ALL GUIDES

Consuming a RESTful Web Service

This guide walks you through the process of creating an application that consumes a RESTful web service.

What You Will Build

You will build an application that uses Spring's `RestTemplate` to retrieve a random Spring Boot quotation at http://localhost:8080/api/random.

What You Need

- About 15 minutes
- A favorite text editor or IDE
- Java 17 or later
- Gradle 7.5+ or Maven 3.5+
- You can also import the code straight into your IDE:
  - Spring Tool Suite (STS)
  - IntelliJ IDEA
  - VSCode

How to complete this guide

Like most Spring Getting Started guides, you can start from scratch and complete each step or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to Starting with Spring Initializr.

To **skip the basics**, do the following:

- Download and unzip the source repository for this guide, or clone it using Git: `git clone https://github.com/spring-guides/gs-consuming-rest.git`
- cd into `gs-consuming-rest/initial`
- Jump ahead to fetching a REST Resource.

When you finish, you can check your results against the code in `gs-consuming-rest/complete`.

Starting with Spring Initializr

You can use this pre-initialized project and click Generate to download a ZIP file. This project is configured to fit the examples in this tutorial.

To manually initialize the project:

1. Navigate to https://start.spring.io. This service pulls in all the dependencies you need for an application and does most of the setup for you.
2. Choose either Gradle or Maven and the language you want to use. This guide assumes that you chose Java.
3. Click **Dependencies** and select **Spring Web**.
4. Click **Generate**.
5. Download the resulting ZIP file, which is an archive of a web application that is configured with your choices.

> If your IDE has the Spring Initializr integration, you can complete this process from your IDE.

> You can also fork the project from Github and open it in your IDE or other editor.

Fetching a REST Resource

With project setup complete, you can create a simple application that consumes a RESTful service.

Before you can do so, you need a source of REST resources. We have provided an example of such a service at https://github.com/spring-guides/quoters. You can run that application in a separate terminal and access the result at http://localhost:8080/api/random. That address randomly fetches a quotation about Spring Boot and returns it as a JSON document. Other valid addresses include http://localhost:8080/api/ (for all the quotations) and http://localhost:8080/api/1 (for the first quotation), http://localhost:8080/api/2 (for the second quotation), and so on (up to 10 at present).

If you request that URL through a web browser or curl, you receive a JSON document that looks something like this:

```
{
  type: "success",
  value: {
    id: 10,
    quote: "Really loving Spring Boot, makes stand alone Spring apps easy."
  }
}
```

That is easy enough but not terribly useful when fetched through a browser or through curl.

A more useful way to consume a REST web service is programmatically. To help you with that task, Spring provides a convenient template class called `RestTemplate`. `RestTemplate` makes interacting with most RESTful services a one-line incantation. And it can even bind that data to custom domain types.

First, you need to create a domain class to contain the data that you need. The following listing shows the `Quote` record class, which you can use as your domain class:

`src/main/java/com/example/consumingrest/Quote.java`

```
package com.example.consumingrest;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = true)
public record Quote(String type, Value value) { }
```

This simple Java record class is annotated with `@JsonIgnoreProperties` from the Jackson JSON processing library to indicate that any properties not bound in this type should be ignored.

To directly bind your data to your custom types, you need to specify the variable name to be exactly the same as the key in the JSON document returned from the API. In case your variable name and key in JSON doc do not match, you can use `@JsonProperty` annotation to specify the exact key of the JSON document. (This example matches each variable name to a JSON key, so you do not need that annotation here.)

You also need an additional class, to embed the inner quotation itself. The `Value` record class fills that need and is shown in the following listing (at `src/main/java/com/example/consumingrest/Value.java`):

```
package com.example.consumingrest;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = true)
public record Value(Long id, String quote) { }
```

This uses the same annotations but maps onto other data fields.

