**Микросервисы Spring Boot с Eureka, Zuul прокси с Feign Client, Spring Data c SQL.**

[30 мая 2019 г.](https://medium.com/@iroshan.du/spring-boot-micro-services-with-eureka-and-zuul-proxy-with-fegin-client-68a3ad78453b?source=post_page-----68a3ad78453b--------------------------------)

Я хотел бы проиллюстрировать реализацию микросервиса spring-boot с помощью шлюза eureka и zuul. Кроме того, объясните, как настроить Feign Client для связи между микросервисами. Примерный проект содержит четыре приложения. Обратите внимание, что я не собираюсь объяснять большую часть теоретических аспектов заголовка, скорее, я хочу обсуждать вещи с помощью базы кода.

1. **Серверное приложение Eureka**

Сервер Eureka или реестр служб - это приложение, в котором хранятся все детали микросервисов, например каталог.

2. **User-Service Application**

Микро-сервис, который отвечает за обработку CRUD-операций с данными пользователя, задействовав уровень базы данных.

**3. Resource-Service Application**

Микро-сервис, который внутренне взаимодействует с User-Service, чтобы конечный пользователь мог получать данные о пользователях.

**4. Приложение Gateway-Service (Zuul proxy).**

Gateway-Service или прокси-сервер zuul - это тот, который остается у входной двери в любом HTTP-запросе. Конечный пользователь видит только URL-адрес прокси-сервера zuul, и как только пользователь запускает HTTP-запрос, прокси-сервер zuul более разумным образом запрашивает конкретную микрослужбу.

Тут будет:

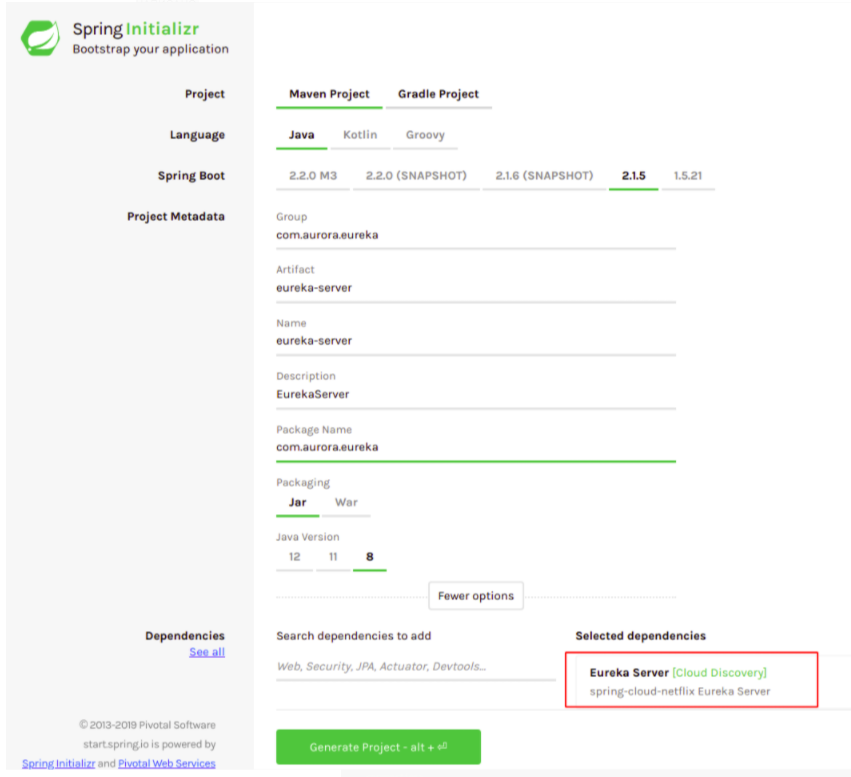
- Eureka сервер,

- Микросервис 1 который регистрируется в Eureka сервере, в котором прописана работа с БД,

- Микросервис 2 который регистрируется в Eureka сервере, в котором, используя механизм Feign клиент, будут вызывать методы из Контроллера Микросервиса 1,

- Zuul сервер, который регистриру**ется** в Eureka сервере, регистриру**ет** у себя два микросервиса и выполняет функцию, который только осуществляет маршрутизацию (*без фильтров и остального*).

