Лабораторная работа №3

По предмету Java

Выполнил: Нечман Дмитрий Андреевич,  
 РИМ-220971

Екатеринбург

2023

**Цель работы:** Доработать простой REST сервис

2. Расширение функционала

Расширение функционала производилось в соответствии с указаниями в лабораторной работе, поэтому принято решение не дублироваться.

**3. Реализация Дополнительного функционала**

1) Добавьте вывод логов уровня INFO везде, где изменяется response и request таким образом чтобы по логам можно было отследить всю логику работы приложения.

Логика добавлена. Пример – запуск сервиса и запрос request:

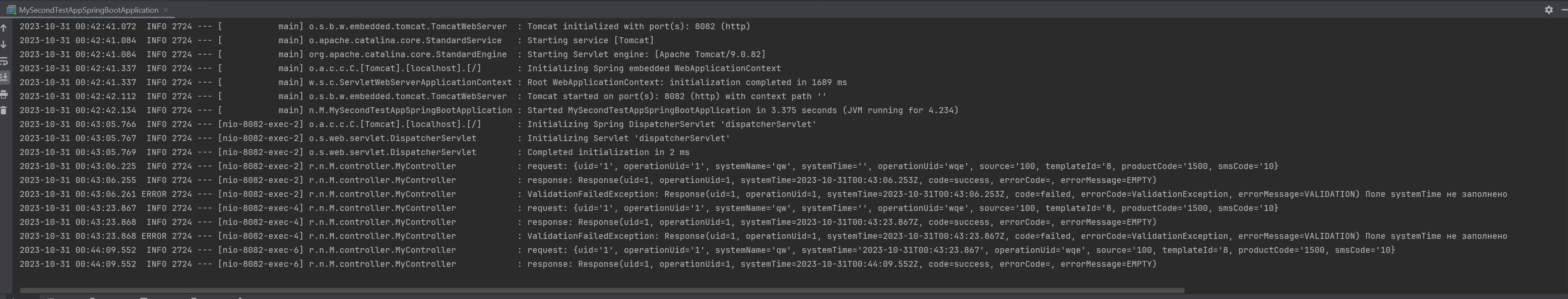


Рисунок 1 – Логгирование request

Код с добавлением логгирования:

@PostMapping(value = "/feedback")  
public ResponseEntity<Response> feedback(@Valid @RequestBody Request request, BindingResult bindingResult) {  
  
 *log*.info("request: {}", request);  
  
 Response response = Response.*builder*()  
 .uid(request.getUid())  
 .operationUid(request.getOperationUid())  
 .systemTime(DateTimeUtil.*getCustomFormat*().format(new Date()))  
 .code(Codes.*SUCCESS*)  
 .errorCode(ErrorCodes.*EMPTY*)  
 .errorMessage(ErrorMessages.*EMPTY*)  
 .build();  
  
 *log*.info("response: {}", response);

2) Добавьте вывод логов уровня ERROR в те места программы, где выбрасываются исключения, а также добавить сообщения которые возникают при ошибке в bindingResult.

Код:

try {  
 validationService.isValid(bindingResult);  
} catch (ValidationFailedException e) {  
 response.setCode(Codes.*FAILED*);  
 response.setErrorCode(ErrorCodes.*VALIDATION\_EXCEPTION*);  
 response.setErrorMessage(ErrorMessages.*VALIDATION*);  
 *log*.error("ValidationFailedException: {} {}", response, Objects.*requireNonNull*(bindingResult.getFieldError()).getDefaultMessage());  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
} catch (UnsupportedCodeException e) {  
 response.setCode(Codes.*FAILED*);  
 response.setErrorCode(ErrorCodes.*UNSUPPORTED\_CODE\_EXCEPTION*);  
 response.setErrorMessage(ErrorMessages.*UID123*);  
 *log*.error("UnsupportedCodeException: {} {} ", response, Objects.*requireNonNull*(bindingResult.getFieldError()).getDefaultMessage());  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
}  
catch (Exception e) {  
 response.setCode(Codes.*FAILED*);  
 response.setErrorCode(ErrorCodes.*UNKNOWN\_EXCEPTION*);  
 response.setErrorMessage(ErrorMessages.*UNKNOWN*);  
 *log*.error("Exception: {} {}", response, Objects.*requireNonNull*(bindingResult.getFieldError()).getDefaultMessage());  
 return new ResponseEntity<>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
}  
modifyResponseService.modify(response);

Пример получения ValidationException при некорректном отправлении запроса:

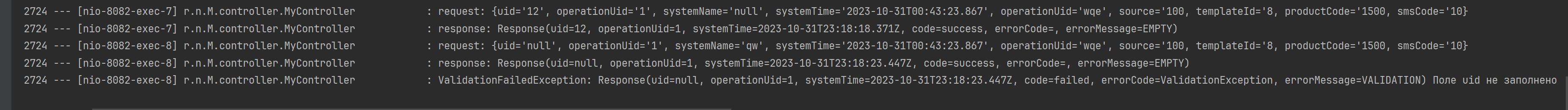
****

Рисунок 2 – Пример логгирования ошибки

Запрос был следующий:

{

  "operationUid": "1",

  "systemName": "qw",

  "systemTime": "2023-10-31T00:43:23.867",

  "source": "wqe",

  "communicationId": 100,

  "templateId": "8",

  "productCode": 1500,

  "smsCode": 10

}

3. В классе Request измените тип поля systemName с типа String на тип созданного самостоятельно перечисления Systems. Элементами перечисления сделайте следующие значения ERP («Enterprise Resource Planning»), CRM («Customer Relationship Management»), WMS (Warehouse Management System).

Код:

package ru.nechman.MySecondTestAppSpringBoot.model;  
  
import com.fasterxml.jackson.annotation.JsonValue;  
  
public enum Systems {  
 *ERP*("Enterprise Resource Planning"),  
 *CRM*("Customer Relationship Management"),  
 *WMS*("Warehouse Management System");  
  
 private final String name;  
  
 Systems(String name) {  
 this.name = name;  
 }  
  
 @JsonValue  
 public String getName() {  
 return name;  
 }  
 @Override  
 public String toString() {  
 return name;  
 }  
}

**4. Вопросы**

2. Что такое CRM система? Приведите примеры.

CRM (Customer Relationship Management) система - это программное обеспечение, которое помогает организациям эффективно управлять отношениями с клиентами. Она объединяет данные о клиентах, историю взаимодействия, продажи, обращения и другую информацию в одном месте, чтобы предоставить комплексный обзор о клиентах и оптимизировать взаимодействие с ними.

Вот несколько примеров популярных CRM систем:

Salesforce: Одна из самых известных и широко используемых CRM платформ, которая предлагает различные инструменты для управления продажами, маркетингом и обслуживанием клиентов.

HubSpot: CRM система, которая предлагает бесплатное ПО для управления контактами, сделками и маркетинговыми кампаниями, а также инструменты для установления и поддержки отношений с клиентами.

Zoho CRM: Интегрированная CRM платформа, которая предлагает модули для управления продажами, маркетингом, обслуживанием клиентов и другими аспектами бизнеса.

Microsoft Dynamics 365: Гибкая CRM система, которая позволяет управлять продажами, маркетингом, обслуживанием клиентов и другими бизнес-процессами, используя интегрированные сервисы Microsoft.

4.Какие библиотеки логгирования бывают в Spring?

В Spring Framework есть несколько популярных библиотек для логгирования. Вот некоторые из них:

Logback: Это быстрая и гибкая библиотека логгирования, являющаяся преемником популярной библиотеки log4j. Logback предоставляет широкие возможности конфигурации и поддерживает различные уровни логгирования.

Log4j: Это одна из самых широко используемых библиотек логгирования в Java. Log4j предлагает гибкую настройку журналирования, поддержку различных уровней логгирования и возможность записи логов в разные цели (файлы, консоль, база данных и т.д.).

SLF4J (Simple Logging Facade for Java): SLF4J - это фасад для различных библиотек логгирования, таких как Logback, Log4j и других. Он предлагает единый интерфейс для логгирования и позволяет легко переключаться между различными реализациями логгера.

JDK Logging: Это встроенный механизм логгирования в Java, который использует java.util.logging пакет. JDK Logging не требует дополнительных зависимостей и предоставляет базовые возможности логгирования.

6. Что означает аннотация @Qualifier?

Аннотация @Qualifier в Spring Framework используется для разрешения неоднозначности при внедрении зависимостей. Когда в контексте приложения есть несколько бинов одного типа, а необходимо выбрать конкретный бин для внедрения, @Qualifier позволяет указать, какой именно бин должен быть выбран.

Применение аннотации @Qualifier выглядит следующим образом:

@Autowired

@Qualifier("beanName")

private SomeType bean;

В приведенном примере, если в контексте приложения существует несколько бинов типа SomeType, то аннотация @Qualifier указывает, что нужно внедрить бин с именем "beanName".

Можно также использовать аннотацию @Qualifier вместе с другими аннотациями, такими как @Autowired или @Inject, для точного определения бинов, которые должны быть внедрены.

**Вывод:** в результате работы ознакомился с Spring Boot и тем, как создать простейшее веб-приложение с помощью фреймворка.

**Ссылка на репозиторий:**

https://github.com/NechD/Java\_homework/tree/master/2%20курс