# Краткое описание

## Задача

– построить микросервисную систему используя spring security oauth2.0 + okta

## Структура микросервисов:

* Admin-service – необходим администратору для слежения за работой системы
* Eureka-service (consul???)
* Gateway-service(using zuul???)
* Config-service
* Statistic-service – сервис, который выдает статистику (пять лучших студентов – средняя оценка студента + количество работ)
* Student-service – сервис, который хранит данные о студентах
* Article-service – сервис, который хранит данные о статьях

## В проекте используются

* MySql
* Swagger (???)
* ELK (в докере)
* GitHub
* Docker

## API

|  |  |  |
| --- | --- | --- |
| **Statistic**-**service** | | |
| getAllStatistic | Возвращает всю статистику | ADMIN |
| getBestStatistic | Возвращает статистику пяти лучших студентов | ADMIN |
| ping | Проверка работоспособности сервиса | ALL |
| **StudentService** | | |
| getAllStudents | Возвращает всех студентов | ADMIN |
| getStudent | Возвращает одного студента | USER |
| addStudent | Добавляет студента | USER |
| ping | Проверка работоспособности сервиса | ALL |
| **Article-service** | | |
| addArticle | Добавляет статью | USER |
| addGrade | Добавляет оценку конкретной статье | TEACHER |
| getAllArticles | Возвращает все статьи | TEACHER, ADMIN |
| Ping | Проверка работоспособности сервиса | ALL |

# ПОСТРОЕНИЕ МИКРОСЕРВИСОВ

1. Создание проекта на гитхабе

https://github.com/natalyaKh/student-statistic-service

Изображение выглядит как текст

Автоматически созданное описание

1. Добавляем statistic-service (зависимости – WEB, LOMBOOK)

На данный момент еще н6ет безопасности. Кроме этого сейчас, пока еще нет структуры микросервисов – статистика для выборки создается HardCode в package runner)

1. Добавляем student-service – (зависимости – WEB, LOMBOOK)

Так же как и в statistic-service студенты будут созданы herd-code в раннере

1. Добавляем article-service
2. Добавляем eureka-service + actuator – ветка – eureka-service

Теперь, если зайти в ЭврикаДашборд и нажать на один из сервисов из списка – мы перейдет на страницу, которая покажет основные точки actuator.

