ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

ассистент		Кочин Д.А.
	подпись, дата	

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Разработка ресурса REST/JSON сервиса

по дисциплине: ТЕХНОЛОГИИ РАЗРАБОТКИ СЕРВЕРНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4936		Назаров М.Р.
	подпись, дата	фамилия, инициалы

Санкт-Петербург 2021

Лабораторная работа №2

Разработка ресурса REST/JSON сервиса

Цель работы:
Разработать сервер-сервис, обеспечивающий взаимодействие со сторонними клиентскими приложениями посредством REST с использованием для передачи информации формата данных JSON.

Задание на лабораторную работу:

1. Подключите к проекту Maven фреймворк Spring (spring-boot)
2. Определите перечень Rest-сервисов, выполняющих те же действия, что и в лабораторной работе 1. Внимательно отнеситесь к вопросу какой HTTP метод использует тот или иной сервис и какие коды HTTP он может возвращать. Реализуйте эти сервисы.

Вариант №15
Сдача недвижимости в аренду

Ход работы:

1. Создадим проект на основе Sring с помощью среды разработки. В разработке использовался язык программирования Kotlin и сборщик Gradle.
2. Создадим базу данных с помощью Exposed (H2) на основе разработанной сущности в прошлой лабораторной работе. Все дальнейшее взаимодействие с информацией будет происходить через базу данных.
3. Создадим все основные функции необходимые для работы Rest-сервиса: добавление, получение, обновление и удаление данных (CRUD) из базы.
4. Для каждой функции определим путь (начиная с localhost/api/collections) и статусы, которые эта функция может вернуть при разных запросах:
   - GET на список коллекций – OK(200) в случае возврата списка и No Content(204) в случае отсутствиях данных в базе.

- GET на один элемент – Bad Request(400) в случае некорректного запроса, Not Found(404) в случае, если такой элемент не был найден в базе, и OK(200) при успешном выполнении запроса.
- POST – Bad Request(400), если запрос некорректен, и Created(201) в случае успешного выполнение добавления.
- PUT – Bad Request(400), если запрос был некорректный или такого элемента нет в базе, и OK(200) при успешном обновлении состояния выбранного элемента.

- DELETE - Bad Request(400), если запрос был некорректный, Not Found(404), если такого элемента нет в базе, и Accepted(202) при принятии запроса на обработку удаления (или просто начале процесса удаления).

Полный код программы находится в Приложении 1.

5. Данные на сервер(POST, PUT) и с него (GET) передаются в текстовом формате JSON.

6. Проверим работы программы с помощью http запросов в Postman:

Просмотр данных (GET http://127.0.0.1:8080/api/):

Status: 200 OK

[{"id":1,"name":"First apartments","city":"Saint Petersburg","address":"Nevsky Prospect, 3","area":28.0,"cost":27000}, {"id":2,"name":"Second apartments","city":"Saint Petersburg","address":"Nevsky Prospect, 27","area":31.8,"cost":31000}, {"id":3,"name":"Third apartments","city":"Saint Petersburg","address":"Bolshoy prospect, 73","area":70.5,"cost":72999}, {"id":4,"name":"Not apartments","city":"Moscow","address":"Lenin Street, 78","area":25.0,"cost":55000}, {"id":5,"name":"Good apartments","city":"Vorkuta","address":"Peace Street, 27","area":69.0,"cost":26500}]

Просмотр одного элемента (GET http://127.0.0.1:8080/api/1):

Status: 200 OK

{"id":1,"name":"First apartments","city":"Saint Petersburg","address":"Nevsky Prospect, 3","area":28.0,"cost":27000}

Просмотр данных из одного города (http://127.0.0.1:8080/api/?city=Moscow):
Status: 200 OK

[{"id":4,"name":"Not apartments","city":"Moscow","address":"Lenin Street, 78","area":25.0,"cost":55000}]

Добавление данных (POST http://127.0.0.1:8080/api/):

Запрос:

Headers: Content-Type: application/json
Body:

{
"id": 10,
"name": "fdshudgfysdgfy",
"city": "Moscow",
"address": "Lenin Street, 78",
"area": 25.0,

"cost": 55000

}

Ответ:
Status: 201 Created

Body: The new estate was created

Обновление данных (PUT http://127.0.0.1:8080/api/10):

Запрос:

Headers: Content-Type: application/json

Body:

{

"id": 10,

"name": "faruhgurhgkeufghi",

"city": "Moscow",

"address": "Lenin Street, 78",

"area": 25.0,

"cost": 55000

}

Ответ:
Status: 200 OK

Body: The estate is saved

Удаление данных (DELETE http://127.0.0.1:8080/api/collections/10):

Status: 202 Accepted

Body: The estate was deleted

Просмотр результирующих данных (GET http://127.0.0.1:8080/api/collections):

Status: 200 OK
Body:

[{"id":1,"name":"First apartments","city":"Saint Petersburg","address":"Nevsky Prospect, 3","area":28.0,"cost":27000}, {"id":2,"name":"Second apartments","city":"Saint Petersburg","address":"Nevsky Prospect, 27","area":31.8,"cost":31000}, {"id":3,"name":"Third apartments","city":"Saint Petersburg","address":"Bolshoy prospect, 73","area":70.5,"cost":72999},

{"id":4,"name":"Not apartments","city":"Moscow","address":"Lenin Street, 78","area":25.0,"cost":55000}, {"id":5,"name":"Good apartments","city":"Vorkuta","address":"Peace Street, 27","area":69.0,"cost":26500}]

Вывод: в ходе лабораторной работы были получены основные навыки разработки REST сервисов с помощью фрейморков Spring , изучен текстовый формат передачи данных JSON, а также закреплена работа базами данных. Детально разобраны коды состояния HTTP и применены на практике.