1.Create Eureka server with Spring Initializr( https://start.spring.io/)



1.1 pom.xml в Eureka Server



1.2 Основной класс приложения на сервере Eureka.

@ **EnableEurekaServer** : включить сервер **eureka** .

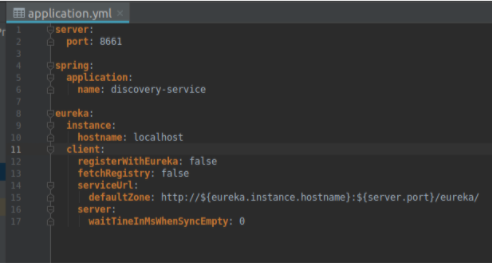


1.3 Файл application.yml сервера Eureka.

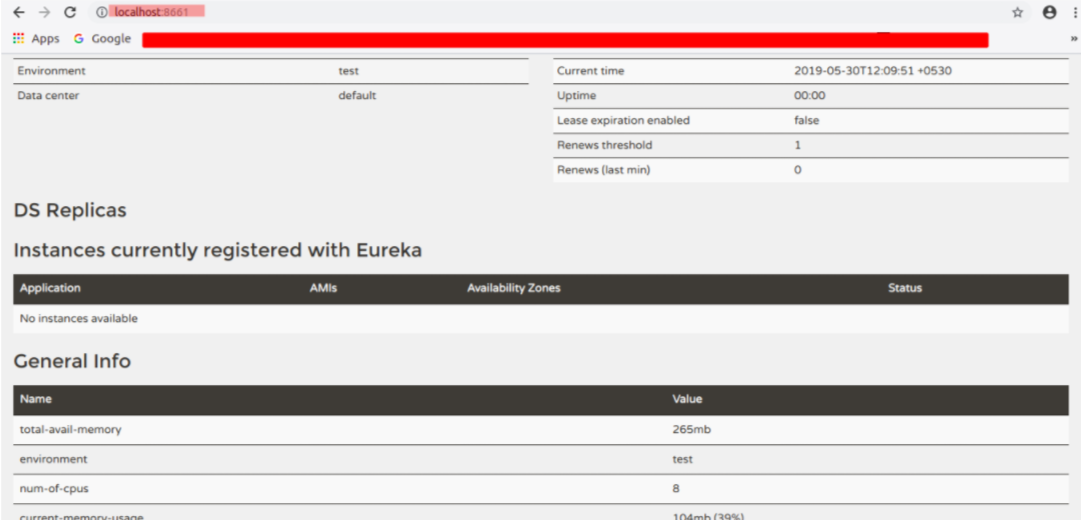
*Строки 1-> 3:* Назначенный порт службы или приложения

*Строки 4-> 6:* Название службы или приложения

*Строки 8-> 17:* Эти конфигурации используются для регистрации данной микрослужбы на сервере eureka. Но самому серверу eureka не нужно регистрироваться на нем. Вот почему атрибуты ***registerWithEureka*** и ***fetchRegistry*** имеют значение false.



1.4. Запуск и запуск Eureka Server с настроенным портом в файле application.yml 8661.



2. Создайте ресурс-сервис с помощью spring initialzr, добавив следующие зависимости. Микросервис 1.

* Eureka Discovery (Облачное открытие)
* RestRepositories (Интернет)
* JPA (SQL)
* MySQL (SQL)

Микросервис 1 состоит из:

1.  ***pom.xml***,

2. ***application.yml***, с номером порта, именем для регистрации на серверах, данными для подключения к БД и данными для регистрации на Eureka,

3. ***UserDetails*** *класса-сущности*,

4. ***UserRepository*** *класса-репозитория*, который будет выполнять запрос к БД для получения из сущности необходимых полей,