Изображение выглядит как текст

Автоматически созданное описание

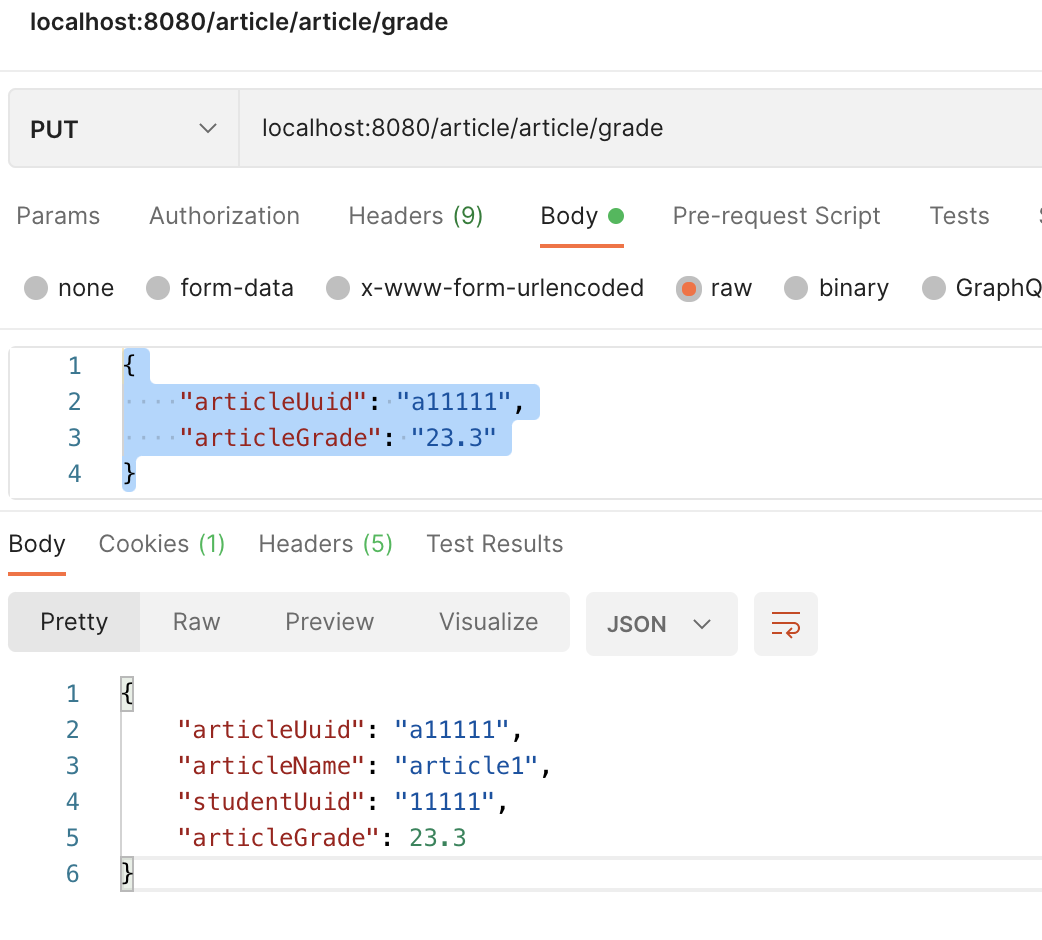
1. Добавляем gateway-service

Запускаем все 5 проектов.

Изображение выглядит как стол

Автоматически созданное описание

Пробуем попасть в микросервисы из gateway



Работает! Линки ->

STATISTIC\_SERVICE  
  
 - localhost:8080/statistic-service/statistic/all  
 - localhost:8080/statistic-service/statistic/best  
 - localhost:8080/statistic-service/ping  
  
STUDENT\_SERVICE  
  
 - localhost:8080/statistic-service/ping  
 - localhost:8080/student-service/students/all  
 - localhost:8080/student-service/students/22222  
  
ARTICLE\_SERVICE  
  
 - localhost:8080/article-service  
 - localhost:8080/article-service/article/all  
 - localhost:8080/article-service/article/grade  
 - body -> {  
 "articleUuid": "a11111",  
 "articleGrade": "23.3"  
 }  
 - PUT

# Добавление oauth2

## Okta - Настраиваем приложение в

1. Логинимся ([email-testprogramsjava@gmail.com](mailto:email-testprogramsjava@gmail.com) password- yellow00

<https://developer.okta.com/>

1. Cоздаем апликацию

Изображение выглядит как текст

Автоматически созданное описание

1. Заполняем данные

|  |  |
| --- | --- |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |
| Изображение выглядит как текст  Автоматически созданное описание |  |

## Gateway Настраиваем приложение

1. Добавляем во все микросервисы

|  |
| --- |
| <dependency>  <groupId>com.okta.spring</groupId>  <artifactId>okta-spring-boot-starter</artifactId>  <version>2.1.0</version> </dependency> <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  <version>2.5.4</version> </dependency> |

1. Добавляем в api-gateway (это у нас сервис ресурсов). Данные берём из того приложения, которое мы создали в okta

okta.oauth2.issuer=https://dev-7817355.okta.com/oauth2/default  
okta.oauth2.client-id=0oa1m531tai5pWqZC5d7  
okta.oauth2.client-secret=Ly-GjdzApRu37vV\_rDjh0zydBebCSSjfrAoFd5x8

1. Добавляем вход по паролю – class SecurityConfig

|  |
| --- |
| @Configuration @EnableGlobalMethodSecurity(prePostEnabled = true) public class SecurityConfig extends WebSecurityConfigurerAdapter {  protected void configure(final HttpSecurity http) throws Exception {  http  .authorizeRequests().anyRequest().authenticated()  .and()  .oauth2Login()  .and()  .oauth2ResourceServer().jwt();  } } |

