Spring Boot REST Internationalization

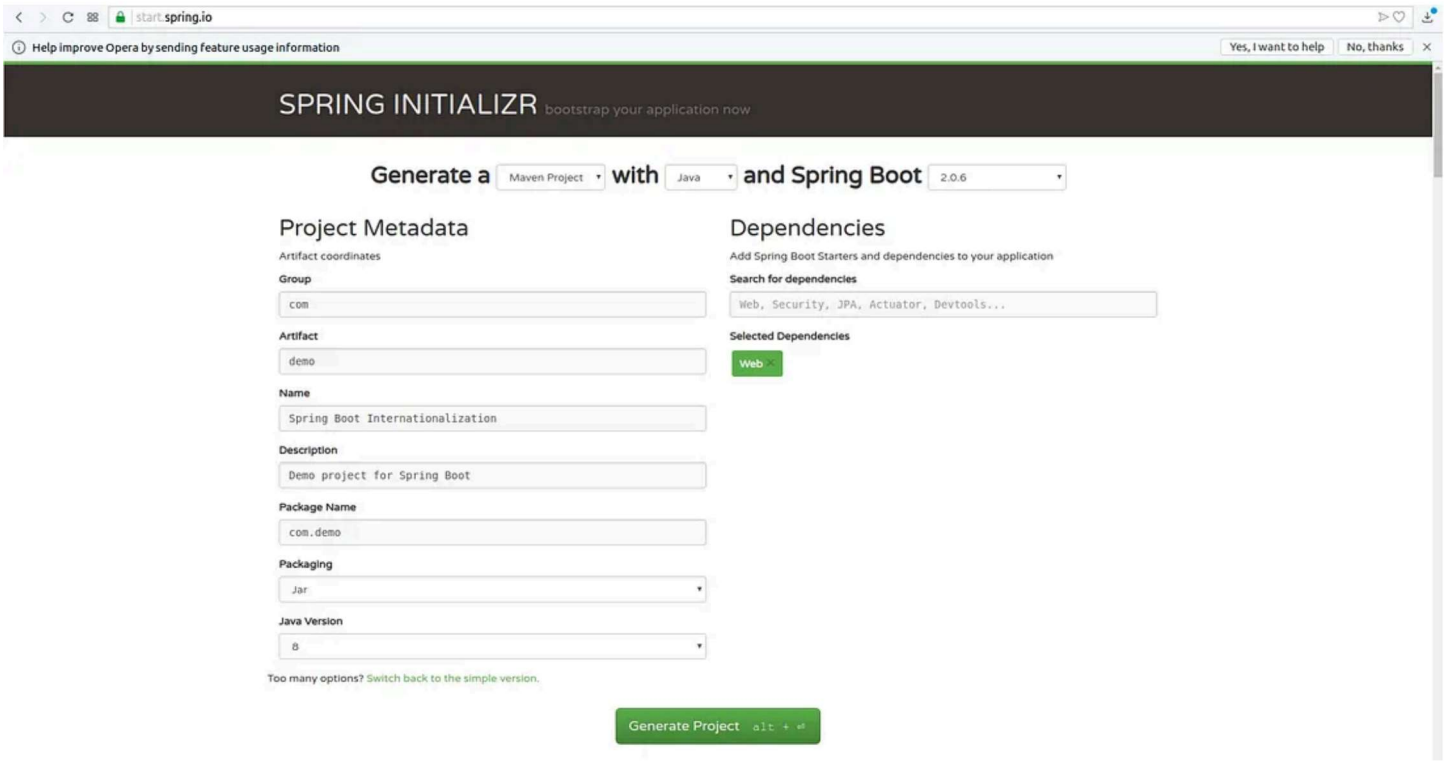
Ihor Kosandiak · Follow
4 min read · Oct 30, 2018

Have you ever faced with challenging task of providing service to your users in their native language? If so — then you may be interested in finding a good approach of doing this.

This guide will show you how is easy, in just couple straightforward steps to make your Spring Boot application to have internationalization. And to handle locales in one single place.

Nice to see you here guys! Here we will speak about adding internationalization to your existing Spring Boot project. The question of application internationalization becomes not trivial when you are working on the project that should serve users from different countries with different languages. In this cases you will need to provide your user from China with response message in Chinese, and user from France with message in French, so everyone would feel comfortable with your application. So let's take a look how to implement this in Spring Boot.

Let's create project using **Spring Initializer** that makes project creation much easier.



Then click download project, unzip it and open with your favorite Idea. I'll use IntelliJ IDEA.

