# Custom Spring Boot Library for Binding Contextual Data

## The Problem

Our company decided to rewrite our existing platform using a modularized architecture. The new platform is made up of Dockerized Spring Boot apps deployed to Kubernetes. The modules can communicate with each other via REST APIs and JMS messaging, both requiring JWT authentication. The JWT contains reserved claims and custom claims that identify the user and provide contextual data (iss, sub, exp, user Id, user permissions, etc). We wanted to authenticate the JWT for each received REST call and JMS message and make the contextual data available throughout the app. Unfortunately, Spring’s @RequestScope can only be used in web requests so we needed another solution that would work for both REST and JMS.

### Our Solution

Our solution was simple. We decided to build a common library that would authenticate the JWT for each received REST call and JMS message and then bind the contextual data to each thread via ThreadLocal to make the contextual data available throughout the app. This is actually what Spring does when @RequestScope is used.

An example of the common library and how to use it can be found here: <https://github.com/valeriearena/contextdatalib-for-http-and-jms>.

### The Common Library

The most important classes in the common library are ExampleContextData, ContextData, and ContextService in the “context” package. These classes are responsible for managing the context data. In addition to the “context” package, CommonServletFilter, the CommonWebClient, and the CommonJmsInterceptor propagate the context data across HTTP and JMS. Each class is described below.

The library also applies custom Spring Boot autoconfigurations that are required to add the JWT to outgoing REST calls and JMS messages (in addition to customizing HttpSecurity so that each web request is authenticated).

The library does not have a Spring Boot launch class because the common library is not intended as a deployable app.

Please see [https://github.com/valeriearena/contextdatalib-for-http-and-jms/tree/master/common/src/main/java/com/example/common](https://github.com/valeriearena/contextdatalib-for-http-and-jms/tree/master/common/src/main/java/com/example/commont).

#### ExampleContextData, ContextData, and ContextService

ExampleContextData contains the specific context data for the user. It contains the actual JWT, as well as the parsed claims contained in the JWT.

ContextData is responsible for managing ExampleContextData and binds it to ThreadLocal, retrieves it from ThreadLocal, and then removes it from ThreadLocal.

ContextService manages ContextData and abstracts the use of ThreadLocal. When ExampleContextData needs to be added, ContextService adds it. When ExampleContextData needs to be removed, ContextService removes it.

Please see <https://github.com/valeriearena/contextdatalib-for-http-and-jms/tree/master/common/src/main/java/com/example/common/context>.

#### CommonServletFilter

CommonServletFilter intercepts each incoming web request to retrieve the JWT Bearer token from the Authorization header. CommonServletFilter authenticates the JWT and parses the claims to build ExampleContextData (via the JwtService). It then uses ContextService to bind ExampleContextData to ThreadLocal via ContextData.

When the request completes, CommonServletFilter uses ContextService to remove ExampleContextData from ThreadLocal. Removing ExampleContextData from ThreadLocal when processing completes is very important in order to avoid memory leaks!

Please see <https://github.com/valeriearena/contextdatalib-for-http-and-jms/blob/master/common/src/main/java/com/example/common/filter/CommonFilter.java>.

#### CommonWebClient

The CommonWebClient retrieves the JWT Bearer token from ContextService and adds it to the Authorization header of the outgoing web request before submitting the request.

Please see <https://github.com/valeriearena/contextdatalib-for-http-and-jms/blob/master/common/src/main/java/com/example/common/rest/webclient/CommonWebClient.java>.

#### CommonJmsInterceptor

CommonJmsInterceptor intercepts each published JMS message and retrieves the JWT Bearer token from CotextService and adds it to the Authorization property of the JMS message.

Just before the JMS message is received by a consumer, CommonJmsInterceptor intercepts the messaging to retrieve the JWT Bearer from the Authorization property. CommonJmsInterceptor authenticates the JWT and parses the claims to build ExampleContextData (via the JwtService). It then uses ContextService to bind ExampleContextData to ThreadLocal via ContextData.

When the JMS message has been processed, CommonJmsInterceptor uses ContextService to remove ExampleContextData from ThreadLocal. Removing ExampleContextData from ThreadLocal when processing completes is very important in order to avoid memory leaks!

Pleas see <https://github.com/valeriearena/contextdatalib-for-http-and-jms/blob/master/common/src/main/java/com/example/common/aop/CommonJmsInterceptor.java>.

### Using the Common Library

Modules that use the common library must declare it as a dependency in the gradle script. Modules must also specify the base package of the common library in the @SpringBootApplication annotation of the Spring Boot launch class so that services in the common library are automatically discovered and registered by Spring.

And then you are ready to use the common library! Just inject ContextData into your REST controllers, services, JMS publishers and subscribers and use the getters to access the context data!

Please see <https://github.com/valeriearena/contextdatalib-for-http-and-jms/tree/master/moduleA> and https://github.com/valeriearena/contextdatalib-for-http-and-jms/tree/master/moduleB.

### Github Example

Please try it out for yourself! Follow the steps in the README and add the specified breakpoints to see how it works!