!!! ветка -> oauth

## Statistic-service Настраиваем приложение

Тут прямой доступ не предполагается, поэтому будем настраивать так, что бы все запросы шли только от apiGateway

1. Добавление зависимостей в pom

После этого, если перезапустить – увидим форму регистрации

1. Добавляем security

@Configuration  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
 protected void configure(final HttpSecurity http) throws Exception {  
 http  
 .authorizeRequests(authorize -> authorize  
 .anyRequest().authenticated()  
 )  
 .oauth2ResourceServer(OAuth2ResourceServerConfigurer::jwt);  
 }  
}

Тут мы указываем, что все запросы должны быть авторизованы, а запросы, которые приходят от сервиса ресурсов (в нашем случае это gateway должны содержать в себе токен.

Теперь зайти напрямую на сервис statistic-service мы уже не можем.

1. Для того, что бы все заработало как нужно, меняем application.properties

Настройки для security

spring:  
 application:  
 name: statistic-service  
 security:  
 oauth2:  
 resourceserver:  
 jwt:  
 issuer-uri: https://dev-7817355.okta.com/oauth2/default  
 client:  
 registration:  
 pricing-client:  
 provider: okta  
 authorization-grant-type: client\_credentials  
 scope: profile  
 client-id: 0oa1lmz0aaF9NkKCi5d7  
 client-secret: roPg\_Q2e26vHB5qfB4VXr-0gyhDPR75LMOB8SBsK  
 provider:  
 okta:  
 issuer-uri: https://dev-7817355.okta.com/oauth2/default

настройки для okta

okta:  
 oauth2:  
 issuer: https://dev-7817355.okta.com/oauth2/default  
 audience: api://default

отключаем ribbon иначе ReactorLoadBalancerавтоконфигурация не будет выполнена.

cloud:  
 loadbalancer:  
 ribbon:  
 enabled: false

все, теперь наш сервис защищен и доступ к нему возможен исключительно из gateway-service