First thing what I'll create is our *CustomLocaleResolver* class that will be responsible for defining user's locale.

```
@Configuration
public class CustomLocaleResolver
             extends AcceptHeaderLocaleResolver
             implements WebMvcConfigurer {

   List<Locale> LOCALES = Arrays.asList(
         new Locale("en"),
```

```
            new Locale("fr"));

    @Override
    public Locale resolveLocale(HttpServletRequest request) {
        String headerLang = request.getHeader("Accept-Language");
        return headerLang == null || headerLang.isEmpty()
                ? Locale.getDefault()
                : Locale.lookup(Locale.LanguageRange.parse(headerLang),
LOCALES);
    }

    @Bean
    public ResourceBundleMessageSource messageSource() {
        ResourceBundleMessageSource rs = new
ResourceBundleMessageSource();
        rs.setBasename("messages");
        rs.setDefaultEncoding("UTF-8");
        rs.setUseCodeAsDefaultMessage(true);
        return rs;
    }
}
```

So, here we told that we have 2 locales supported in the project: ***en*** and ***fr***. The locale should be passed in the header called "**Accept-Language**". So, if the header is present and it is not empty, we will use locale from it, otherwise — we'll use default locale, which is ***en***.

Next let's create our class that will be responsible for choosing right message according to specified locale. I'll call it ***Translator***, and it will have one single method, that will accept message code that should be translated.

```
@Component
public class Translator {

    private static ResourceBundleMessageSource messageSource;

    @Autowired
    Translator(ResourceBundleMessageSource messageSource) {
        Translator.messageSource = messageSource;
    }

    public static String toLocale(String msgCode) {
        Locale locale = LocaleContextHolder.getLocale();
        return messageSource.getMessage(msg, null, locale);
    }
}
```

As you can see the messageSource.getMessage(...) accepts parameter *"msg"*. But this is not exactly the message that should be translated. It is just message code. For now we don't have any message codes, so let's create them.

Under the *resources* folder, create two files: *messages.properties* and *messages_fr.properties*.

Here is the content of messages.properties:

```
hello=Hello World!
welcome=Welcome to this guide!
```

And here is content of messages_fr.properties:

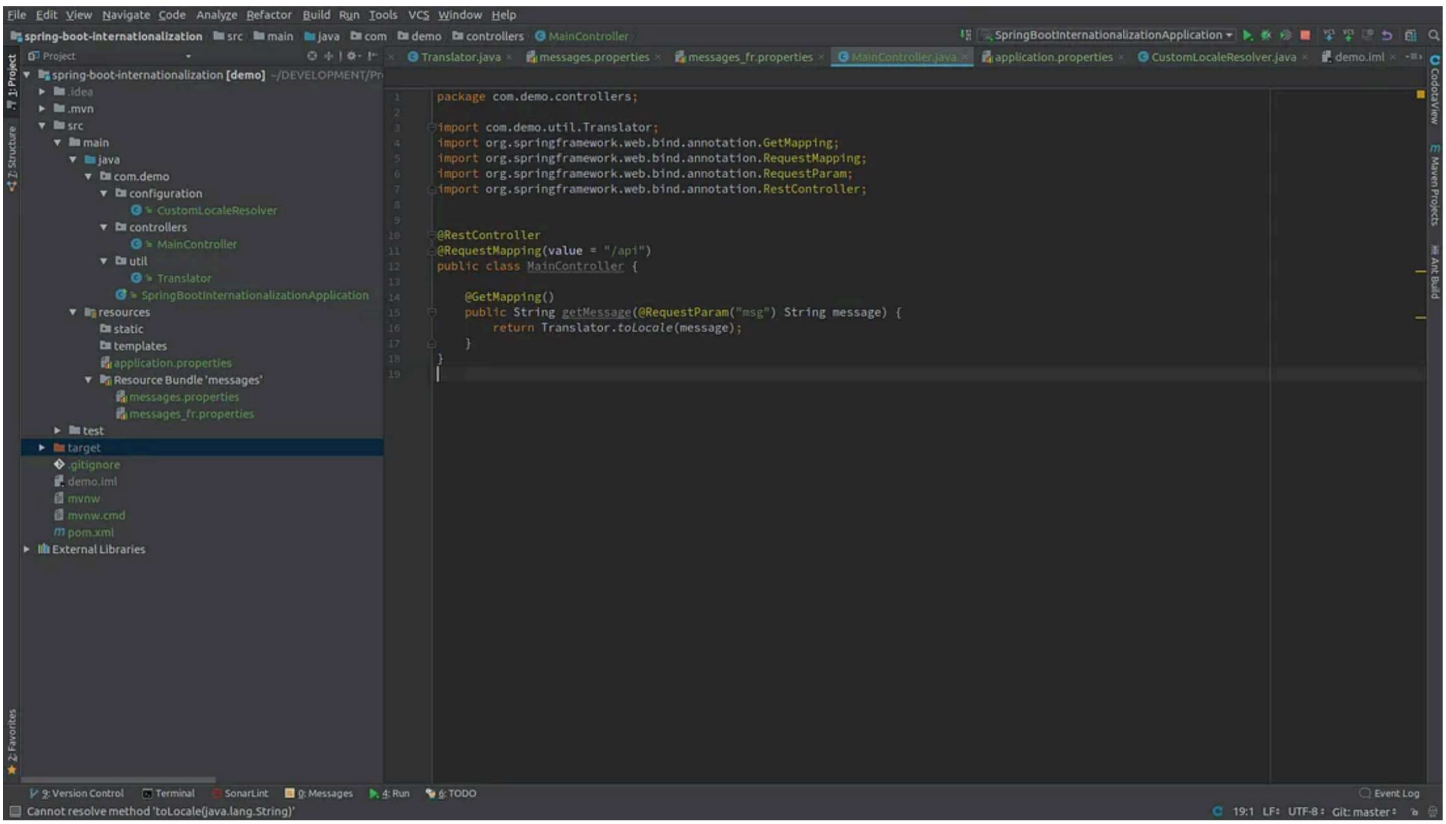
```
hello=Bonjour le Monde!
welcome=Bienvenue dans ce guide!
```