Приложение 1.

Serv2NazarovApplication.java:

```
package com.example.serv2nazarov;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Serv2NazarovApplication {

    public static void main(String[] args) {
        SpringApplication.run(Serv2NazarovApplication.class, args);
    }

}
```

EstateController.java:

```
package com.example.serv2nazarov.controllers;

import com.example.serv2nazarov.models.Estate;
import com.example.serv2nazarov.services.EstateService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class EstateController {

    @Autowired
    private EstateService service;

    @GetMapping
    public ResponseEntity<String> getEstates(@RequestParam(required = false) String city) {
        if (city != null) return service.getAllEstates(city);
        else return service.getAllEstates();
    }

    @GetMapping("/{id}")
    public ResponseEntity<String> getEstate(@PathVariable Integer id) { return service.getEstate(id); }

    @PostMapping
    public ResponseEntity<String> createEstate(@RequestBody Estate estate) { return service.createEstate(estate); }

    @PutMapping("/{id}")
    public ResponseEntity<String> updateEstate(@PathVariable Integer id, @RequestBody Estate estate) {
        return service.updateEstate(id, estate);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> removeEstate(@PathVariable Integer id) { return service.removeEstate(id); }
}
```

Estate.java:

```
package com.example.serv2nazarov.models;

import com.google.gson.Gson;

public class Estate {

    private final Integer id;
    private final String name;
    private final String city;
    private final String address;
    private final Double area;
    private final Integer cost;

    public Estate(
            Integer id,
            String name,
            String city,
            String address,
            Double area,
            Integer cost) {

        this.id = id;
        this.name = name;
        this.city = city;
        this.address = address;
        this.area = area;
        this.cost = cost;
    }

    public Integer getId() { return id; }
    public String getName() { return name; }
    public String getCity() { return city; }
    public String getAddress() { return address; }
    public Integer getCost() { return cost; }
    public Double getArea() { return area; }

    @Override
    public String toString() { return new Gson().toJson(this); }
}
```

EstateService.java:

```
package com.example.serv2nazarov.services;

import com.example.serv2nazarov.models.Estate;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

    @Service
    public class EstateService {
```

```
private final static ArrayList<Estate> estateList = new ArrayList(List.of(
        (new Estate(1, "First apartments", "Saint Petersburg", "Nevsky Prospect, 3", 28.0, 27000)),
        (new Estate(2, "Second apartments", "Saint Petersburg", "Nevsky Prospect, 27", 31.8, 31000)),
        (new Estate(3, "Third apartments", "Saint Petersburg", "Bolshoy prospect, 73", 70.5, 72999)),
        (new Estate(4, "Not apartments", "Moscow", "Lenin Street, 78", 25.0, 55000)),
        (new Estate(5, "Good apartments", "Vorkuta", "Peace Street, 27", 69.0, 26500))
));


public ResponseEntity<String> createEstate(Estate estate) {
    Estate value = estateList.stream().filter(e -> estate.getId().equals(e.getId())).findAny().orElse(null);
    if (value != null) return new ResponseEntity<>("The estate is already exist", HttpStatus.BAD_REQUEST);
    else {
        estateList.add(estate);
        return new ResponseEntity<>("The new estate was created", HttpStatus.CREATED);
    }
}

public ResponseEntity<String> getAllEstates() {
    if (estateList.isEmpty()) return new ResponseEntity<>("List of estates empty", HttpStatus.NO_CONTENT);
    else return new ResponseEntity<>(estateList.toString(), HttpStatus.OK);
}

public ResponseEntity<String> getAllEstates(String city) {
    var list = estateList.stream().filter(e -> city.equals(e.getCity())).collect(Collectors.toList());
    if (list.isEmpty()) return new ResponseEntity<>("List of estates empty", HttpStatus.NO_CONTENT);
    else return new ResponseEntity<>(list.toString(), HttpStatus.OK);
}

public ResponseEntity<String> getEstate(Integer id) {
    Estate e = estateList.stream().filter(estate -> id.equals(estate.getId())).findAny().orElse(null);
    if (e == null) {
        return new ResponseEntity<>("The estate does not exist", HttpStatus.NOT_FOUND);
    } else {
        return new ResponseEntity<>(e.toString(), HttpStatus.OK);
    }
}

public ResponseEntity<String> removeEstate(Integer id) {
    Estate e = estateList.stream().filter(estate -> id.equals(estate.getId())).findAny().orElse(null);
    if (e == null) return new ResponseEntity<>("The estate does not exist", HttpStatus.NOT_FOUND);
    else  {
        estateList.removeIf(estate -> id.equals(estate.getId()));
        return new ResponseEntity<>("The estate was deleted", HttpStatus.ACCEPTED);
    }
}

public ResponseEntity<String> updateEstate(Integer id, Estate estate) {
    Estate oldValue = estateList.stream().filter(e -> id.equals(e.getId())).findAny().orElse(null);
    if (oldValue == null) return new ResponseEntity<>("The estate does not exist", HttpStatus.NOT_FOUND);
    else {
        estateList.removeIf(e -> id.equals(e.getId()));
        estateList.add(estate);
        return new ResponseEntity<>("The estate is saved", HttpStatus.OK);
    }
```

```
        }
    }
```