|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации |
| Федеральное государственное автономное образовательное учреждение высшего образования |
| «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина» |
| Институт радиоэлектроники и информационных технологий – РтФ |
|  |
|  | |

**ОТЧЕТ**

О ВЫПОЛНЕНИИ

ЛАБОРАТОНОЙ РАБОТЫ № 6

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-230971 Е.В. Треглазов

Екатеринбург 2024

**Цель работы:** разработать REST full service.

1. **Разработка REST сервиса**

Интерфейс StudentDAO:

package ru.example.MySpringBoot2Dbase.dao;

import org.springframework.stereotype.Repository;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@Repository

public interface StudentDAO {

    List<Student> getAllStudents();

    Student saveStudent(Student student);

    Student getStudent(int id);

    void deleteStudent(int id);

}

Класс его имплементирующий:

package ru.example.MySpringBoot2Dbase.dao;

import jakarta.persistence.EntityManager;

import jakarta.persistence.Query;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@Slf4j

@Repository

public class StudentDAOImpl implements StudentDAO{

    @Autowired

    private EntityManager entityManager;

    @Override

    public List<Student> getAllStudents() {

        Query query = entityManager.createQuery("from Student");

        List<Student> allStudents = query.getResultList();

        log.info("getAllStudents" + allStudents);

        return allStudents;

    }

    @Override

    public Student saveStudent(Student student) { return entityManager.merge(student); }

    @Override

    public Student getStudent(int id) { return entityManager.find(Student.class, id); }

    @Override

    public void deleteStudent(int id) {

        Query query = entityManager.createQuery("delete from Student "

                + " where id =:studentId");

        query.setParameter("studentId", id);

        query.executeUpdate();

    }

}

Класс описывающий сущность студент:

package ru.example.MySpringBoot2Dbase.entity;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@Entity

@NoArgsConstructor

@AllArgsConstructor

@Table(name = "STUDENTS")

public class Student {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "name")

    private String name;

    @Column(name = "surname")

    private String surname;

    @Column(name = "faculty")

    private String faculty;

    @Column(name = "age")

    private Integer age;

}

Интерфейс StudentService:

package ru.example.MySpringBoot2Dbase.service;

import org.springframework.stereotype.Service;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@Service

public interface StudentService {

    List<Student> getAllStudents();

    Student saveStudent(Student student);

    Student getStudent(int id);

    void deleteStudent(int id);

}

Класс его имплементирующий:

package ru.example.MySpringBoot2Dbase.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import ru.example.MySpringBoot2Dbase.dao.StudentDAO;

import ru.example.MySpringBoot2Dbase.entity.Student;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class StudentServiceImpl implements StudentService{

    @Autowired

    private StudentDAO studentDAO;

    @Override

    @Transactional

    public List<Student> getAllStudents() {

        return studentDAO.getAllStudents();

    }

    @Override

    @Transactional

    public Student saveStudent(Student student) {

        return studentDAO.saveStudent(student);

    }

    @Override

    @Transactional

    public Student getStudent(int id) {

        return studentDAO.getStudent(id);

    }

    @Override

    @Transactional

    public void deleteStudent(int id) {

        studentDAO.deleteStudent(id);

    }

}

Класс MyController:

package ru.example.MySpringBoot2Dbase.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import ru.example.MySpringBoot2Dbase.service.StudentService;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@RestController

@RequestMapping("/api")

public class MyController {

    @Autowired

    private StudentService studentService;

    @GetMapping("/students")

    public List<Student> allStudents() {

        List<Student> allStudents = studentService.getAllStudents();

        return allStudents;

    }

    @GetMapping("/students/{id}")

    public Student getStudent(@PathVariable("id") int id) {

        return studentService.getStudent(id);

    }

    @PostMapping("/students")

    public Student saveStudent(@RequestBody Student student) {

        return studentService.saveStudent(student);

    }

    @PutMapping("/students")

    public Student updateStudent(@RequestBody Student student) {

        studentService.saveStudent(student);

        return student;