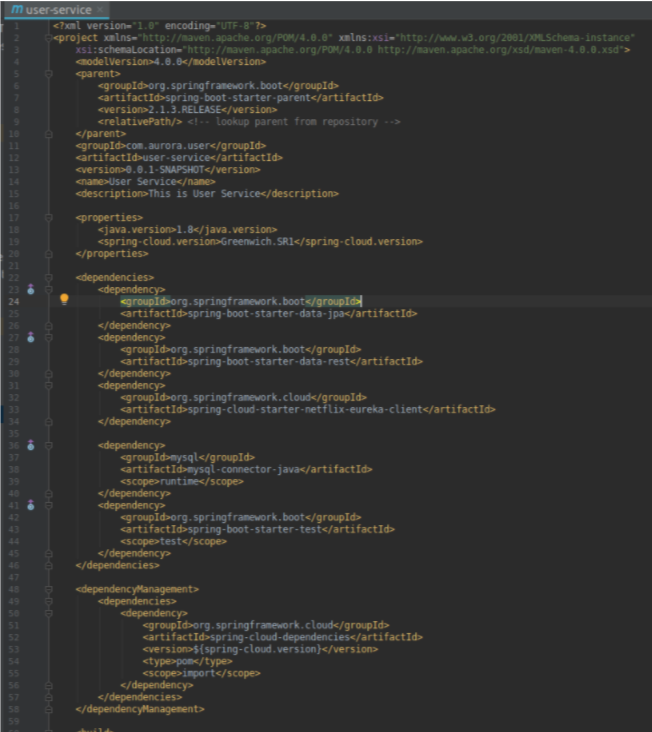
5. ***UserDetailsDTO*** *класса-прокладки* *DTO*, который, скорее всего, всегда будет “целиком и полностью” состоять только из данных полученных из запроса класса-репозитория,

6. ***UserService*** *класса-сервиса*, в который из класса-контроллера вынесена вся логика, причем он в своих методах будет принимать на вход и отдавать на выход данные класса-прокладки DTO,

7. ***UserController*** *класса-контроллера*, подтягивающего механизмы работы из класса-сервиса,

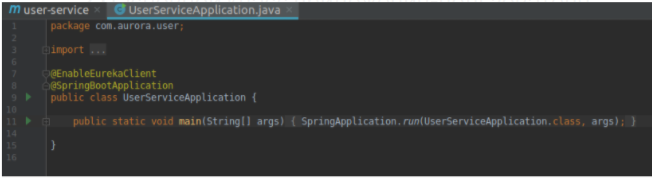
8. **UserServiceApplication** *main-класса*, с аннотацией для регистрации на Eureka сервере.

Файл pom.xml должен быть таким.



2.1 Основной класс приложения-пользователя.

**@EnableEurekaClient** : включить возможности регистрации в качестве микросервиса на сервере eureka.



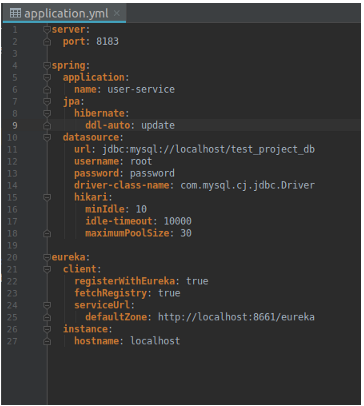
2.2 Файл user-service application.yml.

*Строка 7-> 18:*

Эта конфигурация используется для обработки подключения к базе данных с использованием имени пользователя и пароля (см. Также свойства Hikari).

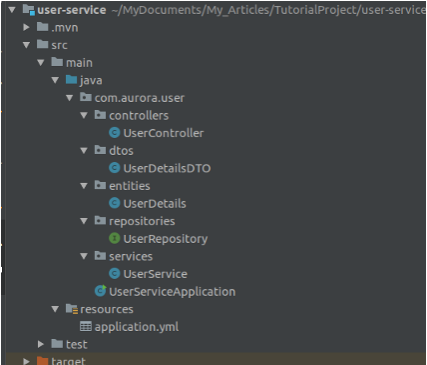
*Строка 22-> 23:*

Флаги установлены в значение true, чтобы зарегистрировать службу пользователя на сервере eureka как микросервис.