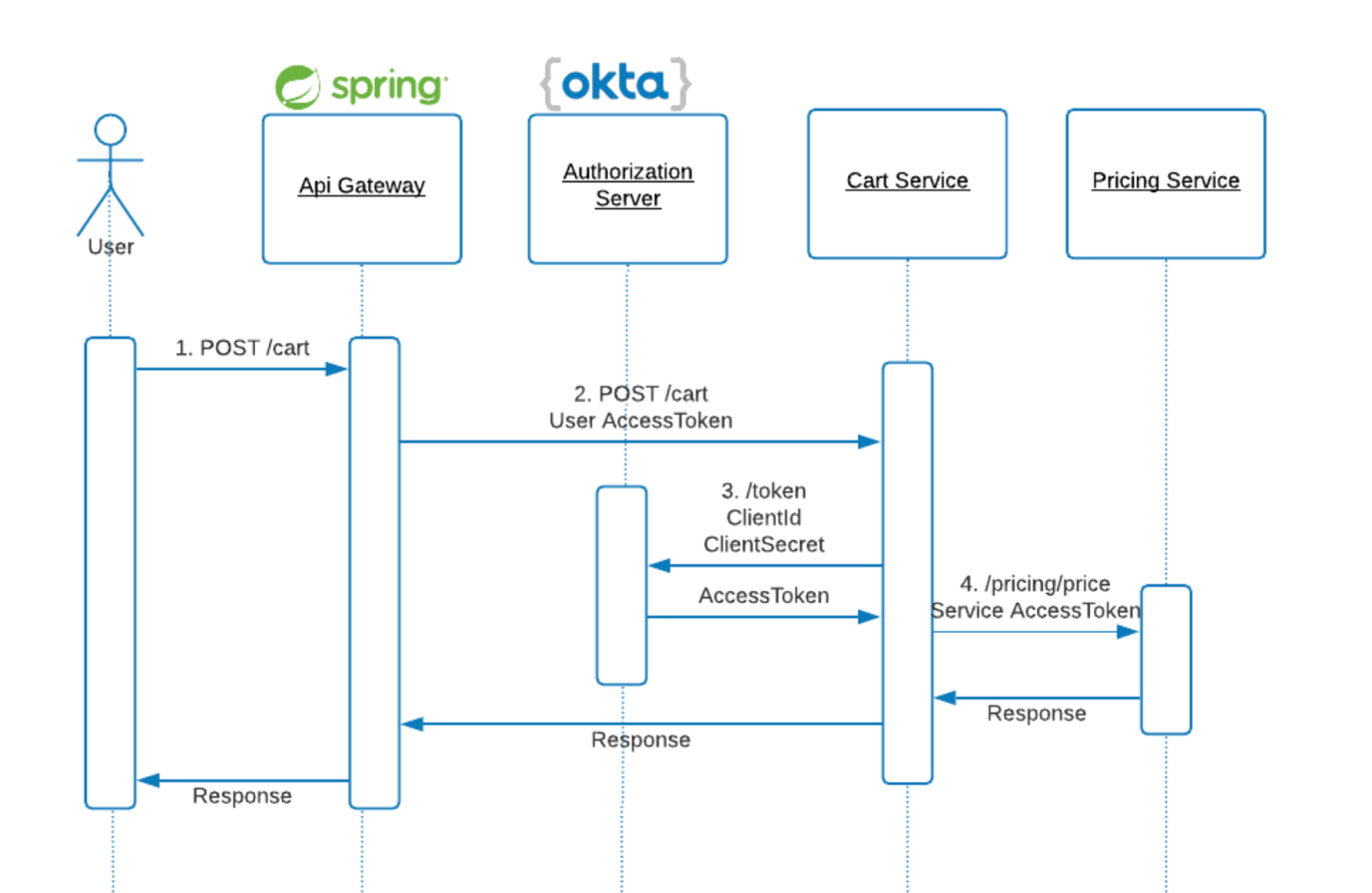
Следующий этап – запросы от микросервиса к микросервису

## Микросервис-Микросервис – настраиваем oauth2

**Пример для statistic-service -> student-service в ветке stat-stud-oauth**

По идеологии наших микро сервисов – сейчас статистка выдается просто hard-code

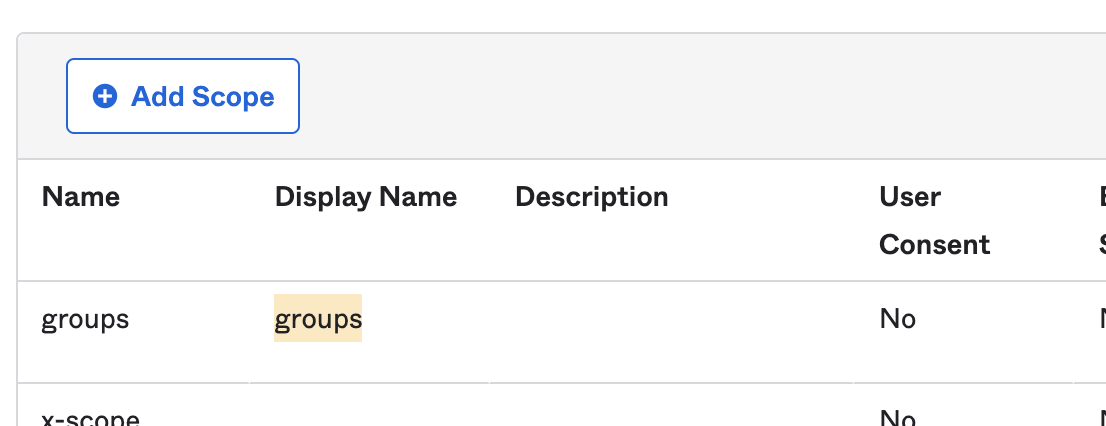
Нам же нужно, что бы перед выдачей сервис получил данные из student-service и из article-service, обработал их и выдал реальный результат.



