

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 «Автоматизация тестирования»

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гижевская В.Д.

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алкеев М.Г.

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

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146727791)

[2 Результаты работы 4](#_Toc146727792)

[2.1 Результаты задания 1 4](#_Toc146727793)

[2.2 Результаты задания 2 4](#_Toc146727794)

[2.3 Результаты задания 3 4](#_Toc146727795)

[ПРИЛОЖЕНИЕ А Листинг программы 5](#_Toc146727796)

1. Постановка задачи

В процессе выполнения заданий ознакомиться с возможностями Spring Cloud Config Server.

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

**Создание файлов настроек**

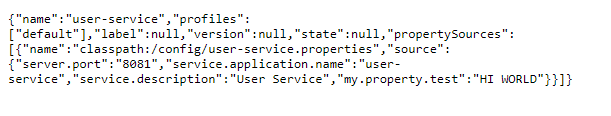
В проекте "lab1" были созданы конфигурационные файлы. Файл "configserver.properties" содержит общие настройки, а файлы "user-service.properties" и "company-service.properties" содержат настройки для соответствующих микросервисов. Каждый файл определяет порт, на котором запустится микросервис, и описание микросервиса.

**Настройка Spring Cloud Config Server**

В проекте "lab1", класс Lab1Application, который является Spring Cloud Config Server, был аннотирован @EnableConfigServer. Класс указывает на использование нативного хранилища конфигураций.

**Запуск Spring Cloud Config Server**

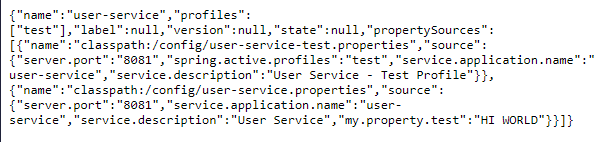
Spring Cloud Config Server запускается, и он готов предоставлять конфигурации для микросервисов.



* 1. Результаты задания 2

**Создание настройки для тестового профил**

В файле "user-service-test.properties" была создана настройка для тестового профиля "test", переопределяя описание микросервиса с использованием environment файла.



* 1. Результаты задания 3

**Создание микросервисов**

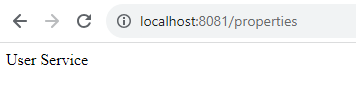
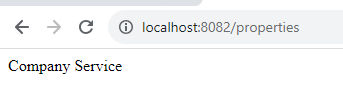
В проектах "user-service" и "company-service" были созданы микросервисы с аннотацией @SpringBootApplication. Убедились, что они правильно настроены для подключения к Spring Cloud Config Server, указав spring.cloud.config.uri в файлах "user-service.properties" и "company-service.properties".

**Создание REST-контроллера**

Для каждого микросервиса создали REST-контроллеры, которые возвращают описание микросервиса из конфигурации. Например, в "user-service" создан контроллер PropertyController, который использует Config для доступа к настройкам.

**Проверка корректности возврата описания сервисов**

Запустили оба микросервиса и убедились, что контроллеры возвращают описание микросервисов, считанное из конфигурации.



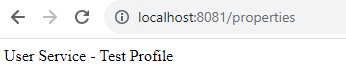
* 1. Результаты задания 4

**Добавление конфигурации для тестового профиля**

В файле "user-service-test.properties" переопределили описание микросервиса для профиля "test".

**Проверка корректности возврата описания сервиса**

Запустили "user-service" с активным профилем "test" и убедились, что контроллер возвращает описание микросервиса, считанное из конфигурации, соответствующей тестовому профилю.



ПРИЛОЖЕНИЕ А  
Листинг программы

**Проект lab1**

spring.config.name=configserver

server.port=8888

spring.application.name=configserver

spring.profiles.active=native

spring.cloud.config.server.native.search-locations=classpath:/config

#spring.cloud.config.server.git.uri=https://github.com/gileraa/config

#spring.cloud.config.server.git.username=gileraa

#spring.cloud.config.server.git.password=fn9Ab37XY36ujuP

#spring.cloud.config.server.git.clone-on-start=true

#spring.cloud.config.server.git.default-label=main

server.port=8081

service.application.name=user-service

service.description=User Service

my.property.test=HI WORLD

server.port=8082

service.application.name=company-service

service.description=Company Service

server.port=8081

spring.active.profiles=test

service.application.name=user-service

service.description=User Service - Test Profile

package com.example.lab1;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableConfigServer

public class Lab1Application {

public static void main(String[] args) {

SpringApplication.run(Lab1Application.class, args);

}

}

**Проект user-service**

spring.application.name=user-service

spring.cloud.config.uri=${CONFIG\_SERVER\_URL}

#spring.cloud.config.server.git.uri=https://github.com/gileraa/microservices-lab1

#spring.cloud.config.server.git.username=gileraa

#spring.cloud.config.server.git.password=fn9Ab37XY36ujuP

#spring.cloud.config.server.git.clone-on-start=true

#spring.cloud.config.server.git.default-label=main

#Properties service

PROPERTIES\_SERVICE\_PORT=8081

#Cloud config server

CONFIG\_SERVER\_URL=http://localhost:8888

package com.userservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class UserServiceApplication {

public static void main(String[] args) {

SpringApplication.run(com.userservice.UserServiceApplication.class, args);

}

}

package com.userservice.controller;

import com.userservice.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.userservice.configuration;

import lombok.Data;

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

import org.springframework.context.annotation.Configuration;

@Data

@Configuration

public class Config {

@Value("${service.description}")

private String testProperty;

}

**Проект company-service**

spring.application.name=company-service

spring.cloud.config.uri=${CONFIG\_SERVER\_URL}

#Properties service

PROPERTIES\_SERVICE\_PORT=8082

#Cloud config server

CONFIG\_SERVER\_URL=http://localhost:8888

package com.companyservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class CompanyServiceApplication {

public static void main(String[] args) {

SpringApplication.run(CompanyServiceApplication.class, args);

}

}

package com.companyservice.controller;

import com.companyservice.configuration.Config;

import lombok.RequiredArgsConstructor;

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.companyservice.configuration;

import lombok.Data;

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

import org.springframework.context.annotation.Configuration;

@Data

@Configuration

public class Config {

@Value("${service.description}")

private String testProperty;

}