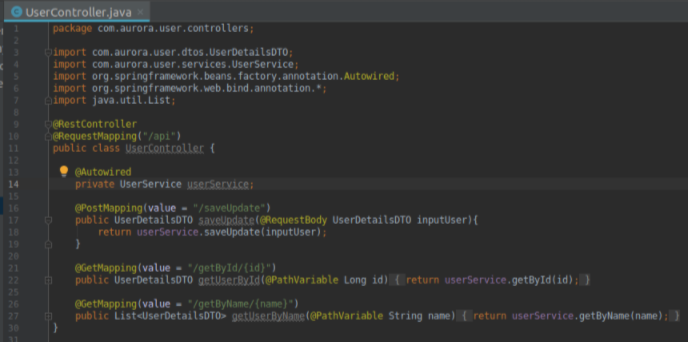


2.3 Структура проекта пользователь-сервис

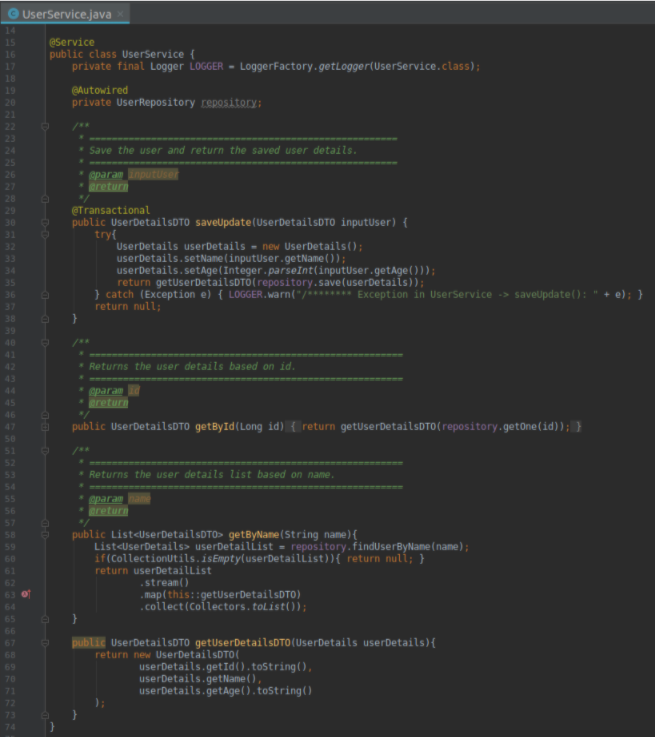
Этот фиктивный микросервис имеет собственную базу данных для управления пользовательскими данными. Многоуровневая структура проекта состоит из следующих компонентов.



* **Контроллеры** : предоставьте конечные точки API для операций CRUD, где Сервер с Клиентом обмениваются перечнем данных из которых состоит класс-прокладка DTO (UserDetailsDTO)



* **Услуги** : обработка бизнес-логики



сохраняет, но и возвращает сохраненную сущность

используем встроенный метод, который не только

делаем метод транзакционным

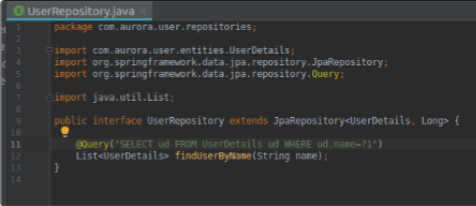
получаем коллекцию List, состоящую из объектов DTO класса

для всех элементов stream-а, из объекта текущего класса (т.к. “this”), вызывается метод getUserDetailsDTO(), в который элементы stream-а вставляются в виде параметра

используем встроенный метод

используем Собственный метод

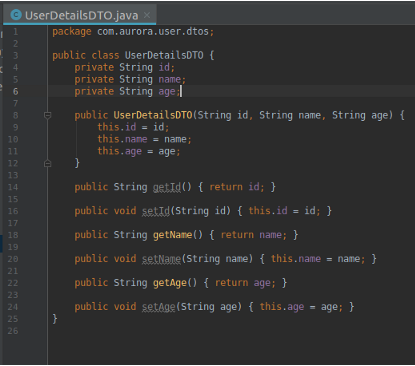
* **Хранилища** : данные сохраняются после повторного запроса в / из базы данных, с использованием механизма Spring Data. Этот интерфейс будет использоваться в Сервисе, в который вынесена логика из Контроллера. В интерфейсе в методе findUserByName() делаться запрос к БД написанный в @Query(). Причем данные, переданные в качестве параметра метода findUserByName(), будут подставлены в запрос вместо цифры 1:



* **Entites** : представляют таблицу базы данных как объекты.



