

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №1 по дисциплине   
«Разработка микросервисных приложений»

Обучающийся группы 6232-020402D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Александров А.А.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобанков А.А.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc147329731)

[2 Результаты работы 4](#_Toc147329732)

[2.1 Результаты задания 1 4](#_Toc147329733)

[2.2 Результаты задания 2 8](#_Toc147329734)

[2.3 Результаты задания 3 9](#_Toc147329735)

[2.4 Результат задания 4 10](#_Toc147329736)

[3 Вывод 11](#_Toc147329737)

[ПРИЛОЖЕНИЕ А Листинг программы 12](#_Toc147329738)

1. Постановка задачи

Задание 1

Создать Spring Cloud Config Server приложение. Помимо файла с общими настройками описать файл с настройками для user-service и company-service. Файлы с настройками должны содержать: порт, на котором запуститься микросервис, описание микросервиса.

Задание 2

Создать настройку для test профайла spring для user-service. Переопределить в ней описание микросервиса c использованием environment файла.

Задание 3

Создать два микросервиса: user-service, company-service. Настроить их подключение к Config Server. Создать файл для чтения конфигурации. Создать rest контроллер с get методом, который будет возвращать описание микросервиса из конфигурации. Убедиться, что описание сервисов возвращается корректно.

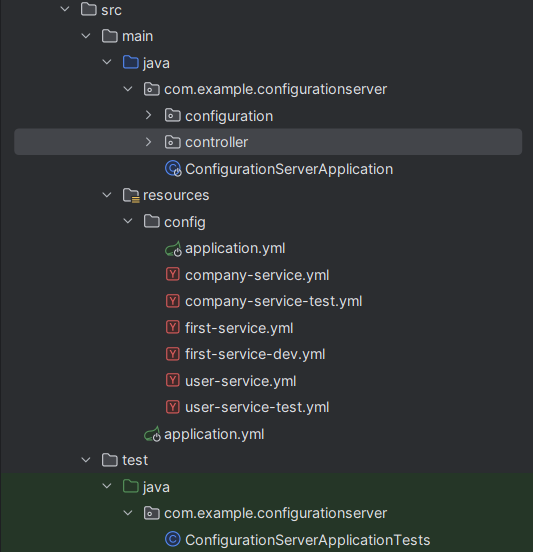
Задание 4

Добавить конфигурацию для запуска user-service с тестовым профайлом. Убедиться, что описание сервиса возвращается корректно.