    }

    @DeleteMapping("/students/{id}")

    public void deleteStudent(@PathVariable("id") int id) {

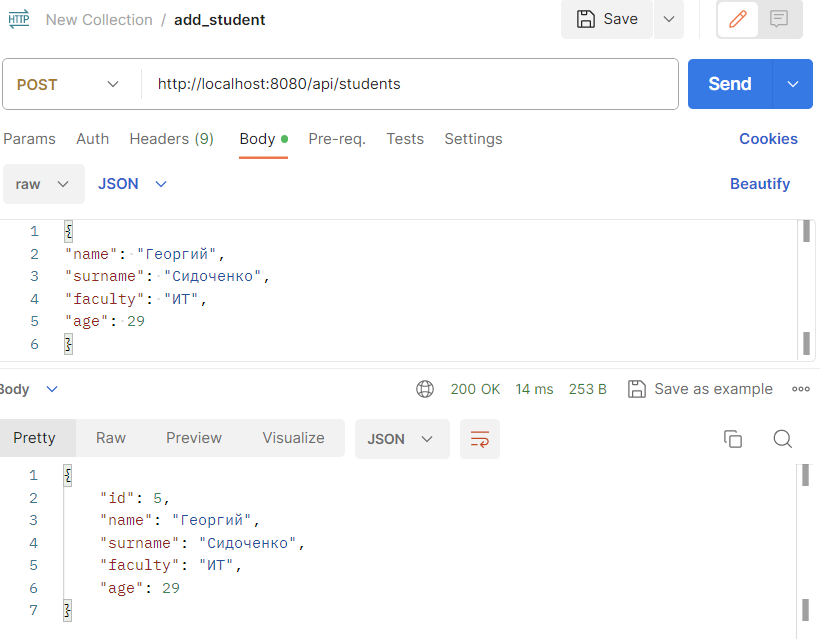
        studentService.deleteStudent(id);