* **Dtos** : представление объекта или набора объектов.



3. Создайте ресурс-сервис с помощью spring initialzr, добавив следующие зависимости. Микросервис 2.

* Eureka Discovery (Облачное открытие)
* RestRepositories (Интернет)
* Feign (облачная маршрутизация)

Микросервис 2 состоит из:

1. ***pom.xml***,

2. ***application.yml***, с номером порта, именем для регистрации на серверах и данными для регистрации на Eureka,

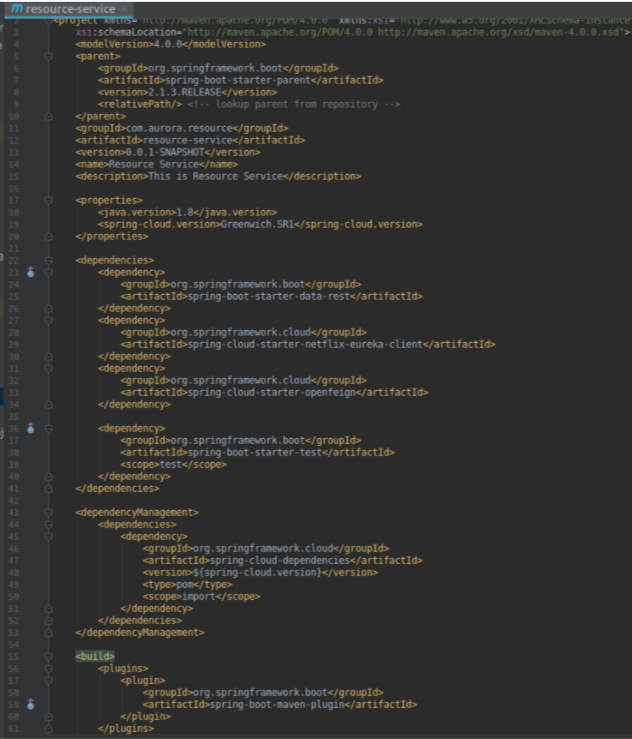
3. ***UserDetailsDTO*** класса-сущности или класса-DTO *(если необходимый класс в запрашиваемом микросервисе использует для принятия/возврата информацию класс-DTO)* аналогичные таким же в запрашиваемом микросервисе,

4. ***UserServiceFeign*** интерфейса со всеми методами необходимого класса-контроллера в запрашиваемом микросервисе,

5. ***UserResourceController*** класса-сервиса или класса-контроллера, в котором мы создаем объект созданного ранее интерфейса и потом вызываем из него необходимый метод.

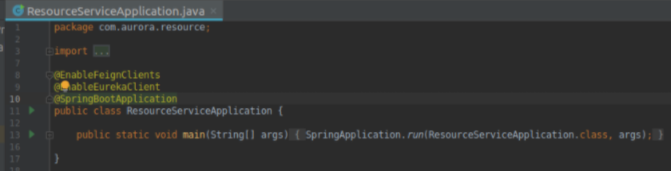
6. ***ResourceServiceApplication*** класса-main, с аннотацией для регистрации на Eureka сервере и аннотацией для запуска механизма Feign.

Тогда pom.xml должен быть таким.



3.1 Основной класс ресурс-сервисного приложения.

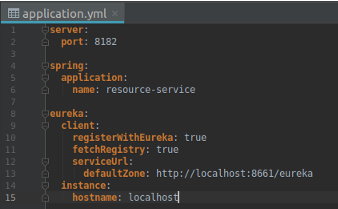
**@EnableFeignClient**: Включить возможности связи между зарегистрированными микросервисами на сервере eureka. Этот фиктивный микросервис имеет собственную базу данных для управления пользовательскими данными. Многоуровневая структура проекта состоит из следующих компонентов.



