Министерство науки и высшего образования Российской Федерации   
Федеральное государственное бюджетное образовательное   
учреждение высшего образования   
«Алтайский государственный технический   
университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Третьяков А.А.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Отчёт

по лабораторной работе № 2

«Java Spring Framework»

по дисциплине «Современные средства разработки Web приложений»

Студент группы ПИ-92 Шульпов В.М.

Старший преподаватель Третьяков А.А.

Барнаул 2022

Общая цель лабораторных по фреймворкам - научиться осваивать новые инструменты программирования.

Для одного (самого сложного) объекта из задачи на лабораторные реализовать четыре операции CRUD:

1. создание объектов (Create)
2. отображение списка объектов и одного объекта (Read)
3. изменение объектов (Update)
4. удаление объектов (Delete)

Требования

1. Должна использоваться модель данных из первой лабораторной
2. Должна использоваться адаптивная верстка из первой лабораторной
3. Используется уровень абстракции данных из фреймворка
4. Используется шаблонизатор из фреймворка
5. Студент отвечает на вопрос "Что это?" в любом месте своего или сгенерированного кода (или взятого из примера)
6. Студент отвечает на вопросы по структуре фреймворка
7. Студент понимает, в чем особенности языка программирования

Отчет должен содержать:

* титульный лист,
* задание (краткое общее задание из этих методических указаний и вариант задания, который выполняет студент),
* исходный код контроллера,
* исходный код шаблонов,
* исходный код модели (если применимо),
* скриншоты всех операций (для различных экранов).

Код можно печатать шрифтом 10pt с одинарным интервалом.

Подробнее о заданиях тут: http://lms.altstu.ru/ilias/goto.php?target=lm\_5334&client\_id=AltSTU

Код:

PersonController:

package org.app1.controllers;  
  
  
  
import org.app1.dao.PersonDAO;  
import org.app1.models.Person;  
import org.app1.util.PersonValidator;  
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.\*;  
  
import javax.validation.Valid;  
import java.util.Optional;  
  
@Controller  
@RequestMapping("/people")  
public class PeopleController {  
  
 private final PersonDAO personDAO;  
 private final PersonValidator personValidator;  
  
 @Autowired  
 public PeopleController(PersonDAO personDAO, PersonValidator personValidator) {  
 this.personDAO = personDAO;  
 this.personValidator = personValidator;  
 }  
  
 //запрос на получение страницы со списком всех людей  
 @GetMapping()  
 public String index(Model model) {  
 model.addAttribute("people", personDAO.index());  
 return "people/index";  
 }  
  
 //запрос на получение страницы с определенным человеком  
 @GetMapping("/{id}")  
 public String personPage(@PathVariable int id, Model model) {  
 Optional<Person> person = personDAO.getPerson(id);  
 if (person.isPresent()) {  
 model.addAttribute("person", person.get());  
 model.addAttribute("books", personDAO.getBooksByPerson(id));  
 return "people/person";  
 }  
 else {  
 return "redirect:/people";  
 }  
 }  
  
 //запрос на получение страницы добавления человека  
 @GetMapping("/new")  
 public String newPerson(@ModelAttribute("person") Person person) {  
 return "people/new-person";  
 }  
  
 //запрос на добавление человека  
 @PostMapping()  
 public String create(@ModelAttribute("person") @Valid Person person,  
 BindingResult bindingResult) {  
 personValidator.validate(person, bindingResult);  
 if (bindingResult.hasErrors())  
 return "people/new-person";  
  
 personDAO.save(person);  
 return "redirect:/people";  
 }  
  
 //запрос на получение страницы изменения человека  
 @GetMapping("/{id}/edit")  
 public String editPersonPage(Model model, @PathVariable int id) {  
 Optional<Person> selectedPerson = personDAO.getPerson(id);  
 if (selectedPerson.isPresent()) {  
 model.addAttribute("person", selectedPerson.get());  
 return "/people/edit-person";  
 }  
 return "redirect:/people";  
 }  
  
 //запрос на редактирование человека  
 @PatchMapping("/{id}")  
 public String edit(@PathVariable int id, @ModelAttribute("person") @Valid Person updatedPerson,  
 BindingResult bindingResult) {  
 personValidator.validate(updatedPerson, bindingResult);  
 if (bindingResult.hasErrors())  
 return "people/edit-person";  
  
 personDAO.update(updatedPerson, id);  
 return "redirect:/people/" + id;  
 }  
  
 //запрос на удаление человека  
 @DeleteMapping("/{id}")  
 public String delete(@PathVariable int id) {  
 personDAO.delete(id);  
 return "redirect:/people";  
 }  
}

BooksController

package org.app1.controllers;  
  
  
import org.app1.dao.BookDAO;  
import org.app1.dao.PersonDAO;  
import org.app1.models.Book;  
import org.app1.models.Person;  
import org.jetbrains.annotations.NotNull;  
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.\*;  
  
import javax.validation.Valid;  
import java.util.Optional;  
  
