УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Рефакторинг баз данных и приложений»

Лабораторная работа №1

Выполнили:

Бордун Анастасия

Иванов Евгений

P34111

Преподаватель

Логинов И. П.

Санкт-Петербург, 2023 г.

Задание:

Провести рефакторинг приложения(курсовая работа по ИСБД). Описать планируемые изменения, описать для чего они необходимы, провести их.

Выполнение:

В качестве изменений были выбраны следующие:

1. Добавление глобальных обработчиков ошибок (ExceptionHandler). Мы добавляем глобальную обработку ошибок для того, чтобы было меньше кода, который обрабатывает ошибки, что уменьшает код, например, в контроллерах, делая их более читаемыми. При этом также гарантируется одинаковая обработка одной и той же ошибки, вызванной в разных местах, так как обработчик один и тот же. Это влияет на гибкость системы — теперь при изменении обработки ошибки все места появления данной ошибки изменят обработку в соответствие с новой логикой (уменьшение связности).

Изменения в коде:

Добавление новых обработчиков ошибок и пример ошибки:

```
@ControllerAdvice
@ResponseBody
public class BookingExceptionHandler {
    @ExceptionHandler(BookingNotFoundException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public ErrorDTO handleBookingNotFoundException(BookingNotFoundException ex) {
        return new ErrorDTO(
                HttpStatus.BAD_REQUEST.name(), ex.getMessage(), LocalDateTime.now());
    }

    @ExceptionHandler(NotEnoughMoneyToBookException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public ErrorDTO handleNotEnoughMoneyToBookException(
                            NotEnoughMoneyToBookException ex
    ) {
        return new ErrorDTO(
                HttpStatus.BAD_REQUEST.name(), ex.getMessage(), LocalDateTime.now());
    }
}
```

Ошибки:

```
package com.example.demo.booking.exceptions;

public class BookingNotFoundException extends RuntimeException {
    public BookingNotFoundException(Long bookingId) {
        super("Booking with id[" + bookingId.toString() + "] not found");
    }
}
```

```
package com.example.demo.booking.exceptions;

public class NotEnoughMoneyToBookException extends RuntimeException {
    public NotEnoughMoneyToBookException() {
        super("Not enough money to book");
    }
}
```

Изменения в контроллерах(удаления кода обработки конкретных ошибок):

```
        try {
            model.put("message", bookingService.getBookingById(id).toString());
        } catch (BookingNotFoundException ex) {
            model.put("message", "BOOKING_NOT_FOUND");
            return new ResponseEntity<>(model, HttpStatus.BAD_REQUEST);
        }
```

И так далее для всех ошибок, а также ошибок добавленной в проект валидаций при чтении из JSON и составления из входных данных различных DTO…

2. Изменения в контроллерах: изменили названия в ручках(endpoint’ах) в соответствие с REST спецификацией. Изменение ответов ручек на ResponseEntity<...> для удобства создания ответов на вызываемые ручки и для удобства читаемости кода(теперь не надо создавать и возвращать объекты типа `Map<Object, Object> model = new HashMap<>();`, а можно вернуть объект типа DTO.

Изменения в коде:

Изменения ручек и возврат новых типов:

```
@RequestMapping("/booking")
@RequestMapping("/bookings")
```

```
@GetMapping(value = "/byId", produces = "application/json;charset=UTF-8")
public ResponseEntity<?> getBookingById(
        @RequestParam(name = "id") Long id
) {
    System.out.println("/booking/byId");
    Map<Object, Object> model = new HashMap<>();

    try {
        model.put("message", bookingService.getBookingById(id).toString());
    } catch (BookingNotFoundException ex) {
        model.put("message", "BOOKING_NOT_FOUND");
        return new ResponseEntity<>(model, HttpStatus.BAD_REQUEST);
    }

    return new ResponseEntity<>(model, HttpStatus.OK);
```

```
@GetMapping(value = "/{id}",
        produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Booking> getBookingById(@PathVariable("id") Long id) {
    log.info("get /bookings/{" + id + "}");
    return ResponseEntity.ok(bookingService.getBookingById(id));
}
```

И так далее для каждого контроллера/вызываемой ручки(endpoint’а)...