So, here we already can see our message codes. They are "*hello*" and "*welcome*". And now you can understand what code should we pass to *toLocale(String msgCode)* method in order to get appropriate message according to user's locale.

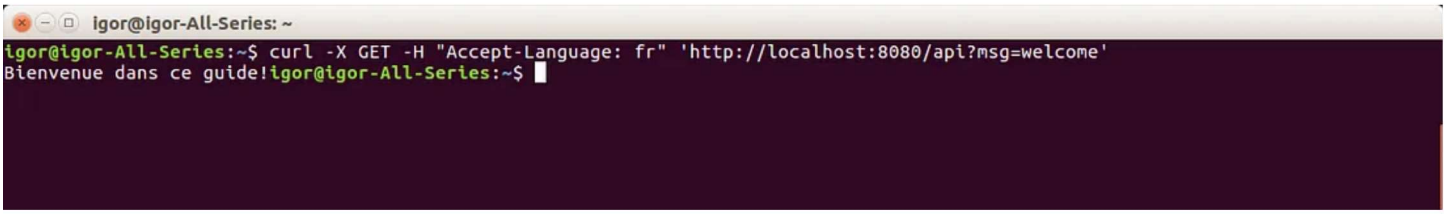
And probably the last step is to create simple controller, let's name it *MainController* that will have just one endpoint, that will accept message code, which we'll pass into the HTTP request as a request parameter.

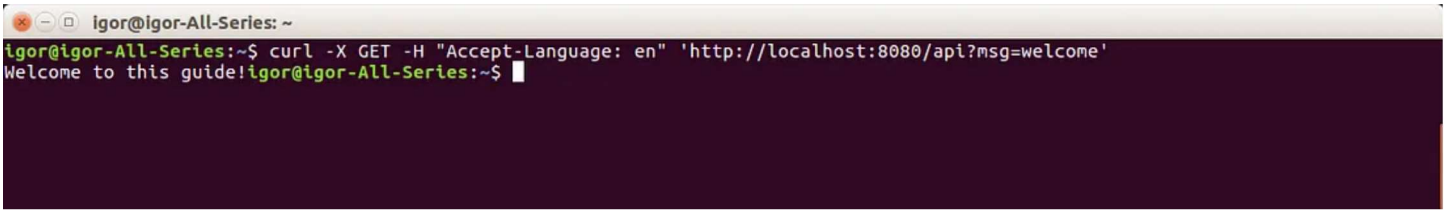
```
@RestController
@RequestMapping(value = "/api")
public class MainController {

   @GetMapping()
   public String getMessage(@RequestParam("msg") String msg) {
      return Translator.toLocale(msg);
   }
}
```

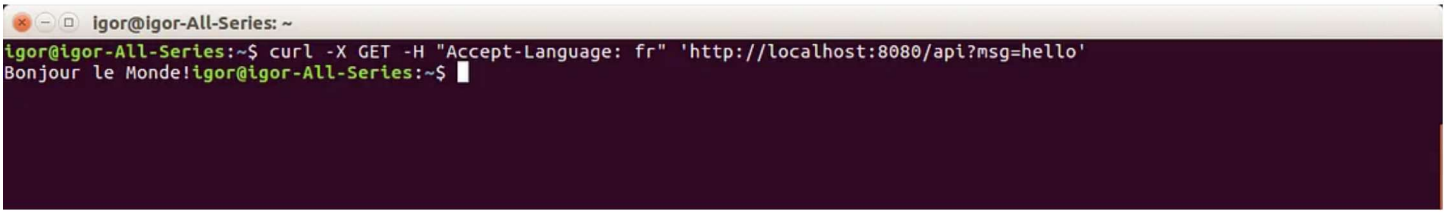
So now, let's take a look at our project structure.

