

( / )

# Spring REST with a Zuul Proxy

Last modified: November 5, 2018

by baeldung (<https://www.baeldung.com/author/baeldung/>)

**REST** (<https://www.baeldung.com/category/rest/>)

**Spring Cloud** (<https://www.baeldung.com/category/spring/spring-cloud/>)

**Zuul** (<https://www.baeldung.com/tag/zuul/>)

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

**>> CHECK OUT THE COURSE** (</ls-course-start>)

## 1. Overview

In this article, we'll explore the **communication between a front-end application and a REST API that are deployed separately**.

The goal is to work around CORS and the Same Origin Policy restriction of the browser and allow the UI to call the API even though they don't share the same origin.



We'll basically create two separate applications – a UI application and a simple REST API, and we'll use **the Zuul proxy** in the UI application to proxy calls to the REST API.

Zuul is a JVM based router and server side load balancer by Netflix. And Spring Cloud has a nice integration with an embedded Zuul proxy – which is what we'll use here.

**Further reading:** To find out more, you can read the full [Privacy and Cookie Policy](/privacy-policy/) (</privacy-policy/>)

Ok

## An Example of Load Balancing with Zuul and Eureka (<https://www.baeldung.com/zuul-load-balancing>)

See how load-balancing with Netflix Zuul looks like.

**Read more** (<https://www.baeldung.com/zuul-load-balancing>) →

## Setting Up Swagger 2 with a Spring REST API (<https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>)

Learn how to document a Spring REST API using Swagger 2.

**Read more** (<https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>) →

## Introduction to Spring REST Docs (<https://www.baeldung.com/spring-rest-docs>)

This article introduces Spring REST Docs, a test-driven mechanism to generate documentation for RESTful services that is both accurate and readable.

**Read more** (<https://www.baeldung.com/spring-rest-docs>) →

## 2. Maven Configuration

First, we need to add a dependency to the zuul support from Spring Cloud to our UI application's *pom.xml*:

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-zuul</artifactId>
4   <version>1.0.4.RELEASE</version>
5 </dependency>
```

## 3. Zuul Properties

Next – we need to configure Zuul, and since we're using Spring Boot, we're going to do that in the *application.yml*:

```
1 zuul:
2   routes:
3     foos:
4       path: /foos/**
5       url: http://localhost:8081/spring-zuul-foos-resource/foos/
```

Note that:

- We are proxying to our resource server *Foos*.
- All requests from the UI that starts with `/foos/` will be routed to our *Foos* Resource server at `http://localhost:8081/spring-zuul-foos-resource/foos/`

Ok

## 4. The API

Our API application is a simple Spring Boot app.

Within this article, we're going to consider the API deployed in a server running **on port 8081**.

Let's first define the basic DTO for the Resource we're going to be using:

```
1 public class Foo {  
2     private long id;  
3     private String name;  
4  
5     // standard getters and setters  
6 }
```

And a simple controller:

```
1 @Controller  
2 public class FooController {  
3  
4     @RequestMapping(method = RequestMethod.GET, value = "/foos/{id}")  
5     @ResponseBody  
6     public Foo findById(  
7         @PathVariable long id, HttpServletRequest req, HttpServletResponse res) {  
8         return new Foo(Long.parseLong(randomNumeric(2)), randomAlphabetic(4));  
9     }  
10 }
```

## 5. The UI Application

Our UI application is also a simple Spring Boot application.

Within this article, we're going to consider the API deployed in a server running **on port 8080**.

Let's start with the main *index.html* – using a bit of AngularJS:

```

1 <html>
2 <body ng-app="myApp" ng-controller="mainCtrl">
3 <script src="angular.min.js"></script>
4 <script src="angular-resource.min.js"></script>
5
6 <script>
7 var app = angular.module('myApp', ["ngResource"]);
8
9 app.controller('mainCtrl', function($scope,$resource,$http) {
10     $scope.foo = {id:0 , name:"sample foo"};
11     $scope.foos = $resource("/foos/:fooId",{fooId:'@id'});
12
13     $scope.getFoo = function(){
14         $scope.foo = $scope.foos.get({fooId:$scope.foo.id});
15     }
16 });
17 </script>
18
19 <div>
20     <h1>Foo Details</h1>
21     <span>{{foo.id}}</span>
22     <span>{{foo.name}}</span>
23     <a href="#" ng-click="getFoo()">New Foo</a>
24 </div>
25 </body>
26 </html>

```

The most important aspect here is how we're accessing the API **using relative URLs!**

Keep in mind that the API application isn't deployed on the same server as the UI application, **so relative URLs shouldn't work**, and won't work without the proxy.

With the proxy, however, we're accessing the *Foo* resources through the Zuul proxy, which is of course configured to route these requests to wherever the API is actually deployed.

And finally, the actually Boot enabled application:

```

1 @EnableZuulProxy
2 @SpringBootApplication
3 public class UiApplication extends SpringBootServletInitializer {
4
5     public static void main(String[] args) {
6         SpringApplication.run(UiApplication.class, args);
7     }
8 }

```

Beyond the simple Boot annotation, notice that we're using the enable-style of annotation for the Zuul proxy as well, which is pretty cool, clean and concise.

## 6. Test The Routing

Now – let's test our UI application – as follows:

```

1 @Test
2 public void whenSendRequestToFooResource_thenOK() {
3     Response response = RestAssured.get("http://localhost:8080/foos/1 (http://localhost:8080/foos/1)");
4
5     assertEquals(200, response.getStatusCode());
6 }

```

## 7. A Custom Zuul Filter

We use cookies to improve your experience with the site. To find out more, you can read the full [Privacy and Cookie Policy](#) ([/privacy-policy](#))

Ok

There are multiple Zuul filters (<https://github.com/spring-cloud/spring-cloud-netflix/tree/master/spring-cloud-netflix-zuul/src/main/java/org/springframework/cloud/netflix/zuul/filters>) available, and we can also create our own custom one:

```

1  @Component
2  public class CustomZuulFilter extends ZuulFilter {
3
4      @Override
5      public Object run() {
6          RequestContext ctx = RequestContext.getCurrentContext();
7          ctx.addZuulRequestHeader("Test", "TestSample");
8          return null;
9      }
10
11     @Override
12     public boolean shouldFilter() {
13         return true;
14     }
15     // ...
16 }

```

This simple filter just adds a header called "Test" to the request – but of course, we can get as complex as we need to here augment our requests.

## 8. Test Custom Zuul Filter

Finally, let's test make sure our custom filter is working – first we will modify our *FooController* at Foos resource server:

```

1  @Controller
2  public class FooController {
3
4      @GetMapping("/foos/{id}")
5      @ResponseBody
6      public Foo findById(
7          @PathVariable long id, HttpServletRequest req, HttpServletResponse res) {
8          if (req.getHeader("Test") != null) {
9              res.addHeader("Test", req.getHeader("Test"));
10         }
11         return new Foo(Long.parseLong(randomNumeric(2)), randomAlphabetic(4));
12     }
13 }

```

Now – let's test our it out:

```

1  @Test
2  public void whenSendRequest_thenHeaderAdded() {
3      Response response = RestAssured.get("http://localhost:8080/foos/1 (http://localhost:8080/foos/1)");
4
5      assertEquals(200, response.getStatusCode());
6      assertEquals("TestSample", response.getHeader("Test"));
7  }

```

## 9. Conclusion

In this write-up, we focused on using Zuul to route requests from a UI application to a REST API. We successfully worked around CORS and the same-origin policy and we also managed to customize and augment the HTTP request in transit.

We use cookies to improve your experience with the site. To find out more, you can read the full [Privacy and Cookie Policy \(/privacy-policy\)](#)

Ok

The **full implementation** of this tutorial can be found in the GitHub project (<https://github.com/eugenp/tutorials/tree/master/spring-zuul>) – this is a Maven-based project, so it should be easy to import and run as it is.

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

>> CHECK OUT THE COURSE ([/ls-course-end](#))



## Build your Microservice Architecture with Spring Boot and Spring Cloud



Download Now



Guest

B SRIDHAR



Hi, I found your article interesting and useful for what I am currently working. I have some queries as below. I currently perform authentication using AUTH service based on spring security oauth2 and generate tokens. Each of the micro services have the @enableResourceServer annotation and so they are protected and decode the tokens using the AUTH service. Also, within every micro service I also do the url check to check if the user has the required authorities for a specific url pattern using spring security for http. I also have another custom service which I used for Access Control. Read more »

+ 0 -

🕒 3 years ago ^



Guest

Eugen Paraschiv (<https://www.baeldung.com/>)

You can certainly do that, yes – but I'm sure you know it's not going to be a trivial implementation. You'll need to run Spring Security in the app that's running the proxy, and then you'll have to move the ACL implementation there as well.

Also – the ACL entries are one thing, but these tie into your actual domain entities, so these will need to be accessible from this edge service as well.

Best of luck with the implementation – it sounds like it would indeed simplify the way you approach security by doing some consolidation. Cheers,

Eugen.

+ 0 -

🕒 3 years ago ^



Guest

antony raja



Hi

It is possible to have the Zuul with Hystrix , Ribbon and proxy as Standalone instead of Embedded proxy

+ 0 -

🕒 2 years ago ^



Guest

Grzegorz Piwowarek



It was not possible some time ago but there was an issue raised which seems to be closed for now so there is a huge chance that yes. <https://github.com/spring-cloud/spring-cloud-netflix/issues/14> (<https://github.com/spring-cloud/spring-cloud-netflix/issues/14>)

+ 0 -

🕒 2 years ago



Guest

Rudy Bonefas



Eugen,

Searched the web but can't figure out the following. Given a pre filter how do I block certain requests to my proxy and possibly send a response with a specific status and response body? I'm guessing throwing an exception in run() would block the request, but I would like to return a specific error response.

+ 1 -

🕒 3 years ago



Guest

reddynr



Hi,

Excellent example for proxy implementation. I have implemented. But my current requirement required to support multiple countries like below configurations. Please advise any sample example.

zuul:

routes:

foos:

path: /foos/\*\*

url: <http://localhost:8081/spring-zuul-foos-resource/CountryCode/foos> (<http://localhost:8081/spring-zuul-foos-resource/CountryCode/foos>)

+ 0 -

🕒 2 years ago ^

We use cookies to improve your experience with the site. To find out more, you can read the full [Privacy and Cookie Policy \(/privacy-policy\)](#)


Eugen Paraschiv (<https://www.baeldung.com/>)

Ok

- Guest

OK, that's pretty similar to the example here. What's the issue you're running into?  
Cheers,  
Eugen.

+ 0 -


🕒 2 years ago
- 

Guest

Shah Faisal

Hi Mr. Baeldung, Is it possible to send data in RequestBody while I'm using zuul proxy? I'm trying to send some data as json i.e. \$http({ method: 'POST', url: 'xyz/prod', headers : headers, data : requestData }).success(function(data) { console.log(data); }); But it seems zuul is removing request InputStream. Looking forward for your suggestion

+ 0 -

🕒 2 years ago ^
- 

Eugen Paraschiv (https://www.baeldung.com/)

Zuul is perfectly fine to handle an HTTP body, sure. That being said – there may be other reasons why the data is not being forwarded – it's hard to say without looking at the code.  
  
My suggestion is to isolate whatever is happening into a failing test and then post the question on StackOverflow. Feel free to followup with the link and I'd be happy to have a look as well.  
Cheers,  
Eugen.

+ 0 -

🕒 2 years ago

Comments are closed on this article!

 **ezoic** (https://www.ezoic.com/what-is-ezoic/)

report this ad

CATEGORIES

- SPRING (https://www.baeldung.com/category/spring/)
- REST (https://www.baeldung.com/category/rest/)
- JAVA (https://www.baeldung.com/category/java/)
- SECURITY (https://www.baeldung.com/category/security-2/)
- PERSISTENCE (https://www.baeldung.com/category/persistence/)
- JACKSON (https://www.baeldung.com/category/json/jackson/)
- HTTP CLIENT-SIDE (https://www.baeldung.com/category/http/)
- KOTLIN (https://www.baeldung.com/category/kotlin/)

SERIES

- JAVA "BACK TO BASICS" TUTORIAL (/java-tutorial)
  - JACKSON JSON TUTORIAL (/jackson)
  - HTTPCLIENT 4 TUTORIAL (/httpclient-guide)
- We use cookies to improve your experience with the site. To find out more, you can read the full [Privacy and Cookie Policy \(/privacy-policy\)](#)
- Ok



[REST WITH SPRING TUTORIAL \(/REST-WITH-SPRING-SERIES\)](#)  
[SPRING PERSISTENCE TUTORIAL \(/PERSISTENCE-WITH-SPRING-SERIES\)](#)  
[SECURITY WITH SPRING \(/SECURITY-SPRING\)](#)

## ABOUT

[ABOUT BAELDUNG \(/ABOUT\)](#)  
[THE COURSES \(HTTPS://COURSES.BAELDUNG.COM\)](https://courses.baeldung.com)  
[CONSULTING WORK \(/CONSULTING\)](#)  
[META BAELDUNG \(HTTP://META.BAELDUNG.COM/\)](http://meta.baeldung.com/)  
[THE FULL ARCHIVE \(/FULL\\_ARCHIVE\)](#)  
[WRITE FOR BAELDUNG \(/CONTRIBUTION-GUIDELINES\)](#)  
[EDITORS \(/EDITORS\)](#)  
[OUR PARTNERS \(/PARTNERS\)](#)  
[ADVERTISE ON BAELDUNG \(/ADVERTISE\)](#)

[TERMS OF SERVICE \(/TERMS-OF-SERVICE\)](#)  
[PRIVACY POLICY \(/PRIVACY-POLICY\)](#)  
[COMPANY INFO \(/BAELDUNG-COMPANY-INFO\)](#)  
[CONTACT \(/CONTACT\)](#)