    }

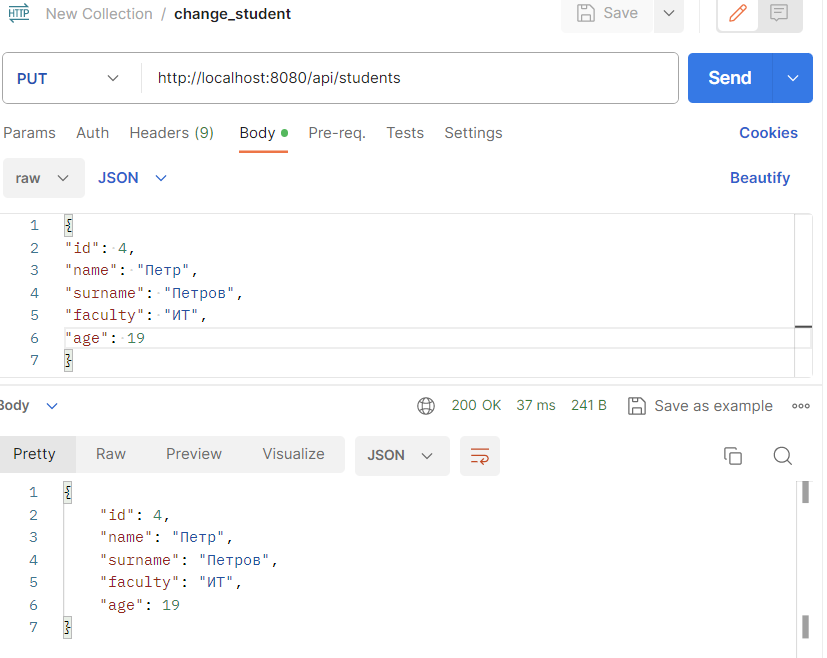
}

1. **Запуск и тестирование приложения**

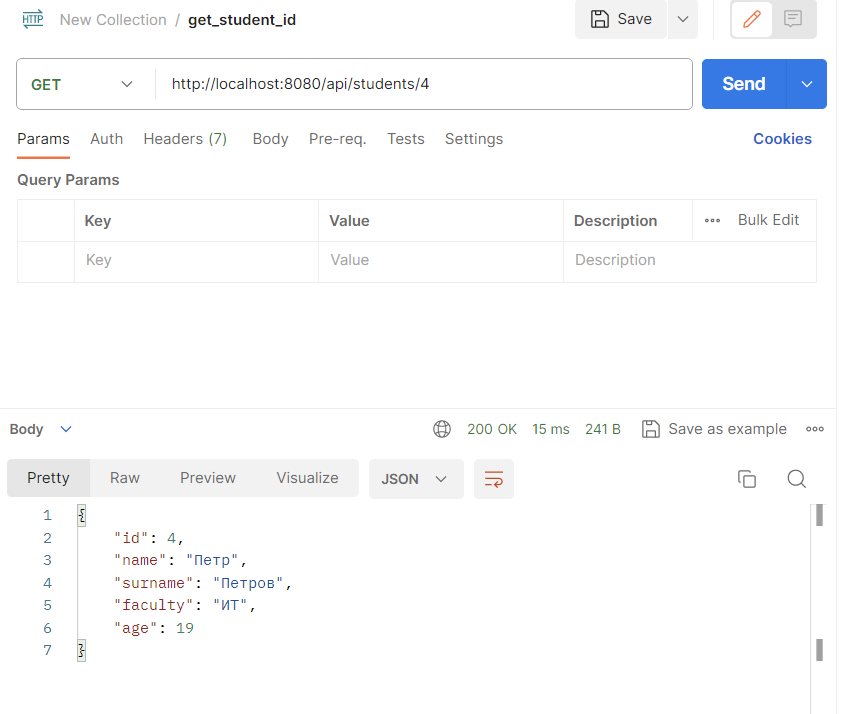
POST запрос для добавление студента:



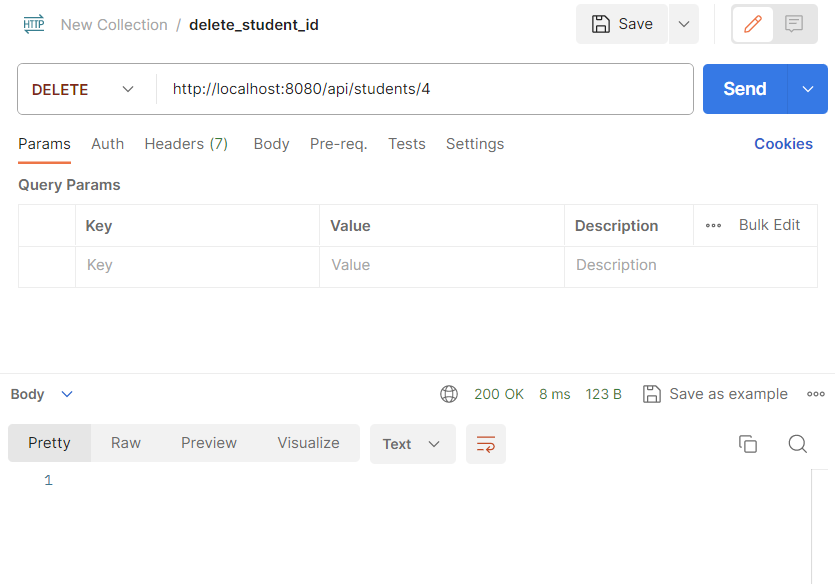
PUT запрос на изменение:



GET запрос на получение студента по id:



DELETE запрос на удаление студента по id:



1. **Реализация дополнительного функционала**
2. Доработайте методы, возвращающие значение, чтобы возвращался ответ об успехе или неуспехе завершения операции.

Доработанный класс MyController:

package ru.example.MySpringBoot2Dbase.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import ru.example.MySpringBoot2Dbase.service.StudentService;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@RestController

@RequestMapping("/api")

public class MyController {

    @Autowired

    private StudentService studentService;

    @GetMapping("/students")

    public ResponseEntity<?> allStudents() {

        try {

            List<Student> allStudents = studentService.getAllStudents();