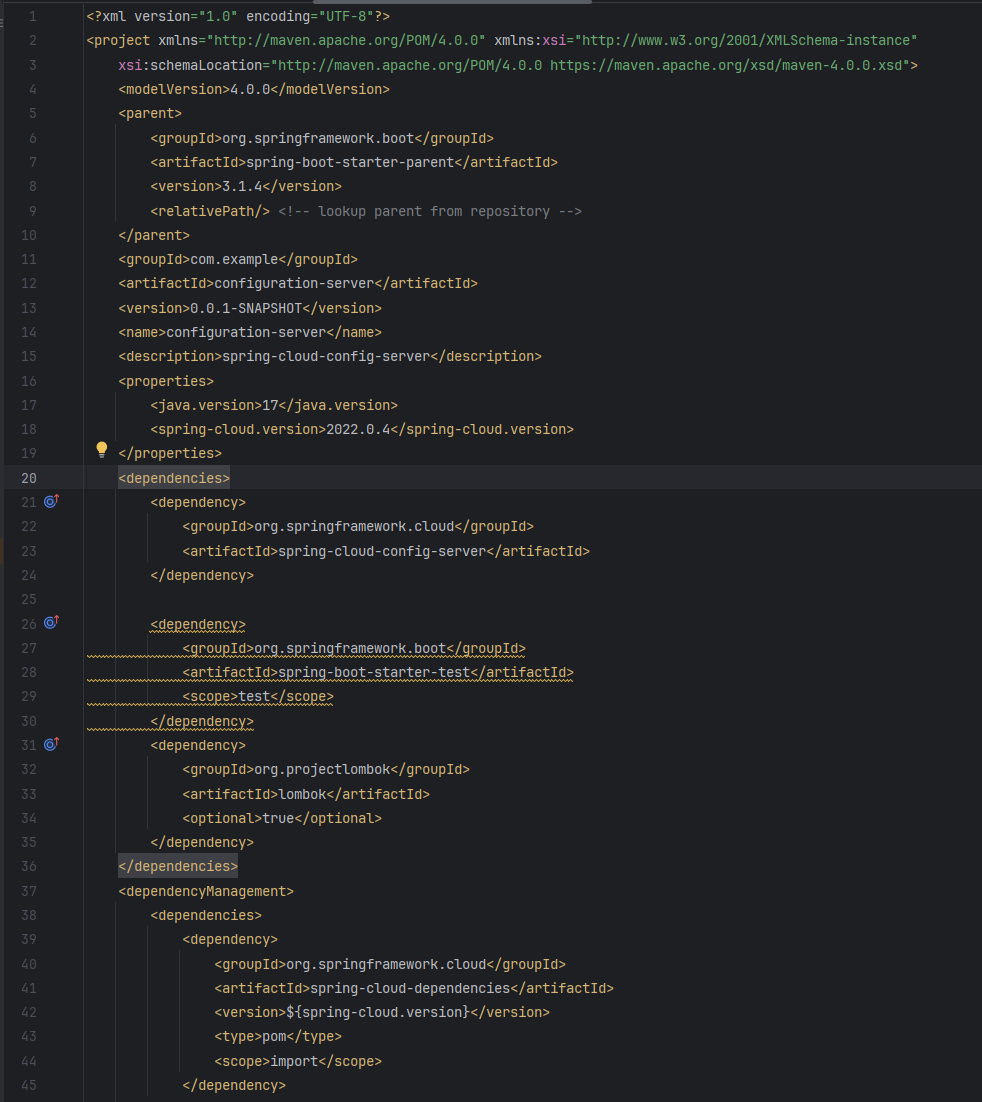
1. Результаты работы
   1. Результаты задания 1

Spring Cloud Config Server представляет собой приложение Spring Boot, поэтому запуск производится из своей IDE (IntelliJ IDEA), основным классом является ConfigurationServerApplication. В качестве сборщика проекта использовался фреймворк Apache Maven.

Структура проекта:

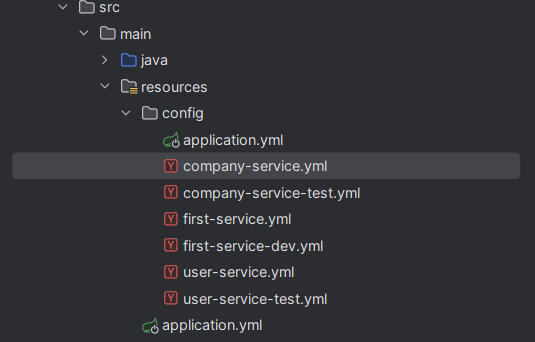


pom.xml:

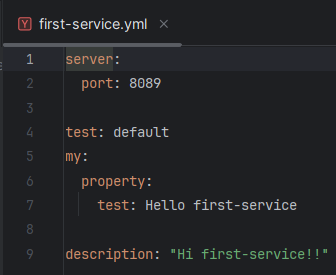




Конфигурационные файлы:



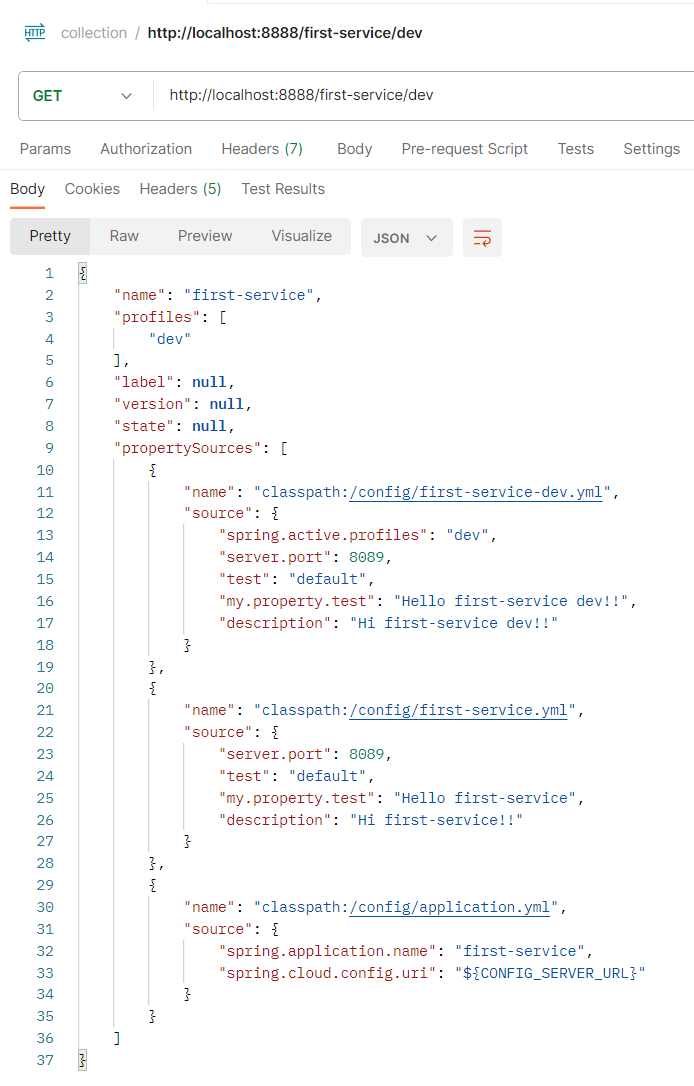
Описание конфигураций:



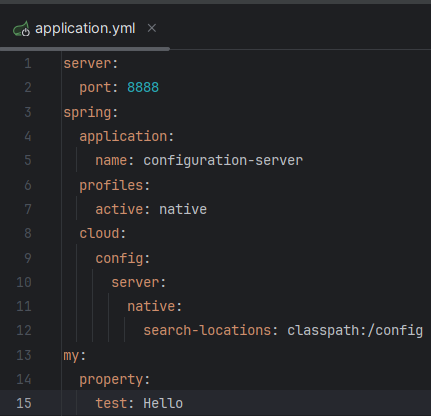


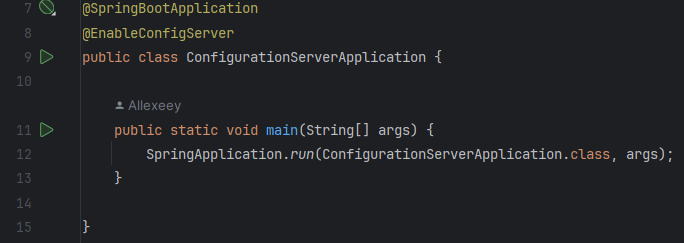
Для остальных конфигураций аналогично, но уже с профилем test.

Получаемая конфигурация:



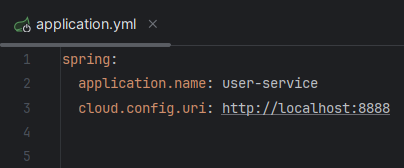
Настройка и запуск Spring Cloud Config:





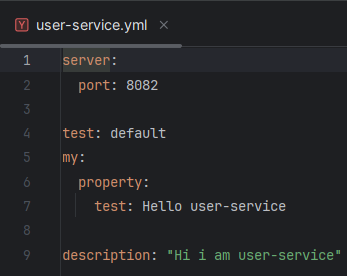
* 1. Результаты задания 2

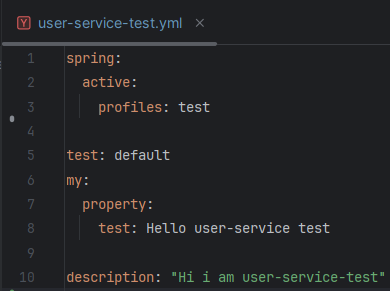
Настройки подключения к Spring Cloud Config Server для каждого сервиса идентичны и заключаются в указание адреса и имени приложения, а также дальнейшей регистрации при старте приложения в качестве провайдера.