@Controller  
@RequestMapping("/books")  
public class BooksController {  
  
 private final BookDAO bookDAO;  
 private final PersonDAO personDAO;  
  
 @Autowired  
 public BooksController(BookDAO bookDao, PersonDAO personDAO) {  
 this.bookDAO = bookDao;  
 this.personDAO = personDAO;  
 }  
  
 //запрос на получение страницы со списком всех книг  
 @GetMapping()  
 public String allBooksPage(@NotNull Model model) {  
 model.addAttribute("books", bookDAO.index());  
 return "books/all-books";  
 }  
 //запрос на получение страницы с определенной книгой  
 @GetMapping("/{id}")  
 public String bookPage(@PathVariable int id, Model model, @ModelAttribute("person") Person person) {  
 Optional<Book> book = bookDAO.getBook(id);  
 if (book.isPresent()) {  
 model.addAttribute("book", book.get());  
 Optional<Person> personOwner = bookDAO.getBookOwner(id);  
 if (personOwner.isPresent()) {  
 model.addAttribute("person\_owner", personOwner.get());  
 }  
 else {  
 model.addAttribute("people", personDAO.index());  
 }  
 return "books/book";  
 }  
 else {  
 return "redirect:/books";  
 }  
  
 }  
 //запрос на получение страницы добавления книги  
 @GetMapping("/new")  
 public String newBookPage(@ModelAttribute("book") Book book) {  
// model.addAttribute("book", new Book());  
 return "books/new-book";  
 }  
 //запрос на добавление новой книги  
 @PostMapping()  
 public String createBook(@ModelAttribute("book") @Valid Book book,  
 @NotNull BindingResult bindingResult) {  
 if (bindingResult.hasErrors())  
 return "books/new-book";  
  
 bookDAO.save(book);  
 return "redirect:/books";  
 }  
 //запрос на получение страницы изменения книги  
 @GetMapping("/{id}/edit")  
 public String editBookPage(@PathVariable int id, Model model) {  
  
 Optional<Book> selectedBook = bookDAO.getBook(id);  
 if (selectedBook.isPresent()) {  
 model.addAttribute("book", selectedBook.get());  
 return "/books/edit-book";  
 }  
 return "redirect:/books";  
 }  
  
 //запрос на редактирование данных книги  
 @PatchMapping("/{id}")  
 public String edit(@PathVariable int id, @ModelAttribute("book") @Valid Book updatedBook,  
 BindingResult bindingResult) {  
 if (bindingResult.hasErrors())  
 return "books/edit-book";  
  
 bookDAO.update(updatedBook, id);  
 return "redirect:/books/" + id;  
 }  
  
 //запрос на удаление книги  
 @DeleteMapping("/{id}")  
 public String delete(@PathVariable int id) {  
 bookDAO.deleteBook(id);  
 return "redirect:/books";  
 }  
  
 @PatchMapping("/{id}/assign")  
 public String assign(@PathVariable int id, @ModelAttribute("person") Person selectedPerson) {  
 bookDAO.assign(id, selectedPerson);  
 return "redirect:/books/" + id;  
 }  
  
 @PatchMapping("/{id}/release")  
 public String release(@PathVariable int id) {  
 bookDAO.release(id);  
 return "redirect:/books/" + id;  
 }  
}

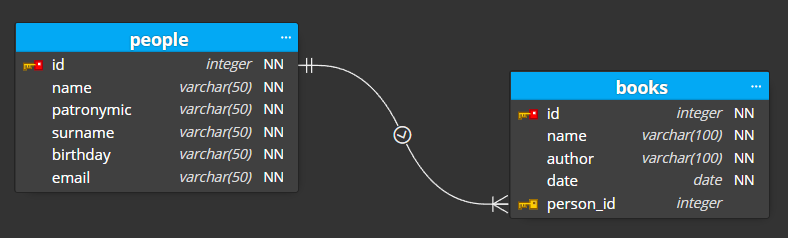
PersonDAO

package org.app1.dao;  
  
import org.app1.models.Book;  
import org.app1.models.Person;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
import java.util.Optional;  
  
@Component  
public class PersonDAO {  
 private final JdbcTemplate jdbcTemplate;  
  
 @Autowired  
 public PersonDAO(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 //получить список всех людей  
 public List<Person> index() {  
 return jdbcTemplate.query("SELECT \* FROM People", new PersonRowMapper());  
 }  
  
 //добавить человека  
 public void save(Person person) {  
 jdbcTemplate.update("INSERT INTO public.people(name, patronymic, surname, birthday, email)" +  
 "VALUES (?, ?, ?, ?, ?);",  
 person.getName(),  
 person.getPatronymic(),  
 person.getSurname(),  
 person.getBirthday(),  
 person.getEmail());  
 }  
  