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

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @GetMapping("/students/{id}")

    public ResponseEntity<?> getStudent(@PathVariable("id") int id) {

        try {

            return new ResponseEntity<>(studentService.getStudent(id), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PostMapping("/students")

    public ResponseEntity<?> saveStudent(@RequestBody Student student) {

        try {

            return new ResponseEntity<>(studentService.saveStudent(student), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PutMapping("/students")

    public ResponseEntity<?> updateStudent(@RequestBody Student student) {

        try {

            return new ResponseEntity<>(studentService.saveStudent(student), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @DeleteMapping("/students/{id}")

    public ResponseEntity<?> deleteStudent(@PathVariable("id") int id) {

        try {

            studentService.deleteStudent(id);

            return new ResponseEntity<>(HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

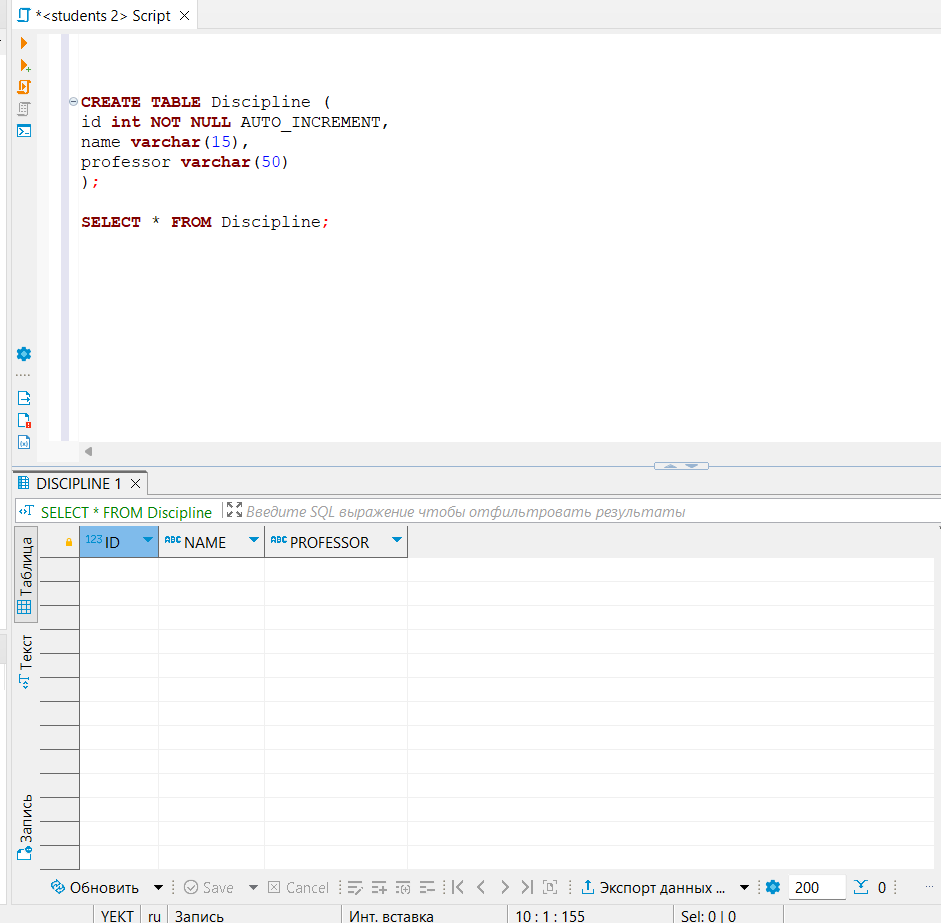
        }

    }

}

1. Создайте вторую таблицу в базе данных «Учебные дисциплины» и опишите для нее необходимые контроллер, сущность, DAO и сервисный слой.

Создание таблицы:



Класс Discipline описывающий сущность в пакете entity:

package ru.example.MySpringBoot2Dbase.entity;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@Entity

@NoArgsConstructor

@AllArgsConstructor

@Table(name = "academic\_disciplines")

public class Discipline {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "name")

    private String name;

    @Column(name = "professor")

    private String professor;

}

Интерфейс DisciplineDAO:

package ru.example.MySpringBoot2Dbase.dao;

import org.springframework.stereotype.Repository;

import ru.example.MySpringBoot2Dbase.entity.Discipline;

import java.util.List;

@Repository

public interface DisciplineDAO {

    List<Discipline> getAllDisciplines();