**ВАЖНО:** Добавление обоих spring-boot-starter-webиspring-boot-starter-webflux модулей результаты в Спринг ботинке автоматической настройки Spring MVC, не WebFlux. Это позволяет приложениям Spring MVC использовать реактивный WebClient.

<https://developer.okta.com/blog/2020/08/14/spring-gateway-patterns#pattern-3-service-to-service-client-credentials-grant>

1. В yml statistic-service ставим scope: groups

Таким образом мы добавляем client scope-groups, который разрешен в нашем приложении в Okta. Те, у кого другой scope не смогут зайти в приложение.

1. В контроллере statistic-service

@Autowired  
StudenService studenService;  
  
  
@GetMapping("/test")  
public List<StudentDto> getStudents(){  
 List<StudentDto> s = studenService.price();  
 return s;  
}

1. Добавляем сервис для отправки запроса в statistic-service

@Service  
public class StudenService {  
  
 @Autowired  
 private WebClient webClient;  
  
 public List<StudentDto> price() {  
 Mono<StudentDto[]> response = webClient.get()  
 .accept(MediaType.*APPLICATION\_JSON*)  
 .retrieve()  
 .bodyToMono(StudentDto[].class).log();  
  
 StudentDto[] objects = response.block();  
 return Arrays.*asList*(objects);  
 }  
}

1. Statistic-servise добавляем StudentDto
2. Самое важное – нужно правильно сконфигурировать WebClient
3. @Configuration  
   public class WebClientConfig {  
     
    @Autowired  
    private ReactorLoadBalancerExchangeFilterFunction lbFunction;  
     
    @Autowired  
    private ObjectMapper objectMapper;  
     
    @Bean  
    public WebClient webClient(ClientRegistrationRepository clientRegistrations,  
    OAuth2AuthorizedClientRepository auth2AuthorizedClients){  
     
    ExchangeStrategies exchangeStrategies = ExchangeStrategies.*builder*()  
    .codecs(configurer ->  
    configurer.defaultCodecs().jackson2JsonDecoder(new Jackson2JsonDecoder(objectMapper)))  
    .build();  
     
    ServletOAuth2AuthorizedClientExchangeFilterFunction oauth2 =  
    new ServletOAuth2AuthorizedClientExchangeFilterFunction(clientRegistrations, auth2AuthorizedClients);  
     
    oauth2.setDefaultClientRegistrationId("pricing-client");  
     
    return WebClient.*builder*().apply(oauth2.oauth2Configuration())  
    .exchangeStrategies(exchangeStrategies)  
    .filter(lbFunction).baseUrl("http://student-service/students/all").build();  
    }  
   }

В приведенном выше коде мы устанавливаем собственный json-декодер, начиная с того, objectMapperкоторый включает доап модули которые мы сможем использовать, если захотим, чтобы денежные суммы были правильно сериализованы и десериализованы. Кроме того, мы установили pricing-clientOAuth 2.0 по умолчанию registrationId. Для обнаружения службы [ReactorLoadBalancerExchangeFilterFunction](https://cloud.spring.io/spring-cloud-commons/reference/html/" \l "webflux-with-reactive-loadbalancer)необходимо добавить в WebClient.

Теперь при запросе <http://localhost:8080/stat/statistic/test> мы получаем список студентов из student-service

Следующий шаг – привести код в порядок, и сделать так, что бы статистика отдавалась уже с данными, которые были получены из сервиса студентов.

Делаем то же самое с article-service

## Исправление кода

Теперь исправляем код так, что бы работала наша логика – при вызове статистики, должна показываться реальная статистика, а не хард код

Теперь при запуске url- localhost:8080/stat/statistic.all мы получаем сначала все данные по клиентам, потом все данные по статьям каждого студента, и после этого только отдаем общую статистику или статистику по 5 лучшим студентам.

Ветка -stat-code-review

Следующий этап – создание сервиса конфигурации и возвращение к одному общему pom файлу