Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
Имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИВлГУ)**

Факультет ИТР

Кафедра ПИн

ЛАБОРАТОРНАЯ РАБОТА №3

по Разработка корпоративных приложений

Тема Валидация данных в Spring

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИн-122
(группа)

Загребин Д. А.
(фамилия, инициалы)

(подпись) (дата)

Муром 2025

Лабораторная работа №3

Тема: Валидация данных в Spring

Цель: Научиться применять встроенные и кастомные механизмы валидации для проверки целостности и корректности данных, поступающих в Spring Boot приложение.

Задание:

1. Рассмотреть пример из лабораторной работы
2. Создать набор сущностей в зависимости от варианта (либо по теме курсовой

работы)

3. Реализовать web-приложение для работы с соответствующими данными.
4. Базовые действия: вывод информации о всех экземплярах и ввод нового.
5. Реализовать валидацию данных.
6. Реализовать поиск по различным критериям.

3 «Студент»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); ВУЗ; курс; группа; средний бал; специальность.*

Ход работы:

Листинг 1 MainController.java:

```
package ru.zagrebin.laba11;

import java.util.Optional;

import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import ru.zagrebin.laba11.entity.Address;
import ru.zagrebin.laba11.entity.Student;
import ru.zagrebin.laba11.entity.StudentGroup;
import ru.zagrebin.laba11.form.StudentForm;
import ru.zagrebin.laba11.service.ServiceAddress;
import ru.zagrebin.laba11.service.ServiceCourse;
import ru.zagrebin.laba11.service.ServiceStudent;
import ru.zagrebin.laba11.service.ServiceStudentGroup;
import ru.zagrebin.laba11.service.ServiceUniversity;

@Controller
public class MainController {
    private final ServiceStudent serviceStudent;
    private final ServiceAddress serviceAddress;
    private final ServiceStudentGroup serviceStudentGroup;
    @SuppressWarnings("unused")
    private final ServiceUniversity serviceUniversity;
    @SuppressWarnings("unused")
    private final ServiceCourse serviceCourse;

    @Autowired
    public MainController(ServiceStudent serviceStudent, ServiceAddress
serviceAddress, ServiceStudentGroup serviceStudentGroup, ServiceUniversity
serviceUniversity, ServiceCourse serviceCourse) {
        this.serviceStudent = serviceStudent;
```

```
        this.serviceAddress = serviceAddress;
        this.serviceStudentGroup = serviceStudentGroup;
        this.serviceUniversity = serviceUniversity;
        this.serviceCourse = serviceCourse;
    }

    @GetMapping("/institut")
    public String institut(Model model) {
        model.addAttribute("students", serviceStudent.getAllStudents());
        return "main";
    }

    @GetMapping("/details/{id}")
    public String details(@PathVariable("id") Long id, Model model) {
        Optional<Student> selectedStudent = serviceStudent.getStudentById(id);
        if (selectedStudent.isPresent()) {
            model.addAttribute("student", selectedStudent.get());
            return "details";
        }
        return "redirect:/institut";
    }

    @GetMapping("/students/delete/{id}")
    public String delete(@PathVariable("id") Long id) {
        serviceStudent.deleteStudentById(id);
        return "redirect:/institut";
    }

    @GetMapping("/students/addStudentForm")
    public String addStudentForm(Model model) {
        model.addAttribute("studentForm", new StudentForm());
        model.addAttribute("groups",
serviceStudentGroup.findAllStudentGroup());
        return "addStudentForm";
    }

    @PostMapping("/students/addStudentForm")
    public String addStudent(@Valid @ModelAttribute StudentForm studentForm,
BindingResult result, Model model) {

        if (result.hasErrors()) {
            model.addAttribute("groups",
serviceStudentGroup.findAllStudentGroup());
            return "addStudentForm";
        }
```

```
            Student student = new Student();
            convertFormToEntity(studentForm, student);
            serviceStudent.saveStudent(student);