3. Добавление DTO в проект. В большом количестве мест необходимо было добавить data transfer object для нескольких целей. Так мы создаём логическую абстракцию над получаемыми/возвращаемыми данными. Также в таком случае, если есть две ручки, которые возвращают аналогичную информацию(например, информацию о студии, но вызываются в разных случаях), то DTO гарантирует, что объекты будут аналогичны по структуре. Тут же видно, что мы повторно используем код, так как при изменении возвращаемого объекта, нам не надо изменять возвращаемые значения в разных местах.

   Изменения в коде:

Новые объекты(сразу с аннотациями для Spring валидации):

```
@Data
public class StudioDTO {
    @JsonView
    @NotBlank(message = "Name is required")
    @ToUpperCase
    private String name;

    @JsonView
    @NotBlank(message = "Description is required")
    @ToUpperCase
    private String description;

    @JsonView
    @NotBlank(message = "Full-description is required")
    @ToUpperCase
    private String fullDescription;

    @JsonView
    @NotBlank(message = "Mail is required")
    @Email(message = "Incorrect mail pattern")
    private final String mail;

    @JsonView
    @Pattern(regexp = "\\+7[0-9]{10}", message = "Phone-number must start with +7, then
- 10 numbers more")
    private final String phone;

    @JsonView
    @NotBlank(message = "Tin is required")
    @Pattern(regexp = "\\d{4}-\\d{3}", message = "TIN must match the pattern xxxx-xxx")
    private final String tin;

}
```

Теперь данные объекты можно принимать в запросе или передавать в ответе в контроллерах:

```
    @PostMapping(value = "",
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Studio> addStudioWithLegalInfo(@Valid @RequestBody StudioDTO studioDTO) {
        log.info("post /studios");
        return ResponseEntity.ok(studioService.addStudioAndLegalInfo(studioDTO));
    }
```

И так далее в очень большом количестве мест…

4. Добавление Spring Validation. Валидация входных объектов необходима для обработки запросов и составления корректных DTO объектов из json. Также это удобно и убирает необходимость проверки отдельных полей запросов. При помощи ExceptionHandler есть возможность глобальной обработки ошибок в запросах(ошибки лишних и недостающих аргументов в запросе, ошибки приведения типов).

   Изменения в коде:

Добавление зависимости в build.gradle:

```
dependencies {
      implementation 'org.springframework.boot:spring-boot-starter-validation'
}
```

Пример валидации в DTO. В данном случае есть проверка на то, что поля *name, mail, phone, gender* не будут пустыми строками, также для полей представлены шаблоны(можно задать свои, можно использовать предоставленные), которым они должны соответствовать, иначе будет выброшено исключение:

```
@Data
public class InstructorDTO {
    @JsonView
    @NotBlank(message = "Name is required")
    private final String name;

    @JsonView
    @NotBlank(message = "Mail is required")
    @Email(message = "Incorrect mail pattern")
    private final String mail;

    @JsonView
    @NotBlank(message = "Phone is required")
    @Pattern(regexp = "\\+7[0-9]{10}", message = "Phone-number must start with +7, then
- 10 numbers more")
    private final String phone;

    @JsonView
    @NotBlank(message = "Gender is required")
    @ToUpperCase
    @ValueOfEnum(enumClass = Gender.class)
    private String gender;

}
```

5. Добавление сервиса логирования *@Slf4j*. Логирование удобно и помогает в управлении приложением. Можно выводить в консоль сообщения с разными уровнями информации. Формат вызова более удобный, чем вывод через *System.out.println();* С логгером сразу понятно для чего данная строчка(а значит удобство разработки).

Изменения в коде:

Добавлены аннотации над классами `@Slf4j`, в которых используется логгер:

```
+ @Slf4j
  public class StudioController {
```

Пример вызова логгера для обозначения в логах, какая ручка была вызвана:

```
    public ResponseEntity<List<Studio>> getAllStudios() {
        log.info("get /studios");
```

6. Добавление *lombok* аннотаций в сущности для удобства чтения файлов, уменьшения времени написания новых сущностей(не надо писать все *геттеры, сеттеры, конструкторы*).

Изменения в коде:

Добавлены аннотации к сущностям и DTO:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "class")
public class Classes {
```

Код изменений:

Всю проделанную работу(все изменения в коде) с кратким описанием изменений можно найти в PR на гитхабе проекта: https://github.com/3ilib0ba/ITMO-RDBaA/pull/1