Finishing the Application

The Initializr creates a class with a `main()` method. The following listing shows the class the Initializr creates (at `src/main/java/com/example/consumingrest/ConsumingRestApplication.java`):

```
package com.example.consumingrest;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ConsumingRestApplication {

	public static void main(String[] args) {
		SpringApplication.run(ConsumingRestApplication.class, args);
	}

}
```

Now you need to add a few other things to the `ConsumingRestApplication` class to get it to show quotations from our RESTful source. You need to add:

- A logger, to send output to the log (the console, in this example).
- A `RestTemplate`, which uses the Jackson JSON processing library to process the incoming data.
- A `CommandLineRunner` that runs the `RestTemplate` (and, consequently, fetches our quotation) on startup.

The following listing shows the finished `ConsumingRestApplication` class (at `src/main/java/com/example/consumingrest/ConsumingRestApplication.java`):

```
package com.example.consumingrest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Profile;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class ConsumingRestApplication {

	private static final Logger log = LoggerFactory.getLogger(ConsumingRestApplication.class);

	public static void main(String[] args) {
		SpringApplication.run(ConsumingRestApplication.class, args);
	}

	@Bean
	public RestTemplate restTemplate(RestTemplateBuilder builder) {
		return builder.build();
	}

	@Bean
	@Profile("!test")
	public CommandLineRunner run(RestTemplate restTemplate) throws Exception {
		return args -> {
			Quote quote = restTemplate.getForObject(
					"http://localhost:8080/api/random", Quote.class);
			log.info(quote.toString());
		};
	}
}
```

Finally, you need to set the server port. The quoters application uses the default server port, 8080, so this application cannot also use the same port. You can set the server port to 8081 by adding the following line to application properties (which the Initializr created for you):

`server.port=8081`

Running the Application

You can run the application from the command line with Gradle or Maven. You can also build a single executable JAR file that contains all the necessary dependencies, classes, and resources and run that. Building an executable jar makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

If you use Gradle, you can run the application by using `./gradlew bootRun`. Alternatively, you can build the JAR file by using `./gradlew build` and then run the JAR file, as follows:

```
java -jar build/libs/gs-consuming-rest-0.1.0.jar
```

If you use Maven, you can run the application by using `./mvnw spring-boot:run`. Alternatively, you can build the JAR file with `./mvnw clean package` and then run the JAR file, as follows:

```
java -jar target/gs-consuming-rest-0.1.0.jar
```

> The steps described here create a runnable JAR. You can also build a classic WAR file.

You should see output similar to the following but with a random quotation:

```
2019-08-22 14:06:46.586  INFO 42940 --- [           main] c.e.c.ConsumingRestApplication           : Quote{type
```

> If you see an error that reads,
> `Could not extract response: no suitable HttpMessageConverter found for response type [class com.example.consu`
> it is possible that you are in an environment that cannot connect to the backend service (which sends JSON if you can reach it). Maybe corporate proxy. Try setting the `http.proxyHost` and `http.proxyPort` system properties to values appropriate for your environ

Summary

Congratulations! You have just developed a simple REST client by using Spring Boot.

See Also

The following guides may also be helpful:

- Building a RESTful Web Service
- Consuming a RESTful Web Service with AngularJS
- Consuming a RESTful Web Service with jQuery
- Consuming a RESTful Web Service with rest.js
- Accessing Gemfire Data with REST
- Accessing MongoDB Data with REST
- Accessing data with MySQL
- Accessing JPA Data with REST
- Accessing Neo4j Data with REST
- Securing a Web Application
- Building an Application with Spring Boot
- Creating API Documentation with Restdocs
- Enabling Cross Origin Requests for a RESTful Web Service
- Building a Hypermedia-Driven RESTful Web Service

Want to write a new guide or contribute to an existing one? Check out our contribution guidelines.

> All guides are released with an ASLv2 license for the code, and an Attribution, NoDerivatives creative commons license for the writing.

Get the Code

Go To Repo