            return "redirect:/institut";
        }


        @GetMapping("/students/editStudent/{id}")
        public String editStudentForm(@PathVariable("id") Long id, Model model) {
            Optional<Student> student = serviceStudent.getStudentById(id);
            if (student.isPresent()) {
                StudentForm studentForm = new StudentForm();
                convertEntityToForm(student.get(), studentForm);

                model.addAttribute("studentForm", studentForm);
                model.addAttribute("groups",
serviceStudentGroup.findAllStudentGroup());
                return "addStudentForm";
            }
            return "redirect:/institut";
        }

        @PostMapping("/student/editStudent/{id}")
        public String editStudent(@PathVariable(name = "id") Long id, @Valid
@ModelAttribute StudentForm studentForm, BindingResult result, Model model) {
            if (result.hasErrors()) {
                model.addAttribute("groups",
serviceStudentGroup.findAllStudentGroup());
                return "addStudentForm";
            }

            Optional<Student> existingStudent = serviceStudent.getStudentById(id);
            if (existingStudent.isPresent()) {
                Student student = existingStudent.get();
                convertFormToEntity(studentForm, student);
                serviceStudent.saveStudent(student);
            }

            return "redirect:/institut";
        }

        @GetMapping("/students/search")
        public String searchForm(Model model) {
            model.addAttribute("searchType", "lastName");
```

```
            model.addAttribute("ln", "");
            model.addAttribute("ct", "");
            model.addAttribute("mg", "");
            return "search";
        }

        @PostMapping("/students/search")
        public String searchResults(@RequestParam String searchType,
                                    @RequestParam String searchValue,
                                    Model model) {
            switch (searchType) {
                case "lastName":
                    model.addAttribute("students",
serviceStudent.findByLastName(searchValue));
                    break;
                case "city":
                    model.addAttribute("students",
serviceStudent.findByCity(searchValue));
                    break;
                case "gpa":
                    try {
                        Double gpa = Double.parseDouble(searchValue);
                        model.addAttribute("students",
serviceStudent.findByGpaGreaterThanEqual(gpa));
                    } catch (NumberFormatException e) {
                        model.addAttribute("error", "Введите корректное число для
GPA");
                    }
                    break;
                case "group":
                    model.addAttribute("students",
serviceStudent.findByGroupName(searchValue));
                    break;
                case "speciality":
                    model.addAttribute("students",
serviceStudent.findBySpeciality(searchValue));
                    break;
            }

            model.addAttribute("searchType", searchType);
            model.addAttribute("searchValue", searchValue);
            return "search";
        }
```

```
        // Combined search by up to three criteria: lastName (contains), city
(contains), min GPA
        @PostMapping("/students/search/combined")
        public String searchCombined(@RequestParam(required = false) String
lastName,
                                     @RequestParam(required = false) String city,
                                     @RequestParam(required = false) String
minGpa,
                                     Model model) {
            Double parsedGpa = null;
            if (minGpa != null && !minGpa.trim().isEmpty()) {
                try {
                    parsedGpa = Double.parseDouble(minGpa.trim());
                } catch (NumberFormatException e) {