3.2 Файл-ресурс application.yml.

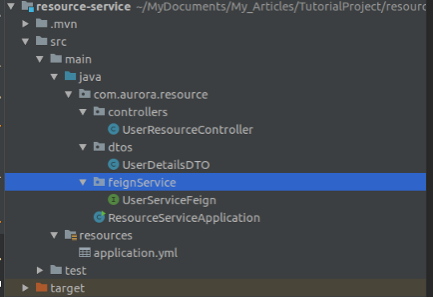
*Строка 10 -> 11:*

Флаги установлены в значение true, чтобы зарегистрировать ресурс-службу на сервере eureka как микросервис.

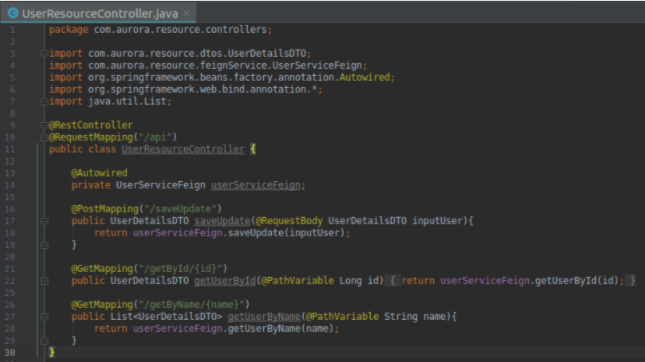


3.3 Структура проекта ресурс-сервис

Это место, где в картину входит симулируемый клиент. Ресурс-служба внутренне взаимодействует с пользовательской-службой для получения и отправки данных между микрослужбой. Посмотрим, как пойдет процесс.



* **UserResourceController** : у него есть конечные точки API, доступные для остального мира. Как только пользователь запускает HTTP-вызов api, ресурс-сервер связывается с пользовательской службой через имитацию клиента. Тут я, используя механизм Feign клиента, вызываю методы из Контроллера первого микросервиса.



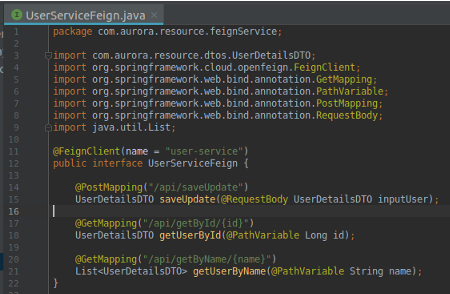
* **UserServiceFeign** : это место, где происходит волшебство.

@ **FeginClient (name = «пользователь-сервис»)** :

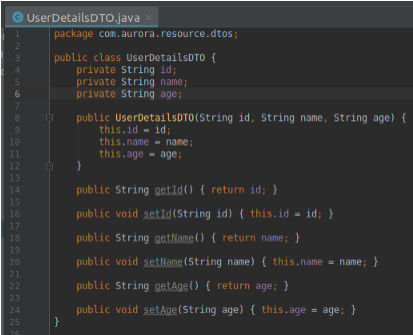
Включите связь с пользовательским сервисом через fegin client. Значение атрибута name должно быть именем приложения службы пользователя, упомянутой в application.yml.

**Сопоставления API GET и POST**

Остальная часть всех сопоставлений GET и POST должна быть такой же, как конечные точки API, доступные в контроллере службы пользователя. (обратитесь к контроллеру службы пользователя). Затем клиент fegin разумно разрешает конечную точку.