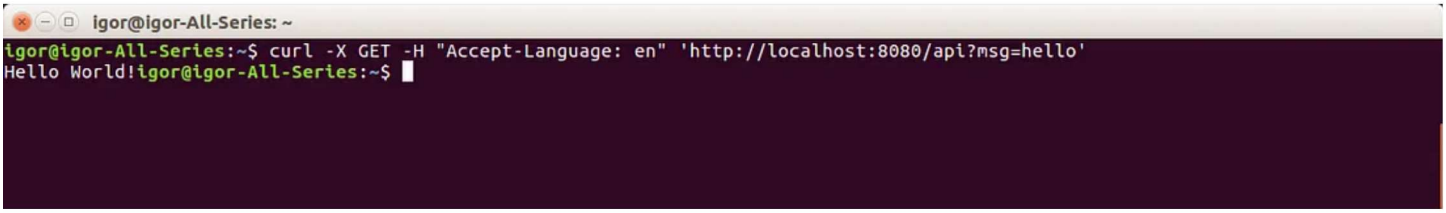


And make simple requests using CURL:









So as you see, responses are different based on value of "*Accept-Language*" header passed in the request. This way, we don't need to check what was passed in the request in each controller method, and then pass it further to service layers. We now can do this in one single place, which is ***CustomLocaleResolver*** *class.*

Basically that's all.

Thank you for your time. Hope you got interesting stuff for you, and enjoyed the reading. As always you can find code sample on my GitHub account.

And if you found this article useful — help others to find it too— press the **"clap"** button. You may press and hold it and give up to 50 claps! 😀

. . .

Java Spring Boot Spring Rest Learning

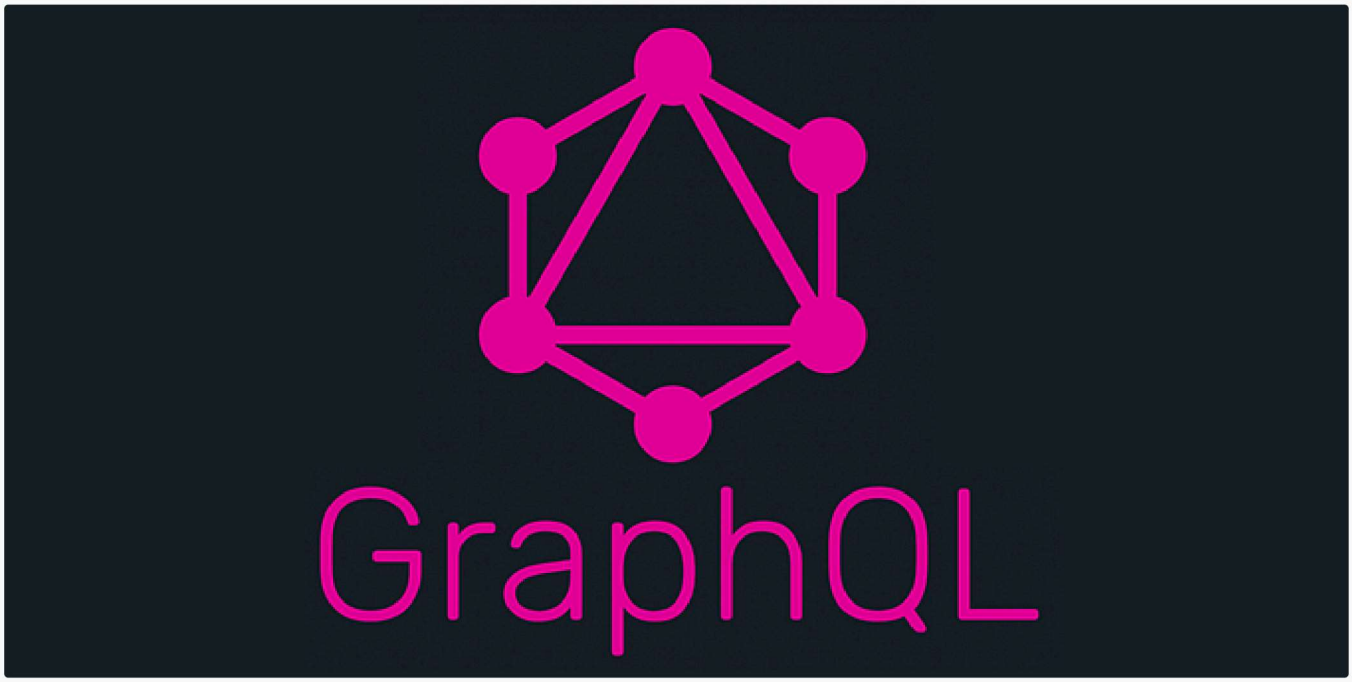
Follow

Written by Ihor Kosandiak

823 Followers

Java Software Developer. Passionate about new technologies. Sharing own experience with others.