                    model.addAttribute("error", "Введите корректное число для
минимального GPA");
                }
            }

            model.addAttribute("students", serviceStudent.searchCombined(lastName,
city, parsedGpa));
            model.addAttribute("ln", lastName);
            model.addAttribute("ct", city);
            model.addAttribute("mg", minGpa);
            model.addAttribute("searchType", "combined");
            return "search";
        }

        private void convertFormToEntity(StudentForm form, Student student) {
            student.setId(form.getId());
            student.setLastName(form.getLastName());
            student.setFirstName(form.getFirstName());
            student.setPatronymic(form.getPatronymic());
            student.setGender(form.getGender());
            student.setNationality(form.getNationality());
            student.setHeight(form.getHeight());
            student.setWeight(form.getWeight());
            student.setBirthDate(form.getBirthDate());
            student.setPhoneNumber(form.getPhoneNumber());
            student.setGpa(form.getGpa());

            boolean hasAddress = (form.getCity() != null &&
!form.getCity().trim().isEmpty())
                    || (form.getStreet() != null &&
!form.getStreet().trim().isEmpty())
```

```
                    ||          (form.getHouse()          !=          null          &&
!form.getHouse().trim().isEmpty())
                    ||          (form.getApartment()          !=          null          &&
!form.getApartment().trim().isEmpty());

            if (hasAddress) {
                Address addressToUse = null;
                if (form.getId() != null) {
                    // Try reuse existing address when editing
                    Optional<Student>                    existing                    =
serviceStudent.getStudentById(form.getId());
                    if (existing.isPresent()) {
                        addressToUse = existing.get().getAddress();
                    }
                }
                if (addressToUse == null) {
                    addressToUse = new Address();
                }
                addressToUse.setCity(form.getCity());
                addressToUse.setStreet(form.getStreet());
                addressToUse.setHouse(form.getHouse());
                addressToUse.setApartment(form.getApartment());

                serviceAddress.SaveAddress(addressToUse);
                student.setAddress(addressToUse);
            } else {
                student.setAddress(null);
            }

            if (form.getStudentGroupId() != null) {
                StudentGroup                          group                          =
serviceStudentGroup.findStudentGroupById(form.getStudentGroupId());
                student.setStudentGroup(group);
            }
        }

        private void convertEntityToForm(Student student, StudentForm form) {
            form.setId(student.getId());
            form.setLastName(student.getLastName());
            form.setFirstName(student.getFirstName());
            form.setPatronymic(student.getPatronymic());
            form.setGender(student.getGender());
            form.setNationality(student.getNationality());
            form.setHeight(student.getHeight());
            form.setWeight(student.getWeight());
```

```
            form.setBirthDate(student.getBirthDate());
            form.setPhoneNumber(student.getPhoneNumber());
            form.setGpa(student.getGpa());

            if (student.getAddress() != null) {
                form.setCity(student.getAddress().getCity());
                form.setStreet(student.getAddress().getStreet());
                form.setHouse(student.getAddress().getHouse());
                form.setApartment(student.getAddress().getApartment());
            }

            if (student.getStudentGroup() != null) {
                form.setStudentGroupId(student.getStudentGroup().getId());
            }
        }

    }
```