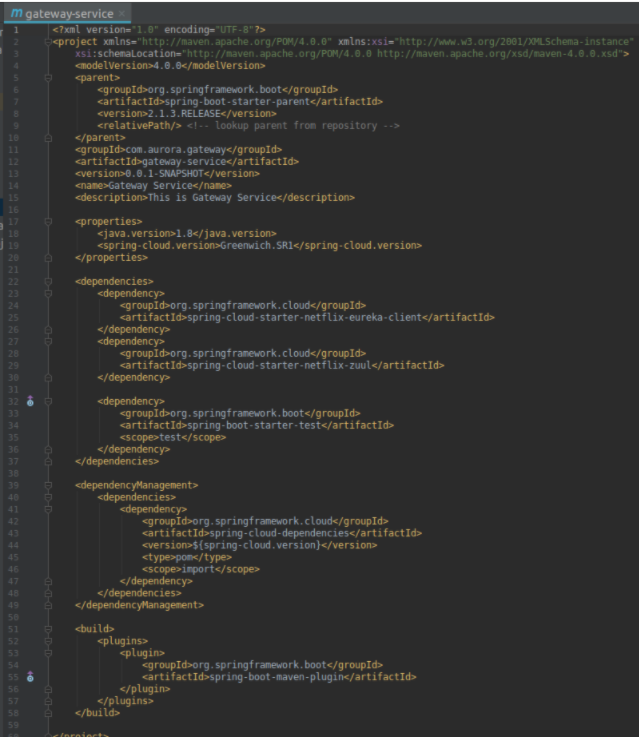
* **UserDetailsDTO:** должно быть таким же, как UserDetailsDTO Микросервиса 1. Создаем именно класс-DTO, а не класс-сущности, т.к. класс-сервис и класс-контроллер в Микросервисе 1 используют в виде получаемых/отдаваемых данных именно класс-DTO



4. Создайте сервис-шлюз (прокси Zuul) с помощью spring initializr, добавив следующие зависимости.

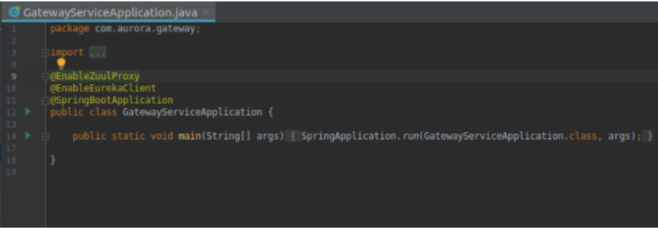
* Eureka Discovery (Облачное открытие)
* Zuul (облачная маршрутизация)

Затем файл pom.xml следующим образом.



**4.1 Основной класс Gateway-service (Zuul proxy).**

@ **EnableZuulProxy** : включить функции прокси-сервера zuul, такие как интеллектуальная маршрутизация.



4.2 Файл application.yml службы шлюза.

*Строка 17-> 30:*

Эта конфигурация отображает конфигурацию интеллектуальной маршрутизации в нескольких микросервисах. Мы называем это интеллектуальной маршрутизацией, потому что конечный пользователь не знает точного IP-адреса каждой микрослужбы, а знает только имена служб. ***ресурс-сервис*** и ***пользователь-сервис****являются* такими микросервисами.

**путь** :

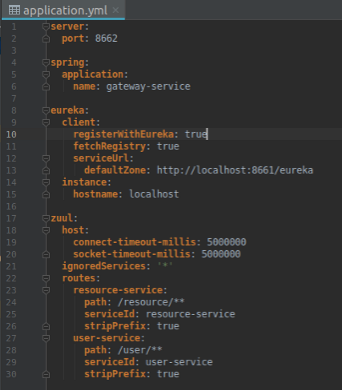
Обратитесь к префиксу URL-адреса конечной точки api, чтобы определить конкретную микросервису

например: [http: // localhost: 8662 / resource / api / saveUpdate](http://localhost:8662/resource/api/saveUpdate)

Zuul (gateway-service- localhost: 8762) достаточно умнее, чтобы маршрутизировать HTTP-запрос в приложение службы ресурсов (микрослужба)