    Discipline saveDiscipline(Discipline discipline);

    Discipline getDiscipline(int id);

    void deleteDiscipline(int id);

}

Класс его имплементирующий:

package ru.example.MySpringBoot2Dbase.dao;

import jakarta.persistence.EntityManager;

import jakarta.persistence.Query;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import ru.example.MySpringBoot2Dbase.entity.Discipline;

import java.util.List;

@Slf4j

@Repository

public class DisciplineDAOImpl implements DisciplineDAO{

    @Autowired

    private EntityManager entityManager;

    @Override

    public List<Discipline> getAllDisciplines() {

        Query query = entityManager.createQuery("from Discipline");

        List<Discipline> allDisciplines = query.getResultList();

        log.info("getAllDisciplines" + allDisciplines);

        return allDisciplines;

    }

    @Override

    public Discipline saveDiscipline(Discipline discipline) { return entityManager.merge(discipline); }

    @Override

    public Discipline getDiscipline(int id) { return entityManager.find(Discipline.class, id); }

    @Override

    public void deleteDiscipline(int id) {

        Query query = entityManager.createQuery("delete from Discipline "

                + " where id =:DisciplineId");

        query.setParameter("DisciplineId", id);

        query.executeUpdate();

    }

}

Интерфейс DisciplineService:

package ru.example.MySpringBoot2Dbase.service;

import org.springframework.stereotype.Service;

import ru.example.MySpringBoot2Dbase.entity.Discipline;

import java.util.List;

@Service

public interface DisciplineService {

    List<Discipline> getAllDisciplines();

    Discipline saveDiscipline(Discipline discipline);

    Discipline getDiscipline(int id);

    void deleteDiscipline(int id);

}

Класс его имплементирующий:

package ru.example.MySpringBoot2Dbase.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import ru.example.MySpringBoot2Dbase.dao.DisciplineDAO;

import ru.example.MySpringBoot2Dbase.entity.Discipline;

import java.util.List;

@Service

public class DisciplineServiceImpl implements DisciplineService{

    @Autowired

    private DisciplineDAO disciplineDAO;

    @Override

    @Transactional

    public List<Discipline> getAllDisciplines() {

        return disciplineDAO.getAllDisciplines();

    }

    @Override

    @Transactional

    public Discipline saveDiscipline(Discipline discipline) {

        return disciplineDAO.saveDiscipline(discipline);

    }

    @Override

    @Transactional

    public Discipline getDiscipline(int id) {

        return disciplineDAO.getDiscipline(id);

    }

    @Override

    @Transactional

    public void deleteDiscipline(int id) {

        disciplineDAO.deleteDiscipline(id);

    }

}

Дополненный класс MyController:

package ru.example.MySpringBoot2Dbase.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import ru.example.MySpringBoot2Dbase.service.DisciplineService;

import ru.example.MySpringBoot2Dbase.entity.Discipline;

import ru.example.MySpringBoot2Dbase.service.StudentService;

import ru.example.MySpringBoot2Dbase.entity.Student;

import java.util.List;

@RestController

@RequestMapping("/api")

public class MyController {

    @Autowired

    private StudentService studentService;

    @GetMapping("/students")

    public ResponseEntity<?> allStudents() {

        try {

            List<Student> allStudents = studentService.getAllStudents();

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

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @GetMapping("/students/{id}")

    public ResponseEntity<?> getStudent(@PathVariable("id") int id) {

        try {

            return new ResponseEntity<>(studentService.getStudent(id), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PostMapping("/students")

    public ResponseEntity<?> saveStudent(@RequestBody Student student) {

        try {

            return new ResponseEntity<>(studentService.saveStudent(student), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PutMapping("/students")

    public ResponseEntity<?> updateStudent(@RequestBody Student student) {

        try {

            return new ResponseEntity<>(studentService.saveStudent(student), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @DeleteMapping("/students/{id}")

    public ResponseEntity<?> deleteStudent(@PathVariable("id") int id) {

        try {

            studentService.deleteStudent(id);

            return new ResponseEntity<>(HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @Autowired

    private DisciplineService disciplineService;