* 1. Результаты задания 3

Конфигурация:

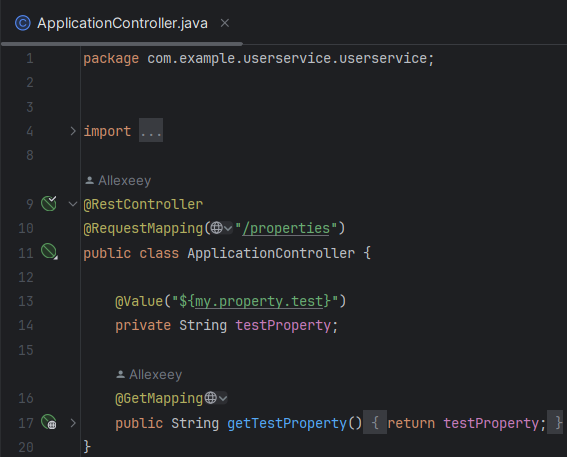




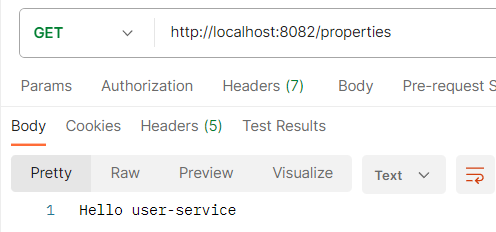
Класс:



rest контроллер:

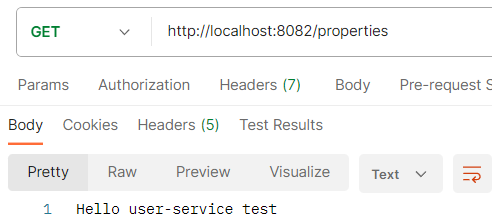


Результат:



* 1. Результат задания 4

Переключаем конфигурацию на test и проверяем конфигурацию:



Для остальных сервисов аналогично, но с другим серверным портом.

1. Вывод

В результате выполнения лабораторной работы был развёрнут Spring Cloud Config Server. Настроено получение и регистрация конфигураций для приложений. Настроены различные конфигурации для разных сред.

Написан отчёт.

[Исходный код доступен по ссылке](https://github.com/Allexeey/microservices)

ПРИЛОЖЕНИЕ А  
Листинг программы

package com.example.configurationserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication

@EnableConfigServer

public class ConfigurationServerApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigurationServerApplication.class, args);

}

}

package com.example.configurationserver.controller;

import com.example.configurationserver.configuration.Config;

import lombok.RequiredArgsConstructor;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequiredArgsConstructor

@RequestMapping("/properties")

public class PropertyController {

private final Config config;

@GetMapping

public String getTestProperty() {

return config.getTestProperty();

}

}

package com.example.configurationserver.configuration;

import lombok.Data;

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

import org.springframework.context.annotation.Configuration;

@Configuration

@Data

public class Config {

@Value("${my.property.test}")

private String testProperty;

}

package com.example.configurationserver;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class ConfigurationServerApplicationTests {

@Test

void contextLoads() {

}

}

server:

port: 8888

spring:

application:

name: configuration-server

profiles:

active: native

cloud:

config:

server:

native:

search-locations: classpath:/config

my:

property:

test: Hello

spring:

application:

name: first-service

cloud:

config:

uri: ${CONFIG\_SERVER\_URL}

server:

port: 8081

test: default

my:

property:

test: Hello company-service

description: "Hi i am company-service"

spring:

active:

profiles: test

test: default

my:

property:

test: Hello company-service test

description: "Hi i am company-service-test"

server:

port: 8089

test: default

my:

property:

test: Hello first-service

description: "Hi first-service!!"

spring:

active:

profiles: dev

test: default

my:

property:

test: Hello first-service dev!!

description: "Hi first-service dev!!"

server:

port: 8082

test: default

my:

property:

test: Hello user-service

description: "Hi i am user-service"

spring:

active:

profiles: test

test: default

my:

property:

test: Hello user-service test

description: "Hi i am user-service-test"

package com.example.userservice.userservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class UserServiceApplication {

public static void main(String[] args) {

SpringApplication.run(UserServiceApplication.class, args);

}

}

package com.example.userservice.userservice;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class UserServiceApplicationTests {

@Test

void contextLoads() {

}

}

package com.example.userservice.userservice;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/properties")

public class ApplicationController {

@Value("${my.property.test}")

private String testProperty;

@GetMapping

public String getTestProperty() {

return testProperty;

}

}

spring:

application.name: user-service

cloud.config.uri: http://localhost:8888