Листинг 2 StudentFormDto.java:

```
package ru.zagrebin.laba11.validateDto;

import java.time.LocalDate;

import jakarta.validation.constraints.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class StudentFormDto {
      private Long id;

    @NotBlank
    @Size(min=3, max=50)
    private String lastName;

    @NotBlank
    @Size(min=3, max=50)
    private String firstName;

    @Size(min=3, max=50)
    private String patronymic;
```

```
@NotBlank
@Pattern(regexp ="Мужской|Женский")
private String gender;

@NotBlank
private String nationality;

@NotNull
@Min(value = 100)
@Max(value = 250)
private Double height;

@NotNull
@Min(value = 25)
@Max(value = 300)
private Double weight;

@NotNull
@Past
private LocalDate birthDate;

@NotNull
@Pattern(regexp = "^\\+7\\d{10}$")
private String phoneNumber;

@NotNull
@DecimalMin(value = "0.0")
@DecimalMax(value = "5.0")
private Double gpa;

@NotBlank
private String city;

@NotBlank
private String street;

@NotBlank
private String house;

private String apartment;

@NotNull
private Long studentGroupId;
```

Демонстрация работы программы:

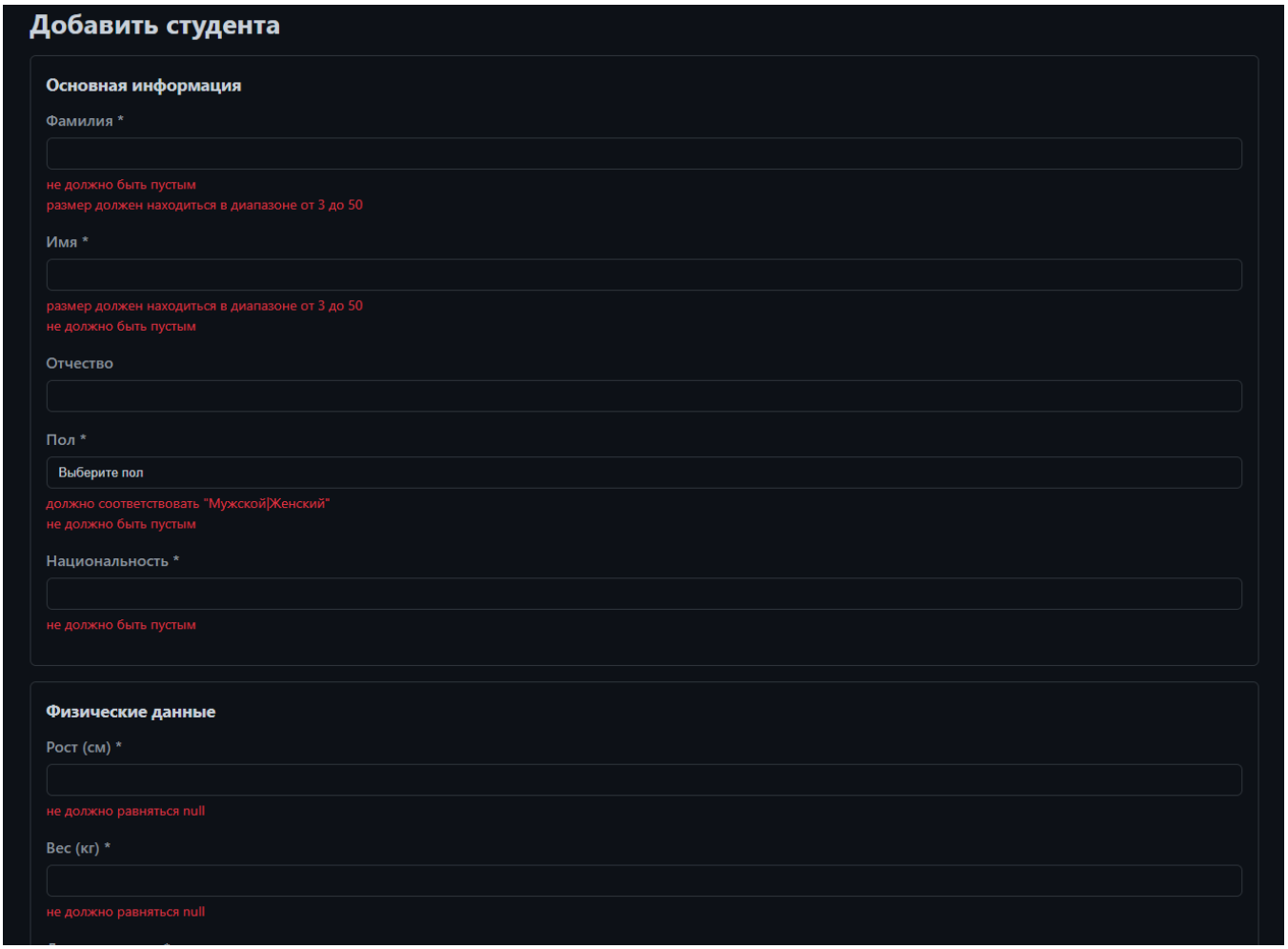

Рисунок 1 – Ошибка при некорректных данных

Вывод: Входе лабораторной работы было изучено применение встроенных и кастомных механизмов валидации для проверки целостности и корректности данных, поступающих в Spring Boot приложение.