More from Ihor Kosandiak

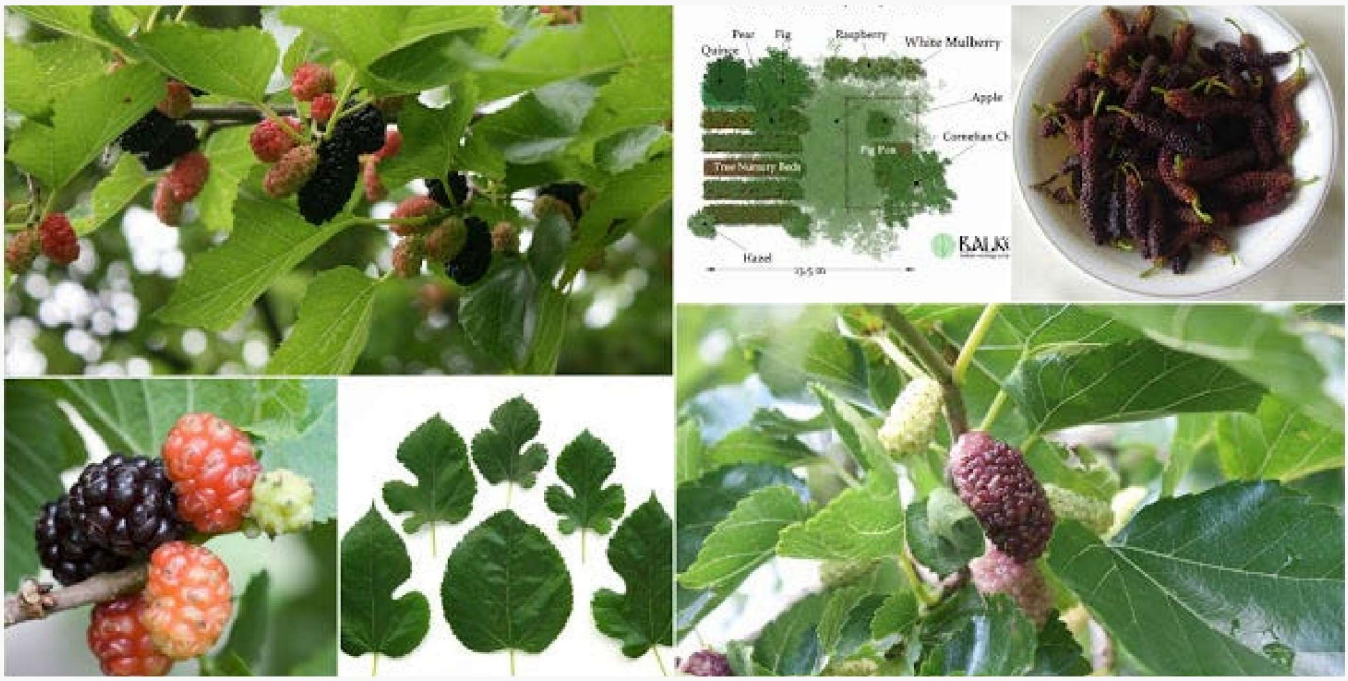


Ihor Kosandiak in ORIL Software

Spring Boot with GraphQL and MongoDB

If you read this article, than you probably heard about GraphQL.

Nov 30, 2017 2.2K 12



Paul Alfrey

Mo' Mulberry — A guide to probably everything you need to know about growing Mulberry

Not many plants offer so much to the grower while demanding so little in return. A tree that requires so little attention and care, that...

Oct 11, 2017 223 4

Shashank Yadav

Understanding Vector Quantized Variational Autoencoders (VQ-VAE)

From my most recent escapade into the deep learning literature I present to you this paper by Oord et. al. which presents the idea of...