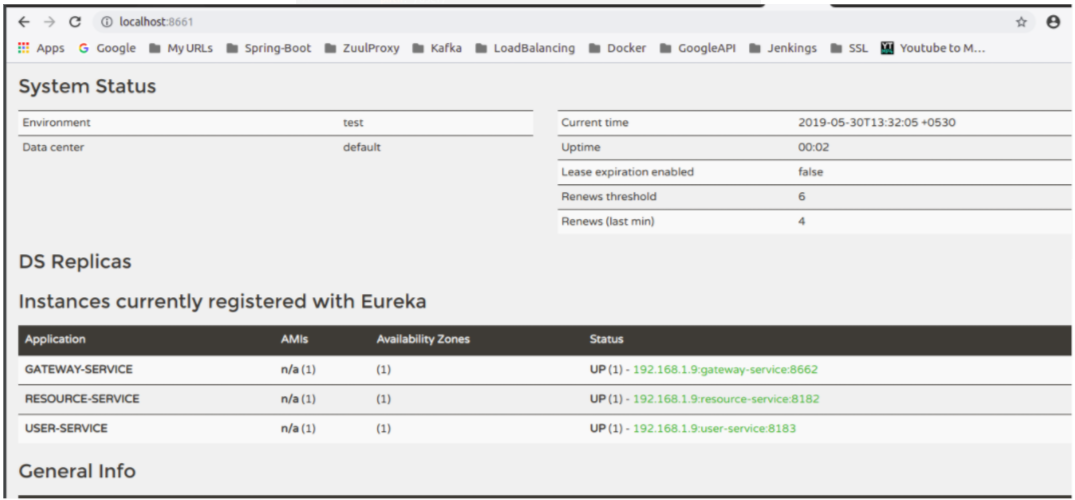
**serviceId** :

Должно быть имя приложения для каждой ***микрослужбы,*** которое уже определено в конкретном файле application.yml как свойство ***«spring.application.name»*** .



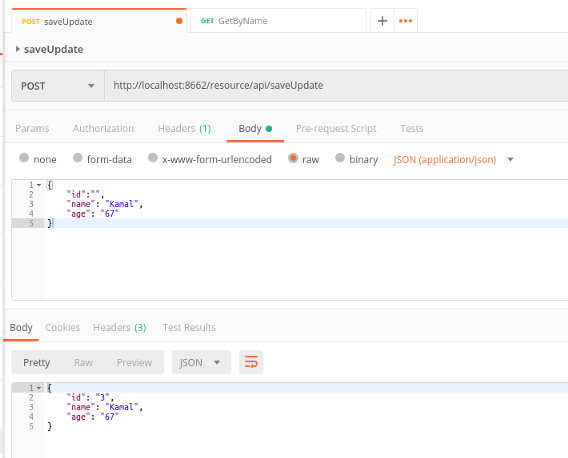
Пришло время проверить нашу тяжелую работу….

Запустите все проекты и посмотрите в Eureka, все ли службы работают…

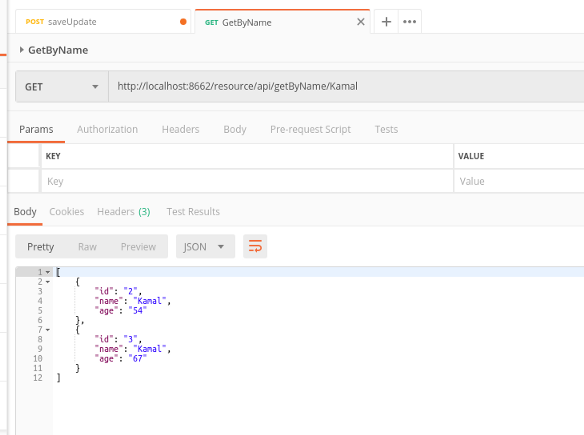


Протестируйте конечные точки API с помощью postman (*первая запись в БД уже есть*)

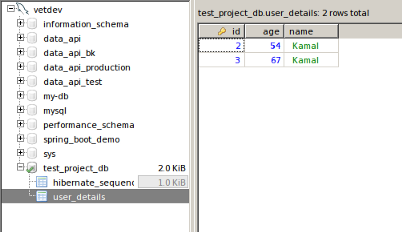
1. **SaveUpdate end point**



**2. GetByName API end point**



How is the database look like ……?????



Have a Nice Day !!!!!!!