Шаблон проектирования DTO

Spring Boot умеет автоматически преобразовывать объекты в JSON, когда они возвращаются из методов контроллера. Для этого внутри используется библиотека Jackson:

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

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.PathVariable;
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.exception.ResourceNotFoundException;
import io.hexlet.spring.model.User;
import io.hexlet.spring.repository.UserRepository;

@RestController
@RequestMapping("/api")
public class UsersController {
    @Autowired
    private UserRepository repository;

    @GetMapping("/users/{id}")
    @ResponseStatus(HttpStatus.OK)
    // Пользователь автоматически преобразуется в JSON
    public User show(@PathVariable Long id) {
        var user = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found"));
        return user;
    }
}
```

Несмотря на удобство, на практике этот механизм используют редко по нескольким причинам:

- **Безопасность**: Обычно у пользователя есть свойства, которые не стоит показывать наружу — например, хэш пароля или количество денег на счету. Автоматическое преобразование не учитывает такие данные и возвращает все доступные свойства.
- **Представления**: В разных ситуациях нужно возвращать разные наборы свойств. Для веб-версии нужно что-то одно, а для мобильной — что-то другое. Кроме того, по разным причинам могут отличаться названия свойств.
- **Схема данных**: Со временем имена полей могут меняться — например, из-за изменений в базе данных. При этом API меняться не должен, потому что на него рассчитывают клиенты. Разделение помогает асинхронно менять названия либо в сущностях, либо в API.
- **Связи**: Если в сущностях появляются связи с другими сущностями, это может вести к исключениям и другим проблемам во время преобразования в JSON.

В Jackson встроена аннотация `@JsonIgnore`, которая в простых случаях помогает решить проблемы с безопасностью. Если пометить этой аннотацией какое-то поле сущности, оно будет проигнорировано при конвертации в JSON:

```
// Остальные импорты
import com.fasterxml.jackson.annotation.JsonIgnore;

public class User {
    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Long id;

    @Column(unique = true)
    @Email
    private String email;

    private String firstName;

    private String lastName;

    @NotBlank
    @JsonIgnore
    private String password;
}
```

У этого механизма есть две проблемы:

- Это антипаттерн, который нарушает саму суть MVC. Модель узнает, как она используется в слое представления.
- Этот механизм не решает проблемы, описанные выше.В разных ситуациях мы работаем с разными наборами полей с точки зрения безопасности и представлений. Аннотация `@JsonIgnore` работает, только когда существует единственное представление — в реальных проектах такое встречается редко.

Для решения этих задач был придуман шаблон проектирования **Data Transfer Object** (*DTO*). По этому паттерну мы должны создавать свой класс с особыми набором полей под каждую конкретную ситуацию, которая требует своего набора полей. Затем необходимые данные из модели нужно копировать в DTO и возвращать наружу.

Для примера выше нам понадобится класс `UserDTO`:

```
// src/main/java/dto/UserDTO.java
package io.hexlet.spring.dto;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class UserDTO {
    private Long id;
    private String userName;
    private String firstName;
    private String lastName;
}
```

DTO — это не часть Spring Boot, поэтому именование и расположение этих классов лежит полностью на программистах. Мы будем хранить эти классы в директории *src/main/java/dto*.

Когда класс написан, остается только внедрить его в контроллер:

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

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.PathVariable;
```

```java
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.exception.ResourceNotFoundException;
import io.hexlet.spring.model.User;
import io.hexlet.spring.repository.UserRepository;
import io.hexlet.spring.dto.UserDTO;

@RestController
@RequestMapping("/api")
public class UsersController {
    @Autowired
    private UserRepository repository;

    @GetMapping("/users/{id}")
    @ResponseStatus(HttpStatus.OK)
    // Пользователь автоматически преобразуется в JSON
    public UserDTO show(@PathVariable Long id) {
        var user = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found"));

        var dto = new UserDTO();
        dto.setId(user.getId());
        dto.setFirstName(user.getFirstName());
        dto.setLastName(user.getLastName());

        return dto;
    }
}
```

Таким же образом мы поступим и во всех остальных ситуациях. Например, со списками:

```java
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.PathVariable;
```

```java
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.UserDTO;
import io.hexlet.spring.exception.ResourceNotFoundException;
import io.hexlet.spring.model.User;
import io.hexlet.spring.repository.UserRepository;

@RestController
@RequestMapping("/api")
public class UsersController {
    @Autowired
    private UserRepository repository;

    @GetMapping("/users")
    public List<UserDTO> index() {
        var users = repository.findAll();
        var result = users.stream()
                .map(this::toDTO)
                .toList();
        return result;
    }

    // Чтобы сделать работу удобнее
    // И избежать дублирования
    private UserDTO toDTO(User user) {
        var dto = new UserDTO();
        dto.setId(user.getId());
        dto.setFirstName(user.getFirstName());
        dto.setLastName(user.getLastName());
        return dto;
    }
}
```

Для удобства мы вынесли преобразование сущности в DTO в отдельный приватный метод. Это помогает немного снизить уровень дублирования, но не освобождает от ручного копирования свойств из одного объекта в другой. В следующих уроках мы познакомимся с библиотекой для автоматического копирования свойств.

В наших примерах для списка и вывода конкретной сущности мы использовали один к `UserDTO`, но это не обязательно. Если набор полей будет разным, то на каждый набор

понадобится свой собственный класс: `UserDTO`, `CreateUserDTO`, `UserListDTO`, `AdminUserListDTO` и так далее.

Далее →