Sep 1, 2019 514 10

Ihor Kosandiak

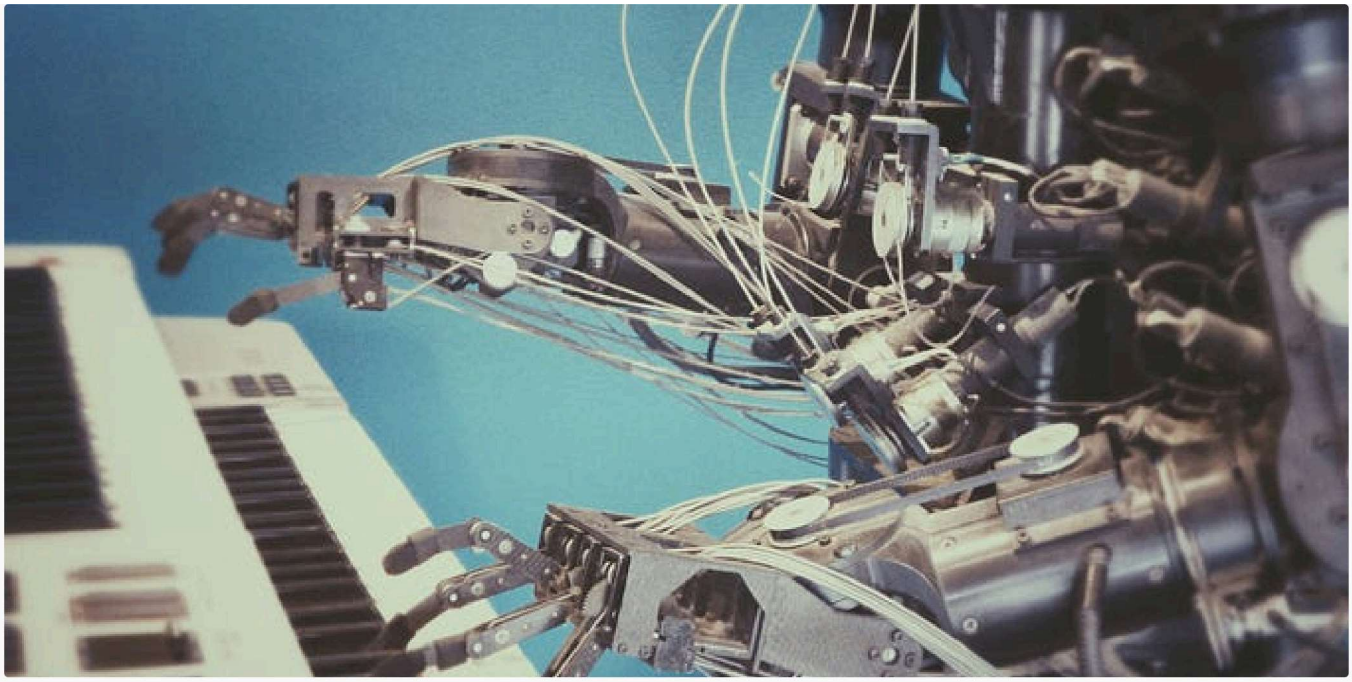
Spring Cloud Gateway security with JWT

There is a clear understanding that everything that is exposed to the Internet should be secured. Especially when you create a software...

Feb 24, 2021 488 2

See all from Ihor Kosandiak

Recommended from Medium

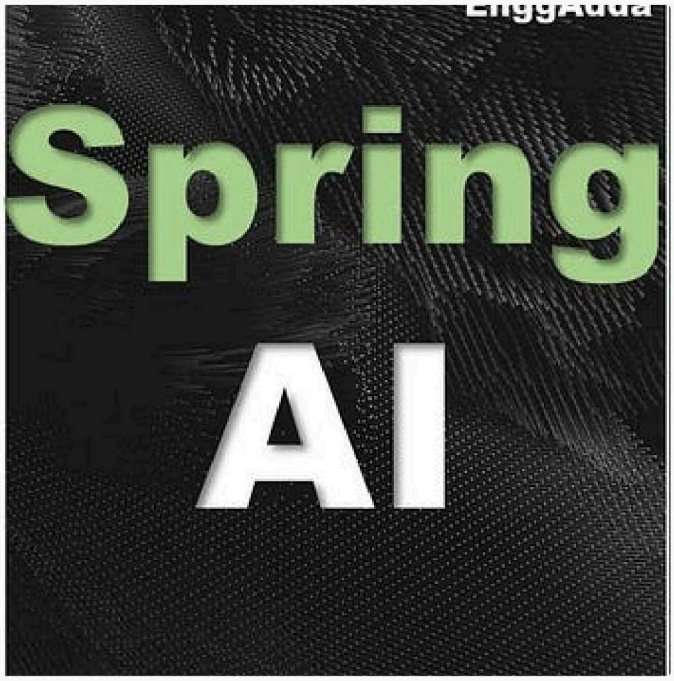


Nasim Salmany

DocChat: Interactive chat with documents using Spring AI, OpenAI

More intelligent access to documents using AI

Apr 29 40



Ethan Leung

Spring AI : Java Integration with Large Language Models Simplified

Hello, I'm Ethan, a Java backend development engineer. I have a keen interest in the application of large models. In this AI spring, we...