    @GetMapping("/disciplines")

    public ResponseEntity<?> allDisciplines() {

        try {

            List<Discipline> allDisciplines = disciplineService.getAllDisciplines();

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

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @GetMapping("/disciplines/{id}")

    public ResponseEntity<?> getDiscipline(@PathVariable("id") int id) {

        try {

            return new ResponseEntity<>(disciplineService.getDiscipline(id), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PostMapping("/disciplines")

    public ResponseEntity<?> saveDiscipline(@RequestBody Discipline discipline) {

        try {

            return new ResponseEntity<>(disciplineService.saveDiscipline(discipline), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @PutMapping("/disciplines")

    public ResponseEntity<?> updateDiscipline(@RequestBody Discipline discipline) {

        try {

            return new ResponseEntity<>(disciplineService.saveDiscipline(discipline), HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    @DeleteMapping("/disciplines/{id}")

    public ResponseEntity<?> deleteDiscipline(@PathVariable("id") int id) {

        try {

            disciplineService.deleteDiscipline(id);

            return new ResponseEntity<>(HttpStatus.OK);

        } catch (Exception e) {

            return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

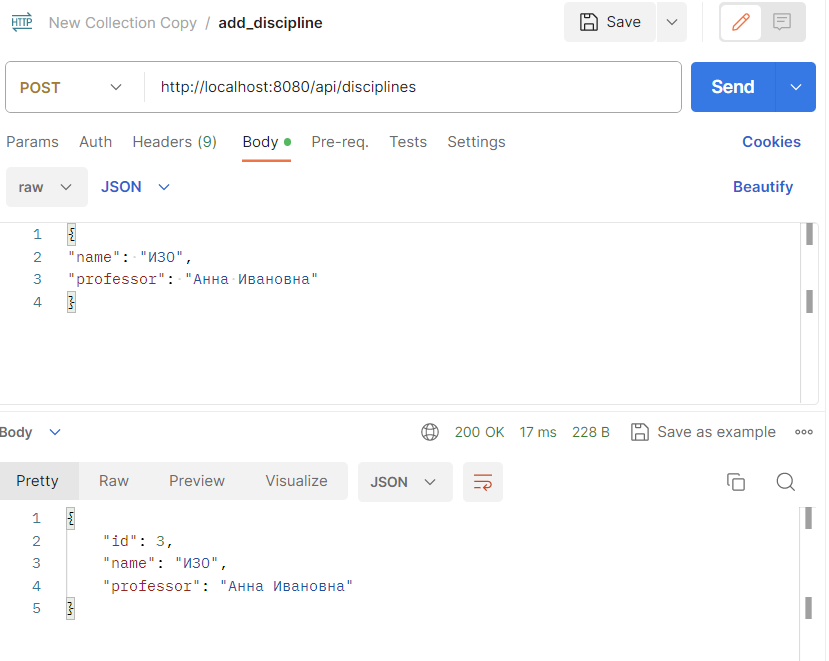
    }

}

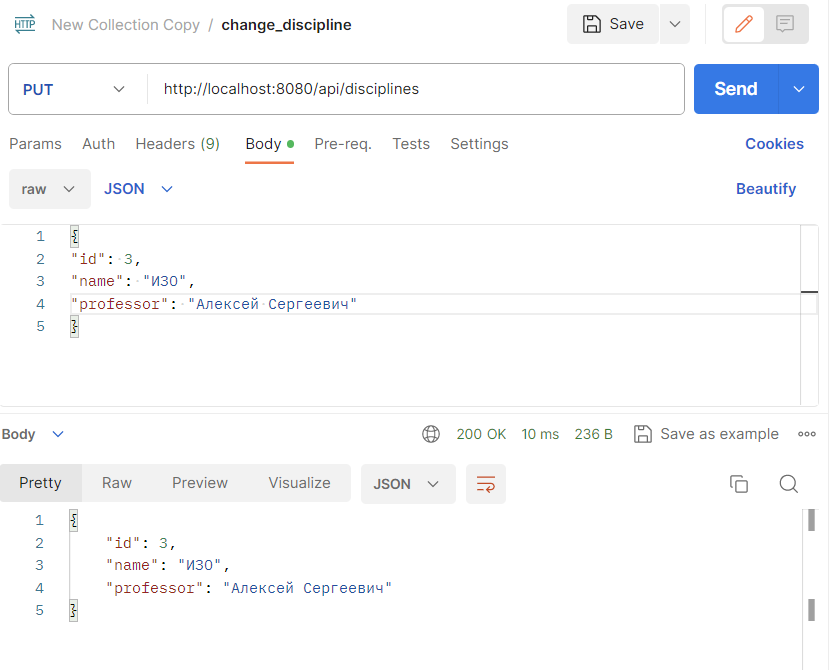
1. Прикрепите скриншоты выполненного ручного тестирования в Postman.