 //получить человека по id  
 public Optional<Person> getPerson(int id) {  
 return jdbcTemplate.query("SELECT \* FROM People WHERE id=?;",  
 new PersonRowMapper(), new Object[]{id}).stream().findAny();  
 }  
  
 //получить человека по id  
 public Optional<Person> getPersonByEmail(String email) {  
 return jdbcTemplate.query("SELECT \* FROM People WHERE email=?;",  
 new PersonRowMapper(), new Object[]{email}).stream().findAny();  
 }  
  
 //получить список книг, взятых определенным человеком  
 public List<Book> getBooksByPerson(int id) {  
 return jdbcTemplate.query("select \* from Books where person\_id = ?", new BookRowMapper(), id);  
 }  
  
 //удалить человека по id  
 public void delete(int id) {  
 jdbcTemplate.update("DELETE FROM People WHERE id = ?;", id);  
 }  
  
 //редактировать данные о человеке  
 public void update(Person updatedPerson, int id) {  
 jdbcTemplate.update("update people set " +  
 "name = ?, patronymic = ?, surname = ?, birthday = ?, email = ? where id = ?;",  
 updatedPerson.getName(),  
 updatedPerson.getPatronymic(),  
 updatedPerson.getSurname(),  
 updatedPerson.getBirthday(),  
 updatedPerson.getEmail(),  
 id);  
 }  
}

Person

package org.app1.models;  
  
  
  
import com.fasterxml.jackson.annotation.JsonFormat;  
  
  
import org.springframework.format.annotation.DateTimeFormat;  
  
import javax.validation.constraints.Email;  
import javax.validation.constraints.NotEmpty;  
import javax.validation.constraints.NotNull;  
import javax.validation.constraints.Size;  
import java.time.LocalDate;  
  
public class Person {  
 private int id;  
 @NotEmpty(message = "Name should not be empty")  
 @Size(min = 2, max = 30, message = "Name should be between 2 and 30 characters")  
 private String name;  
 @NotEmpty(message = "Name should not be empty")  
 @Size(min = 2, max = 30, message = "Name should be between 2 and 30 characters")  
 private String patronymic;  
 @NotEmpty(message = "Name should not be empty")  
 @Size(min = 2, max = 30, message = "Name should be between 2 and 30 characters")  
 private String surname;  
 @NotNull(message = "Date should not be empty")  
 @JsonFormat(shape = JsonFormat.Shape.*STRING*, pattern = "yyyy-MM-dd")  
 @DateTimeFormat(iso = DateTimeFormat.ISO.*DATE*)  
 private LocalDate birthday;  
  
 @NotNull(message = "Email should not be empty")  
 @Email(message = "Email should be valid")  
 private String email;  
  
 public Person() {  
 }  
  
 public Person(int id, String name, String patronymic, String surname, LocalDate birthday, String email) {  
 this.id = id;  
 this.name = name;  
 this.patronymic = patronymic;  
 this.surname = surname;  
 this.birthday = birthday;  
 this.email = email;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getFullName() {  
 return String.*format*("%s %s %s", getName(), getPatronymic(), getSurname());  
 }  
  
 public LocalDate getBirthday() {  
 return birthday;  
 }  
  
 public int getBirthdayYear() {  
 return birthday.getYear();  
 }  
  
 public void setBirthday(LocalDate birthday) {  
 this.birthday = birthday;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", patronymic='" + patronymic + '\'' +  
 ", surname='" + surname + '\'' +  
 ", birthday=" + birthday +  
 ", email='" + email + '\'' +  
 '}';  
 }  
}

Структура БД



Код генерации БД

CREATE TABLE people(

    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

    "name" varchar(50) NOT NULL,

    patronymic varchar(50) NOT NULL,

    surname varchar(50) NOT NULL,

    birthday date NOT NULL,

    email varchar(50) NOT NULL UNIQUE,

);