Apr 17 448 7

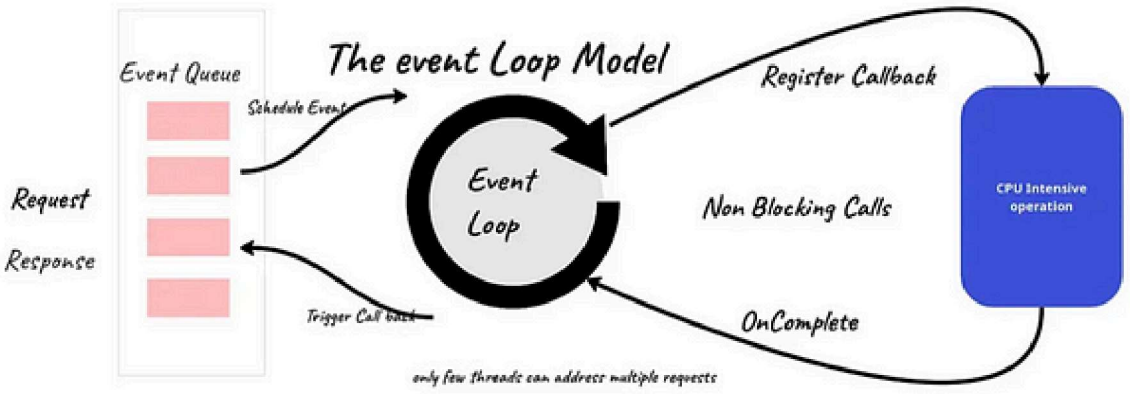
Lists

Self-Improvement 101
20 stories · 2223 saves

How to Find a Mentor
11 stories · 600 saves

Good Product Thinking
11 stories · 615 saves

General Coding Knowledge
20 stories · 1339 saves



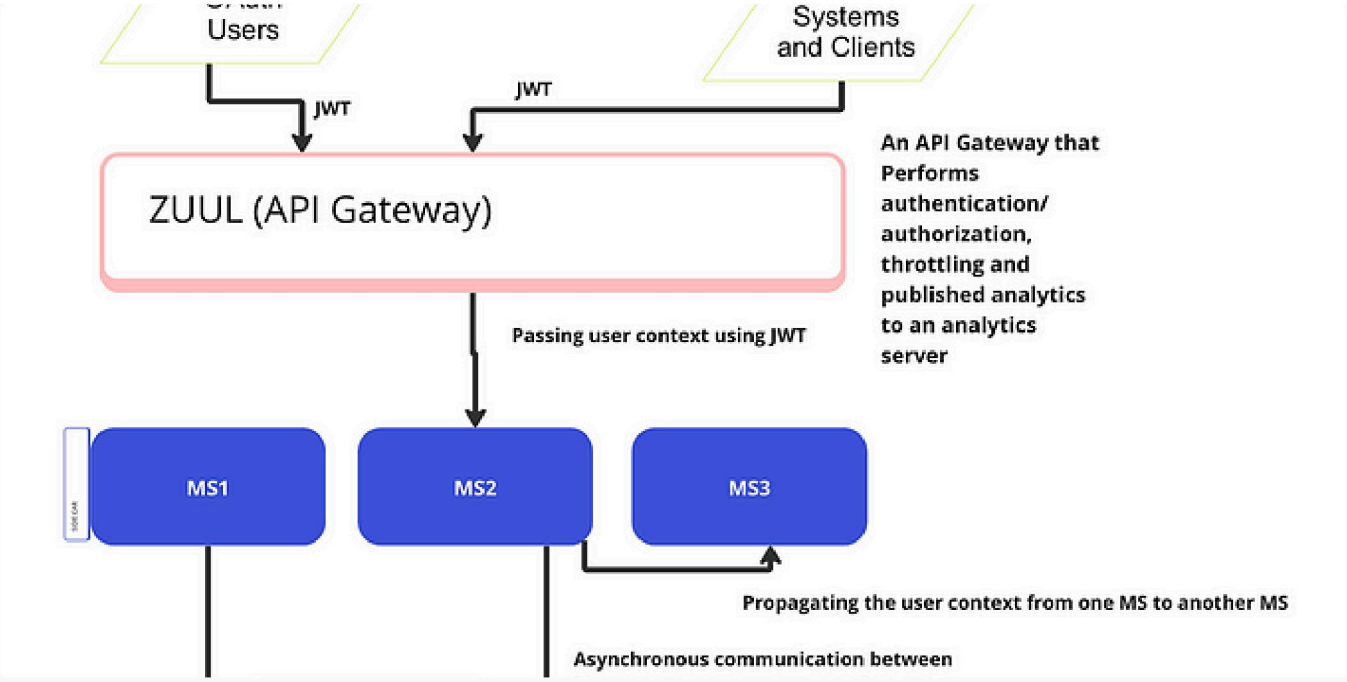
Event Loop Design Construct

Vikas Taank

Understanding the Event Loop Model

Please go through an auxiliary learning if you like before you jump on the event loop.

