но для него можно создать маппер, который решает эту задачу. Здесь мы сразу приведем его код:

```
package io.hexlet.spring.mapper;
```

```java
import org.mapstruct.Mapper;
import org.mapstruct.MappingConstants;
import org.mapstruct.TargetType;
import org.springframework.beans.factory.annotation.Autowired;

import io.hexlet.spring.model.BaseEntity;
import jakarta.persistence.EntityManager;

@Mapper(
    componentModel = MappingConstants.ComponentModel.SPRING
)
public abstract class ReferenceMapper {
    @Autowired
    private EntityManager entityManager;

    public <T extends BaseEntity> T toEntity(Long id, @TargetType Class<T> entityClass) {
        return id != null ? entityManager.find(entityClass, id) : null;
    }
}
```

Чтобы этот код заработал, необходимо внедрить общий базовый интерфейс для всех моделей, на который маппер мог бы ориентироваться и понимать, применять метод `toEntity()`. Мы назовем его `BaseEntity`:

```java
package io.hexlet.blog.model;

// По желанию его можно заполнять
// С точки зрения маппера важно только его наличие
public interface BaseEntity {
}
```

Определение класса `Post` в таком случае выглядит так:

```java
public class Post implements BaseEntity {
    // Код класса
}
```

Создание теста

Код интеграционных тестов не завязан на устройство контроллера. Поэтому сам тест можно написать до реализации контроллера — так мы упростим создание контроллера и проверку его работоспособности. Такой подход называется **TDD** (*Test Driven Development*):

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

import static net.javacrumbs.jsonunit.assertj.JsonAssertions.assertThatJson;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.springframework.security.test.web.servlet.request.SecurityMockMvcRequest
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.instancio.Instancio;
import org.instancio.Model;
import org.instancio.Select;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openapitools.jackson.nullable.JsonNullable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.security.test.web.servlet.request.SecurityMockMvcRequestPostPro
import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.ObjectMapper;

import io.hexlet.blog.model.Post;
import io.hexlet.blog.model.User;

import io.hexlet.spring.dto.PostUpdateDTO;
import io.hexlet.spring.mapper.PostMapper;
import io.hexlet.spring.repository.PostRepository;
import io.hexlet.spring.util.ModelGenerator;
import net.datafaker.Faker;

@SpringBootTest
```

```java
@AutoConfigureMockMvc
public class PostsControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper om;

    @Autowired
    private PostMapper postMapper;

    @Autowired
    private PostRepository postRepository;

    // Нужно создать бин
    @Autowired
    private Faker faker;

    private User testUser;
    private Post testPost;

    @BeforeEach
    public void setUp() {
        testUser = Instancio.of(User.class)
            .ignore(Select.field(Post::getId))
            .supply(Select.field(User::getEmail), () -> faker.internet().emailAddress())
            .create();

        testPost = Instancio.of(Post.class)
            .ignore(Select.field(Post::getId))
            .supply(Select.field(Post::getName), () -> faker.gameOfThrones().house())
            .supply(Select.field(Post::getBody), () -> faker.gameOfThrones().quote())
            .supply(Select.field(Post::getAuthor), testUser)
            .create();
    }

    @Test
    public void testIndex() throws Exception {
        postRepository.save(testPost);

        var result = mockMvc.perform(get("/api/posts")
```

```
                .andExpect(status()
                .isOk())
                .andReturn();
        var body = result.getResponse().getContentAsString();
        assertThatJson(body).isArray();
    }

    @Test
    public void testCreate() throws Exception {
        var dto = postMapper.map(testPost);

        var request = post("/api/posts")
                .contentType(MediaType.APPLICATION_JSON)
                .content(om.writeValueAsString(dto));

        mockMvc.perform(request)
                .andExpect(status().isCreated());

        var post = postRepository.findBySlug(dto.getSlug()).get();
        assertNotNull(post);
        assertThat(post.getName()).isEqualTo(dto.getName());
    }

    @Test
    public void testUpdate() throws Exception {
        postRepository.save(testPost);

        var dto = new PostUpdateDTO();
        dto.setName(JsonNullable.of("new name"));

        var request = put("/api/posts/" + testPost.getId())
                .contentType(MediaType.APPLICATION_JSON)
                .content(om.writeValueAsString(dto));

        mockMvc.perform(request)
                .andExpect(status().isOk());

        post = postRepository.findById(testPost.getId()).get();
        assertThat(post.getName()).isEqualTo(dto.getName());
    }

    @Test
```

```
    public void testShow() throws Exception {
        postRepository.save(testPost);

        var request = get("/api/posts/" + testPost.getId());
        var result = mockMvc.perform(request)
                .andExpect(status().isOk())
                .andReturn();
        var body = result.getResponse().getContentAsString();
        assertThatJson(body).and(
            v -> v.node("slug").isEqualTo(testPost.getSlug()),
            v -> v.node("name").isEqualTo(testPost.getName()),
            v -> v.node("body").isEqualTo(testPost.getBody())
        );
    }

    @Test
    public void testDestroy() throws Exception {
        postRepository.save(testPost);
        var request = delete("/api/posts/" + testPost.getId());
        mockMvc.perform(request)
                .andExpect(status().isNoContent());

        assertThat(postRepository.existsById(testPost.getId())).isEqualTo(false);
    }
}
```

Реализация контроллера

Перейдем к контроллеру. Здесь мы не добавляем ничего нового. Весь его код мы видели частями, а теперь собираем все вместе:

```
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.DeleteMapping;
```

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import io.hexlet.spring.dto.PostCreateDTO;
import io.hexlet.spring.dto.PostDTO;
import io.hexlet.spring.dto.PostUpdateDTO;
import io.hexlet.spring.exception.ResourceNotFoundException;
import io.hexlet.spring.mapper.PostMapper;
import io.hexlet.spring.repository.PostRepository;
import io.hexlet.spring.util.UserUtils;
import jakarta.validation.Valid;

@RestController
@RequestMapping("/api")
public class PostsController {
    @Autowired
    private PostRepository repository;

    @Autowired
    private PostMapper postMapper;

    @GetMapping("/posts")
    @ResponseStatus(HttpStatus.OK)
    public List<PostDTO> index() {
        var posts = repository.findAll();
        var result = posts.stream()
                .map(postMapper::map)
                .toList();

        return result;
    }

    @PostMapping("/posts")
    @ResponseStatus(HttpStatus.CREATED)
    public PostDTO create(@Valid @RequestBody PostCreateDTO postData) {
        var post = postMapper.map(postData);
```

```
        repository.save(post);
        var postDTO = postMapper.map(post);
        return postDTO;
    }

    @GetMapping("/posts/{id}")
    @ResponseStatus(HttpStatus.OK)
    public PostDTO show(@PathVariable Long id) {
        var post = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found: " + id));
        var postDTO = postMapper.map(post);
        return postDTO;
    }

    @PutMapping("/posts/{id}")
    @ResponseStatus(HttpStatus.OK)
    public PostDTO update(@RequestBody @Valid PostUpdateDTO postData, @PathVariable Long i
        var post = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found: " + id));
        postMapper.update(postData, post);
        repository.save(post);
        var postDTO = postMapper.map(post);
        return postDTO;
    }

    @DeleteMapping("/posts/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void delete(@PathVariable Long id) {
        repository.deleteById(id);
    }
}
```

В целом, CRUD не ограничивается только перечисленными методами. По ситуации методов может быть больше:

- Если у нас есть какое-то особое обновление и вывод списка
- Если у нас есть несколько контроллеров для одной и той же сущности (например, для управления постами для пользователей и для администраторов нужно два разных контроллера)

Далее →