CREATE TABLE books(

    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

    "name" varchar(100) NOT NULL,

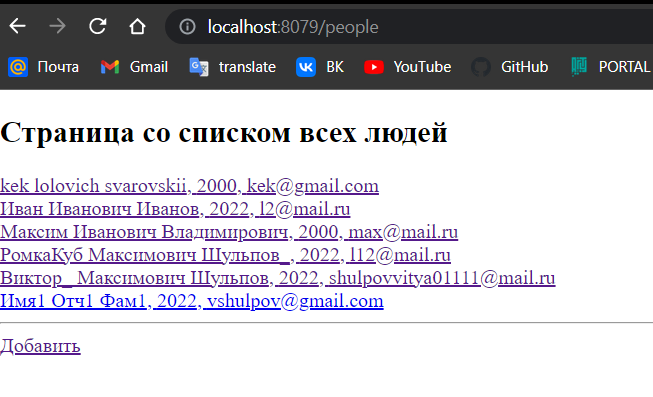
    author varchar(100) NOT NULL,

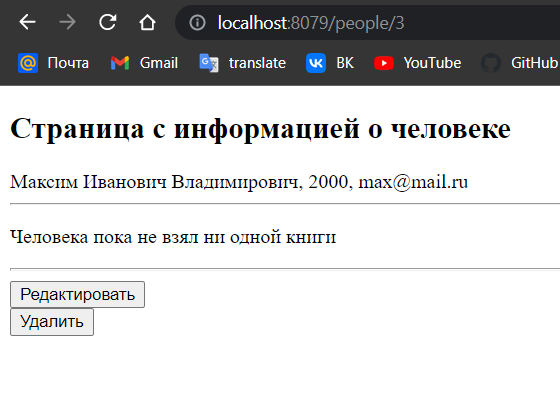
    date date NOT NULL,

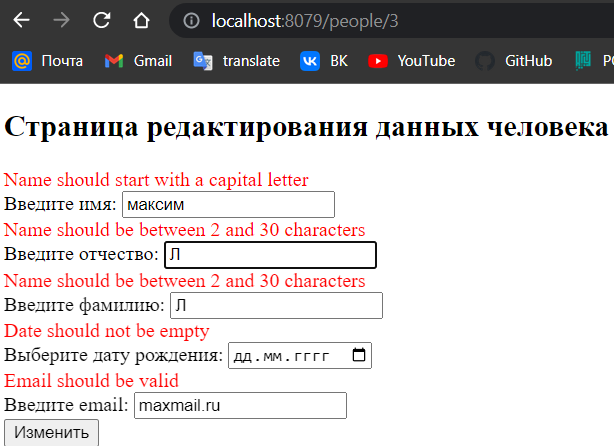
    people\_id int DEFAULT NULL REFERENCES people(id) ON DELETE Set null ON UPDATE Cascade

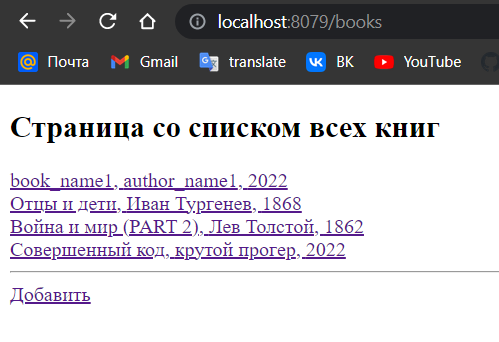
);

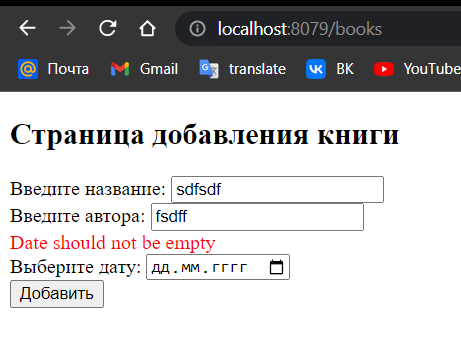
Веб-страницы:

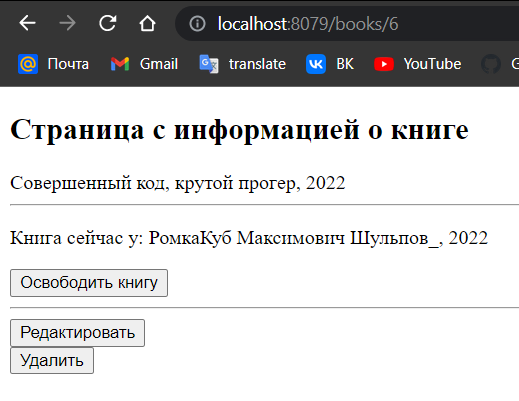
****

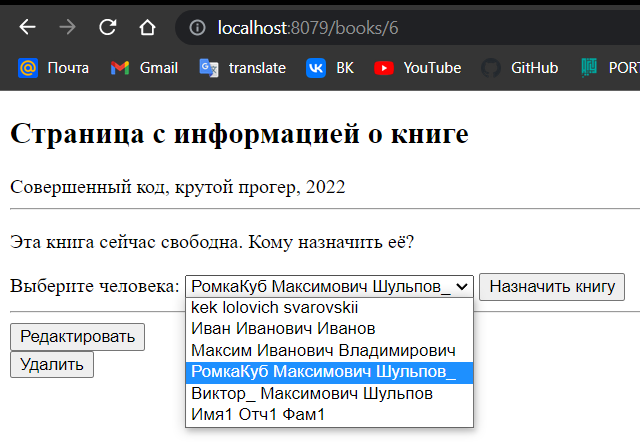
****

****

****

****

****

****