## **Lab 7 - Simple API Gateway with Zuul**

****Part 1 - Startup****

1. Stop ALL of the services that you may have running from previous exercises. If using an IDE you may also wish to close all of the projects that are not related to "lab-7”.
2. Start the common-config-server and common-eureka-server.
3. Lab 7 has copies of the word server. Start 5 separate copies of the lab-9-word-server, using the profiles "subject", "verb", "article", "adjective", and "noun". There are several ways to do this, depending on your preference:
4. If you wish to use Maven, open separate command prompts in the target directory and run these commands:
   * mvn spring-boot:run -Dspring.profiles.active=subject
   * mvn spring-boot:run -Dspring.profiles.active=verb
   * mvn spring-boot:run -Dspring.profiles.active=article
   * mvn spring-boot:run -Dspring.profiles.active=adjective
   * mvn spring-boot:run -Dspring.profiles.active=noun
5. Or if you wish to run from directly within STS, right click on the project, Run As... / Run Configurations... . From the Spring Boot tab specify a Profile of "subject", UNCHECK live bean support, and Run. Repeat this process (or copy the run configuration) for the profiles "verb", "article", "adjective", "noun".
6. Check Eureka at [http://localhost:8010](http://localhost:8010/). Any warnings about running a single instance are expected. Ensure that each of your 5 applications are eventually listed in the "Application" section, bearing in mind it may take a few moments for the registration process to be 100% complete.
7. Optional - If you wish, you can click on the link to the right of any of these servers. Replace the "/info" with "/" and refresh several times. You can observe the randomly generated words.

****Part 2 - Start and Examine Existing System****

1. Open lab-7-gateway. This is a simple Spring Boot web application. We will modify it to be a simple API gateway with Zuul.
2. Examine the templates/sentence.html page. Notice that it contains JavaScript for making AJAX calls to obtain the different parts of the sentence. There are 5 separate calls to make, each on a different server. How can the JavaScript make these calls without encountering cross site scripting restrictions?
3. Run lab-7-gateway. Access [http://localhost:8080](http://localhost:8080/). You should encounter errors as the various AJAX calls cannot be completed successfully. We will fix this next.

****Part 3 - Implement a Zuul Reverse Proxy****

1. Stop the lab7-gateway application.
2. Add the dependency for the config client. org.springframework.cloud / spring-cloud-config-client.
3. Add the dependency for Eureka-based service discovery. org.springframework.cloud / spring-cloud-starter-netflix-eureka-client.
4. Add the dependency for Zuul. org.springframework.cloud / spring-cloud-starter-netflix-zuul.
5. If using Eclipse, the M2E plugin may require you to update your project at this point. Right click on the project / Maven / Update Project
6. Setup the application to obtain configuration from the config server on startup. Do you remember how to do this? Open bootstrap.yml and add the location of the configuration server. For a reminder how to do this, consult the configuration of the word server.
7. Open the main application class and add the annotation to enable Zuul proxy.
8. Save your work. Run the application. Access [http://localhost:8080](http://localhost:8080/). The sentence should build correctly with no errors.

****Part 4 - Add a service prefix**** Our web page expects JavaScript and CSS resources to be located under "/js" and "/css" respectively. Let's adjust our system so that all calls to the back-end microservices are under "/services".

1. Open the templates/sentences.html page. Find the TODO comment around line 30. Change the prefix variable to "/services". Notice how the variable is used in the next few lines.
2. Refresh the page in the browser. We should get errors at this point. Do you understand why?
3. Open application.yml. Set the zuul prefix to "/services". Save all work and restart.

****Part 5 - Add ETag Support**** At present our server is sending back individual word values for the AJAX requests even if the browser already has the value being sent. ETags can be used to eliminate the need to send a payload to the client when nothing has changed.

1. Within your browser, open Developer Tools (Internet Explorer / Chrome), Firebug (Firefox), Web Inspector (Safari), refresh the web page, and examine the network activity. The browser is receiving a 304 code instead of 200 for the JavaScript and CSS files since they are unchanged. Let's add similar support for the word AJAX calls if they are unchanged.
2. Open your main Application class and add this Bean:

```

@Bean

public ShallowEtagHeaderFilter shallowEtagHeaderFilter() {

return new ShallowEtagHeaderFilter();

}

```

1. Save your work and restart. Refresh the browser several times. Notice that we randomly receive 304s for the AJAX requests instead of 200s. Do you understand why this is random?

****Reflection****

1. How does the application know where the individual word services are? Zuul automatically uses Eureka service discovery.
2. How did this application know how to contact Eureka? We used Spring Cloud Config, and the server / repository we are using knows the location.
3. Why do we get 304s randomly on the AJAX requests? Since the word values are randomly generated, 304s only occur in the unlikely event that the server returns an identical value to what it returned in the previous request.
4. ETags are a great way to optimize web / REST applications, but the ETag usage demonstrated here is impractical for two reasons. 1) the values we are receiving are intended to be random, 304s only occur because our set of seed values is relatively small, and 2) the ETag itself is far larger than any of our words, so we actually consume more bandwidth than we save!
5. This web site uses Thymeleaf, JQuery, and Bootstrap, though the usage of each is very rudimentary. The application.properties file has a setting that allows the Thymeleaf template changes to be loaded immediately, which is useful in development.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*