Mar 8 15

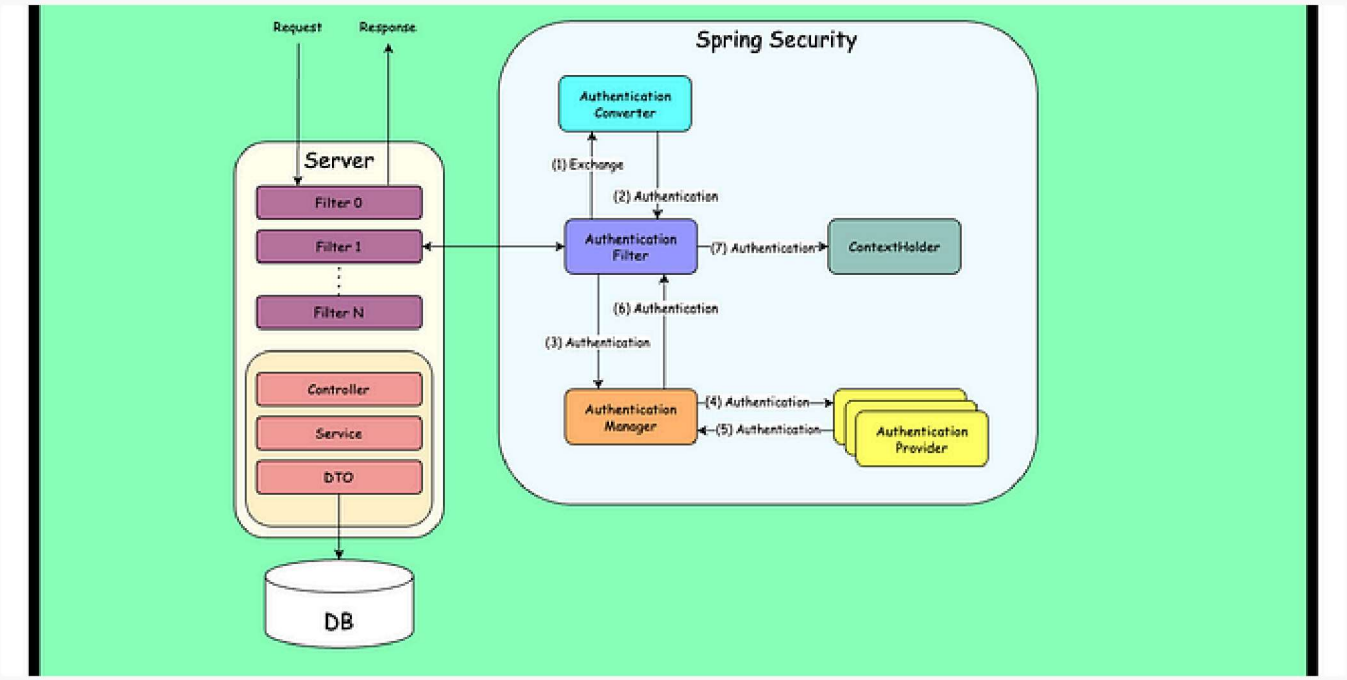


Vikas Taank

Essential Interview Questions for RESTful Micro services

What are the differnt HTTP methods?

Jun 20 10

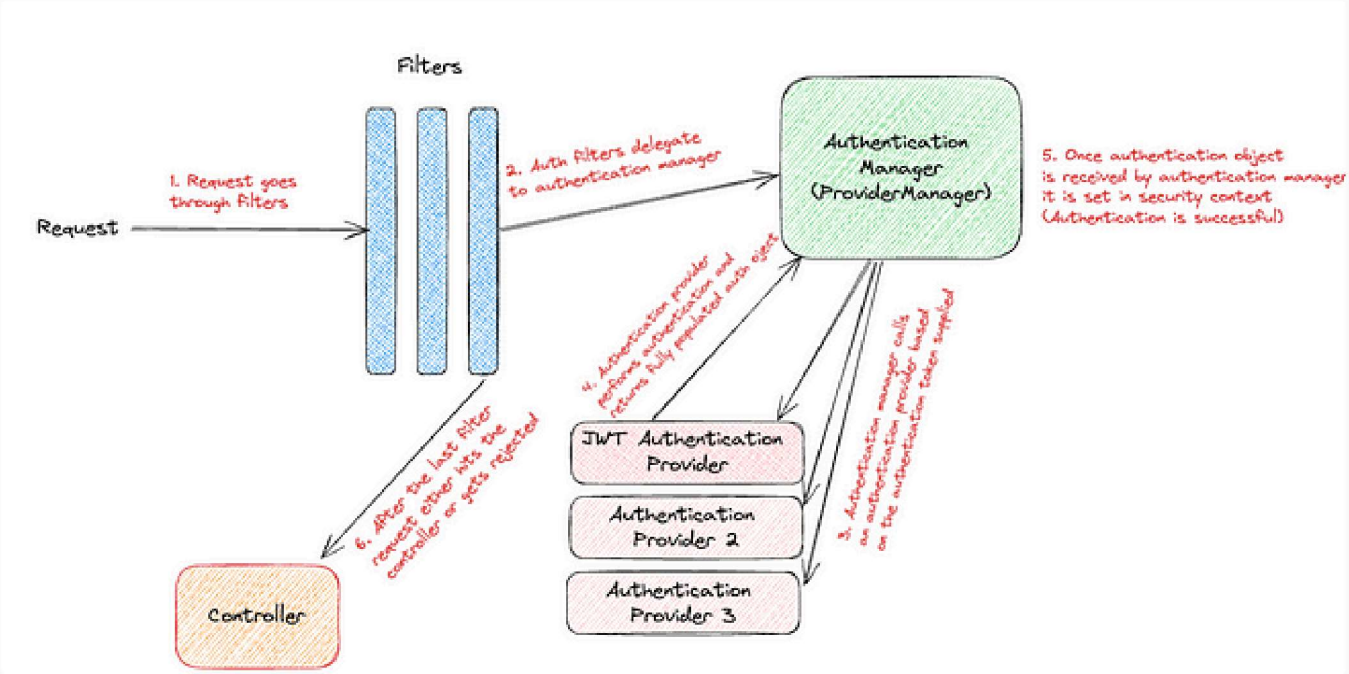


Kritagya Khandelwal

Customizing Spring Security in Springboot WebFlux Services

Implementation of custom authentication flow for spring webflux services, with the power of spring security, method level authorization.

Mar 16 1



Muhammet Özen

Understanding Spring Security

Introduction

Feb 13 26

See more recommendations