Тестирование методов, созданных в доп. задании 2

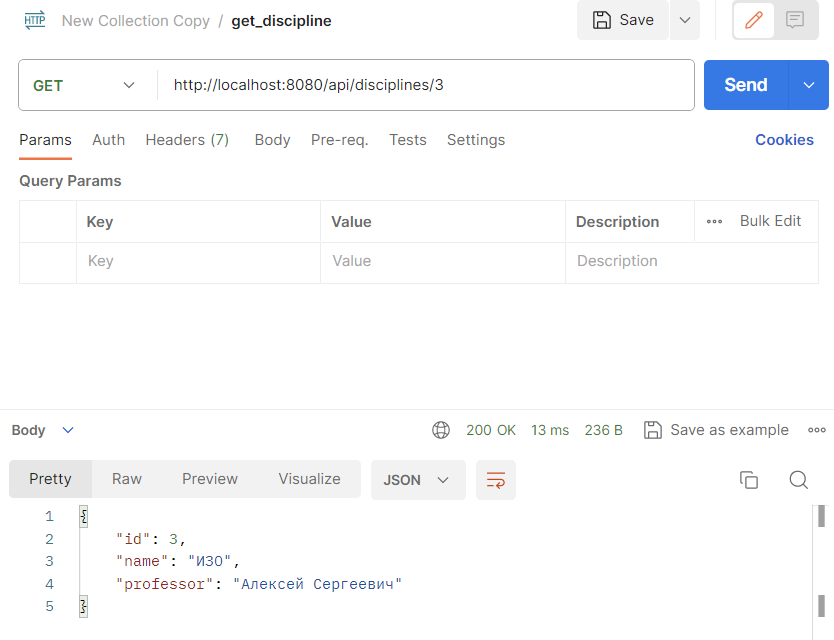
POST запрос для добавления дисциплины:



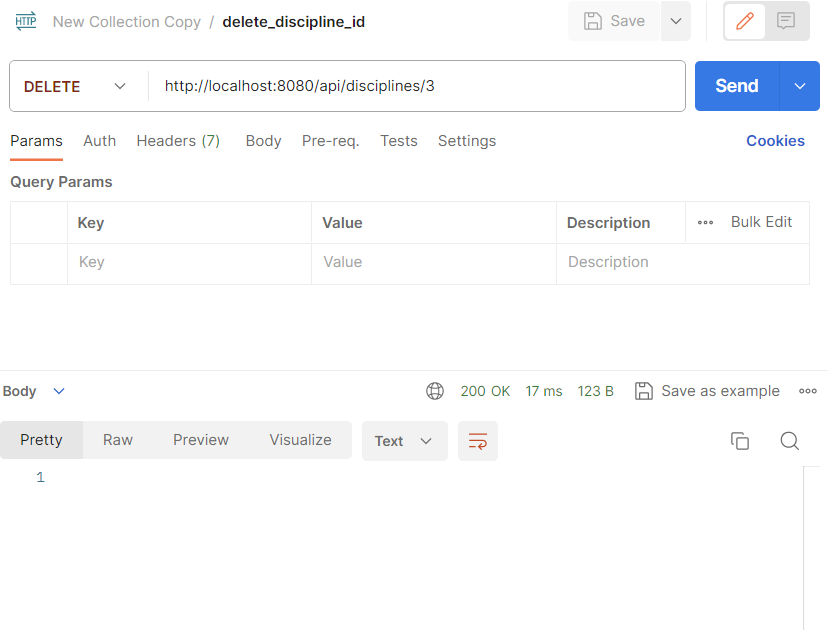
PUT запрос на изменение:



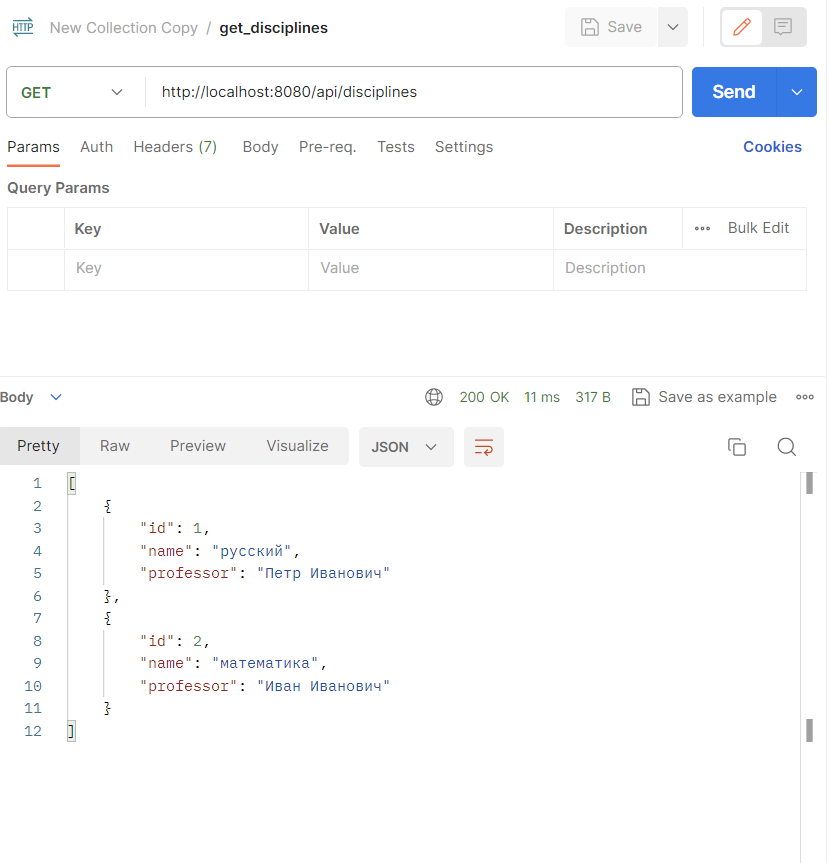
GET запрос на получение дисциплины по id:



GET запрос на удаление дисциплины по id:



GET запрос на получение всех дисциплин:



1. **Ответы на вопросы**

2 вариант

2. Что из себя представляют подходы API-First и Code-First?

Подход API-First предполагает, что разработка начинается с определения API, который будет использоваться для взаимодействия с приложением. При этом сперва определяются требования к API, затем создается его спецификация, и только потом начинается разработка приложения, которое будет использовать этот API. Подход Code-First предполагает, что разработка начинается с создания кода приложения, а затем уже определяется API на основе этого кода.

4. Какие существуют виды интеграции информационных систем? Кратко опишите каждый.

- File-based integration - интеграция на основе обмена файлами;

- Shared database integration - интеграция на основе общей базы данных;

- Messaging integration - интеграция на основе передачи сообщений между системами;

- Remote procedure call (RPC) integration - интеграция на основе удаленных вызовов процедур;

- Service-oriented architecture (SOA) integration - интеграция на основе сервисной архитектуры;

- Application programming interface (API) integration - интеграция на основе использования API.

6. Что такое gRPC, и какая разница между ним и REST?

gRPC — это открытый протокол удаленного вызова процедур, который позволяет эффективно взаимодействовать между разными языками программирования и платформами. Он использует протокол HTTP/2 для передачи данных и поддерживает множество языков программирования. Основное отличие между gRPC и REST заключается в том, что gRPC использует бинарный формат передачи данных, в то время как REST использует текстовый формат (например, JSON или XML). Кроме того, gRPC поддерживает более широкий набор типов данных и более эффективен в использовании ресурсов сети.